MULTI-SCALE DATA STORAGE SCHEMES FOR SPATIAL INFORMATION SYSTEMS

John Mark Ware

Department of Computer Studies

A thesis submitted in partial fulfilment of the requirements of the University of Glamorgan/Prifysgol Morgannwg for the degree of Doctor of Philosophy.

This thesis programme was carried out in collaboration with the British Geological Survey.

April 1994

Table of Contents

Table of Figures and Plates

Acknowledgements

I would like to give my unreserved thanks to my Director of Studies, Dr. Chris Jones, for his continual help and encouragement throughout the course of this research. I am also indebted to the advice and support given to me by my secondary supervisors, Dr. Robert Davies and Dr. Graham Tough.

I would also like to acknowledge the support given by the British Geological Survey, Keyworth. In particular I would like to thank Dr. John Rees, Mr. Konrad Dabek and Mr. Keith Ambrose for many helpful discussions and other assistance provided during the course of the project.

Thanks are also due to the many members of staff at the University who have supported my research. In particular I wish to thank my fellow researchers, Dr. David Kidner and Mr. Geraint Bundy.

Certificate of Research

This is to certify that, except where specific reference is made, the work presented in this thesis is the result of the investigation undertaken by the candidate.

Candidate .....

Director of Studies .....

Declaration

This is to certify that neither this thesis or any part of it has been presented or is being currently submitted in candidature for any degree other than the degree of Doctor of Philosophy of the University of Glamorgan.

Candidate .....

Multi-Scale Data Storage Schemes for Spatial Information Systems

John Mark Ware

The University of Glamorgan

Abstract

This thesis documents a research project that has led to the design and prototype implementation of several data storage schemes suited to the efficient multi-scale representation of integrated spatial data. Spatial information systems will benefit from having data models which allow for data to be viewed and analysed at various levels of detail, while the integration of data from different sources will lead to a more accurate representation of reality.

The work has addressed two specific problems. The first concerns the design of an integrated multi-scale data model suited for use within Geographical Information Systems. This has led to the development of two data models, each of which allow for the integration of terrain data and topographic data at multiple levels of detail. The models are based on a combination of adapted versions of three previous data structures, namely, the constrained Delaunay pyramid, the line generalisation tree and the fixed grid.

The second specific problem addressed in this thesis has been the development of an integrated multi-scale 3-D geological data model, for use within a Geoscientific Information System. This has resulted in a data storage scheme which enables the integration of terrain data, geological outcrop data and borehole data at various levels of detail.

The thesis also presents details of prototype database implementations of each of the new data storage schemes. These implementations have served to demonstrate the feasibility and benefits of an integrated multi-scale approach.

The research has also brought to light some areas that will need further research before fully functional systems are produced. The final chapter contains, in addition to conclusions made as a result of the research to date, a summary of some of these areas that require future work.

Chapter 1

Project Introduction

1.1 Introduction.

This chapter serves as an introduction to the thesis. Section 1.2 provides a brief overview of Geographic Information Systems, and their 3-D counterpart Geoscientific Information Systems. The thesis is primarily concerned with the data models on which such spatial information systems are based. Therefore, early mention is made of the types of data which need to be accommodated by these spatial data models. Some of the limitations of current data models are discussed in Section 1.3, with particular regard to the problem of multi-scale data access. The aims of the project, which are to overcome some of these limitations, are also outlined. The chapter concludes with an overview of the rest of the thesis.

1.2 Background.

Many definitions have been given in the literature to describe what constitutes a Geographic Information System (GIS), some of which are included in Maguire [1]. In summary, a GIS can be thought of as a computer system that can acquire, store, update, process and display geographical (that is, spatially referenced) data (Figure 1.1).

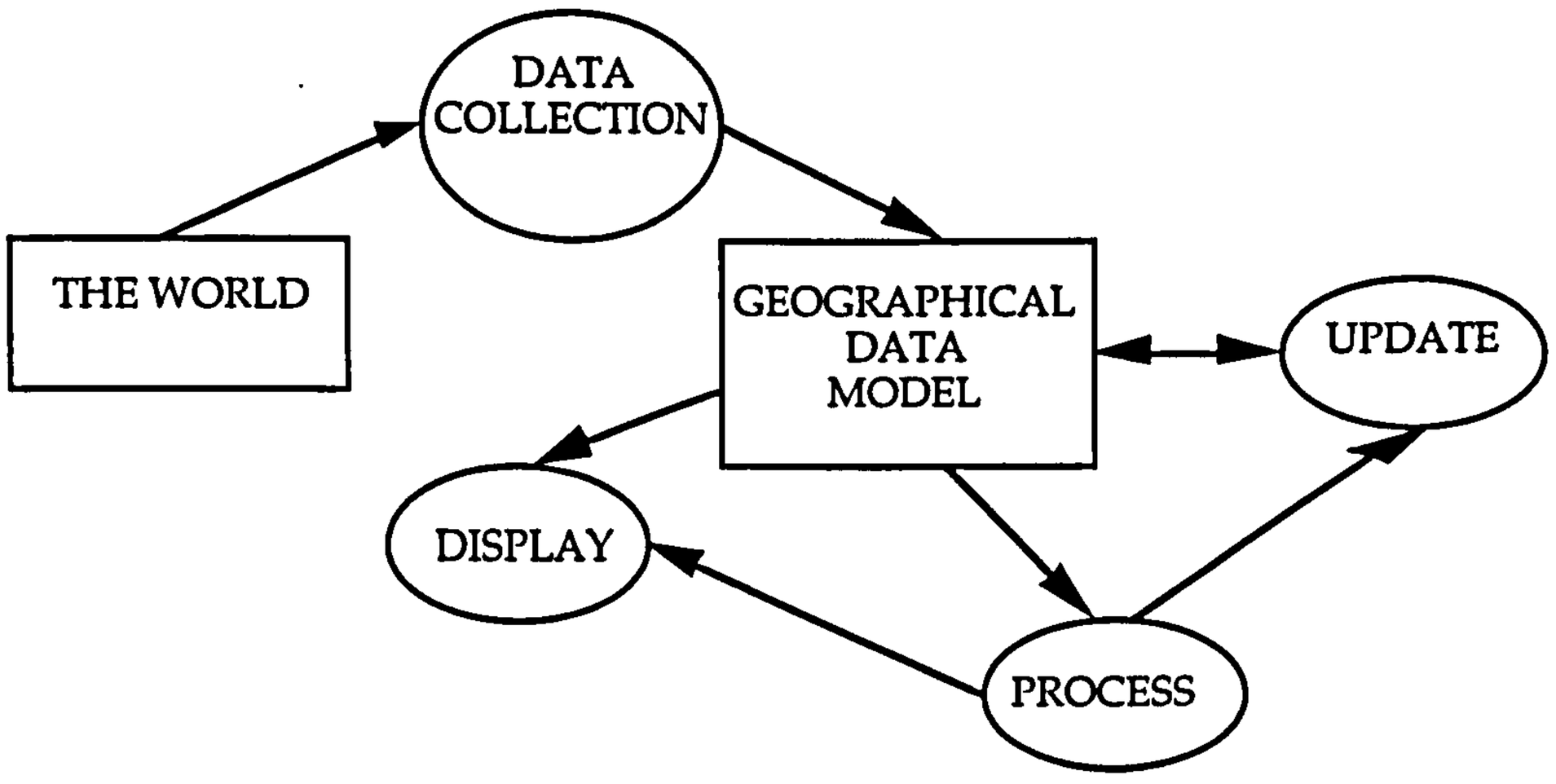


Figure 1.1 The main functions of a GIS.

GIS development began in the 1960s with one of the earliest known implementations being the Canadian Geographic Information System, which was used to assist in urban planning. GIS research and development continued throughout the 1970s and 1980s, and today a wide variety of commercial systems are available. Among the best known are ARC-INFO (ESRI), INTERGRAPH (Intergraph) and SPANS (Tydex Technology Ltd.). These, and other, systems have found a wide range of use in application areas which include environmental resource management, emergency planning and routing, monitoring the built environment (in particular, the utilities industries), market analysis,

and population analysis and prediction. The main function of a GIS, like any other information system, is to improve decision making in areas such as research, planning and management. Decision making is aided by means of the spatial analysis functions available within the GIS. In modern GIS these functions include choropleth mapping, buffer generation, polygon overlay, contouring, network analysis, and area and length calculations.

GIS differ from other types of information system in that they deal primarily with geographic data. This data is characterised by the fact that it is related to a specific location in space. Geographic data can, in the simplest case, be regarded as the digital representation of the information which appears on a conventional map. This data falls into two categories, namely, spatial data and attribute (non-spatial or aspatial) data. Spatial data is used to represent the form and location of objects which appear on the map, such as trees, houses and rivers. Non-spatial data consists of alphanumeric attribute information which describes, in some way, a particular spatial data item, or group of items (for example, the name of a river). Spatial data exists in one of two formats, either vector format or raster format. Vector data defines map objects by means of an x,y coordinate or string of coordinates which refer to object locations and, particularly, boundaries, within a specific spatial referencing system. This format provides flexible and accurate representation. Raster defined data is expressed as an array of pixels which categorise the contents of space on the basis of regular fixed size cells. The resolution of objects is therefore limited to the resolution of the pixel array. This thesis deals primarily with the vector format, the reason for which is described in Chapter 2.

GIS typically deal with large volumes of data. The means by which data is stored is therefore of great importance, with efficiency, both in terms of storage space used and data retrieval performance, being a major objective. This thesis is primarily concerned with the design of the underlying spatial data models on which GIS data storage schemes are based. When considering the design of such data models there are a number of considerations to be kept in mind. The first is that the spatial functions which perform operations on the stored data are usually only concerned, at any one time, with spatially specific subsets of the complete data set. The second issue to consider is that of scale, and the fact that the scale at which data is required is usually dependent on the particular application for which it is being used. It is also important to understand that geographic data comes in a variety of forms and represents many different types of geographic entity.

Attribute data and spatial data, which have already been mentioned, are usually stored separately. At present, the relational database approach is being adopted by many commercial systems to provide a convenient means of storing attribute data. The

data storage schemes adopted for the storage of spatial data tend to be specifically designed for particular data types. Two types of spatial data of importance to this thesis are topographic data and terrain data. Topographic data can itself be broken down into three sub-types, namely, point data, line data and polygon data. These sub-types are combined in a variety of ways to form topographic features corresponding to the real world phenomena which appear on a map. Terrain data, as far as this thesis is concerned, refers to a collection of points, each with a height value associated with it, which when joined together in a particular way forms an approximation to the true ground surface (that is, a digital terrain model). Data structures suited to the storage of spatial data are reviewed in Chapters 2, 3 and 4.

An application area which makes wide use of GIS is that of geology. However, GIS are limited in that they do not offer true 3-D modelling facilities. This is a serious limitation in that geology concerns itself, in the main, with 3-D data. This data comes from a variety of sources, including, well logs, seismic surveys, and gravity and magnetic studies. Data is also made available by the digitising of existing interpreted data, such as contours, cross-sections and outcrop maps. Geological modelling usually involves large volumes of data. Until recently the computer technology available to the majority of the geological community could not cope with such volumes of data. Memory costs were too expensive, computational speeds too slow and graphical displays too low a resolution [2]. During the late 1980s modern workstation technology, which to some extent overcomes these problems, has become available. Today, the research and development of 3-D GIS, or Geoscientific Information Systems (GSIS) as they are commonly known, is accelerating. Several commercial GSIS systems are available, including IVM (Dynamic Graphics) and Vulcan (KRJA Systems). The basic functions of a GSIS are the geological equivalent of those of a GIS, namely, the acquisition, storage, update, processing and display of geological data. Note that greater complexity is involved in supplying each of these functions when compared to the 2-D GIS equivalent. GSIS are reviewed in greater detail in Chapter 8.

1.3 Aims of Project.

The project has two main aims. The first concerns the integration of data of different types to provide an improved spatial data model. The second aim is to provide an efficient multi-scale representation of the spatial data model.

These aims are applied to two specific problems. The first concerns itself with the design of an integrated multi-scale data model suited to GIS. At present, many GIS can be regarded as limited in that 2-D topographic map data and terrain data are stored separately. It is the author's belief that as GIS continue to develop they will benefit from data models which model reality to an ever increasing extent. It is suggested here that a first step towards this goal is the development of a data model which allows for

the integration of terrain data and topographic data. In reality, geographic objects do not all lie on the same flat surface as suggested by current GIS data models. The decisions people make, whether it concerns something relatively trivial (such as, 'should I take a bus or should I walk ?') or something more important (such as, 'where is the new motorway going to go ?') involves all available information, which will sometimes include information regarding terrain. Integrating topographic data and terrain data will benefit such decision making since it provides an opportunity to view and analyse topographic data in a way which bears a closer resemblance to the real world. Data integration can also serve to offer improved terrain modelling capabilities. Topographic data which represent naturally occurring physical objects usually conform to certain surface specific features (for example, a river running through a valley). In addition, certain man-made objects, such as canals, quarries and roads, also usually influence, or have been influenced by, the form of the ground surface. It is hoped that combining the two data types will lead to an improved terrain model, that is, one which more accurately represents the ground surface. Such integration will be of benefit to application areas such as civil engineering, landscape architecture and geology.

GIS would also benefit from having data models which allowed for data to be viewed and analysed at different levels of detail, according to the areal extent, or scale, of the region of interest. A current goal for many GIS researchers is the development of a scaleless database, in which all data is stored at a single, highly detailed source scale. All other scales would be derived from the source scale data at run-time in answer to specific database queries. At present, such a database is not feasible, since not all generalisation functions are available in an automated form, and those which are are often slow and only perform well over small changes in scale. Current GIS cater for different levels of detail by adopting a multiple representation approach, that is, a different set of data representing each scale. There are two main disadvantages to this approach. The first concerns the high storage overheads involved in storing a complete data set for each scale represented. Much information will be repeated between scales. The second disadvantage involves the inconsistency which can develop between different scale versions of the same data. Updating a particular data set at one scale without carrying out an equivalent update at all other scales will lead to inconsistency. It is suggested here that a compromise can be reached by adopting a multi-scale data structure approach. Multi-scale data structures provide means of efficiently storing and retrieving spatial data from databases at levels of detail which are adaptable to different scales of representation. A review of such data structures is given in Chapter 4.

The second specific problem addressed in this thesis concerns the development of an integrated multi-scale 3-D geological model. It is noted that GSIS, like their geographical counterparts, are currently poor with regards to data integration. The need to overcome

this problem is probably greater in the field of GSIS since geological data are usually sparse and the geological objects they are seeking to represent are complex. Therefore there is a need to integrate all available data in order to create accurate geological data models. This thesis addresses a particular problem, namely, the integration of terrain data, geological map data (topographic data) and borehole well log data to assist in the construction of an accurate 3-D geological model. This work forms part of a larger project currently taking place at the British Geological Survey (BGS) which is attempting to integrate data from a variety of sources (geological and geophysical). The ability to efficiently store and access data at multiple levels of detail is of an equal benefit to GSIS as it is to GIS. Therefore the work addresses the design of an integrated multi-scale 3-D geological model. This problem, and the BGS project, are discussed in greater detail in Chapter 8.

Note that with regards to both models, in addition to the design of the data model itself, it is also necessary to develop algorithms which when applied to source data will allow for the implementation of the model. This project therefore addresses both the design of the integrated multi-scale spatial data models and the development of methods by which the models are constructed from source data.

1.4 Thesis Outline.

This chapter has served as an introduction to the thesis. It has given brief background information concerning GIS and GSIS, and noted two shortcomings. The first relates to the lack of data integration facilities within current systems, while the second pertains to the inefficient way in which multiple scale representations are handled. The aims of the thesis have been stated as finding solutions to these problems.

The remaining chapters in this thesis can be divided into three distinct groups. Chapters 2 to 7 concern themselves with the design and implementation of data models and algorithms suited to the efficient multi-scale representation of integrated geographical data. The next two chapters, Chapters 8 and 9, deal with the equivalent geological problem, developing the work described in the previous chapters into 3-D. Finally, the thesis is concluded in Chapter 10.

Chapter 2 provides a review of some basic GIS data structures suited to the storage of either topographic data or terrain data. The following chapter, Chapter 3, deals with the topic of spatial access data structures, providing details of the fixed grid, the quadtree, the R-tree and the grid file methods. Multi-scale data structures are then reviewed in Chapter 4, with special attention given to the line generalisation tree and the Delaunay pyramid. A new data storage scheme, termed the Multiresolution Topographic Surface Model (MTSM), is proposed in Chapter 5. This scheme, based on the fixed grid, the line generalisation tree and the constrained Delaunay pyramid data

structures, provides an efficient integrated multi-scale representation of terrain data and topographic data. Chapter 6 describes two database implementations of the MTSM and gives the results of a series of performance evaluation tests. A new, improved version of the recently developed Implicit TIN data storage scheme is described in Chapter 7. An Implicit TIN version of the MTSM, termed the I_MTSM, is then detailed.

Chapters 8 and 9 concern themselves with the design and implementation of an integrated multi-scale 3-D geological data model. Chapter 8 begins with an introduction to GSIS, outlining the 3-D GIS project currently being carried out by BGS. A review of 3-D object representation techniques is then supplied, with special consideration given to boundary representation and octree methods. Several techniques for providing efficient spatial access to collections of 3-D objects are then discussed. The chapter concludes with the description of a multi-scale 3-D data model (MGM), which, building on the design of the MTSM, provides integrated storage of terrain data, geological outcrop data and borehole data. A prototype database implementation of the MGM, and the results of performance tests, are described in Chapter 9.

The thesis concludes in Chapter 10, which provides a thesis summary, a report on the achievements made by the project and an indication as to how the work might proceed in the future.

Chapter 2

A Review of GIS Data Structures

2.1 Introduction.

The data structures used for the internal representation of geographical data are a concern to both designers and users of GIS. This chapter provides an overview of some of the more well known of these data structures. Section 2.2 addresses the issue of the type of data that a GIS is expected to be able to handle. Two data types of importance to this thesis are topographic data and terrain data. Data structures suited to the storage of each of these data types are reviewed in Section 2.3 and Section 2.4 respectively. Finally, Section 2.5 gives a chapter summary and provides conclusions as to which of the data structures reviewed are of particular benefit to the design of the integrated multi-scale data model proposed in Chapter 1.

2.2 Geographic Data.

It is a fact that GIS must be able to store geographic data. In a first instance, this becomes equivalent to storing the information which appears on a map (although advancing research in areas such as Artificial Intelligence and Hypermedia is now enabling the storage of much more information than just that which appears on a map). A map can be considered to be made up from objects (such as buildings, roads and rivers), which give a graphical representation of reality, and text (YH or SWANSEA for example), which describe what the objects mean. In addition, GIS give the possibility of storing information which might not have been stored on the original map, such as the population of Swansea or the price per night at the Youth Hostel. Also, much of the information held on a map is implied information, such as Cardiff is the nearest city to Swansea or that the Youth Hostel is on an isolated island in the middle of a lake. Since a GIS is required to perform many functions which require implied information, such as produce all Youth Hostels within walking distance of Swansea, it is desirable to somehow include implied information within GIS.

The data types held in a GIS can be classified into two distinct groups, namely spatial data and attribute data. Spatial data can itself be divided into two components, that is, geometric data and topological data. Geometric data are used to represent the metric locational information which appears on a map, such as the position and size of a building, and can be in either vector or raster format. This review restricts itself to the vector format. The main reason for this is that the two data structures which are to form the basis of the integrated multi-scale data model (namely, the constrained Delaunay pyramid and the line generalisation tree) are vector-based. This is not seen as too great a restriction on the usefulness of the proposed data model since vector-formated data is at present widely available and widely used. The 2-D vector data format has three sub-types, namely, point, line and polygon. Other primitives such as circles and splines are possible but these do not usually occur as primary data in a GIS. Topological data describe the relationships between geometric data. The three basic relationships that exist are connectivity, adjacency and inclusion. Topological data are

not always explicitly stored since in principle they can be derived from geometric data. Attribute data are alphanumeric data related to geographic objects, such as the estimated value of a building or the name of a river.

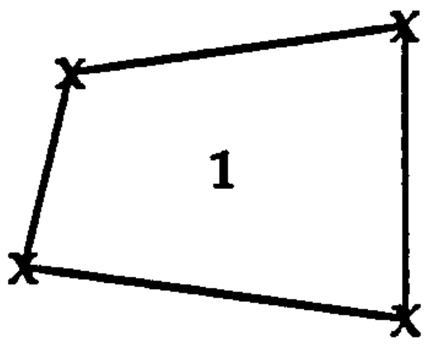
A further geometric data type, which can be regarded as distinct from the others already mentioned, is that which represents terrain. This type of data item can be distinguished from, say, an isolated vector point representing a spot height or elevation point, in that it forms part of a collection of other data items of the same type, which when joined together in some way seek to form an approximation to the ground surface (a terrain model). Data of this type will hereafter be referred to specifically as terrain data, while all other geometric data (polygons, lines, points and, in next section, objects) will be referred to as topographic (physical feature) data. Traditionally, GIS have been primarily concerned with topographic data. When terrain models are present, they are stored separately. As such, the storage of topographic data and the storage of terrain data will now be discussed separately.

2.3 Data Structures for Topographic Data.

The types of topographic features that appear on maps, such as buildings, roads and county borders, can be thought of as comprising of points, lines and polygons. The latter are perhaps the most frequently encoded feature in geographic data systems. The capture and storage of such data commonly takes one of two forms, vector format and raster format, each method having particular advantages and disadvantages when compared with the other. For reasons which have already been mentioned, this section will deal only with vector data.

2.3.1 A Simple Storage Scheme.

A simple data structure for storing map feature data is that of encoding entity by entity, with no account taken of topological relations between entities. In other words, all polygons, lines and points are encoded without regard of the fact that they may intersect or merge with other polygons, lines and points. Names or symbols which define what each entity is are held as a list of attribute text strings, as shown in Figure 2.1. While this method has the advantage of simplicity, it has many disadvantages (see Figure 2.2). Firstly, lines between adjacent polygons must be stored twice, leading to serious errors in matching (giving rise to slivers and gaps along the common boundary) and unnecessary data duplication. A second disadvantage is that there is no topological information, which results in poor performance in operations such as adjacent object queries and the inability to represent islands (except as purely geographical constructions).



Entity Number	Coordinate List	Attribute
1	x1 y1 x2 y2 x3 y3 x4 y4 x1 y1	HOUSE
2	x5 y5 x6 y6	ROAD
:	: : : :	

Figure 2.1 - A simple method for storing topographic feature data.

Finally, there is no simple way of checking if the topology of a polygon is correct, that is, if it is incomplete (dead-ends) or if it makes a topologically inadmissible loop (weird polygons).

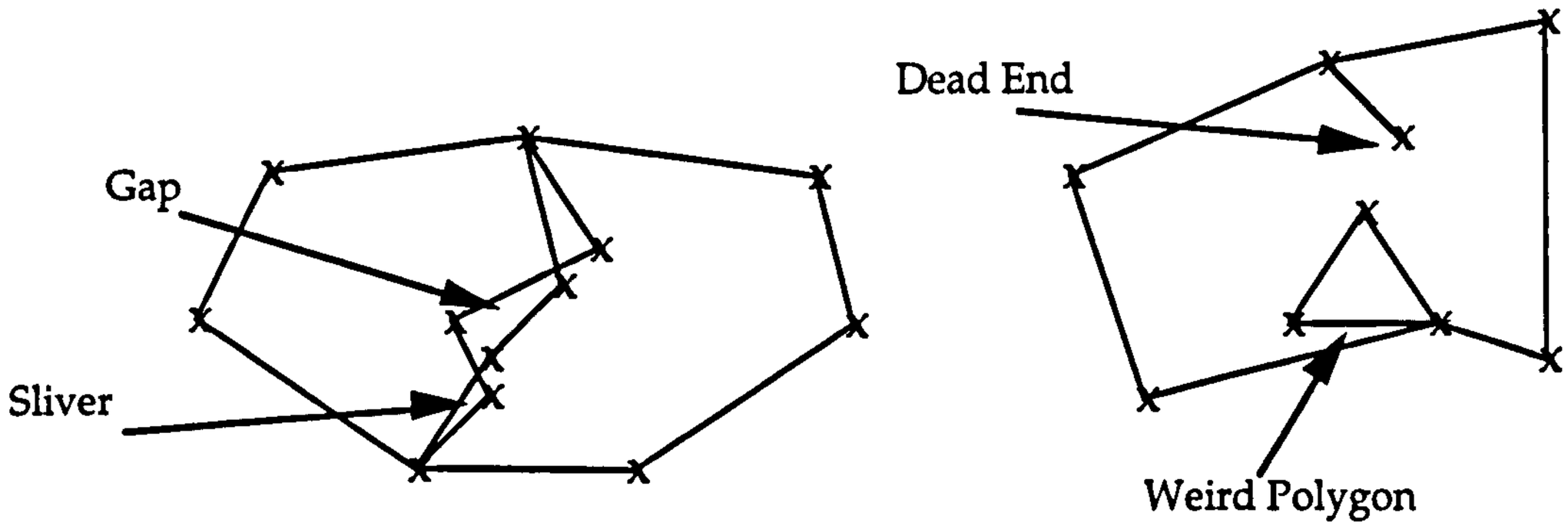
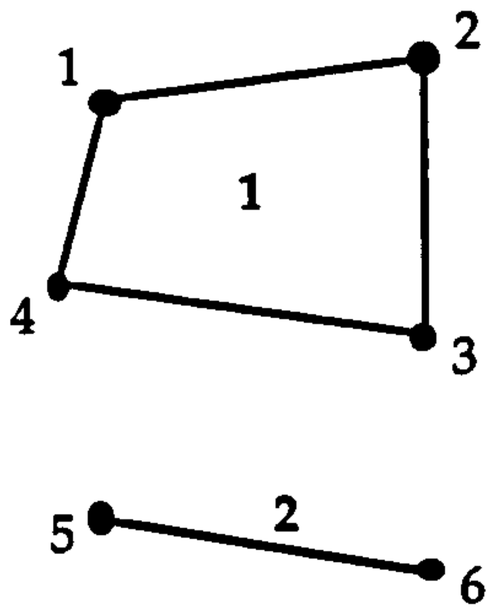


Figure 2.2 - Some of the disadvantages of the simple method for storing data.

2.3.2 An Improved Data Structure.

Some of the limitations of independently encoded structures can be overcome by introducing a point dictionary (Figure 2.3). This dictionary contains the coordinates of every data point on the map, with each coordinate pair being assigned an unique identifier. Polygons, lines or points representing map objects are then made up of lists of coordinate identifiers. This method has the advantages that boundaries between adjacent polygons are unique and therefore slivers and gaps do not occur. A further advantage is that point coordinates are only stored once, thus reducing storage costs.



Point Dictionary

Coordinates	Identifier
x1 y1	1
x2 y2	2
x3 y3	3
x4 y4	4
x5 y5	5
x6 y6	6
: :	:

Entity Number	Point List	Attribute
1	1	HOUSE
	2	
	3	
	4	
	1	
2	5	ROAD
	6	
:	:	

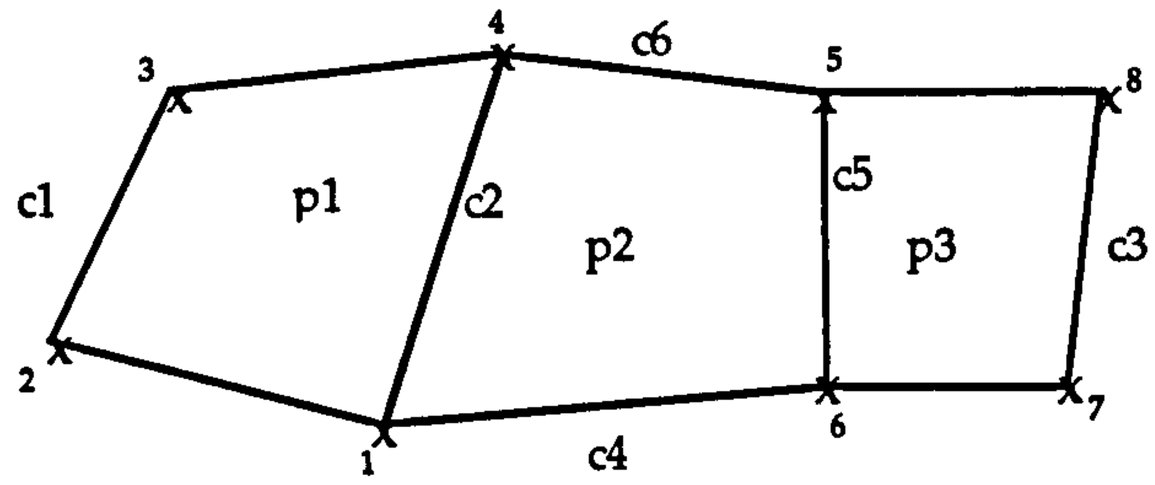
Figure 2.3 - The point dictionary method.

An extension to this method is the addition of a line dictionary and a polygon dictionary. Map objects can also be introduced as the highest level data entity, with each object consisting of a list of identifiers referring to its constituent polygon, line and point parts. This method further reduces storage overheads by removing the need to store multiple copies of the same line or polygon.

2.3.3 Topological Data Structures.

Further data structures have arisen which address the problem of including topological relationships within the data model. One of the first attempts at such a structure was the Duel Independent Map Encoding (DIME) system of the US Bureau of the Census [3]. The basic element of the DIME data file is a simple line segment defined by its two end points, referred to as nodes; more complex lines are represented by a series of segments. The segment has two pointers to the nodes, and codes for the polygon on each side of the segment. Because nodes do not point back to segments, or segments to adjacent segments, laborious searches are needed to assemble the outlines of polygons. Moreover, the simple segment structure makes handling of complex lines very cumbersome because of large data redundancy.

A simple, effective approach is that developed by the Netherlands Soil Survey Institute [4], as shown in Figure 2.4. The polygon map is stored as a segment or chain file in which each chain is stored as a list of x,y coordinate pairs and two pairs of pointers that index the adjacent map areas. The attributes describing each polygon, together with its corresponding index, are stored separately.



Chain File

Chain number	Coordinates	Polygons
c1	x1 y1 x2 y2 x3 y3 x4 y4	p1, -
c2	x4 y4 x1 y1	p1, p2
c3	x5 y5 x8 y8 x7 y7 x6 y6	p3, -
:	: :	

Description File

Polygon Number	Description
p1	House
p2	Shop
p3	Garage

Figure 2.4 - Polygon map stored as a chain file with descriptive data stored separately.

A more recent development, illustrated in Figure 2.5 is the Topologically Integrated Geographic Encoding and Referencing (TIGER) System [5, 6], produced by the US Bureau of the Census as a refined successor to DIME. Here, polygon, line and point primitives are referred to as 2-cells, 1-cells and 0-cells respectively. The first and last points of each 1-cell are called from and to 0-cells. The intermediate points of a 1-cell are called curvature points and are connected by vectors. Each 1-cell points to the two 2-cells positioned directly to its left and right sides. The unbounded region that surrounds the collection of all 1-cells is a special 2-cell, labelled the outside 2-cell. In addition to these definitions, the topological structure must obey two rules. Firstly, the rule of Topological Completeness insists that the topological relationships between cells are complete. For example, this means that each 2-cell, except the outside 2-cell, must be completely surrounded by a set of connected 1-cells. The second rule, that of Topological-Geometric Consistency, requires a consistent relationship between the geometric placement of cells and the pure topological relationships of cells. For example, no two 2-cell interiors can share a common coordinate.

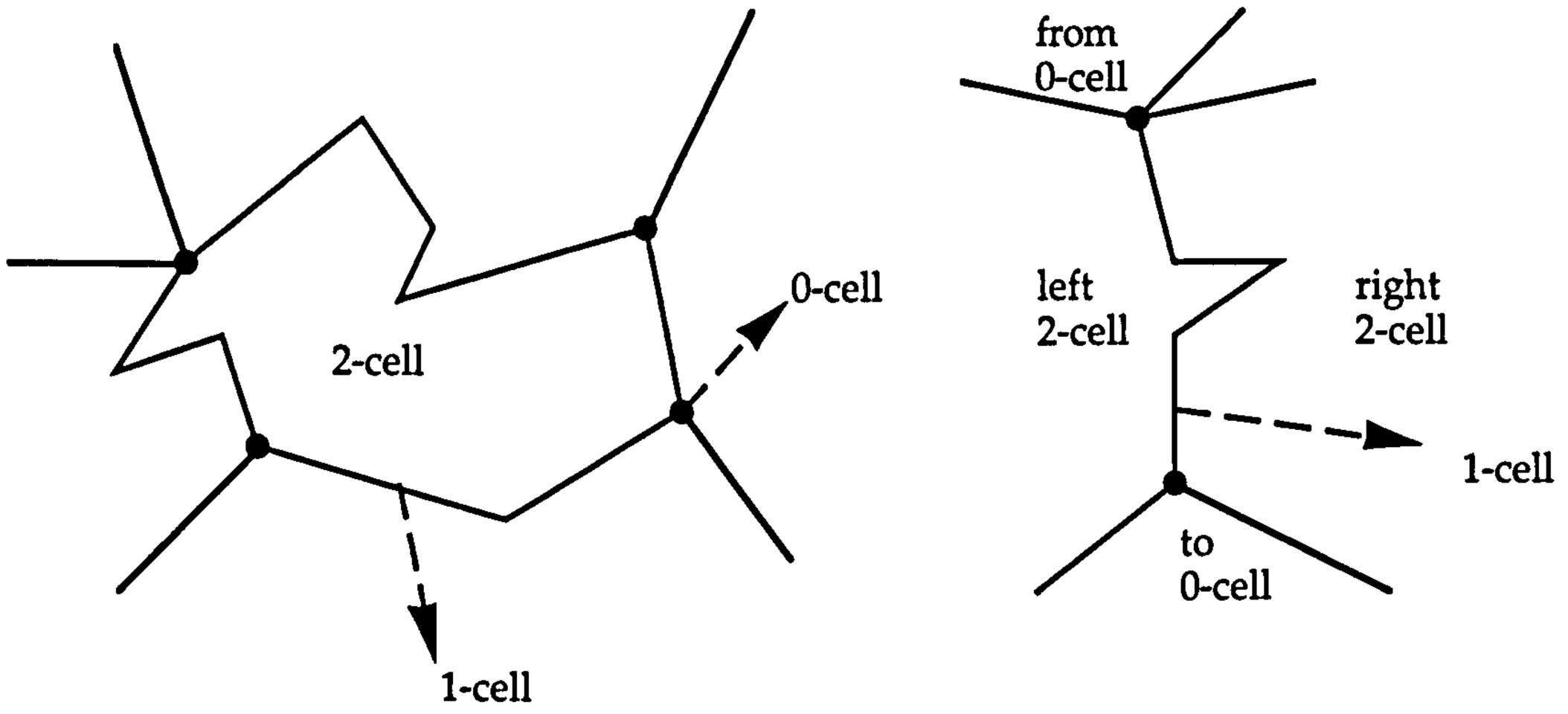


Figure 2.5 - The TIGER structure.

The primary data model for the widely used ARC/INFO GIS consists of four main entities, namely Arcs, Nodes, Label Points and Polygons. Arcs are used to represent line features and the borders of polygons. One line feature or polygon can be made up from many Arcs. The shape of Arcs, which are each assigned an identifier termed a USER_ID, are defined as a series of x and y coordinates. The data model provides the possibility of linking Arcs to their endpoints (Nodes) and to the areas (Polygons) to each side of them. Nodes, which represent Arc endpoints and where line features connect, can be topologically linked to the set of Arcs which meet at the Node. Label Points either represent point features or are used to assign USER_IDs to polygons. Each Label Point is described by an x and y coordinate and a USER_ID. Polygons represent area features as defined by the series of Arcs which compose their borders and by a Label Point positioned inside the Polygon. This Label Point is used to assign the Polygon a USER_ID. Descriptive data concerning Arcs, Label Points and Polygons are held in relational Attribute Tables. This information is related to each Arc, Label Point or Polygon via the USER_ID assigned to each feature.

2.4 Data Structures for Terrain Data.

As the number of GIS users has increased and the range of required uses diversified, a need has arisen for GIS to provide facilities and functions which involve the Earth's surface. Disciplines which would benefit from these facilities and functions include civil engineering, radio path loss analysis, geological surveys and landscape architecture. A means by which the Earth's surface can be adequately represented within a GIS is therefore required. Digital terrain modelling is the term commonly used to describe the range of methods that have been devised to meet this requirement. Since the Earth's surface is an irregular 3-D continuum, it can only be fully defined and depicted by an infinite number of discrete measurements. This approach is not feasible, due to both the problem of data collection and to the finite nature of data storage. Therefore, the

problem of digitally representing the Earth's surface has had to be addressed by adopting methods, both numerical and mathematical, based on a finite set of terrain measurements.

The nature of the terrain data structures adopted depends largely upon the degree to which they attempt to model reality and, or, the intended application of the user. These user-specific approaches have led to the creation of a variety of digital terrain models (DTMs), an overview of some of the most popular of which are given by Peucker [7]. It is noted here that the primary purpose of a DTM is to serve as a substitution for the Earth's surface. The quality of the DTM is therefore primarily determined by the degree to which it approximates to this surface.

2.4.1 The Regular Rectangular Grid.

The most commonly used DTM is the regular rectangular grid (Figure 2.6). Its popularity may be attributed to its simplicity, implicit coordinates, application efficiency and widespread availability of data in this format. The grid does however incur a major disadvantage in that it does not adapt to the changing roughness of terrain. This means that in areas where there is considerable terrain variability the grid will need to have a high point density if it is to approximate accurately. When the same sampling density is applied to flat terrain it will lead to considerable data redundancy. Reaching a suitable compromise between point density and data redundancy cannot always be achieved.

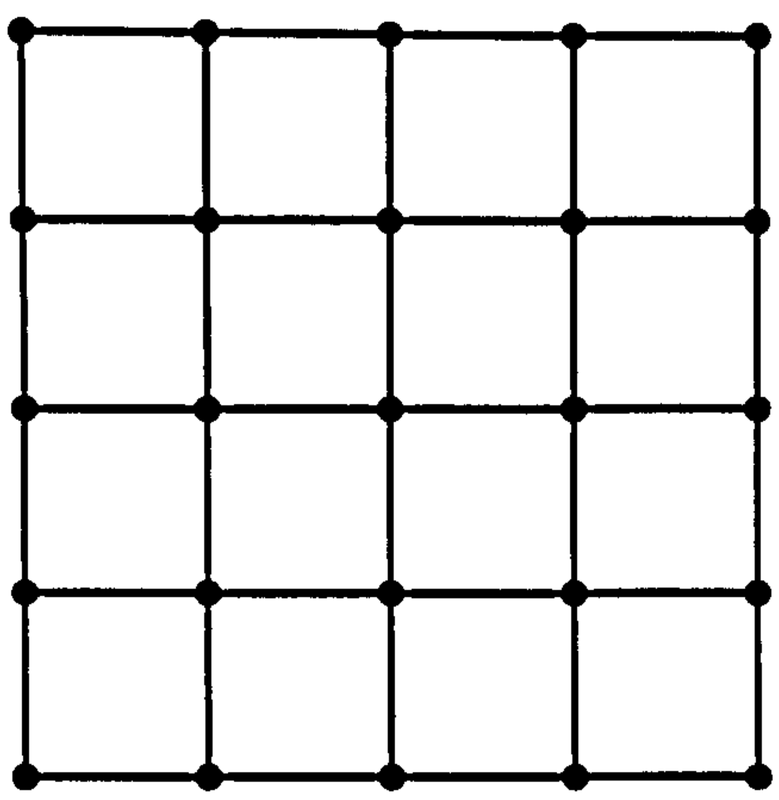


Figure 2.6 - The Regular Rectangular Grid. Each point represents a height measurement. Only the z values need to be stored since the x and y coordinates can be derived from the position of the point in the grid.

2.4.2 The Triangulated Irregular Network.

An alternative DTM, which seeks to overcome the disadvantage of the regular rectangular grid, is the Triangulated Irregular Network, or TIN [8]. The TIN utilises

'surface-specific' points, such as peaks, pits and passes, to form a network of planar, non-overlapping and irregularly shaped triangular facets. The TIN model adapts itself to the roughness of the terrain, with no data redundancy, since all data included in the model represents only critical points. The construction of this model is equivalent to the problem of computing a straight-line graph in the plane, called a planar triangulation, in which the data point projections are joined by straight lines which intersect only at their endpoints.

Methods for constructing the planar triangulation of a set of points have received much attention in the literature (see, for example, [9, 10, 11, 12]). For most applications, such as interpolation, an arbitrary triangulation might not provide an acceptable solution because of the elongated shape of its triangles. A good triangulation can be thought of as one in which the triangles are as equiangular as possible, thus avoiding long and thin triangular facets.

2.4.3 The Delaunay Triangulation.

The Delaunay triangulation has become accepted as the best approach to the creation of a TIN. It is optimal with respect the equiangular requirement and thus has been extensively used as a basis for surface modelling. In addition to producing the most equiangular set of triangles for a given set of points, it has the advantage of producing a unique triangulation of those points.

2.4.3.1 Defining the Delaunay Triangulation.

The 2-D Delaunay triangulation of a set of points S in the plane can be defined as the dual of the Thiessen tessellation (also known as the Dirichlet tessellation or Voronoi tessellation) of S. The Thiessen tessellation is formed as a result of subdividing the plane into polygonal regions, each of which is associated with a point p of S and is defined as the region closer to p than any other point q of S [10]. This is an important concept in geographic applications since a Thiessen polygon can be used to define the region of influence of any point in an areal context. In Figure 2.7, points 1 to 5 are known as the Thiessen neighbours of point p.

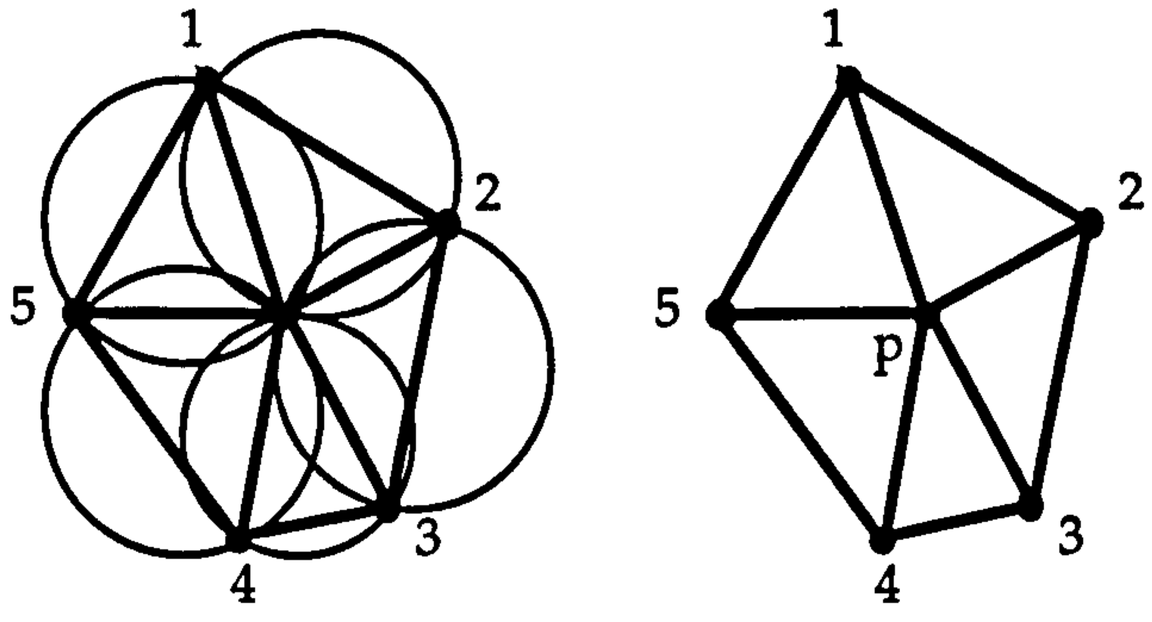


Figure 2.7 - Delaunay triangulation around a point p.

Three lemmas can be distinguished which globally and locally define a Delaunay triangulation [11]. They are -

Lemma 1 : For any triangulation of N vertices, B of which are on the boundary (convex hull), there are 2N-B-2 triangles and a total of 3N-B-3 edges.

Lemma 2 : Two vertices form a Delaunay edge if and only if there exists a circle passing through the vertices that does not contain any other vertex.

Lemma 3 : Three vertices form a Delaunay triangle if and only if its circumcircle does not contain any other vertex in its interior.

By applying Lemma 2 and 3 it is possible to produce algorithms which construct the Delaunay triangulation for S (see Figure 2.7).

2.4.3.2 Data Structures for Storing TINs.

A TIN can be considered as having three primary topological components, namely, vertices, edges and triangles. A data structure suited to encoding a TIN can be regarded as the combination of these basic components and a set of adjacency relations [12]. Woo [13] demonstrates, using an arrow diagram (Figure 2.8), that a total of nine relations can be defined between pairs of these primitive components. Furthermore, it has been stated by De Floriani [12] that any suitably selected subset of these relations can represent, completely and unambiguously, the topology of a TIN.

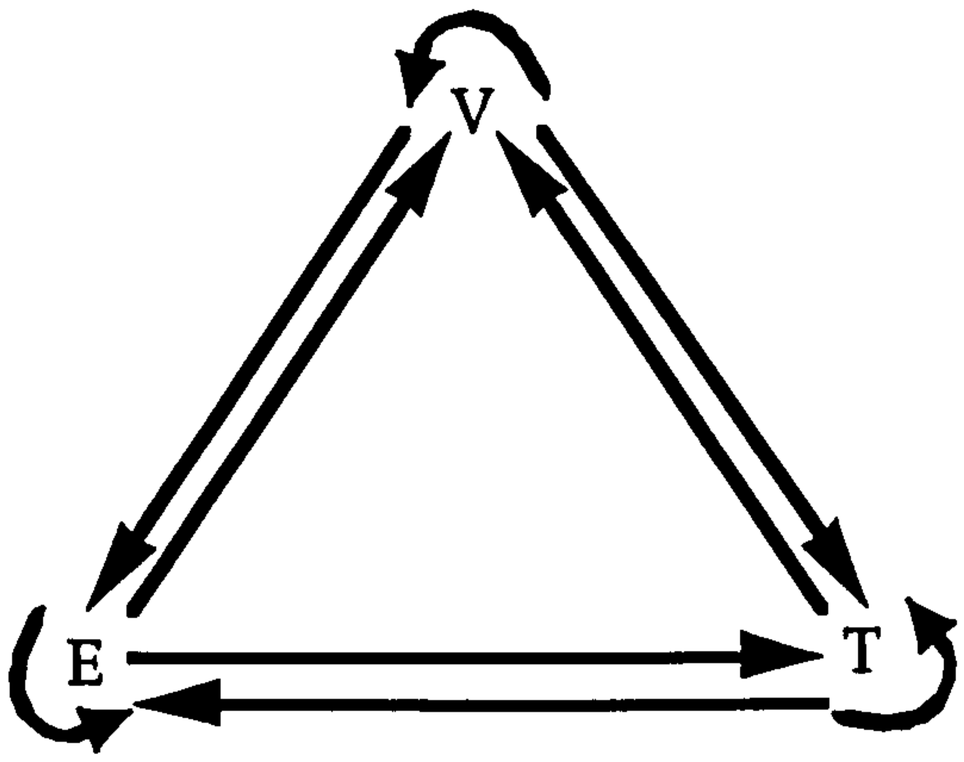
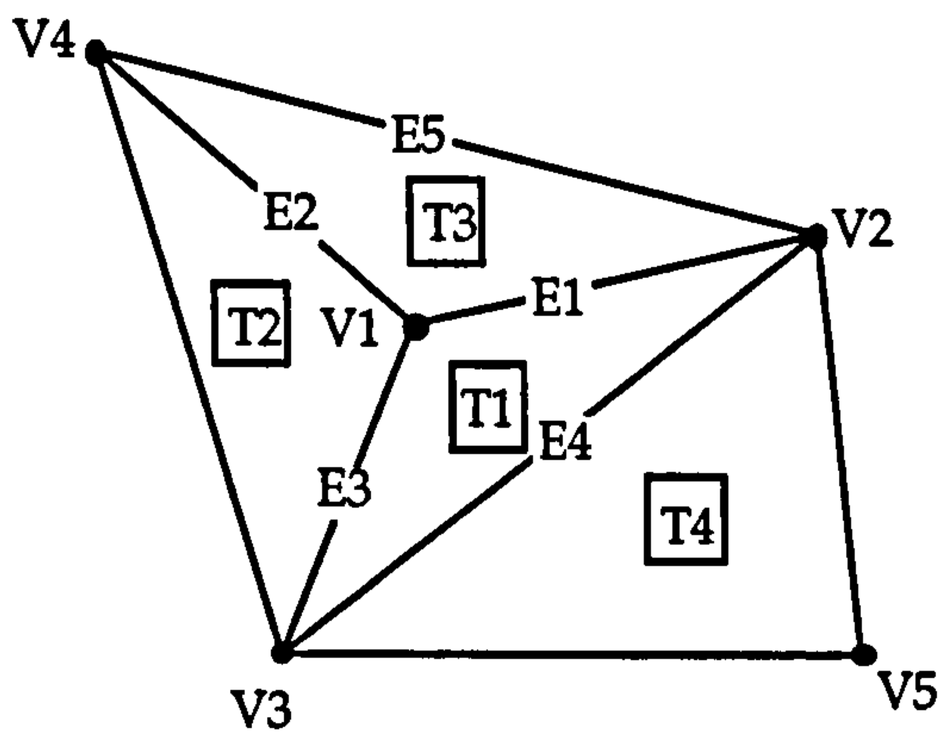


Figure 2.8 - Nine possible relations between pairs of entities in a TIN (V : vertex, E : edge, T : triangle).

In all TIN data structures it is necessary to define the x, y and z coordinates of every surface-specific point. It is also possible to define a point index to provide an unique identifier for each point. In Figure 2.9, it is assumed that having done this, some additional data must be stored to define the structure of the triangulation. Which of the nine schemes is most appropriate is dependent on the requirements of specific applications, so that they each have their own distinct advantages and disadvantages. It should be noted that a hybrid of topological relationships is allowable but any such increase in topological information is directly related to storage costs. For any triangulation of N points, B of which are on the boundary, it can be shown using Euler's theorem that there are 2N-B-2 triangles and a total of 3N-B-3 edges (or 6N-2B-6 directed pointers if stored as links from each vertex). For a vertex-based TIN, each point's coordinates may be stored with a list of pointers to the connected vertices (see vertex-vertex relation in Figure 2.9). If the x, y and z coordinates, index and each pointer require the same unit storage, the total TIN storage will approximate to 10N. The triangle-based TIN will require more storage (approximately 16N) since for each of the 2N-B-2 triangles, pointers to three vertices and three neighbouring triangles are stored (12N-6B-12), together with vertex coordinates and index (4N).

2.4.3.3 Constructing a Delaunay Triangulation.

Efficient algorithms for computing a 2-D Delaunay triangulation have been presented in the literature, a comprehensive review of which is given by De Floriani [12]. Using Lemmas 2 and 3 from Section 2.4.3.1, algorithms can be defined for the construction of the Delaunay triangulation in which the properties of Lemma 1 are implicitly incorporated.



1 Vertex - Vertex : Given V1 store V2, V3, V4
2 Vertex - Edge : Given V1 store E1, E2, E3
3 Vertex - Triangle : Given V1 store T1, T2, T3

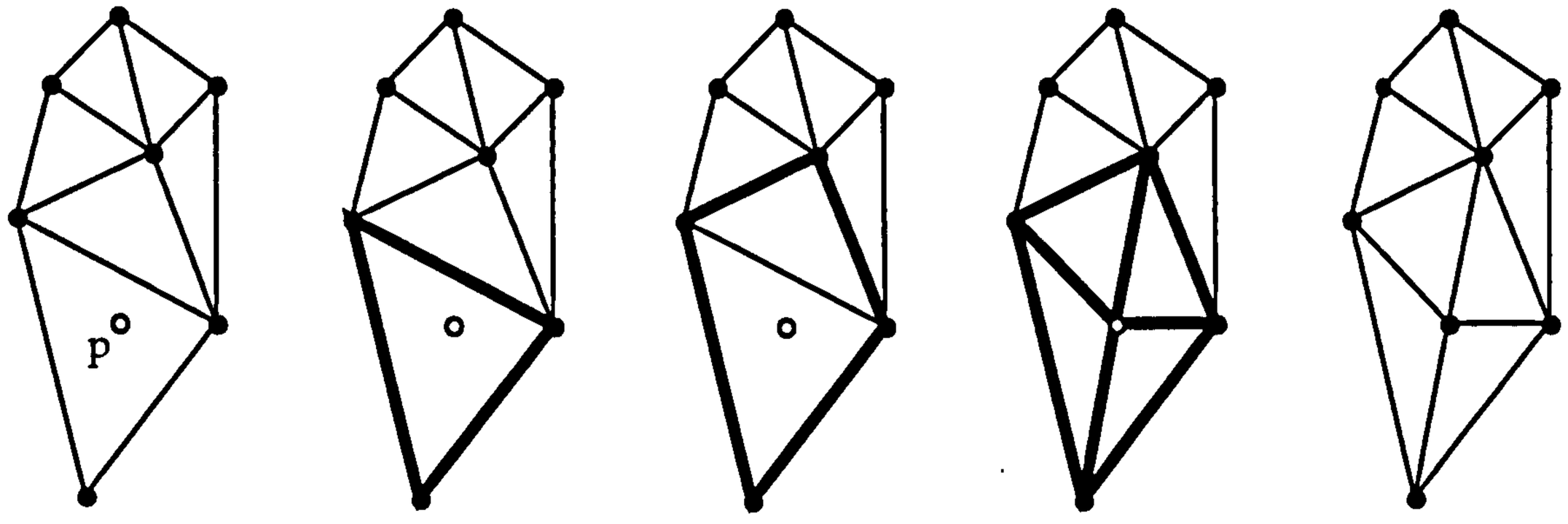
4 Edge - Vertex : Given E1 store V1, V2
5 Edge - Edge : Given E1 store E4, E2, E5, E3
6 Edge - Triangle : Given E1 store T1,T3

7 Triangle - Vertex : Given T1 store V1, V2, V3
8 Triangle - Edge : Given T1 store E1, E4, E3
9 Triangle - Triangle : Given T1 store T2, T3, T4

Figure 2.9 - Illustration of the nine possible relations between pairs of entities in a TIN.

Classical algorithms for constructing the Delaunay triangulation can be classified in a number of ways (see [12]). For instance, an algorithm is termed incremental if it constructs the triangulation by starting from any point and proceeding by adding the remaining points into the subdivision in a stepwise manner. Conversely, divide-and-conquer algorithms recursively split the set of data points into equally-sized subsets until elementary subsets are obtained. These can then be merged to form the complete triangulation. One-step and two-step methods are distinguished by whether they produce a final optimal triangulation in a single step or in two steps (by firstly obtaining an arbitrary triangulation which is then optimised). Also, static and dynamic algorithms differ in that the former assume that all data points to be included in the triangulation are known in advance while the latter make no such demand.

An algorithm of particular relevance to this thesis is the point insertion algorithm described by Watson [14], and more recently adopted in the work of De Floriani [15]. The algorithm is an incremental, dynamic, one-step algorithm based on the stepwise insertion of internal, currently untriangulated points, into an initial enclosing Delaunay triangulation. The initial triangulation can be obtained in a number of ways. A simple method is to employ three dummy points which form an initial, single enclosing triangle. On completion of the triangulation process any triangle which contains a dummy point is removed. An alternative approach consists of producing an initial Delaunay triangulation of those points which define the convex hull of the points to be triangulated. Algorithms for constructing the convex hull of a set of points and computing the Delaunay triangulation of a convex polygon are given by Larkin [16] and Derijver and Maybank [17] respectively.



Triangulation before new point p is inserted.	Locate the triangle in which p lies.	Find the rectangle R_p which bounds the triangles I_p influenced by p.	Delete the triangles in I_p then Delaunay triangulate R_p.	Triangulation after p is inserted.

Figure 2.10 - Inserting a point into a Delaunay triangulation.

The second stage involves sequentially inserting each currently untriangulated point into the current Delaunay triangulation. After each insertion a new Delaunay triangulation will have been formed. Descriptions of methods for inserting a point into an existing Delaunay triangulation have been given in the literature [14, 15]. These methods are based on the premise that according to the circle criterion (Lemma 2), the insertion of a new point p into a Delaunay triangulation T affects only those triangles I_p of T whose circumcircles contain p. The process of inserting the point p can be summarised as locating the triangle t of T in which p lies; recursively examining the neighbours of t until all triangles I_p are found; constructing the polygon R_p formed by the external edges of the triangles in I_p; deleting the triangles in I_p; and finally Delaunay triangulating R_p (this is done by connecting p to each vertex of R_p). This process is illustrated in Figure 2.10.

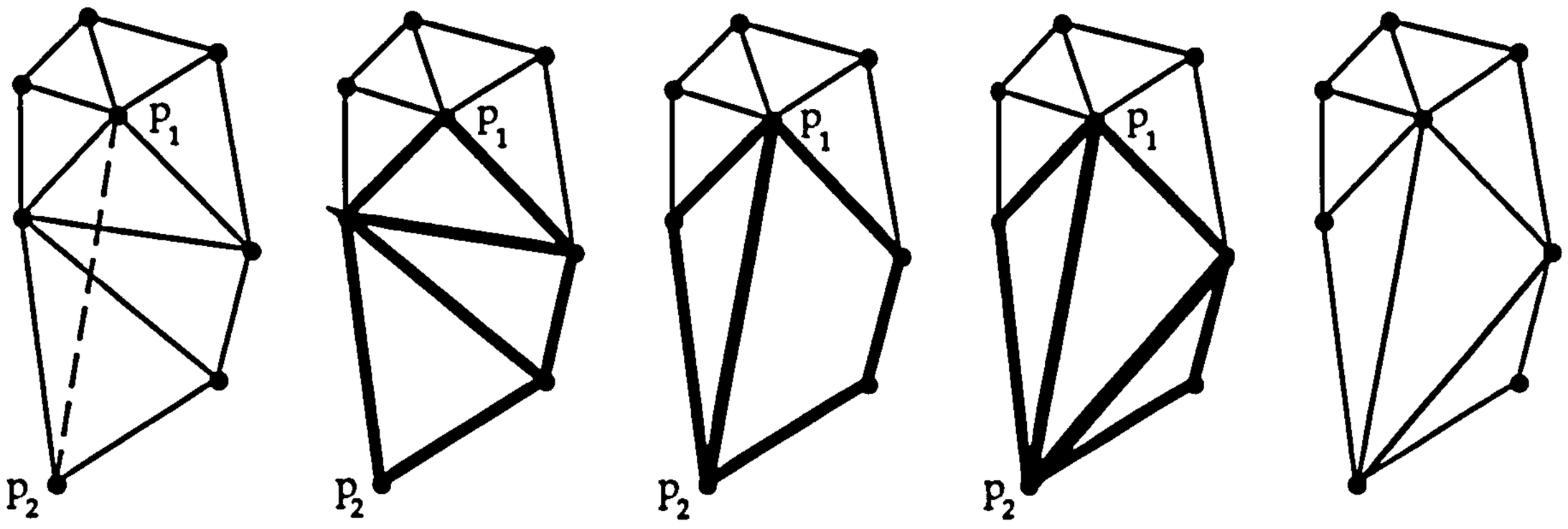
2.4.4 Constrained Delaunay Triangulation.

As has already been mentioned, triangulation algorithms based on Delaunay triangulation have the advantage of producing the set of most equiangular triangles. However, note that Delaunay's method was first developed to solve nearest neighbour problems in the xy-plane, not as a method for surface approximation. As such, Delaunay triangulation algorithms do not consider the third dimension (z coordinate), and may therefore produce triangle edges that contradict the true topology of the surface [18]. To counter this problem, while at the same time attempting to preserve the equiangular property of Delaunay triangulation, algorithms have been developed which produce constrained Delaunay triangulations [19, 20, 21]. These algorithms ensure that any known constraints on the surface, such as lines representing ridges or valleys, are maintained as edges within the triangulation. Retention of these edges results in local violation of the Delaunay circle criteria. However, in doing so, the surface more

accurately models the real world surface. The Delaunay criteria is used to model those parts of the surface for which no structural information is available.

De Floriani and Puppo [22] present a dynamic, two-step method for producing the constrained Delaunay triangulation. The first step is to construct a conventional Delaunay triangulation of all vertices, using any method available. The second stage is to iteratively insert the constraining features into the triangulation as a series of straight line segments. After each insertion, a new constrained Delaunay triangulation is produced.

Algorithms for inserting a line segment L, defined by vertices p_1 and p_2 (which must already be in the triangulation), into a constrained Delaunay triangulation T are given by Heller [21] and De Floriani and Puppo [22]. They can be summarised as firstly locating a triangle t of T which has p_1 as a vertex; proceeding to find all triangles I_L through which L passes; constructing the polygons R_{L1} and R_{L2} either side of L formed by the external edges of the triangles in I_L; deleting the triangles I_L; and finally Delaunay triangulating the polygons R_{L1} and R_{L2} in turn (see Figure 2.11).



Triangulation before the line segment L (p_1,p_2) is inserted.

Find the triangles I_L intersected by L.

Construct polygons R_{L1} and R_{L2} formed by external edges of triangles of I_L Delete triangles of I_L.

Delaunay triangulate R_{L1} and R_{L2}

Triangulation after L is inserted.

Figure 2.11 - Inserting a line segment into a constrained Delaunay triangulation.

2.5 Summary and Conclusions.

This chapter has reviewed a number of data structures suited to the storage of geographic data. The data structures fall into two categories, namely, those used for storing topographic data and those used for storing terrain data. Chapter 1 has stated that the multi-scale data model being designed must accommodate both topographic and terrain data, and integrate both data types in some way. It is suggested here that any topographic data structure used in this work should facilitate the inclusion of the

three vector data format sub-types (point, line and polygon), and objects made up from collections of these sub-types. Also, to avoid matching errors and unnecessary data duplication, it is beneficial to adopt the point, line and polygon dictionary approach, outlined in Section 2.3.2. The inclusion of topographic topological information is not seen as vital at this stage. With regards the storage of terrain data, the TIN approach seems to be appropriate to the design of the integrated multi-scale data model. In addition to the traditional advantages it has over other terrain representation techniques (see Section 2.4.2), it has two properties of particular importance to this thesis. Firstly, certain of the algorithms used for the construction of TINs (for example, [14]) can be adapted and used as surface generalisation algorithms. These algorithms can be used in conjunction with more complex triangle based data structures to produce hierarchical triangulations (see Section 4.3). The second important property a TIN has is its ability in include arbitrarily positioned points and pre-defined edges, thus facilitating the integration of topographic data. Both these properties are explored in greater detail in later chapters.

Chapter 3

Efficient Access to Spatial Data

3.1 Introduction.

The purpose of this chapter is to introduce the concept of spatial access data structures. Section 3.2 provides some indication as to why spatial access data structures are needed in GIS. A review of some of the spatial access data structures which are, at present, in common use is then given in Section 3.3. The subject is looked at purely from a two–dimensional point of view, with special attention being given to the fixed grid, the quadtree, the R–tree and the grid file methods. Section 3.4 provides a chapter summary and indicates which of the reviewed methods are to be adopted in the design of the integrated multi-scale data model.

3.2 The Need for a Spatial Access Data Structure.

A simple way of demonstrating the need for a spatial access data structure is by means of example. Consider a cartographic database consisting of point data, representing cities, and associated attribute data. A typical query to such a database might be to determine all cities within 50 km of some other city, Bristol say, that have a population in excess of 100,000. If the data is stored conventionally, in a relational database for example, one possible approach to answering the query would be to sequentially search through the list of cities, checking for population, and, for those cities with population greater than 100,00, calculating the distance from Bristol. Such an approach appears satisfactory when the database is small or when a large proportion of the cities satisfy the query. However, such a simplistic approach is not suitable for a large database, representing the whole of the UK for example, where only a small proportion of the cities will satisfy the query. It is therefore necessary to design databases where it is possible to retrieve information efficiently according to its spatial location, thus reducing the amount of data accessed, and subsequently processed, as the result of a query. The data structures upon which such databases are based are referred to as spatial access data structures.

3.3 Spatial Access Data Structures.

This section discusses four well-known, and commonly used, spatial access data structures. They are the fixed grid, the quadtree, the R–tree and the grid file. A comprehensive review of these and other spatial data structures is given by Samet [23, 24]. The reason for choosing these four data structures for special scrutiny is that a basic understanding of them is necessary to assist in the description of more complex data storage schemes in later chapters.

It is not the intention of this thesis to give an in–depth evaluation of the relative merits and demerits of each of these data structures. One may benefit by its simplicity, another by its efficiency or applicability to a wide range of data types. Much work in the literature has been produced with regards quantifying some of these relative

attributes (see, for example, [25, 26]), with seemingly no single data structure coming out an overall winner. Suffice to say that when dealing with spatial data, it is necessary to have some kind of data structure providing indexing with regards spatial location. The benefits gained or lost between which of the available structures is used are minimal when compared to the benefits gained by having a spatial access data structure, of any type, in the first instance [26].

3.3.1 The Fixed Grid.

The fixed grid [27] is a data structure based on the concept of the division of xy–space into equal–sized cells. Thus a single areally extensive region of spatial data is partitioned into smaller, equal–sized sub–regions. Each cell, sometimes referred to as a bucket, corresponds to an area of storage (or memory) in which all data lying within the cell, in space, is stored. The data structure is essentially a directory in the form of a two–dimensional array, with one entry per cell.

The fixed grid is particularly suited to storing point data. It is best illustrated by means of a simple example. Consider the following set of conventionally stored point data (Figure 3.1), representing the location of cities.

Point		Attribute
184.0	32.8	Bath
166.5	34.2	Bristol
136.3	38.5	Cardiff
182.0	64.0	Gloucester
176.0	48.0	Newport
102.5	44.0	Swansea
184.5	88.0	Worcester

Figure 3.1 - Conventionally stored point data.

To locate all cities within 20 km of Newport would necessitate examining the coordinates of each of the 6 other cities and performing a distance between points calculation in each case.

Now consider partitioning the data using a fixed grid (Figure 3.2). To answer the previous query, the first step would now be to produce a list of all grid cells which lie within 20 km of Newport, a relatively trivial operation. This gives a list of 9 possible cells, only 2 of which contain relevant data. There is now therefore only the possibility of 2 cities (Bristol and Cardiff) lying within 20 km of Newport, each of which can be checked in turn.

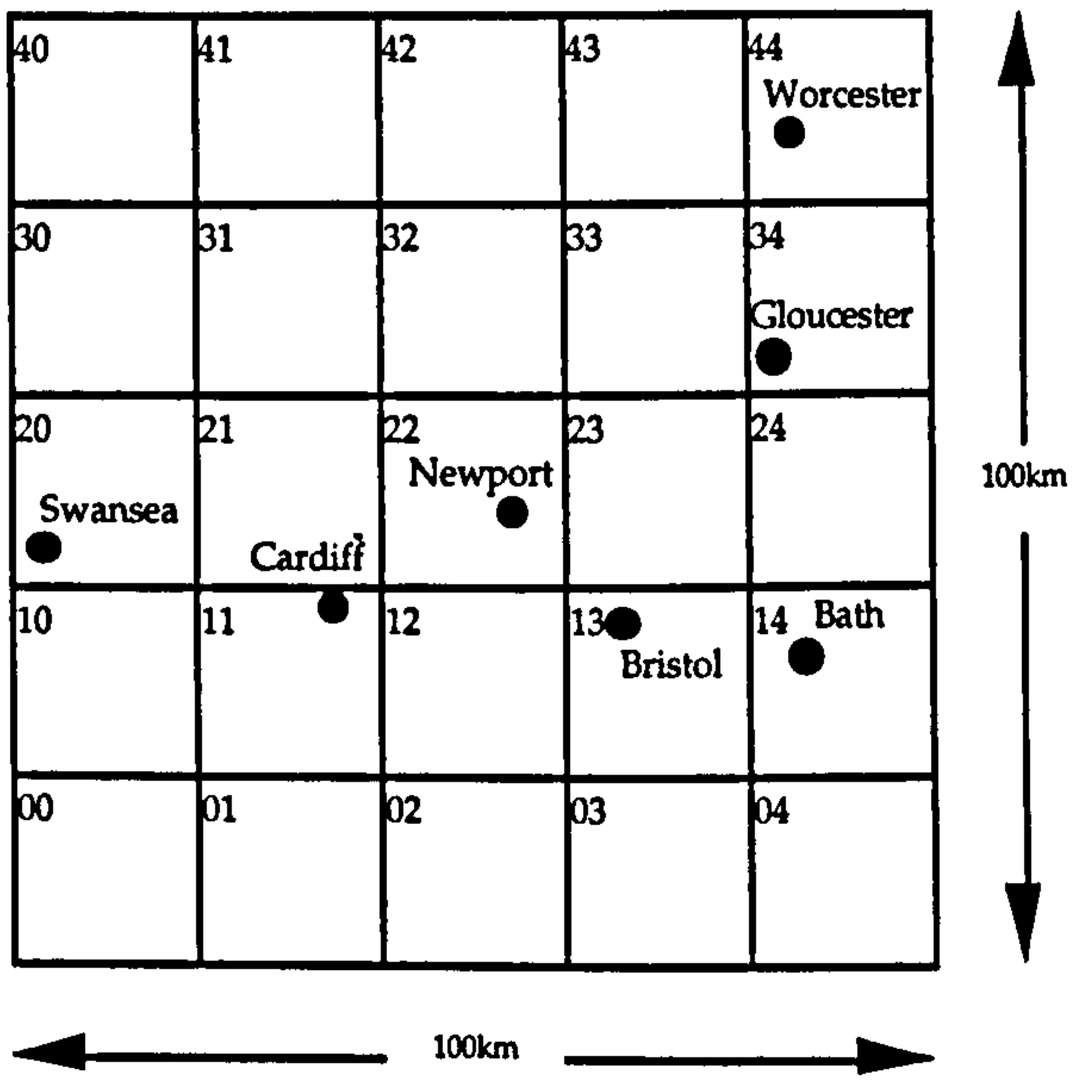


Figure 3.2 - Storage of point data using a fixed grid.

Point data which lies on the edge or corner of two or more cells is catered for by adopting the convention that such a point is always assumed to lie in the cell above and to the right (or some similar convention).

Each cell corresponds to an area of storage, the size of which is fixed. A problem will therefore occur if the number of data items that lie within a cell necessitates more storage than is available. Such occurrences can be minimised by optimising the cell size [28], but there is always the possibility that the number of data items will exceed the storage limit. This problem can be overcome by employing a chaining mechanism. Here, each full cell maintains a pointer to a secondary cell (which is not part of the original grid) in which excess data is stored. The secondary cell may, if necessary, point to a further secondary cell, and so on, thus forming a chain of cells.

It is clear that the fixed grid is best suited to storing point data and there are potential difficulties encountered when dealing with more complex data types such as line and polygon data. For example, it would not be unusual for a polygon to intersect more than one grid cell, or for a large number of complex data items to intersect a single cell. A solution to the first of these difficulties is to segment any data item that intersects more than one cell into a number of new data items, each of which lies within only one cell. This results in an increase in storage due to the increased number of data items and the additional data required to define the points of intersection the new data make with the grid cell edges. The second difficulty can be catered for using the chaining techniques previously described. This, however, is inefficient in that long chains of data items can occur, resulting in increased storage and slower data access.

As previously suggested, a disadvantage of the fixed grid data structure is that it is entirely regular in its subdivision of space, regardless of whether or not this is reasonable for the data being stored. If the data is uniformly distributed over space this does not lead to any problems. However, if the data is non–uniformly distributed over space two particular problems arise [23]. Firstly, some cells will be under–utilised, being empty or nearly empty. This is clearly inefficient with regards storage space. On the other hand, other cells may be over–utilised, leading to long overflow chains. This leads to a large number of storage accesses for all subsequent queries involving any cells that have overflowed, leading to increased access time. Three spatial data structures which seek to overcome these disadvantages are the quadtree, the R–tree and the grid file. These data structures are data adaptive in the manner in which space is divided, that is, division is object–driven rather than space–driven [15].

3.3.2 The Quadtree.

The quadtree, first suggested by Klinger [29], is a data structure which is based on the recursive regular decomposition of a square region into quadrants and sub–quadrants. Much pioneering work regarding the quadtree and its derivatives has been carried out in recent years by Samet (for example, [23, 24, 30]).

Perhaps the simplest form of quadtree is the region quadtree [29, 31], which is concerned with the efficient representation of two–dimensional binary image data. For a binary image, covering a square region, the region quadtree is constructed as follows. If the region under consideration is non–homogeneous, that is, does not consist entirely of 1's or entirely of 0's, it is divided into quadrants. Each quadrant is checked in turn for homogeneity, and any which are found to be non–homogeneous are themselves subdivided. Subdivision continues in a recursive manner until sub–quadrants are obtained that consist entirely of 1's or entirely of 0's. The region quadtree is represented by a tree structure, rooted at the node representing the whole square region. Each son of a node represents a quadrant of the region represented by that node. Each node has out–degree (number of sons) 4, except the leaf nodes, which have out–degree 0 (as they are not subdivided further). A node can therefore be represented by five fields, the first four containing pointers to sons (if there are any), the fifth indicating if the node is empty (0), full (1) or a non–leaf node. As an example, consider the image shown in Figure 3.3a, represented by the $2^3 \times 2^3$ binary image in Figure 3.3b. 1's correspond to image elements, 0's to non–image elements. The resulting quadtree subdivision and region quadtree are shown in Figure 3.3c and Figure 3.3d respectively.

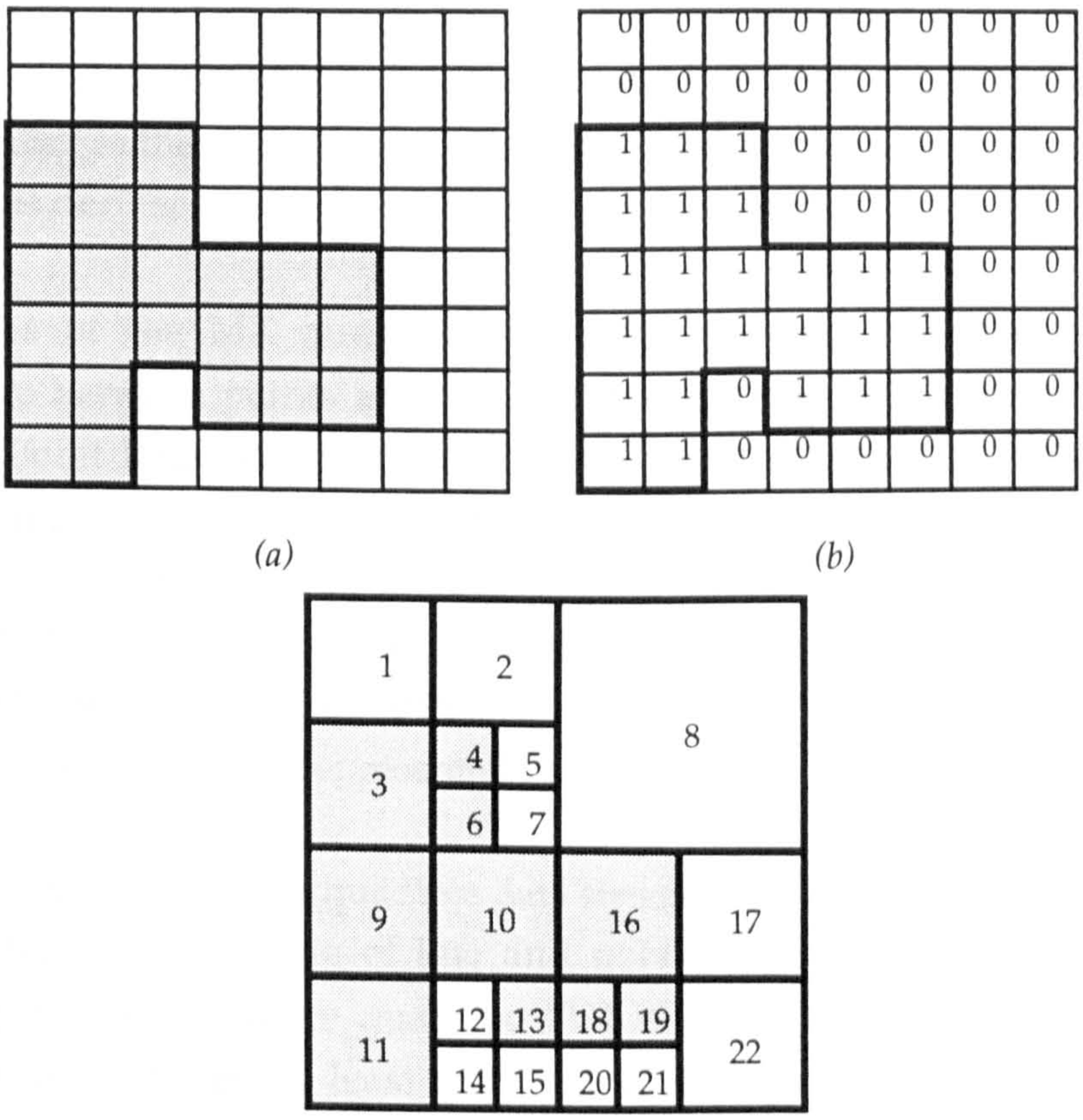


Figure 3.3 - The region quadtree. *(a) Image. (b) Binary image. (c) Quadtree subdivision. (d) Region quadtree.*

The region quadtree, developed for storing binary, or raster, data, has been extended to allow for the storage of point, line and polygon data. The MX quadtree [23] is an attempt to use region quadtree methods for the storing of point data. MX stands for MATRIX and the approach is to treat the point data as existing in a sparse matrix, each point being equivalent to a single matrix element. The matrix is decomposed by a method identical to that used for binary data. However, each node now consists of six

fields. The first four contain pointers to sons, the fifth indicates if the node is empty, full or a non–leaf node and the sixth contains descriptive information about the point being represented at the node. There is no need to store the coordinates of the point data since this is derivable from the path to the node from the root of the tree.

The usefulness of the MX quadtree depends upon there being a one–to–one correspondence between points and the matrix elements. This constraint is relaxed with the PR (Point Region) quadtree [23]. The PR quadtree is organised in a way similar to the region quadtree. The difference is that leaf nodes are either empty or contain a data point (that is, full) and its coordinates. A quadrant contains at most one data point. A PR quadtree node is therefore made up of eight fields. The first six correspond to those of the MX quadtree, with two extra fields required to store the x and y coordinates of the point represented by the node.

Further extensions to the region quadtree data structure have resulted in many quadtree variations suited to the storing of line and polygon data. These include the edge quadtree [32], the least square quadtree [33], the line quadtree [34] and the PM quadtrees [34, 35]. The edge, least square and line quadtrees are all pixel based methods, and as such, will not be dealt with further here.

There are three types of PM quadtree, the PM1, PM2 and PM3 quadtrees, which are each oriented towards storing information about the edges from which lines and polygons are made up. PM quadtrees are arranged in a way similar to the region and PR quadtrees. A region is repeatedly subdivided into four equal–sized quadrants until obtaining leaf nodes that meet a specific criterion. In each case edges are not permitted to exist in more than one leaf node. This necessitates that each edge be divided into sub–edges, termed q–edges. The q–edges are formed by clipping the edge against the borders of each node it intersects. The leaf nodes of a PM1 quadtree (Figure 3.4a) must satisfy the following conditions – (a) contain, at most, one vertex; (b) if it contains a vertex, it cannot include a q–edge that does not include that vertex; and (c) if it contains no vertices, it can contain, at most, one q–edge. The record definition of each tree node is more complex for PM quadtrees than for the quadtrees previously described. It is necessary to include fields containing pointers to descriptions of any point or q–edge data that lies within it, in addition to the usual son pointer and node type fields.

Each of the PM methods differs in its treatment of edge information, but none permits the storage of more than a single vertex in a leaf node. The PM2 quadtree replaces condition (b) of the PM1 leaf node definition by allowing leaf nodes that do not contain a vertex to contain more than a single q–edge, provided they meet at a common vertex (Figure 3.4b). The least restrictive is the PM3 quadtree (Figure 3.4c) which allows

any number of q–edges to be present within a given node. This approach is considered by Samet and Webber [34] to be the most useful of the PM quadtrees and can be used to store points, lines and polygons, without error.

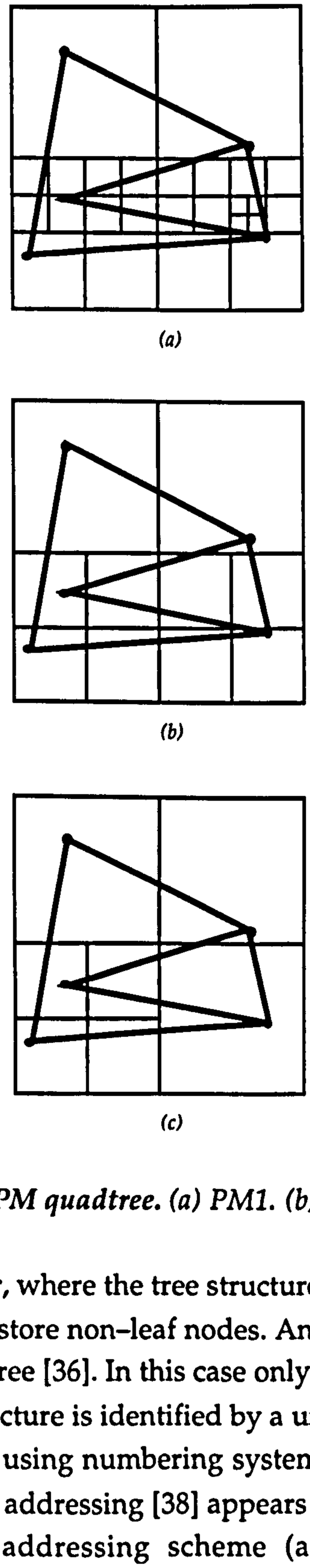


Figure 3.4 -The PM quadtree. (a) PM1. (b) PM2. (c) PM3.

The quadtrees discussed so far, where the tree structure is explicitly defined, incur the storage overhead of having to store non–leaf nodes. An alternative approach is that of the pointerless, or linear, quadtree [36]. In this case only leaf nodes are stored and their position in the overall tree structure is identified by a unique key. The key, or address, of each leaf node is generated using numbering systems known as tesseral addresses [37]. Of these systems, Morton addressing [38] appears to be the most useful and most frequently used [23]. This addressing scheme (as with others) converts the two–dimensional xy-space, that is the quadtree cells represented by the leaf nodes,

into a one–dimensional list of integer keys. If the list of keys is sorted into ascending order, they trace a Peano space–filling curve (Figure 3.5).

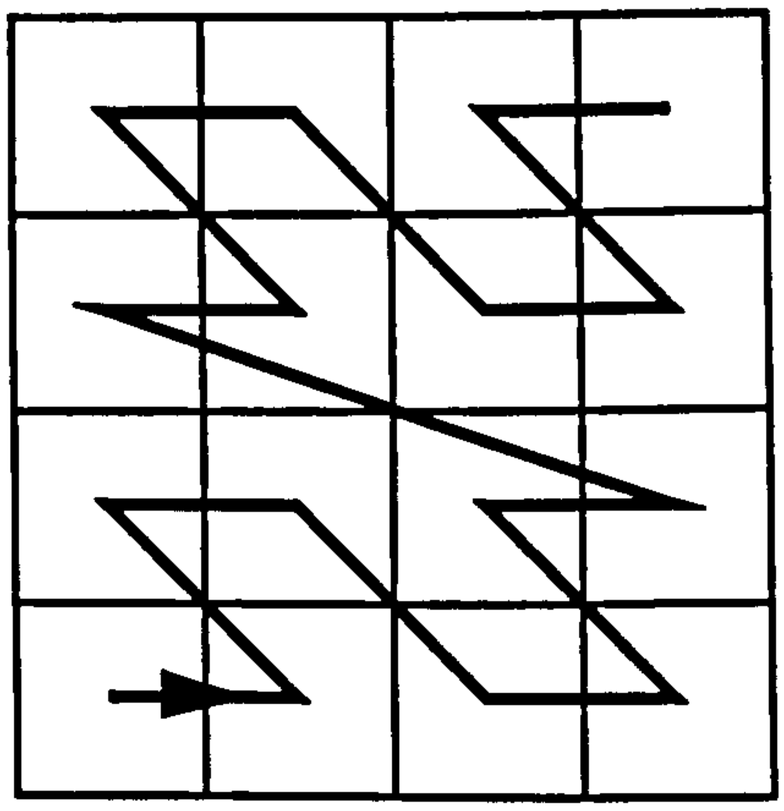


Figure 3.5 - A Peano space-filling curve.

An important characteristic of this method is that cells which are spatially close tend to have addresses close to each other. This is of use since, in general, geographical queries are locationally specific. The list of keys and associated data (which will depend on the type of quadtree being used, region or PM1 for example) can then be stored using a conventional database scheme. The example shown in Figure 3.6 demonstrates how the quadtree given in Figure 3.3 can be represented as a linear quadtree.

The Morton code of a quadtree cell is formed by bit–interleaving the x and y coordinates of its bottom left hand corner (or some similar convention). For example, consider a cell defined by the bottom left hand coordinate pair (x=3, y=4), the binary representation of which is (011, 100). Interleaving the bits, with y arbitrarily deemed to be most significant, results in a binary Morton code of 100101, which converts to a decimal value of 37.

Region	Decimal Coordinate		Level	Binary Coordinate		Binary Morton Code	Decimal Morton Code
1	0	6	2	000	110	101000	40
2	2	6	2	010	110	101100	44
3	0	4	2	000	100	100000	32
4	2	5	3	010	101	100110	38
5	3	5	3	011	101	100111	39
6	2	4	3	010	110	100100	36
7	3	4	3	011	100	100101	37
8	4	4	1	100	100	110000	48
9	0	2	2	000	010	001000	8
10	2	2	2	010	010	001100	12
11	0	0	2	000	000	000000	0
12	2	1	3	010	001	000110	6
13	3	1	3	011	001	000111	7
14	2	0	3	010	000	000100	4
15	3	0	3	011	000	000101	5
16	4	2	2	100	010	011000	24
17	6	2	2	110	010	011100	28
18	4	1	3	100	001	010010	18
19	5	1	3	101	001	010011	19
20	4	0	3	100	000	010000	16
21	5	0	3	101	000	010001	17
22	6	0	2	110	000	010100	20

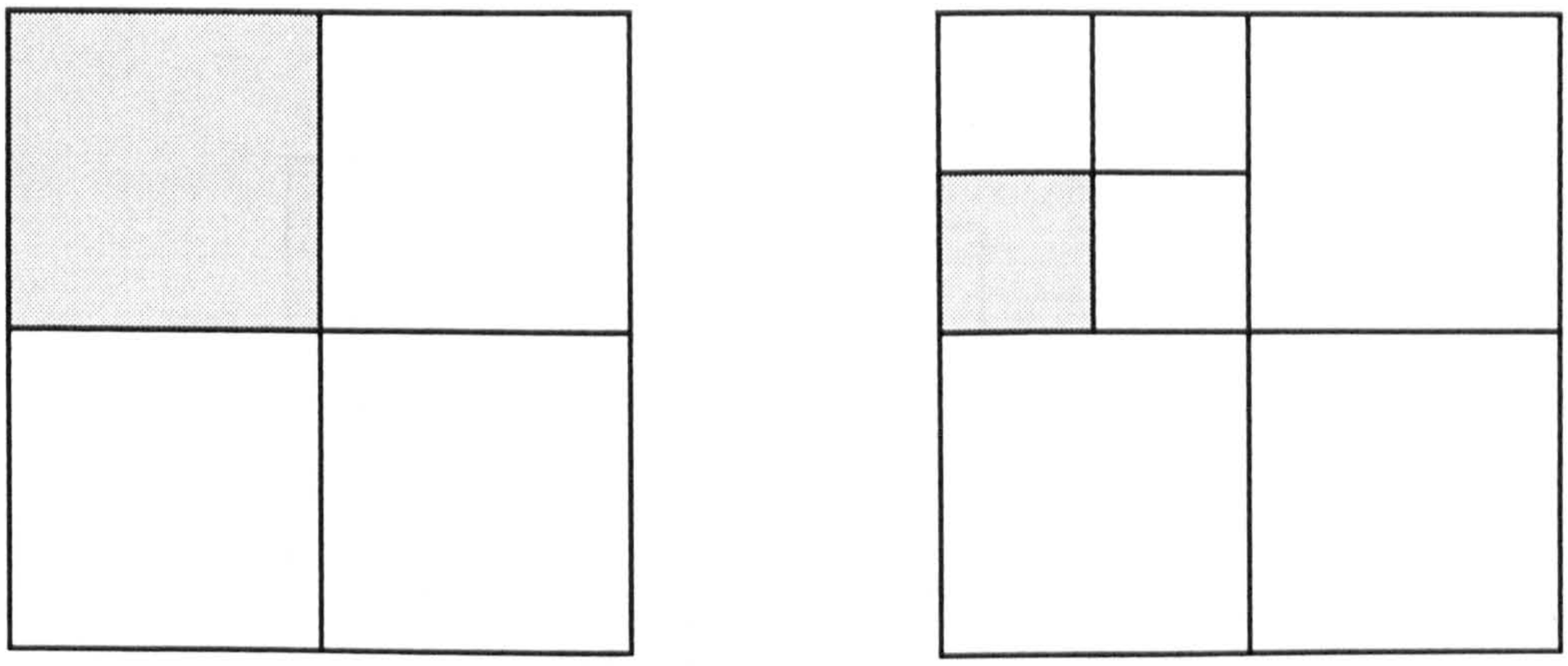
(a)

Decimal	Level
0	2
7	3
8	2
12	2
18	3
19	3
24	2
32	2
36	3
38	3

(b)

Figure 3.6 - A linear quadtree. *(a) Addresses derived by bit-interleaving. (b) Possible storage representation.*

As well as the Morton key, it is also necessary to store for each leaf node the level at which the node appeared in the original tree. Without explicitly recording this information it would be impossible to deduce how large a particular quadtree cell was. For example, the shaded cells in Figure 3.7a and 3.7b are only distinguishable if the level at which they appear is recorded.



(a) *(b)*

Figure 3.7 - Two quadtree cells which cannot be distinguished without the level number being stored. *(a) Level 1. (b) Level 2.*

Other variations of the quadtree have been devised, a comprehensive study of many of which is given by Samet [23, 24]. The quadtree structures that have been reviewed in this section should however be sufficient to assist in a more clear understanding of later chapters.

3.3.3 The R–tree.

The R–tree [39] is a hierarchical data structure, derived from the B–tree [40], that provides an index mechanism which allows data items to be retrieved according to their spatial location. It is again based on the concept of the decomposition of space, in this case the decomposition being dynamic and dictated by the spatial objects themselves.

The rules for building an R–tree are similar to those of a B–tree. All terminal, or leaf, nodes appear at the same level. Each leaf node contains one or more record entries of the form

(I, object_id)

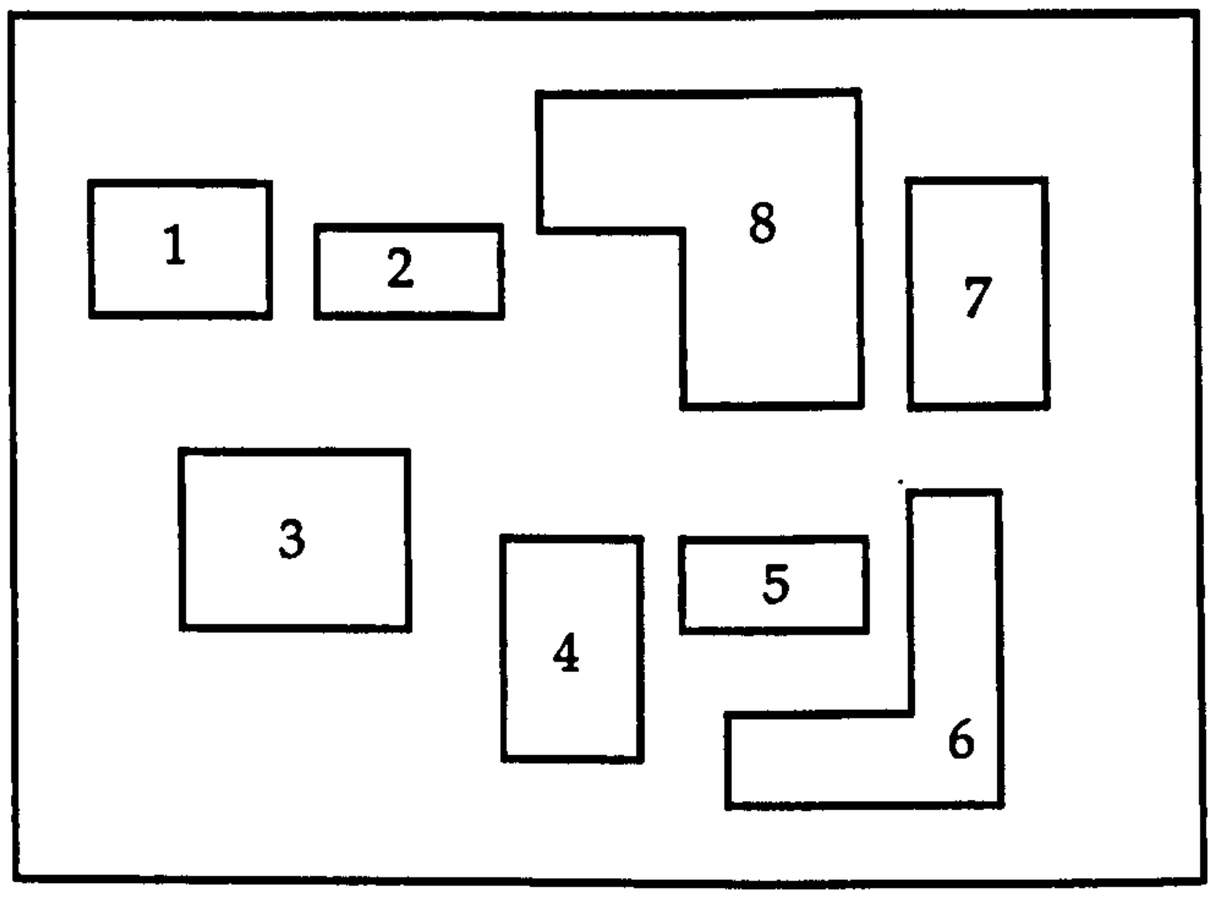
such that I is the smallest rectangle that spatially contains the data object pointed to by the identifier object_id. Non–leaf nodes contain entries of the form

(I, child_id)

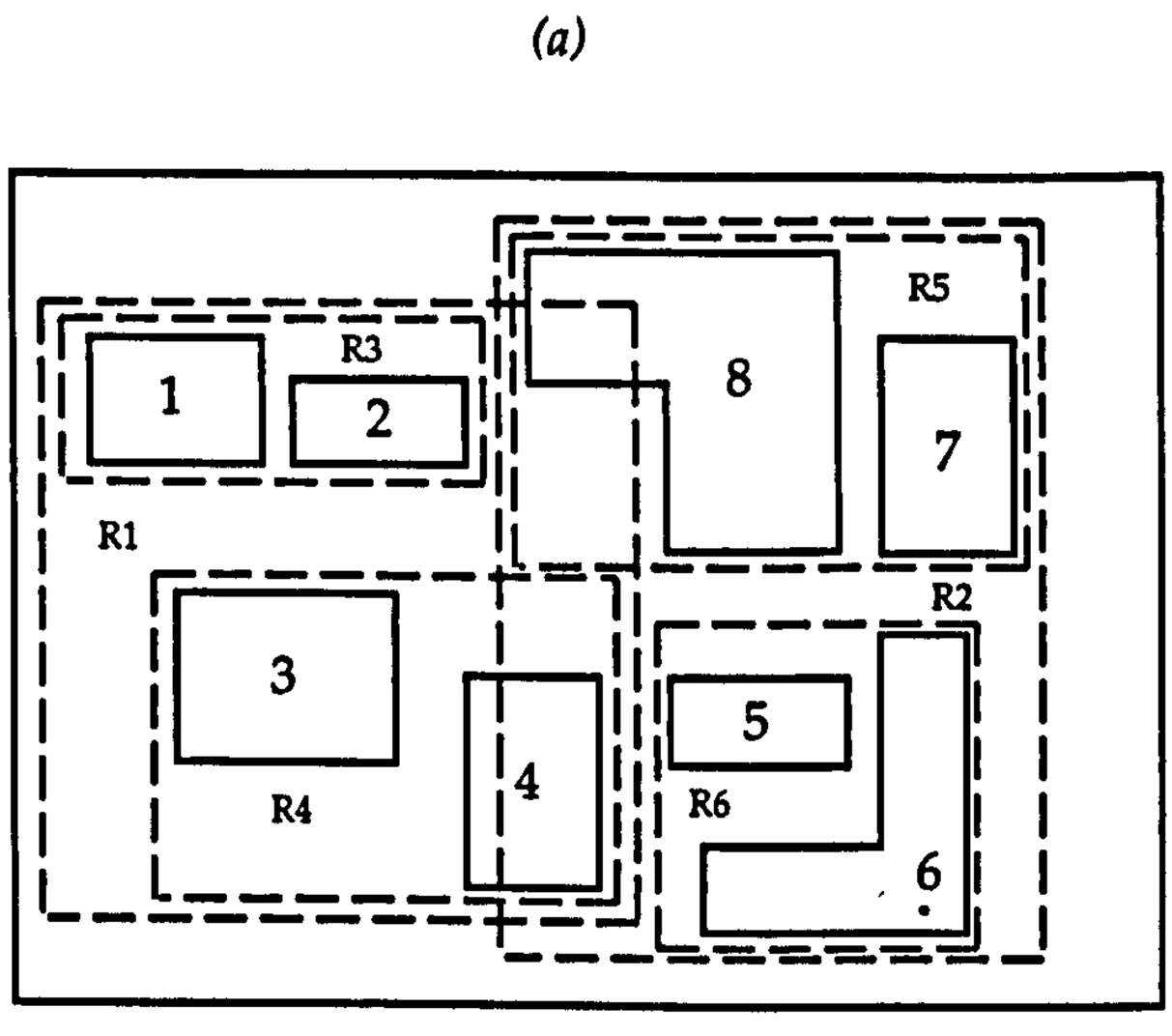
where child_id points to a node in the next lower level of the R–tree and I is the bounding rectangle of all the objects pointed to by the lower node entries.

An R–tree of order (m,M) means that each node in the tree, excluding the root, contains between m (m <= M/2) and M entries. The root node has at least 2 children unless it is

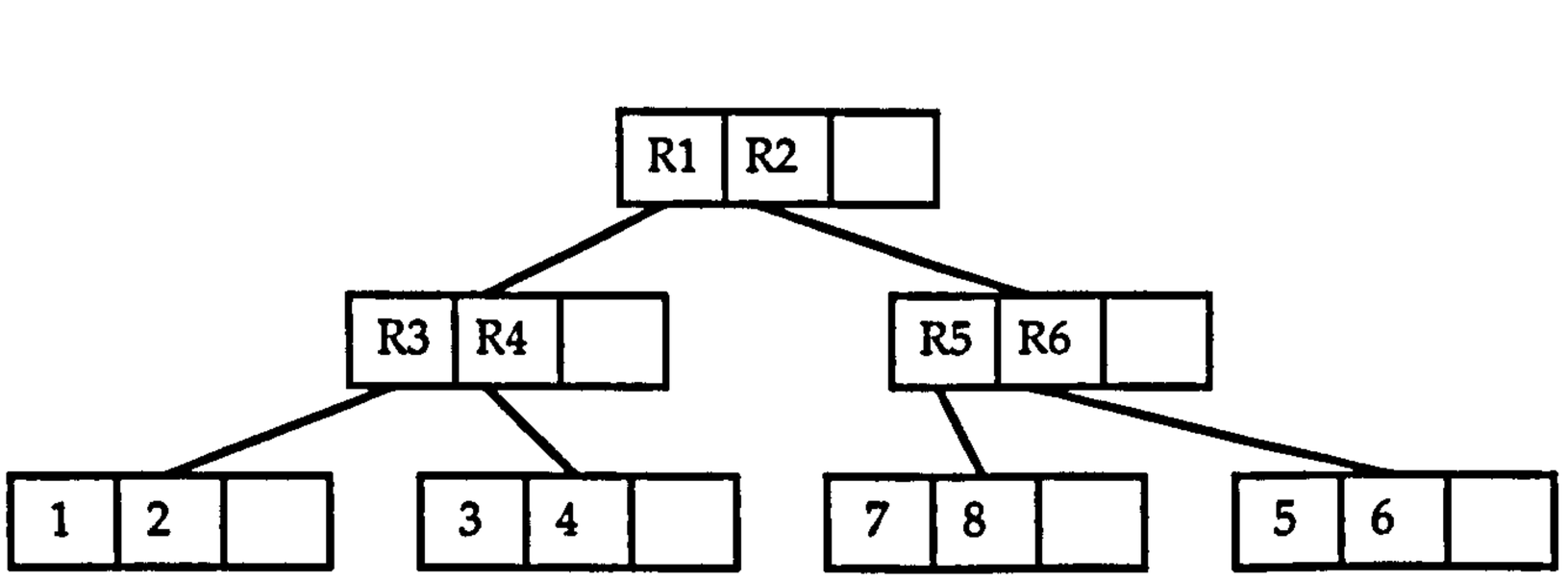
a leaf.



(a)



(b)



(c)

Figure 3.8 - An example R-tree. (a) Eight objects. (b) Decompose into four regions. (c) The resulting R-tree.

The following example demonstrates how an R–tree, with m=1 and M=3 say, is built. Consider the arrangement of objects shown in Figure 3.8a. There are eight objects, greater than the maximum number of entries allowed for each node, which is three.

Therefore the scene has to be decomposed. There are many suggested ways to decompose the object space (three of which are reported by Guttman [39]). One method is to minimise the total area of the bounding rectangles formed as a result of the decomposition. This can have the effect of reducing the average number of nodes visited during subsequent searches of the tree, when compared to other methods.

In the example presented here, the object space is divided into two rectangular areas, R1 and R2, each containing four objects. Both R1 and R2 need to be decomposed further due to the fact that they also contain more than the maximum number of objects allowed. Four new regions, R3, R4, R5 and R6 are thus formed (Figure 3.8b). Each of these regions contains 2 objects. The corresponding nodes are therefore leaf nodes and contain pointers to the appropriate objects (Figure 3.8c).

Any search upon the R–tree descends the tree from the root in a manner similar to a B–tree search (with the difference being that more than one sub–tree under a node visited may need to be searched). For example, consider a search for all objects lying within the query window as shown in Figure 3.9.

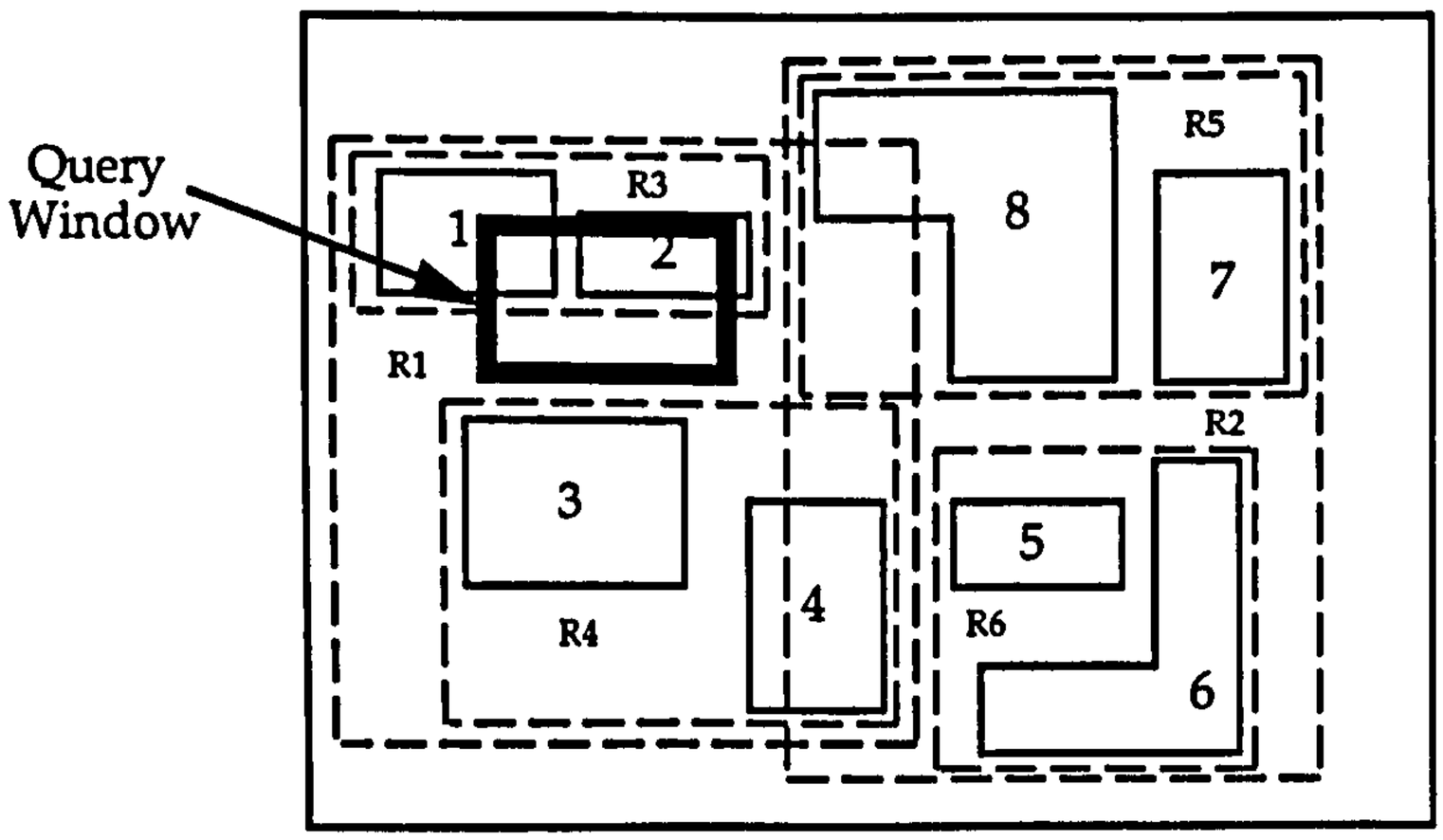


Figure 3.9 - Searching for objects in a query window.

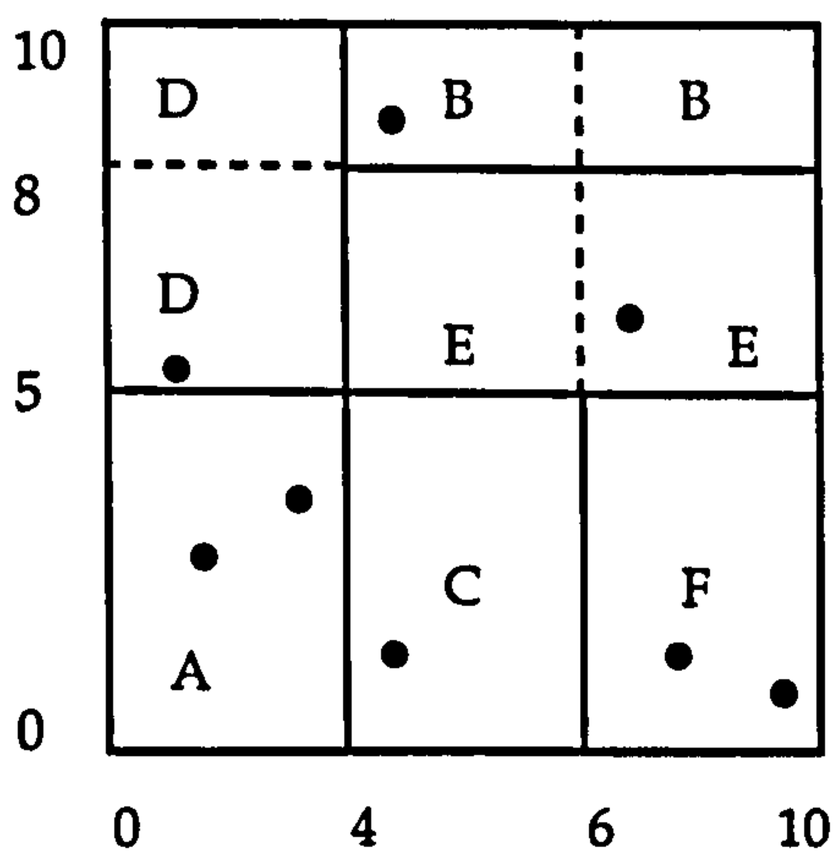
The query window intersects R1 and therefore each of its child nodes need to be searched. R3 is also intersected giving objects 1 and 2 as possibly lying within the area of interest. R4 is not intersected by the query window and therefore objects 3 and 4 can be eliminated from the search. Finally, R2 does not lie within the query window and therefore no further search in this section of the tree need take place.

An alternative to the R–tree is the R^+–tree [41], which avoids overlap among bounding rectangles by splitting them into smaller sub–rectangles. The result is that there may be several paths starting at the root to the same object. This leads to an increase in the height of the tree, thus taking up more space. However, data retrieval time is speeded

up.

3.3.4 The Grid File.

The grid file [42] is a spatial data structure based on the principle of dividing space into rectangles, which are not necessarily equal sized (Figure 3.10). Its main objectives are to retrieve records from disk with at most two disk accesses and to handle range queries efficiently. Rectangles, or grid blocks, are analogous to the cells of the fixed grid method, and each is related to an area of disk storage, known as a bucket. Buckets will have a fixed length, the length of which is defined in terms of the number of primary data items the bucket can accommodate. The form which the primary data item takes, be it a point, a line, a polygon or more complex object, is not of concern here.



Linear scales

x:	0	4	6	10
y:	0	5	8	10

Grid Directory

D	B	B
D	E	E
A	C	F

Data Buckets

Bucket Name	Record 1	Record 2
A	1.9, 2.6	3.8, 3.4
B	4.8, 8.7	
C	4.8, 1.2	
D	1.7, 5.2	
E	6.6, 6.3	
F	7.1, 1.2	0.9, 9.6

Figure 3.10 - The grid file. *In this example there is a bucket size of 2, with 9 grid blocks resulting in 6 storage buckets. Dashed lines partitioning rectangles indicate grid blocks sharing a data bucket.*

Efficient access to data buckets is provided by means of the Grid Directory, which consists of two parts. The first is a dynamic 2-D array containing one entry per grid block. The array element values are pointers to the relevant data buckets, which may in some circumstances be pointed to by more than one array element. Thus a data record may be retrieved in two disk accesses, one for each of the Grid Directory and data bucket. The second part of the Grid Directory consists of two 1-D arrays, called linear scales, which define the actual size of cells (grid blocks) in the x direction and y

direction. Thus spatial queries are supported via an initial search of these linear scales, which are usually located in main memory.

The grid file has good dynamic properties. If a data item is added to a bucket which is not full, the update operation is trivial. Adding to buckets which are full can take one of two courses. If the bucket in question is referenced by more than one grid block, then the bucket may simply be divided into two buckets and the relevant Grid Directory entries updated. If the bucket is referenced by only a single grid block, then the insertion of the new data item will involve the creation of additional grid blocks. This is achieved by adding a division line to one of the linear scales. In the case of deletion, a merging process is carried out analogous to the splitting process for insertion.

3.4 Summary and Conclusions.

The main aim of this chapter has been to introduce the idea of spatial access data structures. This has been achieved through a review of four such data structures, namely, the fixed grid, the quadtree, the R-tree and the grid file. Two of these data structures form the basis of spatial access techniques employed in the data models and database implementations discussed in later chapters. An adaption of the fixed grid is used in the multi-scale data model described in Chapter 5 and the corresponding database implementations described in Chapter 6. The main reason for using the fixed grid is the ease with which it can be implemented. In Chapter 7 a quadtree approach is adopted in the implementation of an Implicit TIN multi-scale database. In this case the quadtree is chosen due to its particular suitability to the Implicit TIN reconstruction algorithms. The reasons for choosing the fixed grid and quadtree methods are discussed in greater detail in the appropriate chapters.

Chapter 4

Multiresolution Representation of Spatial Data

4.1 Introduction.

Maps are required, and therefore produced, at a variety of scales. Section 4.2 is concerned with topographic data and will begin by introducing the notion of scale, before going on briefly to discuss cartographic generalisation. Automated generalisation is then considered, with detailed consideration being given to automated line simplification. In particular, the Douglas-Peucker algorithm is described, its relative advantages and disadvantages being outlined. Following this, methods for storing topographic data at multiple scales are discussed, with mutli-scale data structures being looked at in detail. Section 4.3 concerns itself with the generalisation and multi-scale data representation of terrain data. Special attention is given to hierarchical triangulations, in particular the Delaunay pyramid. Finally, Section 4.4 gives a summary of the chapter and an indication of which of the algorithms and data structures are to be used in the design of the integrated multi-scale data model.

4.2 Generalisation and Scale.

Cartographic generalisation can be thought of as the transformation of the elements of a map so that it will remain legible and meaningful at reduced scale. The scale of a map will depend upon its intended use and the availability of suitable source data from which the generalised map is derived. At present most generalisation is carried out manually by expert cartographers and is arguably 'the most intellectually challenging task for the cartographer' [43].

4.2.1 Generalisation Operations.

There are six main generalisation operations, namely, selection, combination, simplification, exaggeration, symbolisation and displacement [44, 45, 46, 47, 48] (Figure 4.1). For example, consider a group of polygons, representing a village perhaps, on a large scale map. On a small scale map certain of these polygons may not be considered significant enough to appear, and are therefore not included (selection). It may also be appropriate for those polygons that remain to be joined together in some way to form a single polygon (combination). At the next smallest scale the polygon could be replaced by a rectangle (simplification). Depending upon the intended use of the map, this rectangle might also be scaled in some way (exaggeration). At a still smaller scale the rectangle might be replaced by a single point (symbolisation). It may also prove necessary to move a map object (in this case a polygon, a rectangle or a point) during the generalisation process (displacement). This is because many objects in a map appear at a much larger scale than their true ground scale hence causing geographical interference.

4.2.2 Automated Generalisation.

A great deal of cartographic research is currently being carried out in an attempt to find

automated solutions to each of the previously described generalisation operations with much work on the subject appearing in the literature (see, for example, [44, 45, 46, 47, 48]). For practical reasons, this thesis will restrict itself to dealing with line simplification.

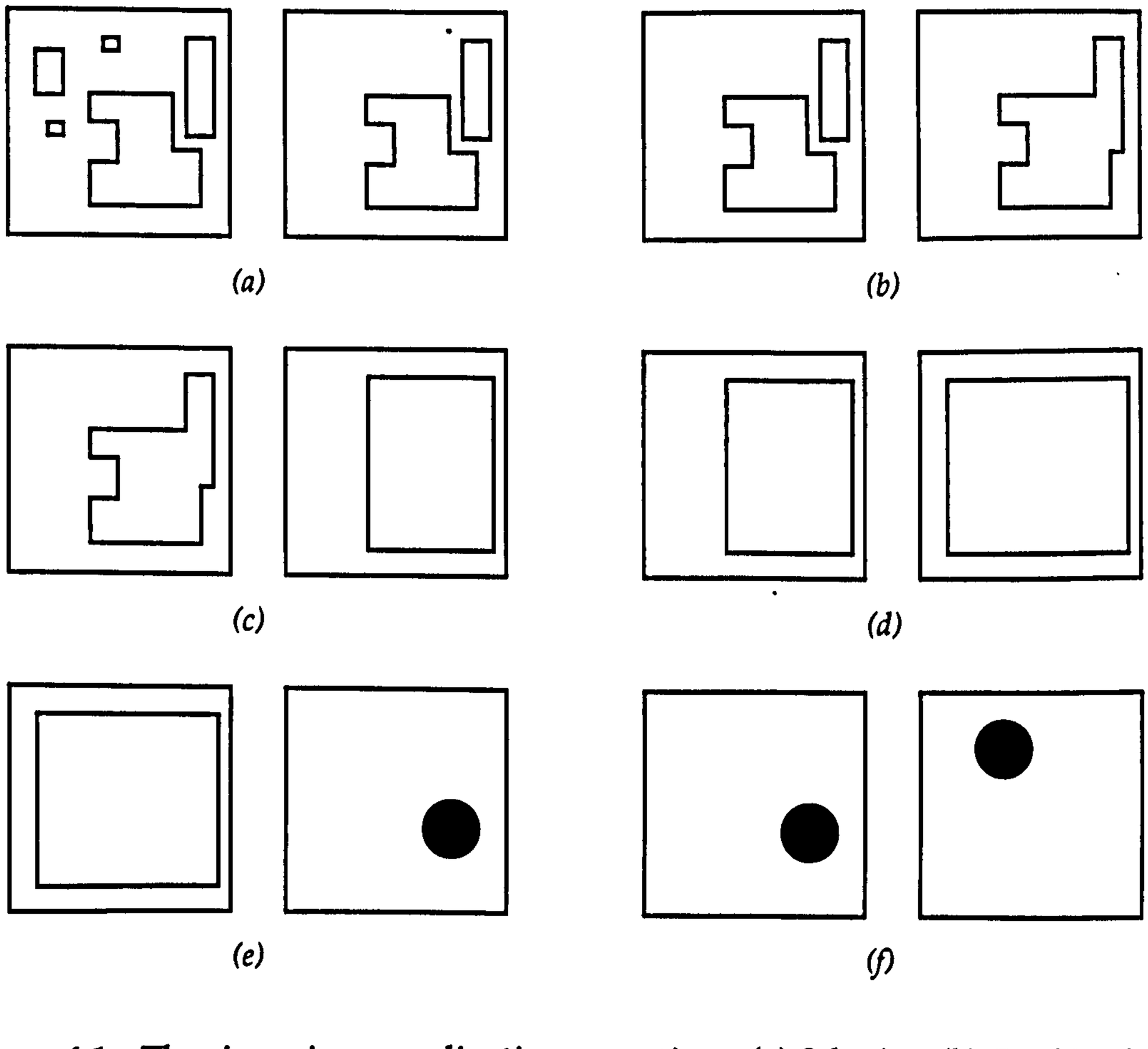


Figure 4.1 - The six main generalisation operations. (a) Selection. (b) Combination. (c) Simplification. (d) Exaggeration. (e) Symbolisation. (f) Displacement.

4.2.3 Automated Line Simplification.

When a small scale map is derived from a large scale geographic data set, only important objects are selected. This section is not concerned with how important objects come to be selected, it is assumed that this has already been done, either automatically or manually. However, the lines from which these objects are made up will often be too detailed at their original scale, the detail being lost due to the limited resolution of the required scale. It is better to use fewer points to represent the lines. It is therefore necessary to apply a line simplification algorithm to each of the lines from which the objects are made up. A number of techniques will now be reviewed. In each case no account is taken of the nature of the phenomena represented by the line (in reality a cartographer might use a different set of rules to simplify a line representing a river to those used for a line representing a road).

4.2.3.1 Algorithm Overview.

The generalisation, or simplification, of a line can be regarded as a reduction in its detail while maintaining its character. The degree of simplification will be reflected in the amount of detail removed and hence the scale suitability of the representation.

When dealing with automated line simplification, two approaches can be taken, namely, point selection and smoothing. Point selection involves choosing points from the original line thought to be relevant at the smaller scale. Smoothing will invariably produce a simplified line which contains points not present in the source line. In this thesis generalisation is being performed in the wider context of a multi-scale data model, wherein data is to be minimised. It therefore follows that a smoothing algorithm is not suitable since points would be produced for inclusion in the smaller scale representation which are not included at the larger scale. This is not in keeping with the hierarchical distribution of points indicative of a multi-scale data model. It appears more appropriate to use a point selection algorithm.

Reviews of line generalisation algorithms have been given by McMaster [49, 50], Zoraster [51] and Abraham [52]. It is reported that point selection algorithms can be broadly classified into two groups, namely, global and local. Global point selection algorithms act upon the entire line as a single entity whereas local algorithms examine subsets of points in turn.

A simple local line simplification algorithm is the n^{th} point algorithm [53] which involves the selection of every n^{th} point from the source line. The value of n can be adjusted to suit a specific scale. It has the advantage of being easy to implement but is widely recognised as not being cartographically correct.

More sophisticated local algorithms have been produced, an example of which is that suggested by Jenks [54]. This considers sets of three points in turn and requires the user to define a minimum distance between points 1 and 2 (MIN1), a minimum distance between points 1 and 3 (MIN2) and a minimum angle (MINA), defined with regards to an angle, A, subtended by the extension of line 12 (from point 1 to point 2) and line 23 (Figure 4.2). If, for any given set of three points, line 12 is less than MIN1 and line 13 is less than MIN2, point 2 is rejected. If line 12 exceeds MIN1 but is less than MIN2, the angle A is inspected. If A is less than MINA then point 2 is rejected.

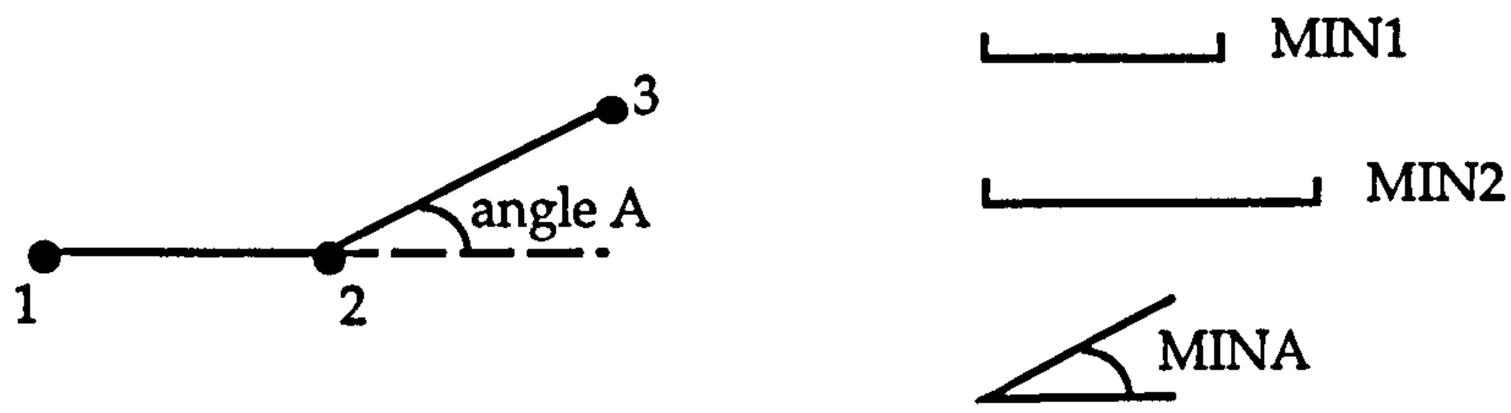


Figure 4.2 - The criteria used in Jenks' line simplification algorithm.

The algorithm starts with the first three points in the line and works its way along the line until point 3 is the last point in the line. A danger with this and similar methods is that slow curves will tend to be over-simplified.

4.2.3.2 The Douglas-Peucker Algorithm.

The global approach appears to offer results which are more satisfactory in cartographic terms. The Douglas-Peucker algorithm [55] is a global line generalisation algorithm which retains the shape information of a line as the number of points describing it is reduced [49] and at present seems to be the most widely used. Harris [56] concludes that this algorithm performs very well in a test designed to see if it would select the same critical points as a trained cartographer.

The algorithm begins by taking the first and last points of the line entity (Figure 4.3a) and joining them by a straight line segment (Figure 4.3b). All intermediate points are then searched to find the point most distant from this line segment (Figure 4.3c). If this distance is less than or equal to a pre-defined tolerance value, the line can be represented by the first and last points. If however the distance exceeds the tolerance value, the line is split in two about the most distant point and two new line segments are formed (Figure 4.3d). One line segment will be defined by the first point and most distant point, the other by the most distant point and the last point. The two line segments are now considered in turn and dealt with in the same way as the original straight line. The procedure is repeated recursively until no line segment has intermediate points lying further away than the tolerance distance (Figures 4.3e to 4.3g).

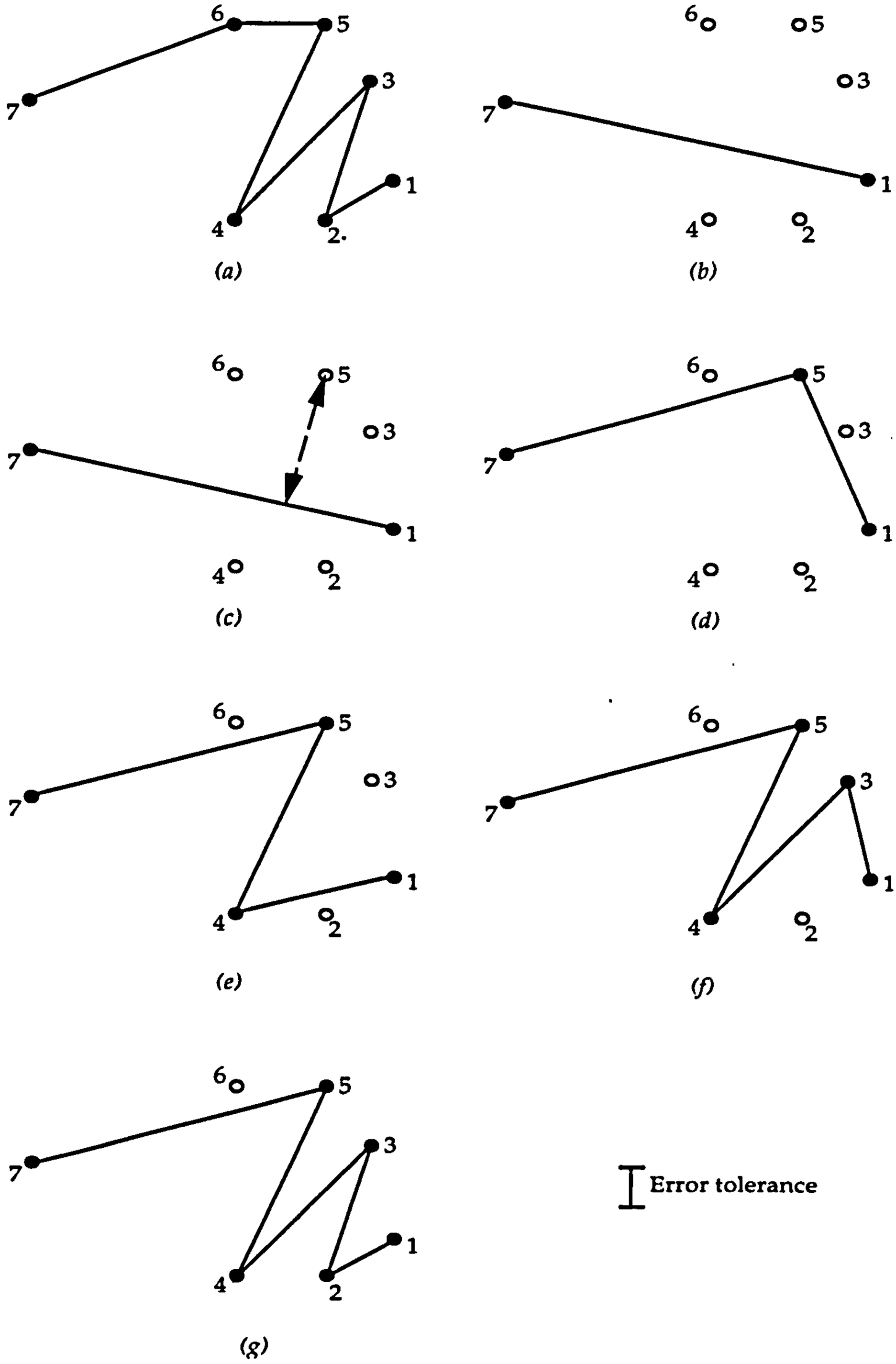


Figure 4.3 - The Douglas-Peucker algorithm applied to a line. (a) Original line. (b) Select first and last point. (c) Point 5 is most distant. (d) Distance of point 5 from line segment (1, 7) exceeds tolerance value and therefore is selected for inclusion. (e) to (g) Repeat, recursively, for line segments (1, 5) and (5, 7).

4.2.4 Representing Line Data at Multiple Scales.

Access to line data at variable scales can be achieved either by storing multiple

versions of each line at predetermined scales; by storing a single large-scale version from which smaller scales are derived using generalisation algorithms; or by means of a multiresolution data structure specifically adapted to retrieving varying degrees of detail. Storage of multiple versions results in significant storage overheads owing to duplication of the constituent vertices between different versions. Retrieval from a single version could incur major processing overheads when deriving a representation of much smaller scale than the original line.

Multiresolution data structures represent a compromise between the approaches. Two such data structures, specifically designed to store cartographic line data at multiple scales, are the line generalisation tree and the BLG-tree.

4.2.4.1 The Line Generalisation Tree.

The line generalisation tree [57, 58, 59], which is closely related to the strip tree [60], is a multiresolution data structure in which vertex duplication is minimised, while providing selective access to those vertices required for a particular scale representation. Each level in the tree corresponds to a particular level of scale significance, the required scales having been determined in advance.

The tree is constructed by firstly assigning to each point a level of scale significance, using the Douglas-Peucker algorithm (or some other point selection algorithm), and then storing that point at its corresponding level in the tree. Therefore, at each level, only those points which are intermediate to points at the previous level in the tree are stored. The order of points within a linear feature can be maintained by either associating with each point a left and right control value which records the number of adjacent intermediate points at the next lower level or by storing a sequence number for each point which records its position in the original line. Although these methods introduce additional data in the form of either the control values or the sequence numbers, it significantly reduces the data overheads of multiple line storage [52]. An example of a line generalisation tree using sequence numbers is shown in Figure 4.4.

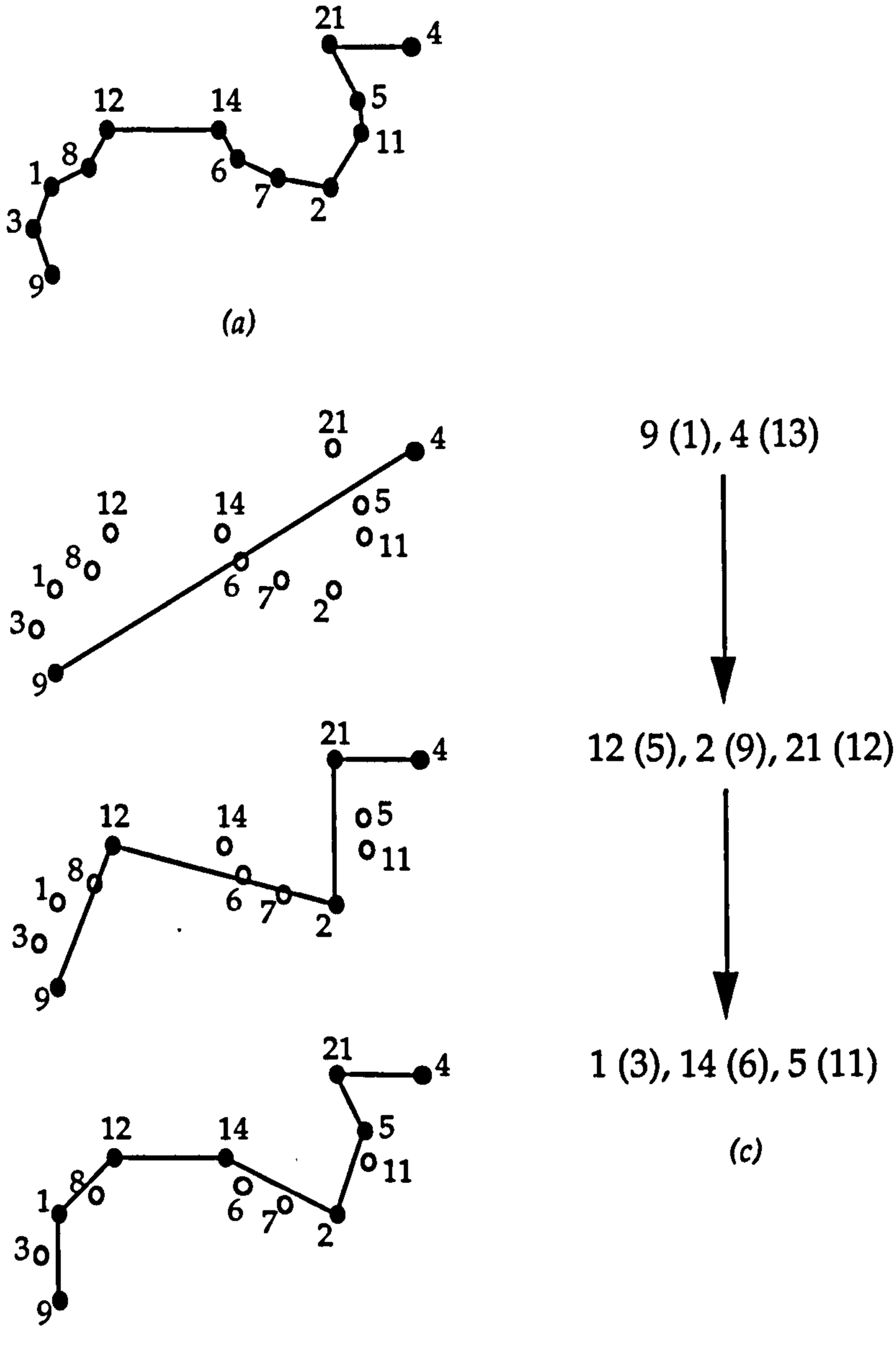


Figure 4.4 - The Line Generalisation Tree. The example shown consists of three levels and adopts the sequence number method. (a) Original line. (b) The result of applying the Douglas-Peucker algorithm. (c) The line generalisation tree representation.

4.2.4.2 The BLG-tree.

A data structure with close similarities to the line generalisation tree is the BLG-tree [61, 62]. This is a binary tree in which each node represents a line segment accompanied by the most distant intermediate point of the original line, its distance, and pointers to the two line segments defined by its current start and end points and the intermediate point. The BLG-tree differs in particular from the line generalisation tree in that the latter employs discrete levels of generalisation. Figure 4.5, taken from van Oosterom [62], shows a line and its corresponding BLG-tree.

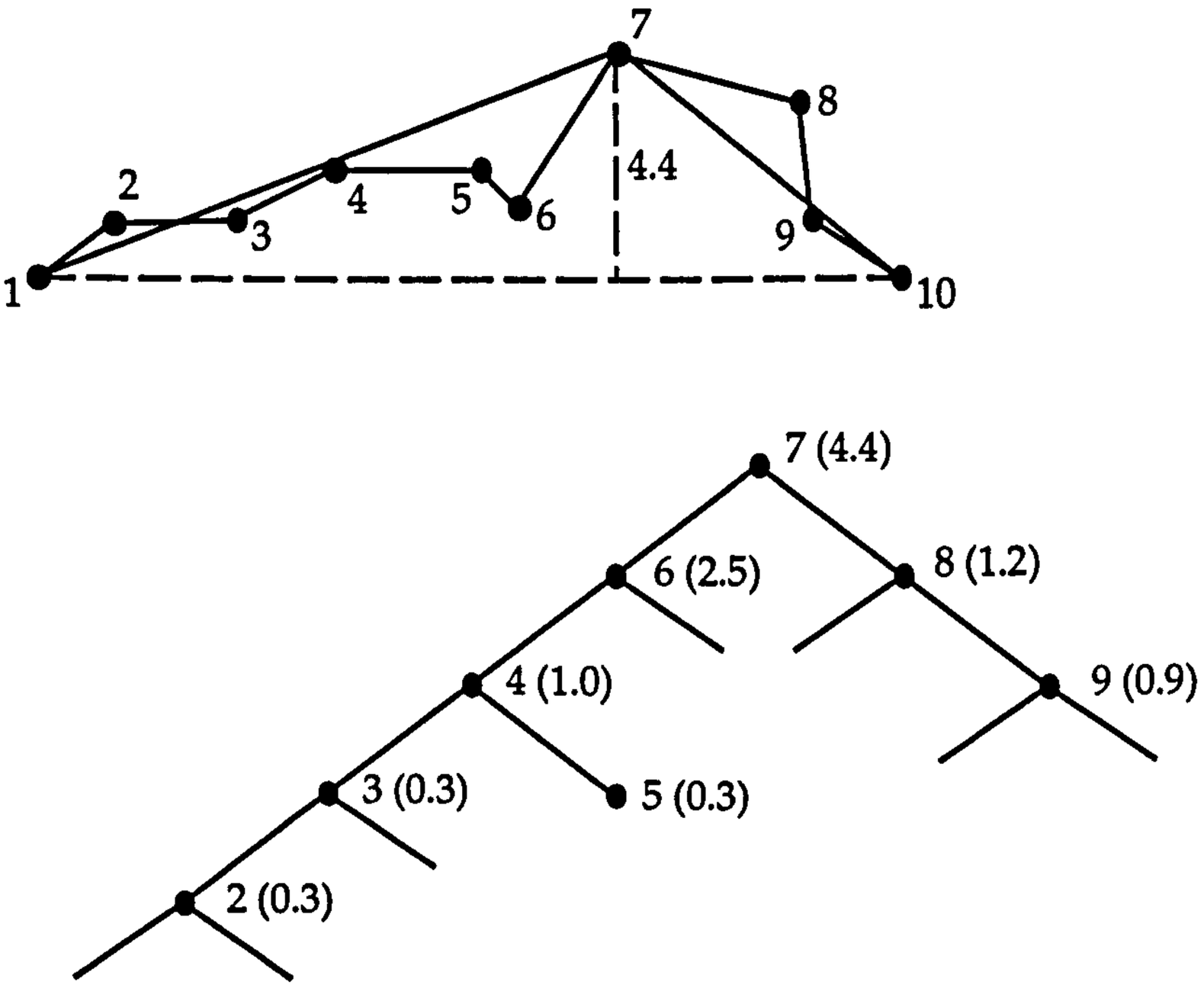


Figure 4.5 - A line and its corresponding BLG-tree.

4.3 Hierarchical Surface Models.

As is the case with topographic data, the resolution at which terrain data is required will often be dependent upon the application for which it is required. The same disadvantages associated with multiple representation and generalisation at run-time also apply. Therefore, a surface model capable of retrieving data according to the need of the application, with limited data duplication and no necessity to perform generalisation at run-time, is required. Thus several hierarchical surface models, which attempt to meet these criteria, have been developed in recent years. Such hierarchical models are usually based on recursively subdividing the surface domain into nested triangulations or rectangles. Hierarchical surface models can therefore be classified into two distinct groups, namely, hierarchical triangulations and quadtree-like models [15]. Quadtree-like models (such as [63, 64]), which make use of domain partition techniques based on rectangles, are suited only to uniformly sampled data points. It is the intention of the data model being designed to cater for irregularly sampled data. It therefore appears that a triangle-based hierarchical model offers the most flexible description of a topographic surface.

4.3.1 Ternary and Quaternary Triangulations.

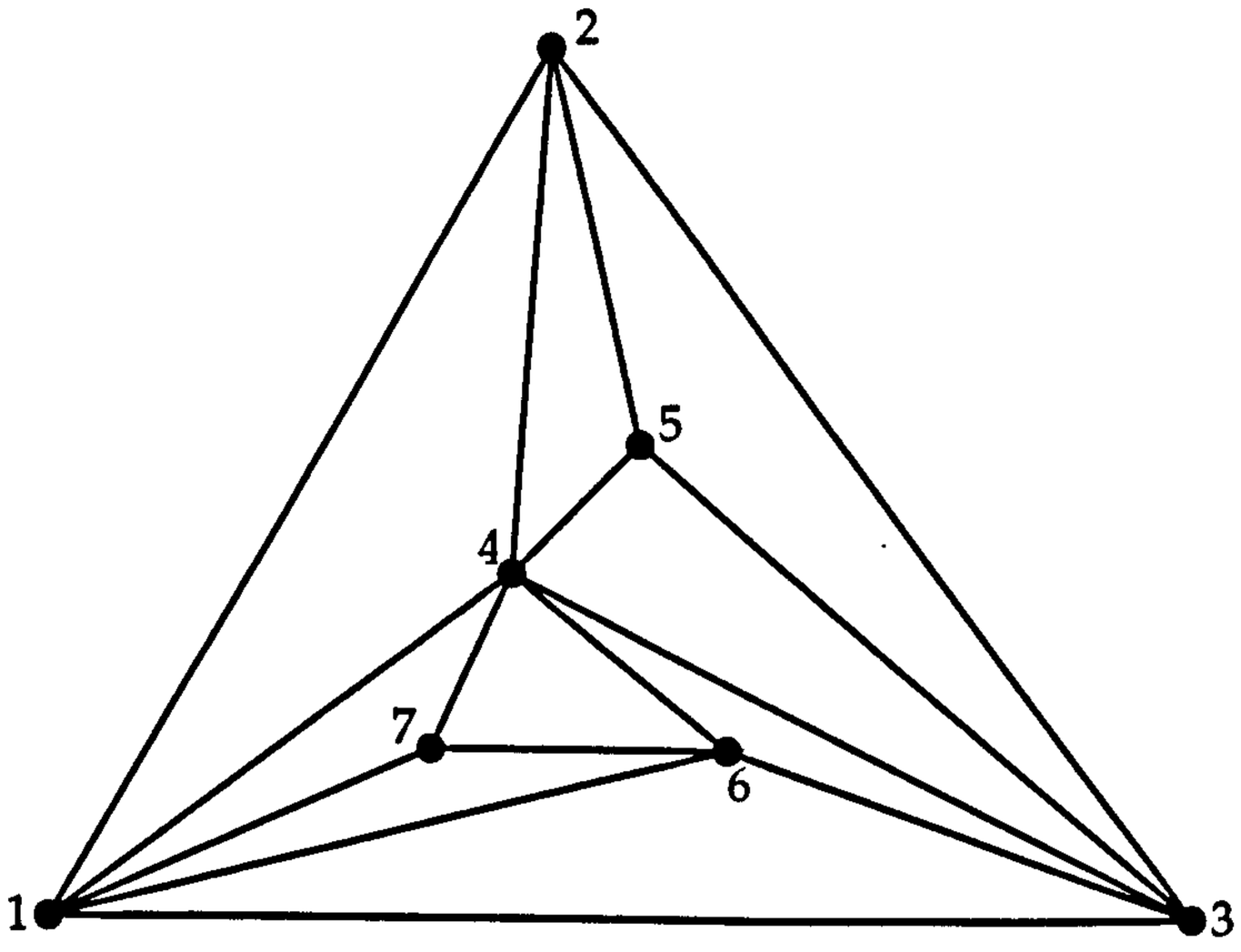
The simplest, and most common, type of hierarchical triangulation schemes are the ternary and quaternary triangulations, examples of which are shown in Figures 4.6 and 4.7 respectively. Consider a set S of points in the plane. A ternary triangulation of S is

represented by a ternary tree, the root of which corresponds to an initial enclosing triangle t_1, whose vertices (p_1, p_2, p_3) belong to S. All projections of unused points, S - {p_1, p_2, p_3} must be contained within t_1. The remaining levels in the tree are a result of combining a currently unused point, p_q, with the vertices (p_i, p_j, p_k) of its containing triangle, t_c, to form three new triangles (p_i, p_q, p_j), (p_j, p_q, p_k) and (p_k, p_q, p_i). These three additional triangles become the child triangles of t_c, and each may in turn become parent triangles to three other child triangles.

In a similar fashion, a quaternary triangulation of S is described by a quaternary tree, the root of which again corresponds to an enclosing triangle t_1. Subsequent child triangles are formed by subdividing a triangle t_c, defined by its vertices (p_i, p_j, p_k) into four subtriangles. This subdivision is achieved by firstly locating three unused points of S, p_q, p_r and p_s, which lie on, or near to, the mid-point of each of the edges of the parent triangle to-be t_c. The points p_i, p_j, p_k, p_q, p_r and p_s are then joined to form the four child triangles (p_q, p_i, p_r), (p_r, p_k, p_s), (p_s, p_j, p_q) and (p_q, p_r, p_s). For both triangulation schemes, the process of subdividing will stop for a particular branch in the tree when either no more suitable remaining points can be found or the error E_i associated with the triangle t_i currently being processed is less than or equal to a pre-defined threshold value E. The value of E_i, the error associated with triangle t_i, can be expressed as follows. Let U_i be the set of currently unused points of S whose projections lie within t_i. Then,

$$E_i = \max \{ e(p_j) \mid p_j \text{ in } U_i \}, \text{ where } e(p_j) = \mid f(x_j, y_j) - z_j \mid .$$

The function f uses the plane defined by the three vertices of t_i to interpolate the z value for a given (x_j, y_j) coordinate pair, the stored z value of which is z_j.



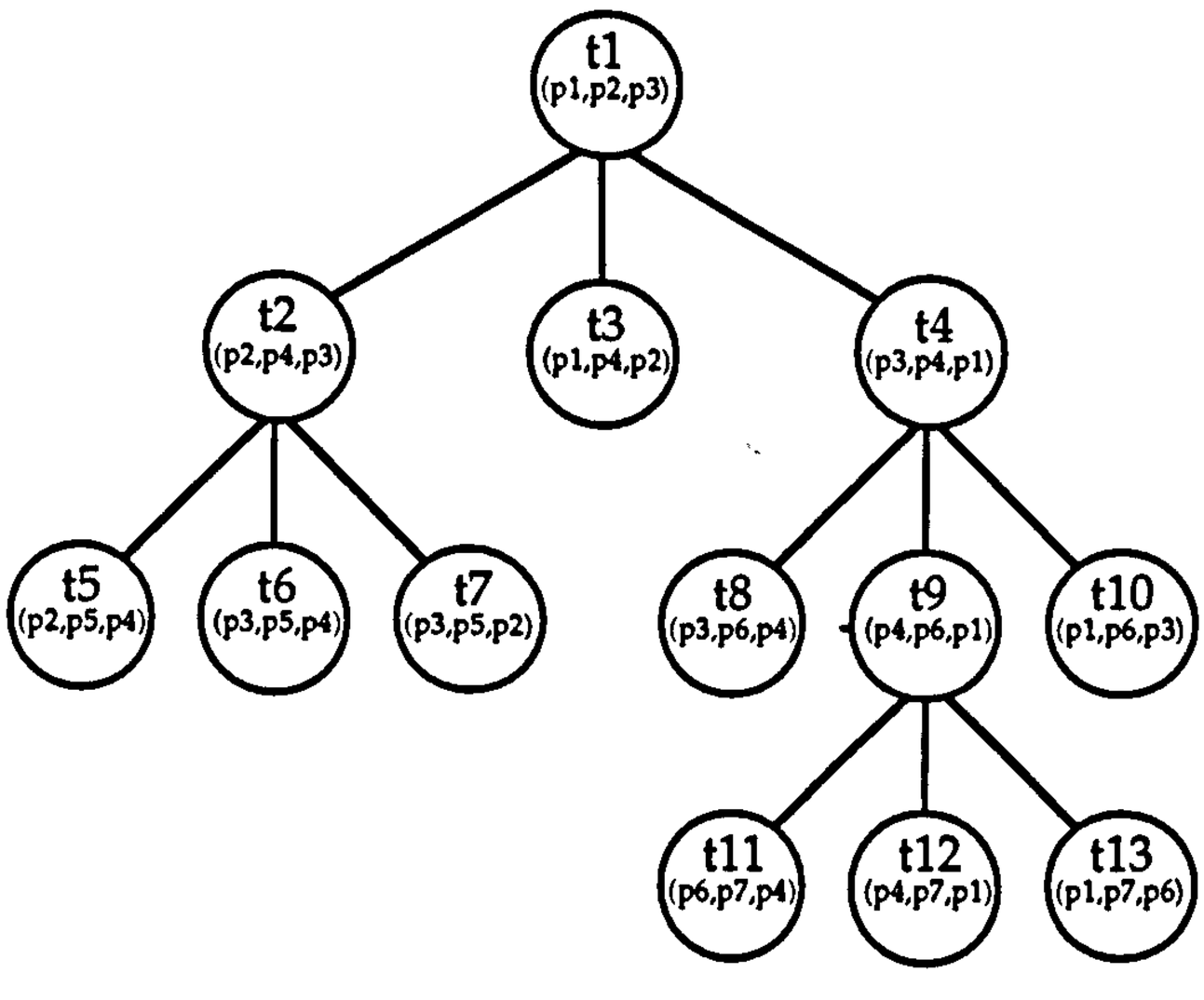


Figure 4.6 - A ternary triangulation and its corresponding ternary tree.

For the hierarchical triangulations described, it is seen that each triangle t_i, not necessarily a leaf triangle, has an error E_i associated with it. Therefore, when a triangulation is required at a specific resolution, defined in terms of an error threshold E_t, it is only necessary to descend each branch of the tree to a depth where the error E_i of the triangle t_i at that level is less than or equal to E_t.

The ternary triangulation is well suited to handling irregularly distributed data but will be prone to produce triangles which are elongated. These long, thin triangles are undesirable since for the purposes of numerical interpolation and visualisation it is more appropriate to have triangles which are as equiangular as possible. When working with data points which are uniformly distributed, the quaternary triangulation is able to provide triangles which behave well with regards the equiangular requirement.

However, since a triangle in a quaternary triangulation may have more than one neighbour along one of its edges, it is liable to produce discontinuous surface approximations.

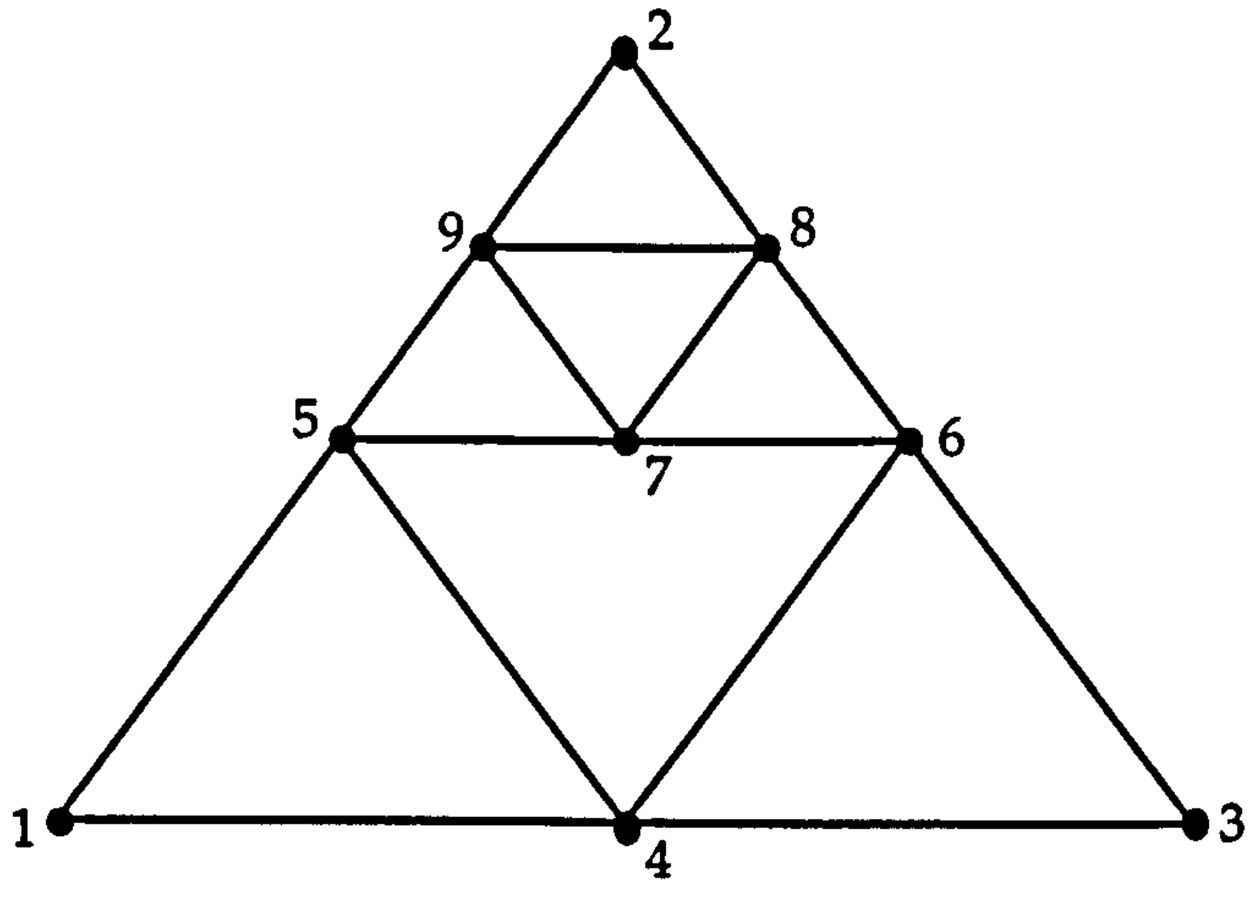


Figure 4.7 - A quaternary triangulation and its corresponding quaternary tree.

4.3.2 Error-Directed Point Selection.

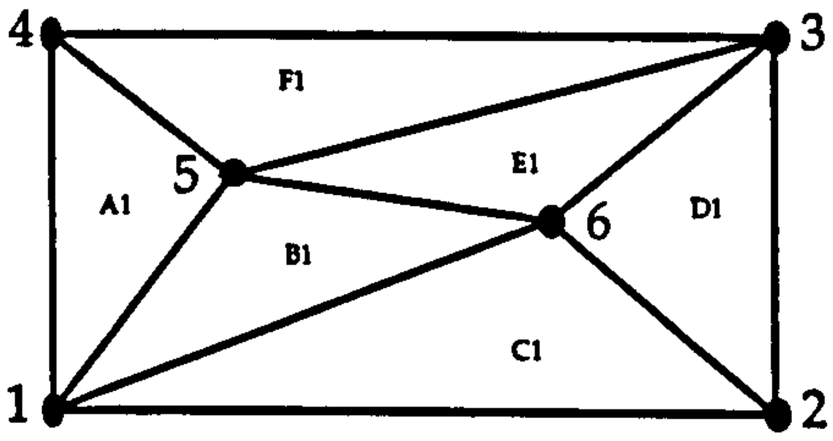
A hierarchical triangulation is a surface approximation based on a subset of a given set of data points S. It is usual to have associated with the triangulation a maximum permissible error E, where E = max $\{E_i\}$. It is desirable to optimise the triangulation with respect to the number of vertices included. Finding the optimal triangulation is not feasible since an exponential number of possible triangulations must be evaluated. However, the error-directed point selection method proposed by De Floriani et al [65] (hereafter referred to as De Floriani's point insertion algorithm), used as part of the process of building a ternary triangulation, while being non-optimal, is satisfactory with regards the number of points needed in the triangulation. In this method, the triangulation is built in a way similar to that of the previously described ternary triangulation. However, in this case the order in which points are inserted into the

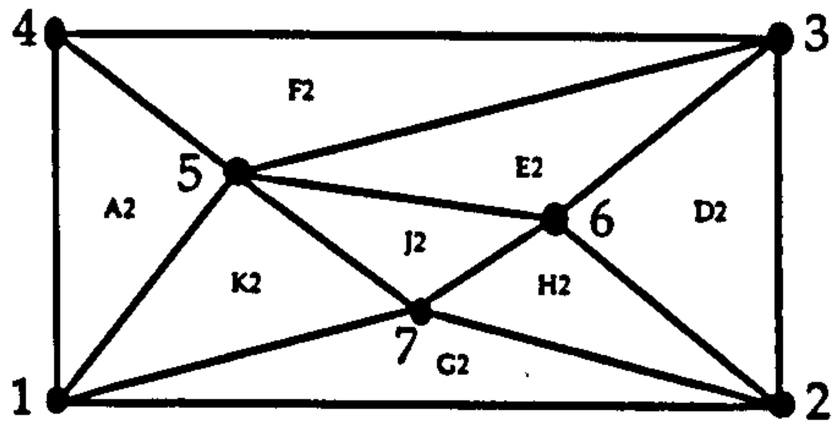
triangulation is considered important. Therefore, for any triangle t_i, with $E_i > E$ and U_i non-empty, a decision has to be made concerning which point p is to be chosen from U_i for insertion into t_i. The principle is to choose that point which is furthest (vertically) from t_i. This method of point selection can be regarded as a 2.5-D equivalent of the Douglas-Peucker algorithm.

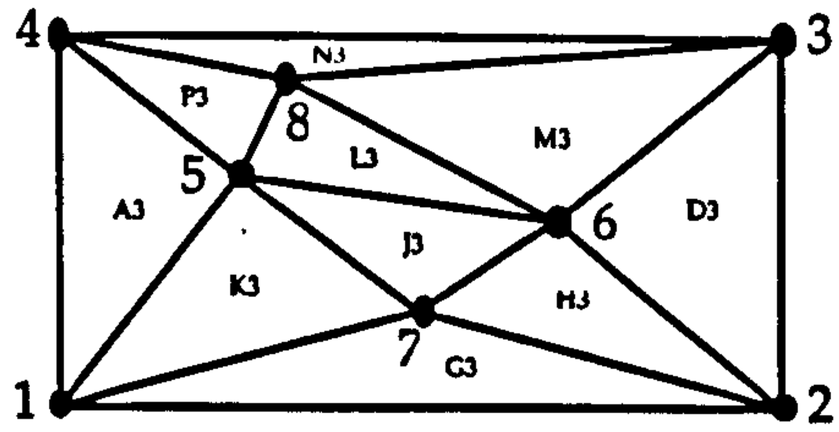
4.3.3 The Delaunay Pyramid.

The ternary and quaternary triangulations, and certain other hierarchical triangulations, such as those presented by Gomez and Guzman [66], Barrera and Vazques [67] and De Floriani et al [65], are each deficient in one of two ways. They either produce approximations which are numerically inaccurate, because of the elongated shape of their constituent triangles, or are well-suited only to regularly sampled data [15]. A hierarchical model, provided by De Floriani, which overcomes these difficulties is the Delaunay pyramid [15]. As the name suggests, this is a hierarchical surface model based on the Delaunay triangulation, and is therefore suited to irregularly sampled data and produces the most equiangular set of triangles. The Delaunay pyramid, in its original form, consists of a hierarchy of Delaunay triangulations, each level of which contains progressively greater detail (Figure 4.8). Each triangulation T_i has an error E_i associated with it. Therefore a pyramid consisting of m levels is represented by a sequence of Delaunay triangulations $[T_0, T_1, ..., T_{m-1}]$ where $E_i <= E_{i-1}$, i = 1, 2, ..., m-1.

The pyramid is built from a set of points S by firstly constructing an initial constrained Delaunay triangulation. This will include those points of S which define the convex hull of S or are the most important surface-specific points (peaks, pits and passes) and lines (ridges and valleys). In the scheme described by De Floriani each triangle is defined by its three vertices and its three adjacent triangles (although it is pointed out that any other of the 9 possible triangle-vertex-point relationships could be used).







Id	Type	Vertices	Adjacent Triangles	Child Triangles
A1	0	1 5 4	B1 F1 -	A2
B1	0	1 6 5	C1 E1 A1	G2 H2 J2 K2
C1	0	1 2 6	- D1 B1	G2 H2
D1	0	6 2 3	C1 - E1	D2
E1	0	3 5 6	F1 B1 D1	E2
F1	0	3 4 5	- A1 E1	F2
A2	1		K2 F2 -	A3
D2	1		H2 - E2	D3
E2	1		F2 J2 D2	L3 M3
F2	2			L3 M3 N3 P3
G2	0	1 2 7	- H2 K2	G3
H2	0	2 6 7	D2 J2 G2	H3
J2	0	6 5 7	E2 K2 H2	J3
K2	0	7 5 1	J2 A2 G2	K3
A3	1		K3 P3 -	
D3	1		H3 - M3	
G3	2			
H3	2			
J3	1		L3 K3 H3	
K3	2			
L3	0	5 6 8	J3 M3 P3	
M3	0	6 3 8	D3 N3 L3	
N3	0	8 3 4	M3 - P3	
P3	0	5 8 4	L3 N3 A3	

Figure 4.8 - An illustration showing the inter-level relationships that exist in the Delaunay pyramid.

Pyramid construction proceeds by taking a currently unused point p from the set S, and adding it to the approximated surface, which is then re-triangulated. The point p is the point furthest away (vertically) from the approximated surface. The process of adding a point and re-triangulating is repeated until each triangle t has an associated error e less than or equal to E_0. The next level, initially identical to the first level, is then created. Further points are added from S until the error threshold for that level is reached. New levels continue to be added to the pyramid until the most detailed level (level m-1) has been created to the required accuracy.

It is likely that some of the triangles at a particular level will be completely retained in the next lower level or will differ only in regard to their adjacent triangles. The Delaunay pyramid overcomes data duplication by storing triangles as either internal, boundary or external. An internal triangle is defined by its vertices and its adjacent triangles. Boundary triangles will consist of a pointer to a parent triangle, from which its vertices will be obtained, and a reference to its three new adjacent triangles. An

external triangle is completely described by a pointer to a higher level triangle. Each triangle within a level also references those triangles at the next more detailed lower level which intersect it. This assists hierarchical spatial search within a Delaunay pyramid to determine which triangle a given point lies in. An example of a three-level Delaunay pyramid is given in Figure 4.8.

It should be noted that the storage benefits gained by adopting the internal, boundary and external triangle approach is fully realised only when changes in detail between levels are small. This is because large changes in scale between successive levels will cause many of the triangles in the parent triangulation to be replaced by new, non space-saving internal triangles in the lower level triangulation.

4.3.4 Adaptive Hierarchical Triangulation.

The Delaunay pyramid, along with all terrain models based solely on Delaunay triangulation, tends to ignore the third dimension when deciding upon the topological relationship between points, and may therefore produce edges that contradict the topology of the surface being modelled. Scarlatos and Pavlidis [68] attempt to overcome this problem by proposing a non-Delaunay triangulation scheme, termed adaptive hierarchical triangulation, which produces a multiresolution terrain model that adapts itself to surface characteristics. However, this method does not appear to be suited to the inclusion of topographic (non-elevation) feature data in the model since the algorithm it employs is dependent upon all data points having an elevation value.

4.3.5 The Constrained Delaunay Pyramid.

To counter the apparent inadequacy of the Delaunay pyramid, De Floriani and Puppo [22] have proposed a dynamic, easy-to-code algorithm to produce a constrained Delaunay pyramid (CDP). The CDP modifies the original Delaunay pyramid by allowing insertion of chains of edges belonging to surface edges. The ability to introduce constraints into a pyramid ensures that specific linear features, such as valleys and ridges, can be retained as connected edges within each level of the pyramid. In principle, this mechanism for constraining the triangulation facilitates the inclusion within it of any point, line or polygon feature, whether physical or cultural. Details regarding the insertion of constraining edges into an existing triangulation have been given in Chapter 2.

4.4 Summary and Conclusions.

This chapter has attempted to provide an introduction to the subject of automated map generalisation. It has suggested that, at present, multi-scale data structures offer the most viable approach by which GIS can cater for spatial data at multiple levels of generalisation. Of the data structures described, two are selected for use in the

integrated multi-scale data model and database implementations described in later chapters. It is thought that at present terrain data can best be catered for by employing the error-directed point selection technique to assist in the construction of a CDP. Having decided to adopt the CDP approach for terrain data storage it follows that the line generalisation tree (in conjunction with the Douglas-Peucker algorithm) offers the most appropriate technique for storing (and producing) multiresolution line data. This is because the CDP and line generalisation tree both employ fixed levels of detail. It might be argued that the BLG-tree offers greater flexibility than the line generalisation tree in that detail levels are not fixed, and therefore a greater range of scales are immediately available for retrieval. However, the BLG-tree does not integrate well with the CDP in its present form. Note also that a partial generalisation at run-time approach, used in conjunction with the CDP and line generalisation tree, would provide for the retrieval of the full range of scales. This approach has been adopted in the scale-independent database presented by Abraham [52].

Chapter 5

A Multiresolution Topographic Surface Model

Chapter 5 A Multiresolution Topographic Surface Model

5.1 Introduction.

This chapter outlines the design of a data model, or data storage scheme, suited to the efficient storage and retrieval of terrain and topographical feature data at multiple scales. The data model, which is based upon several of the data structures described in earlier chapters, is described in Section 5.2. Section 5.3 provides detail, in pseudo-code form, of the data model construction algorithm. In light of recent criticism of the Douglas-Peucker algorithm, Section 5.4 provides a number of reasons as to why it has been adopted for use in this project. Section 5.5 draws attention to some of the limitations of the data model described in Section 5.2, and gives some indication as to how these limitations can be removed. A conclusion and chapter summary in given in Section 5.6.

5.2 A Hierarchical Model for both Topographic and Terrain Data.

Previous chapters have given details of data structures suited to either the efficient multi-scale storage of topographic feature data, such as the line generalisation tree, or to the efficient multi-scale storage of terrain data, such as the constrained Delaunay pyramid (CDP). The algorithms developed to produce generalised versions of topographic data and terrain data have also been reviewed. This section introduces a new data storage scheme, termed the Multiresolution Topographic Surface Model (MTSM), which allows for multi-scale storage of both data types in a single data model.

5.2.1 Model Overview.

The MTSM is a spatial access scheme suited to the efficient storage and retrieval of terrain and topographic data at multiple scales. Two previously described data structures, namely the CDP and the line generalisation tree, form the basis of the model. The CDP is chosen due to its ability to represent surfaces at multiple levels of detail; incorporate points located at arbitrary coordinates; and include constraining features. The line generalisation tree, as indicated in Section 4.4, is preferred to the BLG-tree due to the fact that, as with the CDP, it employs discrete levels of detail.

The vertices, or points, from which the topographic data is made up (representing point, line and polygon features) are merged with the terrain points defining the surface to form a single data set. These combined points are then used to construct a CDP. Each line feature is represented by a line generalisation tree. A unique aspect of the work is the ability to include topologically structured features, such as pylons (point), railways (line) and county borders (polygon) within the pyramid. In the case of line and polygon features, these occur as chains of constrained edges within the pyramid. These are in addition to those surface features necessary to characterise the shape of the surface, such as ridges and valleys.

Point, line and polygon features, all of which are embedded as constraints within the pyramid, are arranged in a hierarchical manner. Each polygon is stored as a collection of references to one or more lines, each of which in turn reference, via a line generalisation tree, vertices within the pyramid. Point features are represented as direct references to these vertices. This approach is based on the point, line and polygon dictionary method described in Section 2.3.2. In certain cases, constrained edges within a triangulation may represent more than one feature. For example, a national boundary very often coincides with some physical boundary, such as a river. Furthermore, objects of interest may consist of sets of point, line and polygon features. This can be illustrated by considering a factory as an object of interest, which is itself made up from point, line and polygon features. To accommodate such occurrences, while at the same time minimising data duplication, an additional entity, referred to here as an object, is introduced into the data structure hierarchy. Each object maintains a list of pointers to the appropriate point, line and polygon features from which it is made up. For the first example, the physical boundary and the political boundary, each stored as a separate object, would refer to the same embedded feature or list of features. The factory, also stored as an object, would refer to each of its constituent point, line and polygon features.

A diagrammatic summary of the MTSM is given in Figure 5.1. The Object List consists of a series of object descriptions, each of which represents a single object. Each object description is composed of a unique object identifier, attribute data which describes what the object is and to what class of object it belongs, plus a list of references to the polygon, line and point features from which the object is made up. Polygon descriptions, which are stored in the Polygon List, are each made up of a unique polygon identifier plus references to the relevant constituent line parts. Line descriptions are held in the Line Lists. The series of n line descriptions for a particular line correspond to the n levels of the line generalisation tree for that line. Each individual line description represents a single level in the line generalisation tree and is made up of a unique line identifier, a list of references to the points which become relevant at that level and a list of corresponding sequence numbers indicating the position of each point in the original line.

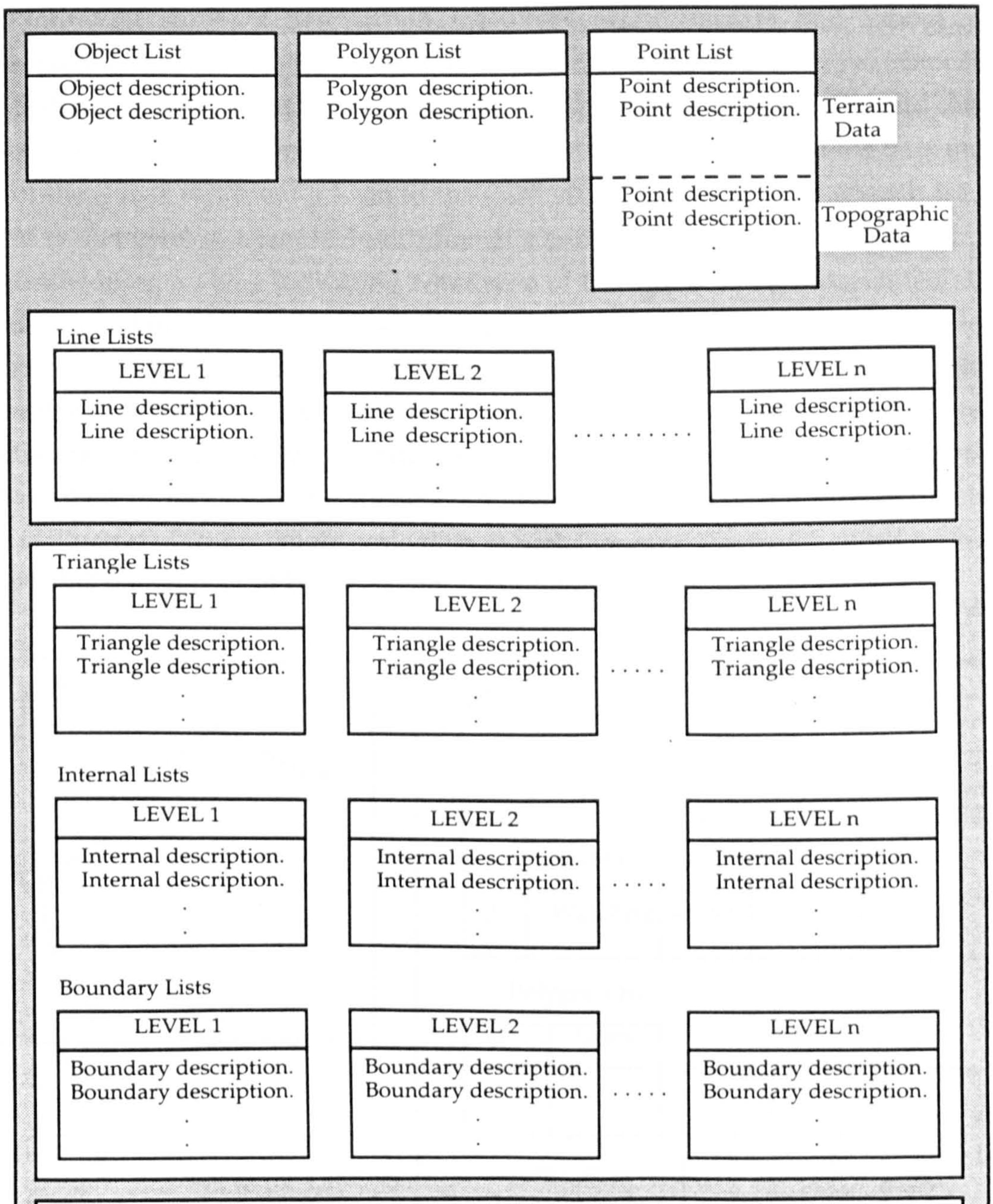
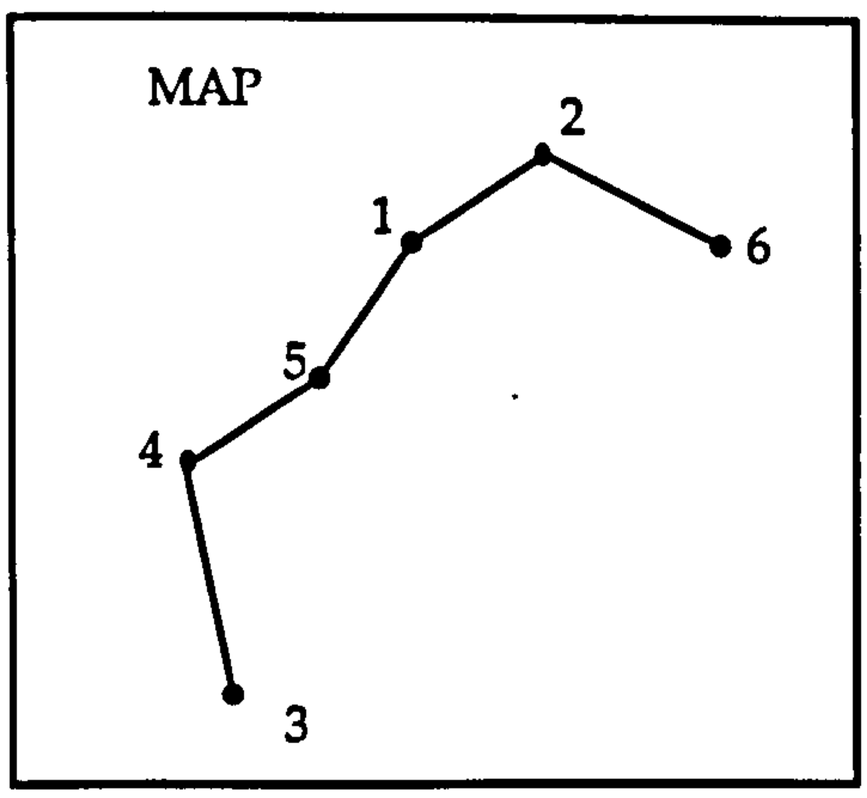


Figure 5.1 - The multiresolution topographic surface model. *In this case the model is shown with n levels of scale.*

The Point List stores a description for every point (terrain and topographic) represented in the data model. A point description comprises of a unique identifier, a value indicating the level at which the point is first included in the model and the x, y and z coordinates of that point. There is a Triangle List at each level of the data model. Each of these lists represent a level in the CDP and stores the details of each triangle present at that level. A triangle description at a particular level is made up of a unique triangle identifier, a value indicating what type of triangle is being represented at that level (internal, boundary or external) plus a reference to the appropriate description in either the Internal List or Boundary List. Each internal description holds the full geometry and adjacency information for internal triangles. Boundary descriptions store the adjacency information for boundary triangles. The Object Grids and Triangle Grids are described in the following section. The relationships between topographic data types is illustrated in the single-scale data model shown in Figure 5.2. With regards to topographic data it should be noted that no topological information is stored.



DATA MODEL

Object List

Id	Description	Class	Polygons	Lines	Points
1	Severn	River	-	1,2	-
2	Wal/Eng	Border	-	1,2	-

Polygon List

Id	Lines

Line List

Id	Points
1	3,4,5
2	6,2,1

Point List

Id	X	Y
1	500.0	200.0
2	520.0	215.0
3	470.0	120.0
4	455.0	160.0
5	485.0	174.0
6	545.0	200.0

Figure 5.2 - A single-scale topographic data model, made up from object, polygon, line and point entities. In this case there are no polygon parts.

It is noted that Kraak and Gazdzicki [69] present a triangle based terrain model capable of representing both the terrain surface and spatial objects related to it. This model is applied to what they term Cartographic Terrain Modelling (CTM). The fundamental difference between CTM and the model presented in this chapter is that CTM is limited to single-scale representation of data.

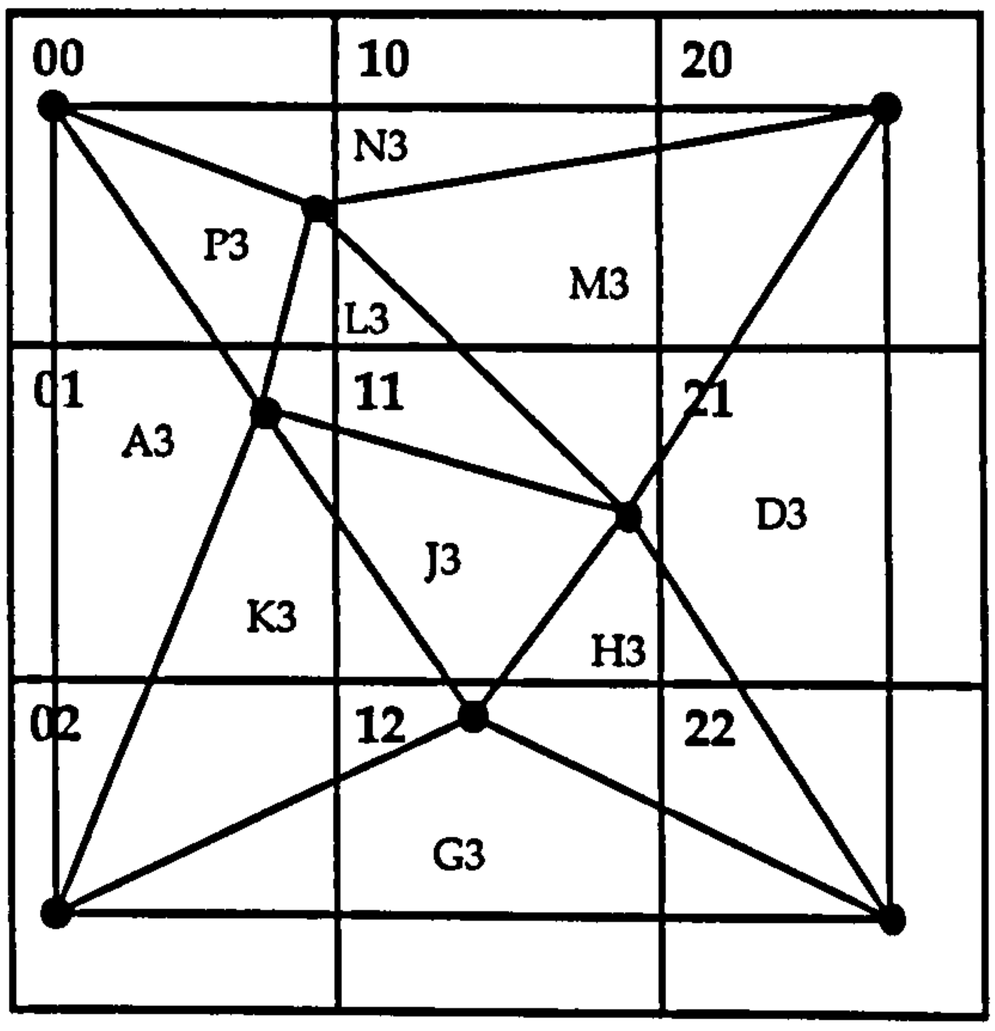
5.2.2 Providing Spatial Access to the Model.

It is important to ensure efficient access to the data model. Since the intended use of the model is to store spatially extensive data, spatial indexing of some sort is required. When deciding on a suitable indexing scheme an important consideration is the type of queries which will typically be presented to the eventual database. Two fundamental queries are of the form 'Retrieve all data of a particular type within a particular area' and 'What data items lie at a particular point ?'. Each of the four data structures reviewed in Section 3.3 have been shown in the literature to be suited to such queries. However, the parent triangle to child triangles pointer method employed by De Floriani [15] in the Delaunay pyramid would appear to be suited to only queries of the latter type. For the purposes of this thesis, two separate indexing techniques have been implemented. The first, a regular grid overlay scheme, is described in the following section. The second, based on the quadtree data structure, will be detailed in Chapter 7.

5.2.2.1 The Regular Grid Overlay Indexing Method.

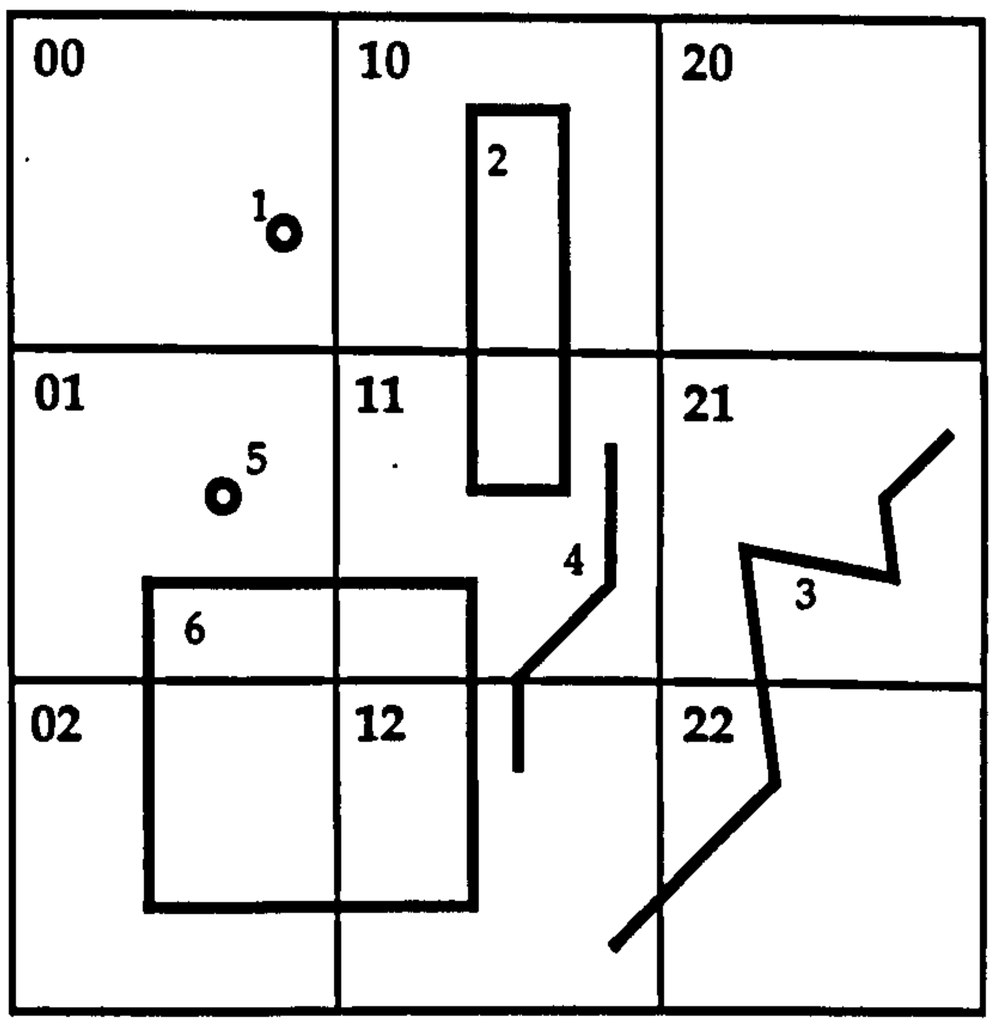
Efficient access to the hierarchical model is provided by introducing spatial indexing at each level in the pyramid. The method described here employs a regular grid overlay scheme, in which each grid cell maintains a list of references to all data which intersect it. This indexing technique, which is based on the fixed grid method (Section 3.3.1), replaces the parent triangle to child triangles pointer method employed by De Floriani [15]. The regular grid overlay technique differs from the fixed grid method in that whereas the cells in the latter correspond to areas of storage in which data items themselves are stored, the cells in the former contain references to data items which themselves are stored elsewhere.

A decision has to be made as to which of the data in the model is to be spatially referenced. In order to arrive at this decision it is again necessary to consider the type of query the data model will have to satisfy. Four typical queries need to be catered for, namely 'Retrieve all objects within a particular area', 'Retrieve all triangles within a particular area', 'What object lies at a particular point ?' and 'What triangle lies at a particular point ?'. Therefore, it becomes apparent that spatial indexing is required on objects and triangles. Also, since the spatial extent of certain objects may differ between levels (due to the generalisation of constituent line parts) and certain triangles may be relevant to some levels in the pyramid and not to others, the spatial access structure has been separated into levels. Therefore, at each level in the pyramid there is a triangle grid (Figure 5.3a), referencing all triangles relevant to that level, and, similarly, an object grid, referencing all objects (Figure 5.3b).



Cell Address	Intersecting Triangles
00	A3 P3 N3 L3 M3
10	N3 L3 M3
20	D3 M3 N3
01	A3 P3 L3 J3 K3
11	L3 J3 K3 M3 H3 D3
21	D3 M3 H3
02	A3 K3 G3
12	K3 G3 H3 J3
22	H3 G3 D3

(a)



Cell Address	Intersecting Objects
00	1
10	2
20	
01	5 6
11	2 6 4
21	3
02	6
12	4 6 3
22	3

(b)

Figure 5.3 - Spatial indexing provided by the regular grid overlay method. (a) The Triangle Grid. (b) The Object Grid.

An object or triangle is deemed to be related to a particular object cell or triangle cell, respectively, if any part of that object or triangle intersects the cell. Each cell description (Figure 5.1) will therefore consist of a x,y coordinate indicating the location of the cell plus a list of references to all objects or triangles which intersect that cell. It is clear that the number of cells in each spatial grid will determine the optimality of searching operations. A dense grid will, in general, be more efficient in terms of search time than a more refined grid. However, this benefit has to be weighed against the resulting increase in storage requirements. McCullagh and Ross [70], when using a similar type of grid structure to assist in constructing the Delaunay triangulation of a set of points, suggest a grid which allows an average of four points per cell. In the

model described here, the number of cells within each grid varies according to the total number of objects or triangles it references, following the approach described by Franklin [71] for indexing lines for detecting intersections. A temporary grid index, which references points, is also used for the purpose of efficient pyramid construction.

5.2.2.2 A Discussion Concerning Spatial Indexing.

Although the indexing scheme described in Section 5.2.2.1 lends itself to referencing all occurrences of objects, and triangles, within a specified area, it does not take into account the spatial extent of individual objects. Thus locationally specific retrievals involving areally extensive objects could lead to large amounts of unwanted data having to be read. This would be particularly true of high resolution data. This problem could be solved by ensuring that individual object components, that is, the point, line and polygon features from which an object is made up, be limited in size. This can be achieved by segmenting any overly extensive line features and polygon features (or to be more precise, the line features from which the polygon features are made up) into a series of less spatially extensive line features. For example, an object, representing a river, might originally have had a single reference to one spatially extensive line feature. After segmentation has taken place, the same object will now have a list of references to a series of less extensive line features, these having replaced the original line feature. In order to take full advantage of the data segmentation it would now also be necessary to spatially index line features, thus ensuring that only those relevant to a particular query are retrieved. A further enhancement of this structure would be to replace the regular sized grid with a data-adaptive indexing method such the bounding quadtree [72]. This would ensure that no single cell references more than a preset maximum amount of data (see Section 3.3.2).

The issue of spatially segmenting line data within a multiresolution model is quite a topical one. The multi-scale line tree [52, 58, 59], an extension of the earlier line generalisation tree [57, 58, 59], provides efficient spatial access to line data at various scales. It achieves this by classifying the internal points of digitised lines into hierarchies of scale-specific levels, which are themselves spatially segmented in a data-adaptive manner, using quadtree cells. A recent data structure, the Reactive-tree [62, 73], also provides efficient storage and retrieval of geometric objects at multiple levels of detail. By combining the R-tree [39], which provides efficient access to data objects by storing bounding rectangle information with each object, and the BLG-tree [61, 62], the Reactive-tree allows both objects and the points making up the objects to be retrieved on the basis of position and scale. A more recently published paper by Becker et al [74] introduces the Priority Rectangle File (PR-file). Here, the points defining line and polygonal objects are assigned levels of scale significance using a line generalisation algorithm. These points are then stored in a data structure which combines certain aspects of the line generalisation tree and the R-file [75]. Here the possible retrieval of

unwanted data is minimised by limiting the maximum number of points contained within a single bounding rectangle, thus limiting the spatial extent of individual object parts.

The type of spatial indexing method employed can be governed to a certain extent by the characteristics of the data that is being included in the model. If individual line features are likely to be spatially extensive it would be wise to consider a scheme which is able to spatially segment individual data items. However, it may be that relative to the total area being modelled, individual line features are not extensive. If this is the case it would appear sensible to remain with a more simple approach. The author, while admitting that the simplicity of his spatial access method is the sole motivating factor for its use, is unsure as to whether a more elaborate scheme would significantly improve the spatial search facilities provided by his model. It may be noted that the recent work by van Oosterom [62, 73] and Becker et al [74] provides examples of multi-scale storage schemes which in the former case do not segment individual line features while in the latter case their component vertices are grouped into rectangular subdivisions. The relative efficiency of the two schemes is not known.

5.2.3 The Selection of Critical Points.

A method of deciding at which level, and then subsequently at all lower levels, a particular point first appears in the MTSM has to be established. It will be governed by either the point's relevance to a particular object or its significance in describing the surface. The Delaunay pyramid selects points by means of an error-directed point insertion algorithm (see Sections 4.3.2 and 4.3.3). Here, a point is included at a particular level if the vertical distance of that point from the approximated surface is greater than a given error tolerance for that level. In the MTSM, any point which does not form part of a topographic object will be dealt with in this way.

Those remaining points, all of which form part of an object, present a more difficult problem. The level at which a particular point of a line feature is inserted can be generated by using a suitable line generalisation algorithm to classify the internal points of the line feature into a specified number of levels of scale-related significance. The method used here is that of Douglas and Peucker [55], which has proved successful in retaining the shape information of a line feature as the number of points describing it is reduced [49]. Another of its properties, essential in allowing a line feature to be stored hierarchically, is that points selected for small scales are a subset of those used in a larger scale representation. This algorithm is also suitable to some extent for simple polygonal shapes. Generalisation of the points of more complex polygonal features, such as buildings, into levels of scale significance cannot easily be achieved automatically. The level at which these points are inserted would, under the present version of our scheme, have to be determined manually.

It should be noted that each point forming part of a feature might also have a height value associated with it which is used to evaluate the point's significance in describing the surface. It is therefore possible that such points may be inserted at a higher level in the pyramid than the level originally indicated by the object generalisation procedure. Any object point which does not have a height value associated with it is assigned a NULL height value.

5.3 An Algorithm for Building the Hierarchical Model.

A pseudo-code algorithm for building the hierarchical model is given in Figure 5.4 (Procedure CREATE_MULTI_SCALE_MODEL). The algorithm makes use of a number procedures, most of which are self explanatory. However, the details of Procedure CREATE_CDP_LEVEL warrant special attention and are shown in Figure 5.5.

The model is constructed from a set of points S, containing all points defining the surface and those forming part of an object, and a list of objects O. Each object is defined by a list of references to its constituent point, line and polygonal features. The algorithm begins by initialising each of the object and triangle grids. Next, a line generalisation tree is constructed for each line feature, individual points having been allocated a level of significance using the Douglas-Peucker algorithm. The algorithm then proceeds to construct a CDP from the set S and the list O. Polygonal features reference line features, which in turn reference points, via a line generalisation tree. Each point which forms part of an object must be included in S and also have a level flag associated with it. This level flag ensures that it is inserted at the correct level of the pyramid, although it is possible that the point is inserted at some higher level according to its importance in approximating the surface. Note that the procedure CREATE_INITIAL_TRIANGULATION only considers those points H of S which have a height value associated with them. The procedure performs a Delaunay triangulation of those points which make up the convex hull of H. This has the effect of omitting from the multi-scale model any point which does not lie within the convex hull of H. The reason for adopting this approach is to ensure accuracy when interpolating height values for those object-defining points which do not initially have a height value.

Each object is inserted into a triangulation by sequentially inserting each of its line and polygonal feature components (point components must already have been included). Line and polygonal features are inserted as a series of straight line segments. Algorithms for inserting points and straight line segments into a Delaunay triangulation are given in the literature (for example, [22, 21]). Brief descriptions of two such methods have been given in Section 2.4.3.3 and Section 2.4.4, respectively.

Global Variables

L - list of line definitions
O - list of object definitions
S - list of point definitions

Procedure CREATE_MULTI_SCALE_MODEL(n, Error_V, Error_L)

/* n - number of levels */
/* Error_V - array of vertical error tolerances */
/* Error_L - array of lateral error tolerances */

For each level i
    INITIALISE_TRIANGLE_GRID().
    INITIALISE_OBJECT_GRID().
Endfor.
For each line l in L
    CREATE_LINE_GENERALISATION_TREE(l, n, Error_L, lgt(l)).
Endfor.
For each level i
    ADD_TO_OBJECT_GRID(O, i).
Endfor.
CREATE_INITIAL_TRIANGULATION(T_1, S).
For each level i
    CREATE_CDP_LEVEL(i, T_i, lgt, Error_V(i)).
    ADD_TO_TRIANGLE_GRID(T_i, i).
    If i ≠ n, then
        COPY_TRIANGULATION(T_i, T_{i+1}).
    Endif.
Endfor.

Endprocedure.

Figure 5.4 - The MTSM algorithm. This algorithm constructs the hierarchical model from a set of points S and list of objects O.

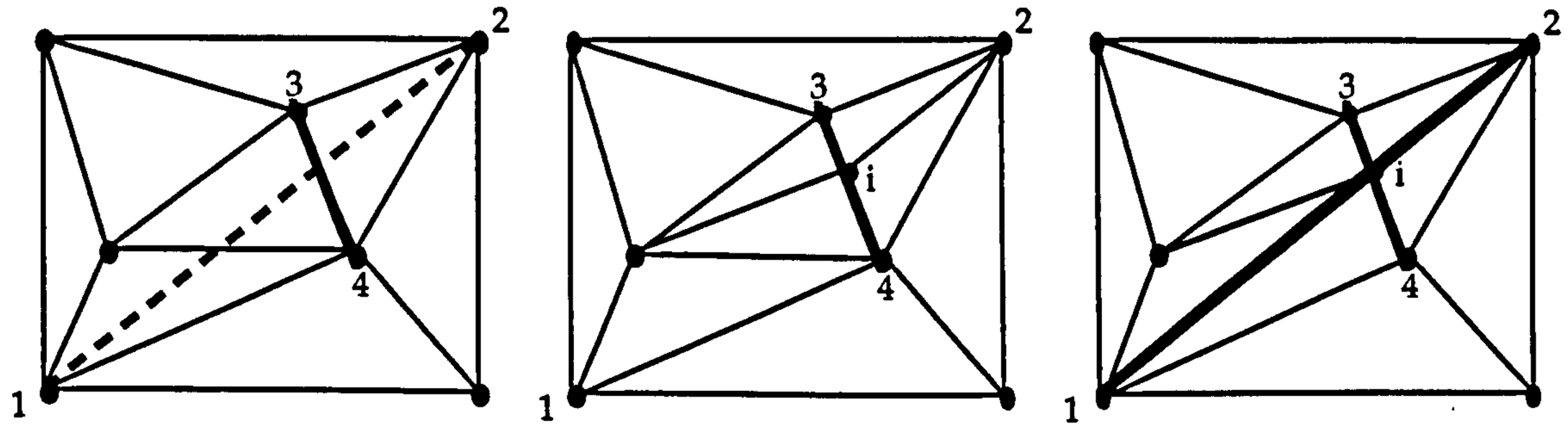
Note that following the insertion of constraints at a particular level it is necessary to check if the triangulation at that level still lies within the required vertical error tolerance threshold. This check is included due to the fact that the insertion of constraining objects which contain, at the time, unused points can sometimes lead to an increase in vertical error. Additional surface describing points are added, if necessary, until the triangulation lies within the required threshold.

```
Procedure CREATE_CDP_LEVEL(i, T_i, lgt, Error_Tol)

/* i -   level number */
/* T_i -  triangulation at level i */
/* lgt - list of line generalisation trees */
/* Error_Tol - vertical error tolerance for level i */

Finished = FALSE.
Do while (not Finished)
    FIND_NEXT_POINT_TO_INSERT(p, error_p).
    If (point found) and (error_p > Error_Tol), then
        INSERT_POINT(p, T_i).
    Else
        Finished = TRUE.
    Endif.
Endif.
For each line l in lgt
    RECONSTRUCT_LINE(lgt(l), i, temp_line).
    For each currently unused point p at level i of lgt(l)
        If z value of p = NULL_VALUE, then
            INTERPOLATE_HEIGHT(p, T_i).
        Endif.
        INSERT_POINT(p, T_i).
    Endfor.
    For each edge (p_1, p_2) in temp_line
        INSERT_EDGE(p_1, p_2, T_i).
    Endfor.
Endfor.
Finished = FALSE.
Do while (not Finished)
    FIND_NEXT_POINT_TO_INSERT(p, error_p).
    If (point found) and (error_p > Error_Tol), then
        INSERT_POINT(p, T_i).
    Else
        Finished = TRUE.
    Endif.
Endif.

Endprocedure.
```

Figure 5.5 - A procedure to create a level in a CDP.

The CDP algorithm as presented by De Floriani and Puppo [22] is restricted in that it only caters for non-intersecting straight line segments. This creates a problem when introducing topographic features into the pyramid because their constituent straight line segments can sometimes intersect each other. For example, this may occur when a road crosses over a county border. The MTSM algorithm makes provision for such occurrences by firstly introducing an additional point into the pyramid at the point of intersection of the two line segments (this point is given an interpolated elevation value), and then substituting the two original line segments with four replacement

segments. This process is illustrated in Figure 5.6.



Edge constraint (p1, p2) cannot be inserted since it intersects constraint (p3, p4).

The solution is to firstly delete constraint (p3, p4) and insert new point i at intersection of edge (p1, p2) and (p3, p4).

Finally, insert four edge constraints, (p1, pi), (pi, p2), (p3, pi) and (pi, p4).

Figure 5.6- Catering for intersecting constraining edges.

5.4 Justification for Using the Douglas-Peucker Algorithm.

In a number of recent papers [76, 77, 78, 79] the Douglas-Peucker algorithm has been subjected to criticism with regards its role as a line generalisation algorithm. The main argument put forward is that the algorithm was originally intended for the purpose of point reduction, not line simplification, and hence cannot be expected to perform well in the role of line generalisation algorithm. Its current use in many GIS is therefore purported to be a misuse. The consensus of opinion, that is in these papers, is that 'the high performance of Douglas-Peucker algorithm in mathematical evaluations (as described by McMaster) may be interpreted as being indicative of its relative merits as a weeding algorithm, but not necessarily as evidence of its superiority as a generalisation algorithm' [76]. Indeed, far from the Douglas-Peucker algorithm being superior, several of the authors proceed to describe their own algorithms which they regard as better suited to the task of simplification.

There are two main criticisms of the Douglas-Peucker algorithm. Firstly, it may only be used successfully when scale changes are small or modest [80, 81, 45]. The second criticism is that it may exhibit closing spikes or crossings that give a topologically distorted view of line morphology [82]. The author of this thesis is aware of these limitations but has persevered with the algorithm for two main reasons. Firstly, it is still, despite the criticism, the most widely used, and generally accepted as best, currently available line simplification algorithm (although this may change in future). Secondly, the Douglas-Peucker algorithm has the property of retaining original points at all stages of simplification and is therefore well suited to the multi-scale aspect of this thesis. In addition, a number of post-simplification routines are available which attempt to resolve some of the inadequacies of the algorithm. Muller [83] presents a

method for the removal of spikes; this thesis (Appendix 1) suggests a means by which spatial integrity can be restored when crossings occur; and several smoothing algorithms [52, 84] have been developed which can extend the range of scale change over which the Douglas-Peucker algorithm may be successfully used.

5.5 Limitations of the MTSM.

It should be noted that with regard to the multiple scale representation of topographic data, the MTSM is restricted to the generalisation, and subsequent multi-scale representation, of line data. This design limitation is deliberate and has been governed by the author's decision to only consider the generalisation of line data in the thesis. However, for the model to be described as truly multi-scale it may be argued that as more complex automated generalisation functions become available then the MTSM must be able to accommodate them. For example, the MTSM at present assumes that an object is present at every level of generalisation, and that it references the same polygon, line and point features in each case. However, in reality, this is not always the case. For example, certain objects at source-scale might not be deemed important enough to be included at all derived scales. Also, an object made up from a number of polygons at source-scale might be represented by a single polygon at a smaller scale. Similarly, a polygon will not always appear at every level, and will not always reference the same line parts at different levels. MTSM also assumes, incorrectly, that all line features appear at every level (albeit in generalised form).

In order to facilitate these types of generalisation in the future an alternative data model is suggested. It differs from the MTSM in that it introduces the concept of object data and polygon data being separated into levels, which may be a subset of the total number of levels in the database. Each level will only reference those objects or polygons which are present at that level. Individual object and polygon descriptions will therefore be allowed to change between levels. The addition and deletion of line features between levels can be accommodated using the present MTSM design.

It should also be pointed that the MTSM (or any multi-scale approach) is inappropriate when applying generalisation operations such as exaggeration, displacement and symbolisation to carry out large scale change. This is because the derived, smaller scale data is no longer, in a geometric sense, a subset of the large scale data. At present, a multiple representation approach would appear to be the only available technique capable of supporting these operations. This does not however rule out the use of the MTSM for separately representing each of the multiple representations across a sub-range of scales (see Section 10.3.1).

5.6 Summary and Conclusions.

This chapter has presented a new data model, or data storage scheme, suited to the efficient multi-scale storage of terrain and topographic data. It has also described an algorithm which when applied to suitable source-scale data will create the data model. The data storage scheme has been based on two previously described multi-scale data structures, namely, the constrained Delaunay pyramid and the line generalisation tree.

The data model is used as a basis for two multi-scale database implementations, descriptions and evaluations of which are given in Chapter 6. At this stage it is possible to make predictions, based on information already known about the data model on which the databases are based, as to what these evaluations will reveal. For example, the worst-case time complexity of the triangulation and constraining methods used in the data model creation algorithm is $O(n^2)$. This suggests that the time taken to create the multi-scale database will increase sharply as the number of points belonging to the model increases. This, however, is not considered to be a serious issue since the intended use of the multi-scale database is as a relatively permanent storage scheme, with database creation being a one-off event. Efficient update of the database is facilitated by the dynamic nature of the data model, thus avoiding the need to re-create the database when updates are required. It seems prudent that a thorough evaluation of the multi-scale databases should involve comparisons with the two alternative representation methods, namely, multiple representation and generalisation at run-time. It is expected that the multi-scale database will require less storage than multiple representation, but at the expense of a slower response time to scale-specific queries. Conversely, it is expected that the multi-scale database will provide a quicker response to database queries than a generalisation at run-time approach, but will incur the data storage overheads inherent in a multi-scale data structure. The extent to which data storage and query processing efficiency differs between the various representation methods will be reported in the next chapter.

Chapter 6

System Implementations

6.1 Introduction.

Two prototype database systems, based on the data model and accompanying algorithm presented in the previous chapter, have been implemented. The first, a relational implementation, named the Multiresolution Topographic Surface Database 1.0 (MTSD 1.0), is described in Section 6.2. The database system has been tested in experiments involving test data obtained from the British Geological Survey (BGS), the results and conclusions of which are included. Section 6.3 concerns itself with an ISAM implementation of the database system, named MTSD 2.0. This database system has been tested with the same test data as that used for MTSD 1.0. Database testing for each of the implementations includes storage-cost and query response-time evaluation. In each case the evaluation is by means of a comparison between the multi-scale database method and two alternative representation techniques, namely, multiple representation and generalisation at run-time. In conclusion, Section 6.4 provides a chapter summary and a comparison of the two implementations.

6.2 A Prototype Relational Database Implementation - MTSD 1.0.

The MTSM and associated algorithm have been implemented in C on a DEC Vax 8800 machine. Objects, polygons, lines (and corresponding line generalisation trees), points, the spatial index grids and the constrained Delaunay pyramid (CDP) are stored as tables within an ORACLE relational database management system (Figure 6.1). The database is built using a C program which makes use of the routines contained in each of five core C libraries. The reason for using the Vax machine for the prototype system was primarily that of convenience. At the start of the project the University's DEC Vax 8800 was the most suitable machine available, offering considerable processing power, a number of high resolution workstations and a wide variety of useful library packages (including GKS and UNIRAS). Having decided to use the Vax, the choice of the ORACLE database as the main data receptacle appeared sensible as it offered a wide range of immediately available data storage and access facilities. The incentive governing the choice of C, as opposed to other available languages (such as Pascal and ADA), was based on the fact that C offered greater scope for future transportability to other machines. This was a particularly important factor in that even at an early stage in the project it was envisaged that all work would eventually be expected to run on a soon to become available UNIX-based SUN workstation.

With this in mind, it was also necessary that the system be programmed in a modular fashion, thus ensuring that any future modifications to code, required as a result of transportation to another machine, could be as localised and restricted as possible.

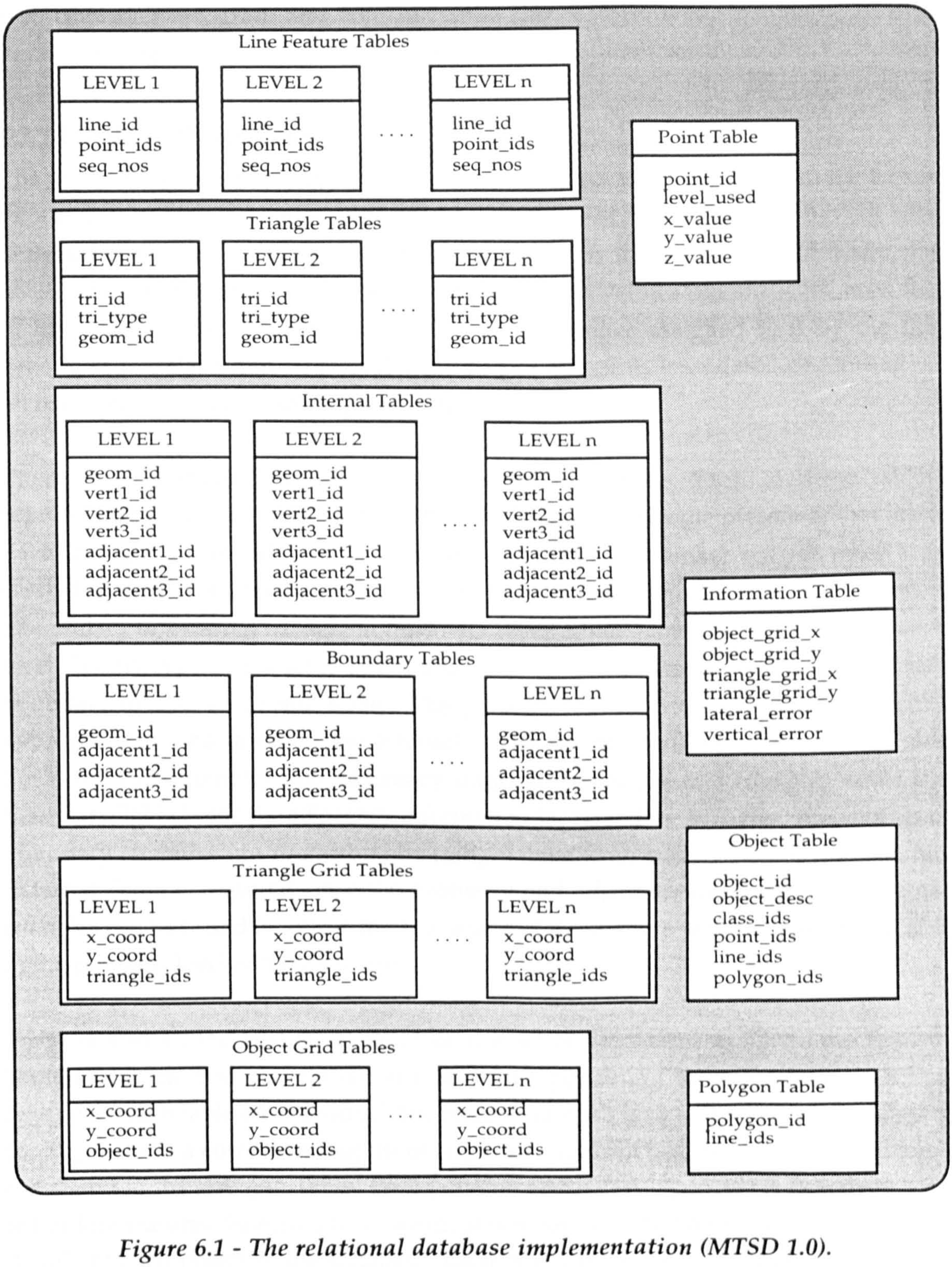


Figure 6.1 - The relational database implementation (MTSD 1.0).

The result of this was the creation of five core C libraries, namely the Data_Retrieval library, the Data_Transfer library, the Geometry library, the Triangulation library and Output library. The benefit of this modularisation becomes immediately apparent in Section 6.3, and again in Chapter 7 where advantage is taken of the re-usability of the routines contained within the core libraries. Also, much use is made of the C 'header file' which allows for global constants and global variables to be held in code separate

from the main program and core libraries. This feature, while not being of vital importance, again assists in easier transition between system versions.

6.2.1 Database Description.

The points from which the topographic data is made up are combined with the terrain data points to form a single set of points S. Each point belonging to S, having been assigned a unique identifier (point_id), is stored in the Point Table. Initially, the level_used flag for all surface points is assigned a null value while the level_used flag of line feature points is set to the level of significance assigned to it by the line generalisation algorithm. The level_used flag of all points is updated depending on which level in the database the point is first used.

There is a Triangle Table at each level of the database. Each of these tables, representing a level in the CDP, stores the details of each triangle present at that level. Each triangle in the database is given an identification number (tri_id) when it is created. It should be noted that if a triangle exists at more than one level (in the form of a boundary or external triangle in the lower level) it will have the same tri_id at each level. The tri_type flag is set to 0, 1 or 2 depending on whether the triangle is internal, boundary or external, respectively. The geom_id field is used as a pointer to the appropriate record in either the Internal or the Boundary Table. The Internal Table holds the full geometry and adjacency information for internal triangles while the Boundary Table holds the adjacency information for boundary triangles. The vertices of boundary triangles are found by obtaining details from a higher level triangle. No External Table is required since the geometry and adjacency information of external triangles can be found by retrieving the details of the triangle with the same tri_id as it in the previous level of the database.

There is also a Line Feature Table at each level of the database. Each Line Feature Table stores the line generalisation tree details pertaining to its level for each line feature. In each table an individual line feature (line_id) is described by a list of points (point_ids) and a corresponding list of sequence numbers (seq_nos). Polygon features are stored in the Polygon Table. Each polygon feature (polygon_id) is described by a list of line features (line_ids) from which it is made up. The Object Table contains the details of each object in the database. Each object is defined by an object identifier (object_id), an object description (object_desc), a list of object class identifiers (class_ids) and lists of the point (point_ids), line (line_ids) and polygon (polygon_ids) features which make up that object. The object class identifiers are used to assist in thematic retrievals of information.

The Object Grid Tables and Triangle Grid Tables correspond to the spatial grids detailed in 5.2.2.1. Each entry in a spatial table consists of the x,y coordinates

(x_coord, y_coord) of the bottom left hand corner of the cell it represents and a list of objects or triangles (object_ids or tri_ids) which intersect that cell. The number of cells in the x and y direction of each object grid (object_grid_x, object_grid_y) and each triangle grid (triangle_grid_x, triangle_grid_y) is stored in the Pyramid Table. This table also stores the lateral error (related to object resolution) and vertical error (related to terrain resolution) associated with each level.

6.2.2 The Core Libraries.

For reasons of portability and re-usability, the routines required to create MTSD 1.0 have been modularised. Routines have been grouped together in a particular library on the basis of the type of operation they perform and upon an evaluation of the likelihood of them having to be modified if they were to be transferred to a different machine. Each library will now be described in turn. Details of the main functions available in each library are given in Appendix 2.

6.2.2.1 Data_Retrieval Library.

This library contains low-level read/write procedures which interact directly with the database tables and make use of the Embedded SQL facilities available on the Vax. The procedures enable a variety of operations, such as retrieve the coordinates for a particular point_id from the Point Table (Figure 6.2), retrieve details for a particular tri_id from the Triangle Table or retrieve the list of object_ids contained within a particular cell of an Object Grid Table. These procedures are specifically designed to interface with the ORACLE database system used on the University's Vax machine.

```
void
get_values(point_id, level_used, x_value, y_value, z_value)
int point_id;
int *level_used;
double *x_value, *y_value, *z_value;
{
    EXEC SQL
    SELECT level_used, x_value, y_value, z_value
    INTO :*level_used, :*x_value, :*y_value, :*z_value
    FROM POINT_TABLE
    WHERE point_id = :point_id;
    return;
    errrpt : handle_error();
}
```

Figure 6.2 - The MTSD 1.0 procedure for retrieving the values associated with a particular point.

6.2.2.2 Data_Transfer Library.

The Data_Transfer library contains higher-level read/write procedures which, instead of interacting directly with the database, interact with the procedures contained within

the Data_Retrieval library. The procedures within the Data_Transfer library carry out operations such as returning the geometry of a particular tri_id or a list of object_ids contained within a particular polygonal area. The library has been designed in such a way as to limit its dependency on where and how the data is actually stored.

6.2.2.3 Geometry Library.

Many of the techniques employed in the construction of MTSD 1.0 require geometrical routines, such as point-in-triangle and line intersection tests. Such routines are used many times and at many different stages of the construction process, and also seem likely to be re-used in any future application programs. It therefore seemed logical to store all such routines in a single library where they could be readily accessed by all other routines.

6.2.2.4 Triangulation Library.

This library contains a number of the basic functions required to perform a constrained Delaunay triangulation of a set of points, such as calculating the convex hull, inserting a point into an existing triangulation and inputting a constraining edge into a triangulation.

6.2.2.5 Output Library.

The Output library consists of a number of high-level computer graphics output primitives. The primitives perform operations such as draw an object or display a triangulation on the screen. In this implementation the routines interact with the UNIRAS graphics routines available on the Vax.

6.2.3 An Implementation Issue.

When creating the multi-scale database it is important to consider the method used for storing intermediate data. Two possible extremes exist, the first involving main memory storage. With this approach, the initial step will be to read all source data into main memory. The database creation program is now applied to this internal data and an internal multi-scale model produced. This multi-scale model, and any updated source data, is then sent to the ORACLE database. The second approach involves the database creation program acting directly on a continuously updated database. The first approach has the disadvantage that the maximum amount of data held in the database is equivalent to the maximum amount of data that can be held in main memory. The major disadvantage of the second approach is that database creation will involve a large number of database accesses, thus increasing the time taken to create the database when compared to the main memory approach (for more detail see [85]). For the purposes of this thesis, the first approach has been adopted, the reason for which is two-fold. Firstly, the data used in the system testing is not extensive, and can

be adequately catered for in main memory. Secondly, the Vax and SUN machines used in developing MTSD 1.0 and MTSD 2.0 (Section 6.3) employ virtual memory management systems, thus considerably increasing the capacity of main memory applications.

6.2.4 Database Retrieval and Update.

In order to test the prototype system a number of basic database retrieval operations have been implemented. The procedures used for these operations are held in the Data_Transfer library. These operations allow for the retrieval of triangles and topographic objects at specific levels of detail and for particular areas of interest. Three parameters are needed to define a query, the first indicating what type of information is to be retrieved. The information type is described in terms of an integer code value, with -1 referring to outcrop objects and 0 to triangles. The second query parameter indicates the level of detail at which the information is required and is defined by an integer value corresponding to the required level. The third parameter defines the area of interest over which information is required. This area is defined in terms of a bounding rectangle which is described by two x,y coordinate pairs, the first corresponding to the bottom left hand corner of the bounding rectangle, the second to the top right hand corner. As an example query, consider the retrieval of triangles at detail level 2, the area of interest being a 500m square region with bottom left hand corner coordinates (42000, 21000). The corresponding MTSD 1.0 query would be (0, 2, 42000, 21000, 42500, 21500).

No database update operations are provided by the MTSD 1.0 system. It should be noted however that the primary data structures on which the database is based (the CDP, the line generalisation tree and the fixed grid) facilitate update. It therefore follows that the MTSD 1.0 will lend itself to the future inclusion of update operations. Note also that the point and edge insertion algorithms used in the creation of the CDP are themselves dynamic and could readily be adapted to allow for the insertion of new data.

6.2.5 Testing MTSD 1.0.

The system described has been tested using data acquired from BGS. Two types of data were involved, namely, geological outcrop map object data (referencing constituent polygon, line and point data) representing outcrop boundaries and fault lines, and terrain data in the form of irregular (x, y, z) point data. The test data covers a 2km x 2km square in the Grantham area of England. Plate 6.1 is a plot of the terrain points, of which there are 380. The outcrop data is shown in Plate 6.2 and consists of 20 objects, which in turn reference a total of 20 polygons and 143 lines. The lines are made up from a total of 896 2-D feature points.

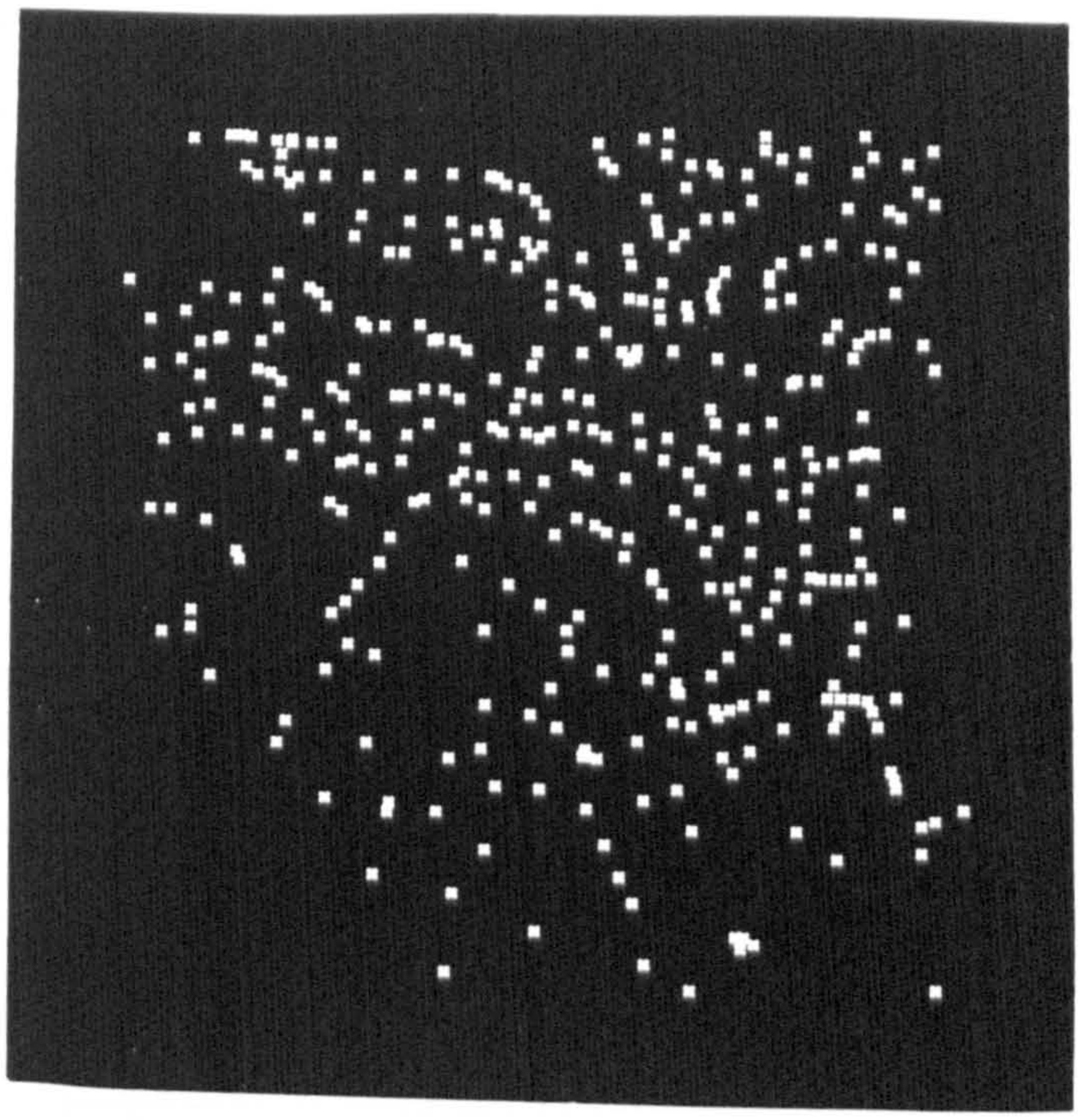


Plate 6.1 - Terrain points (380) forming part of test data set.

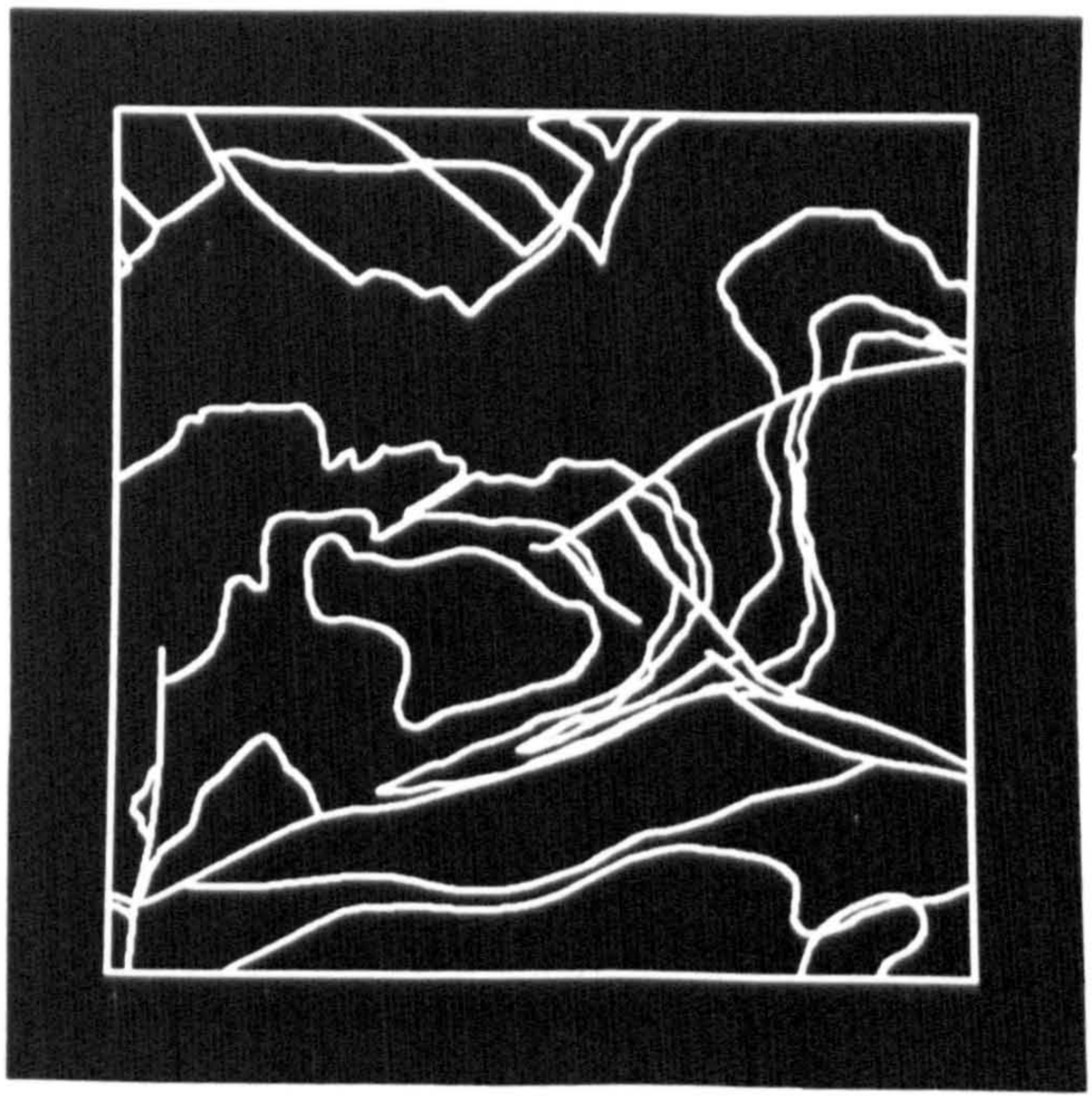


Plate 6.2 - The outcrop data consisting of 20 objects (20 polygons, 143 lines and 896 points).

A series of test databases have been created, details of which are given in Figure 6.3. In each case the lateral and vertical error tolerances have been chosen is such a way as to emphasise the difference in detail between the database levels. No attempt is made here at relating these error tolerances to what professional geographers regard as scale, although some thought is given to this topic in Section 10.3.1. The database creation time, similar for each of the databases, appears satisfactory (particularly in light of the argument presented in Section 5.6). Plates 6.3 to 6.8 show the three levels of the CDP forming part of Database 3. Note that only those object edges which lie completely within the convex hull of the terrain points are included as constraints (see Section 5.3).

Database Name	Number of levels	Vertical Error(m)	Lateral Error (m)	Number of terrain points	Number of topographic points	Number of edges	Creation time (s)
1	3	30.0	5.0	168	216	227	304.0
		12.5	2.5	48	90	317	
		1.0	1.0	148	198	515	
2	3	35.0	8.5	158	157	161	311.0
		10.5	2.5	77	129	317	
		1.0	1.0	129	198	515	
3	3	40.0	10.0	150	138	150	350.0
		7.5	3.0	105	140	289	
		0.5	0.5	114	448	737	

Figure 6.3 - Database creation performance table for MTSD 1.0.

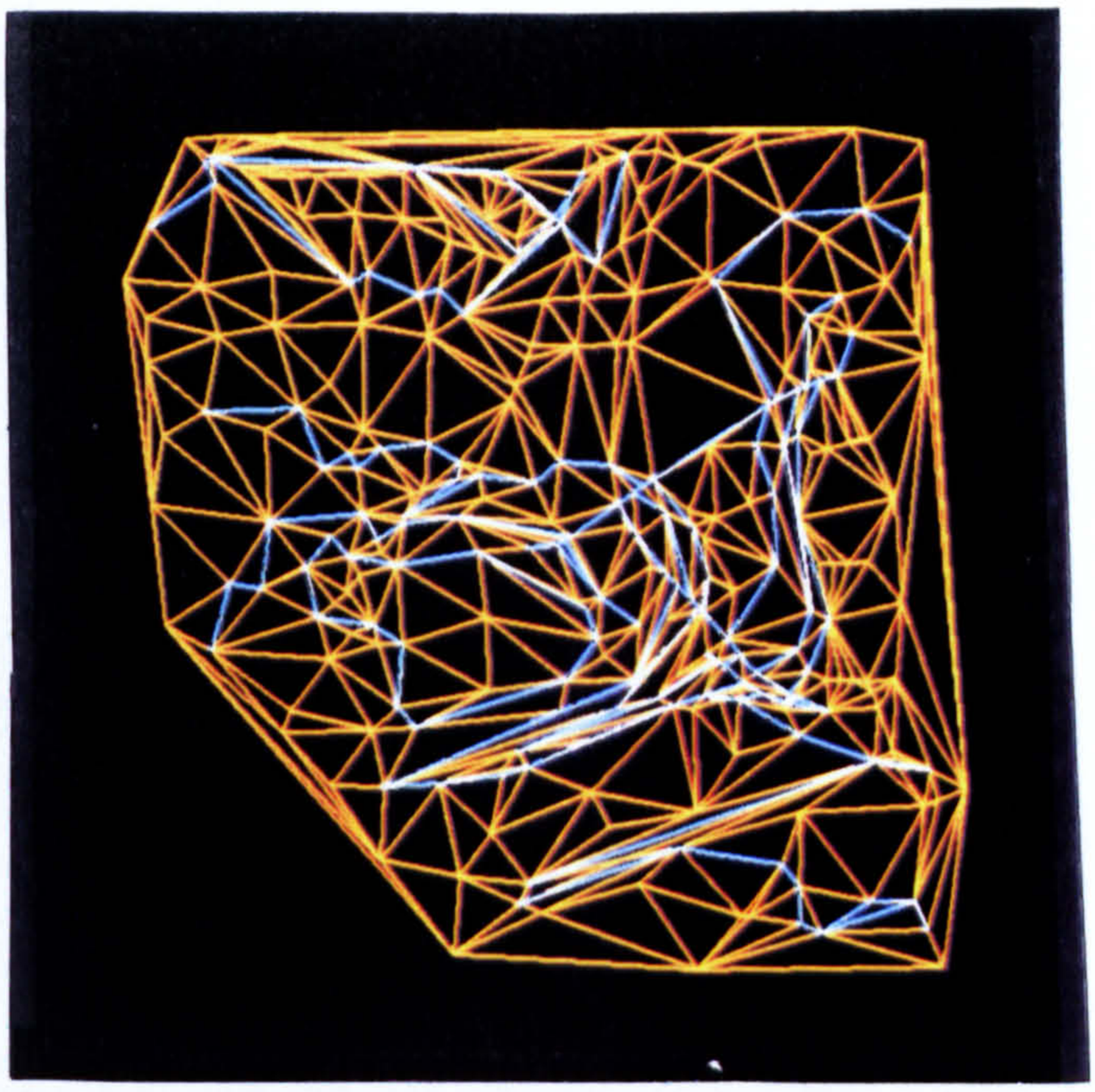


Plate 6.3 - Database 3, level 1 (vertical error = 40.0m, lateral error = 10.0m). *Plan view.*

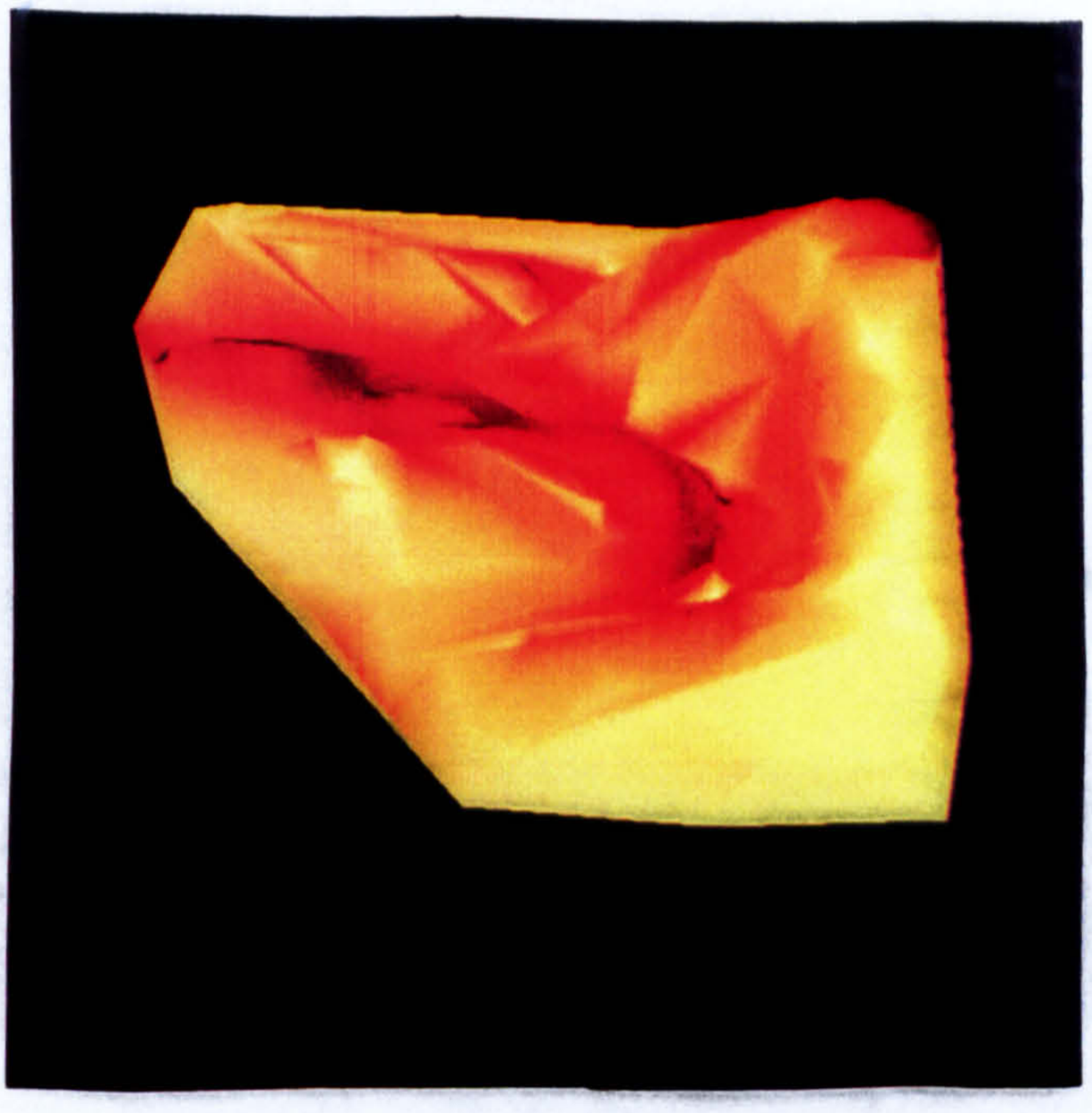


Plate 6.4 - Database 3, level 1 (vertical error = 40.0m, lateral error = 10.0m). *Shaded, perspective view.*

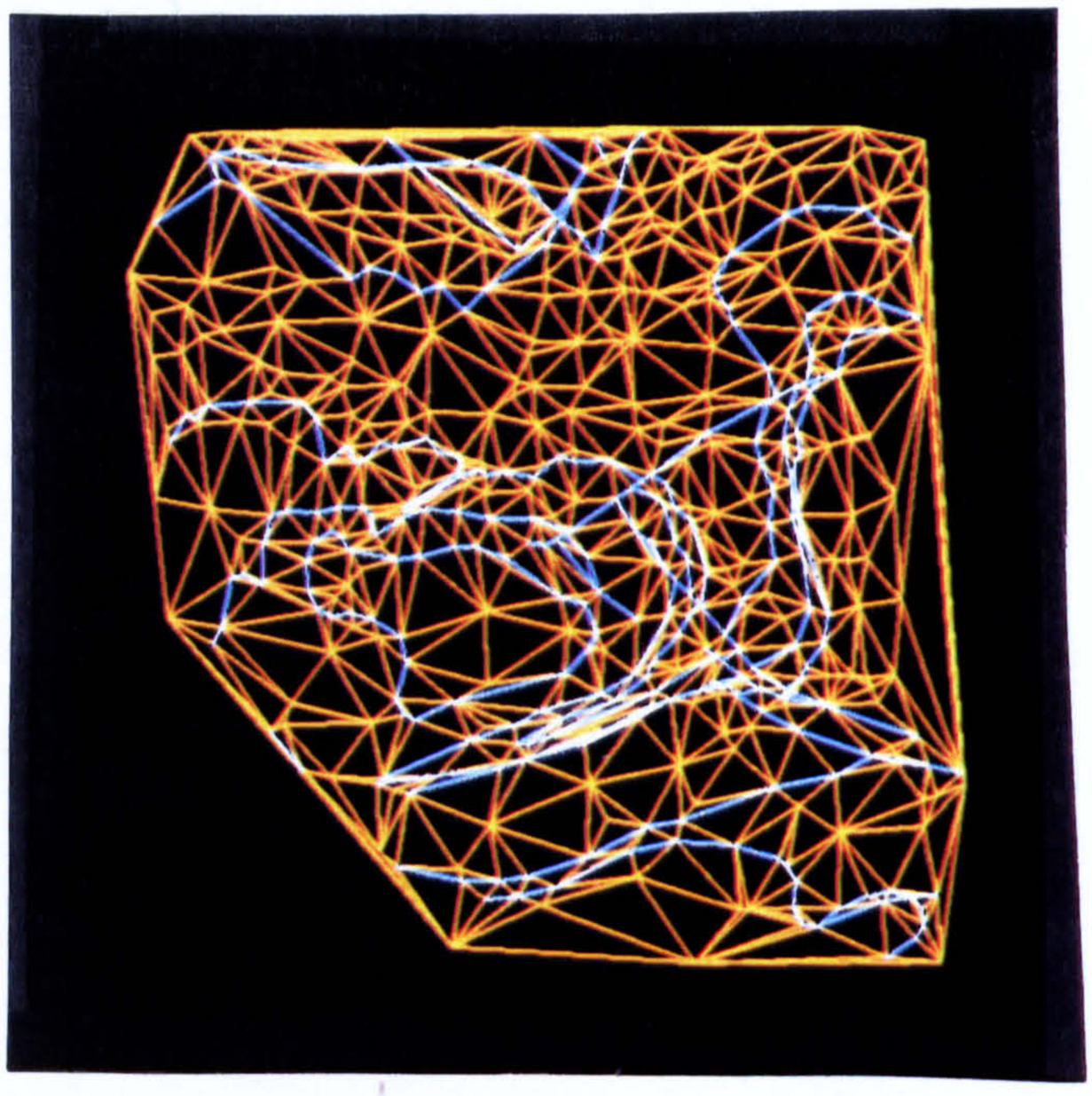


Plate 6.5- Database 3, level 2 (vertical error = 7.5m, lateral error = 3.0m). *Plan view.*

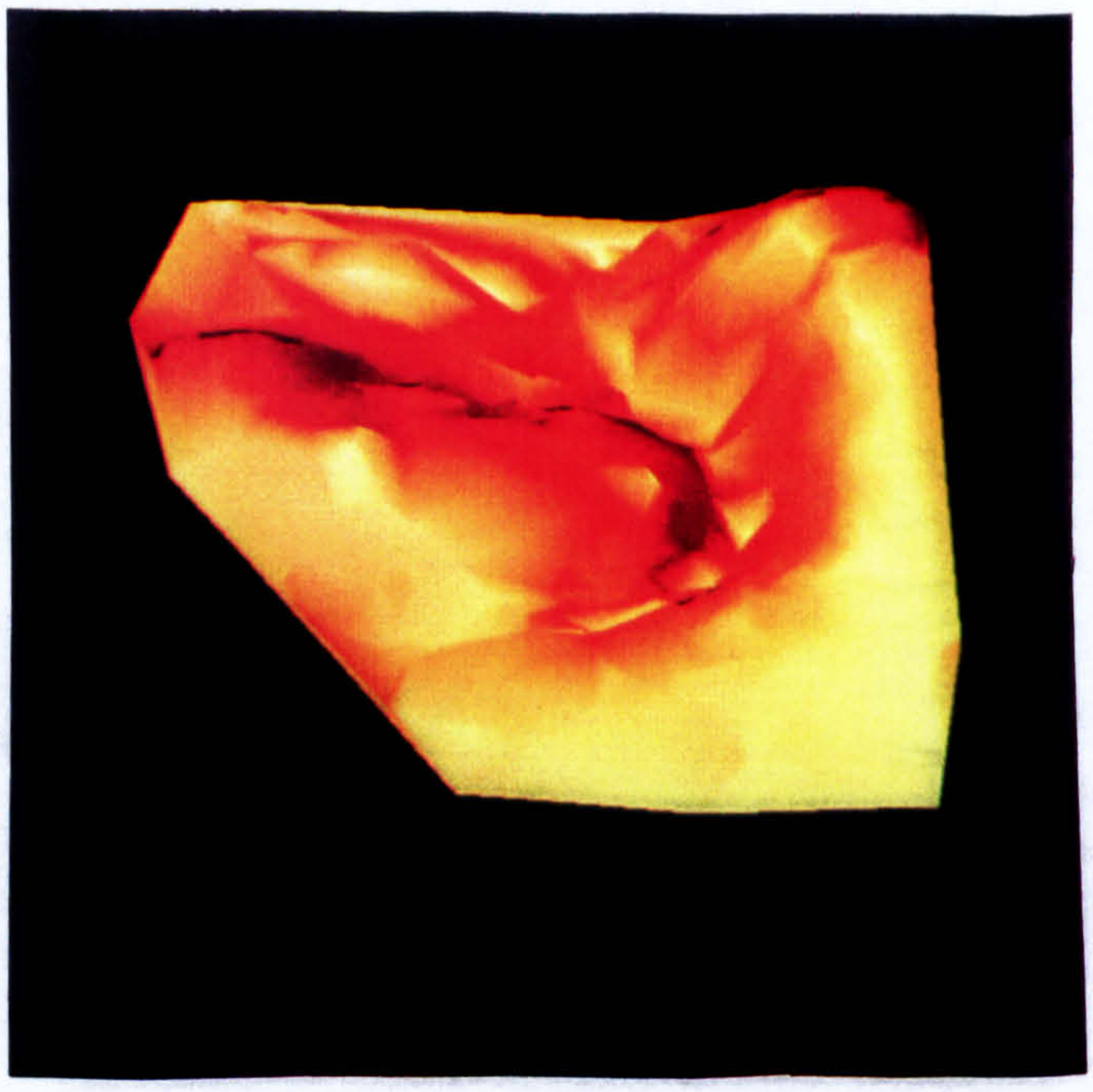


Plate 6.6 - Database 3, level 2 (vertical error = 7.5m, lateral error = 3.0m). *Shaded, perspective view.*

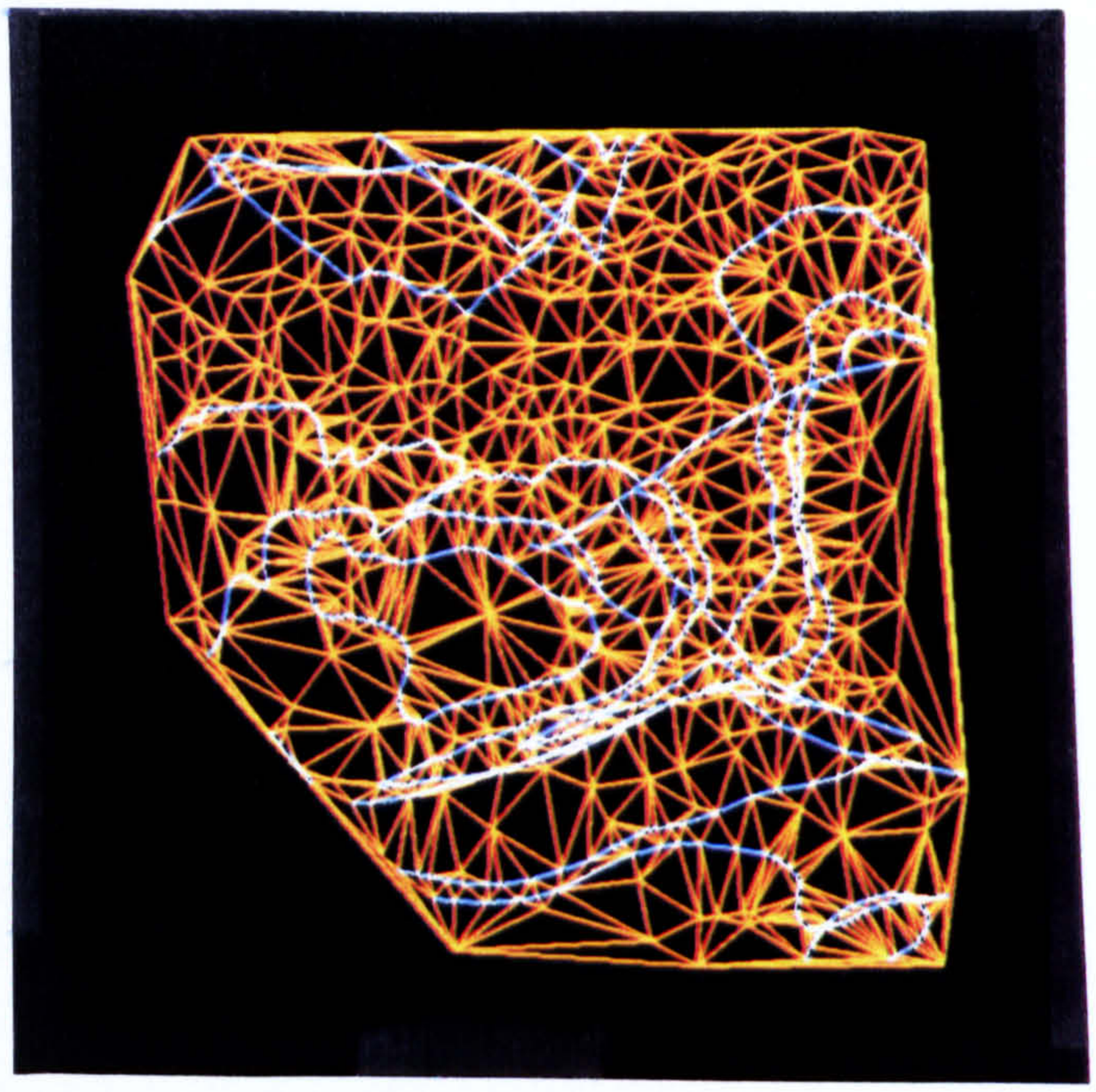


Plate 6.7 - Database 3, level 3 (vertical error = 0.5m, lateral error = 0.5m). *Plan view.*

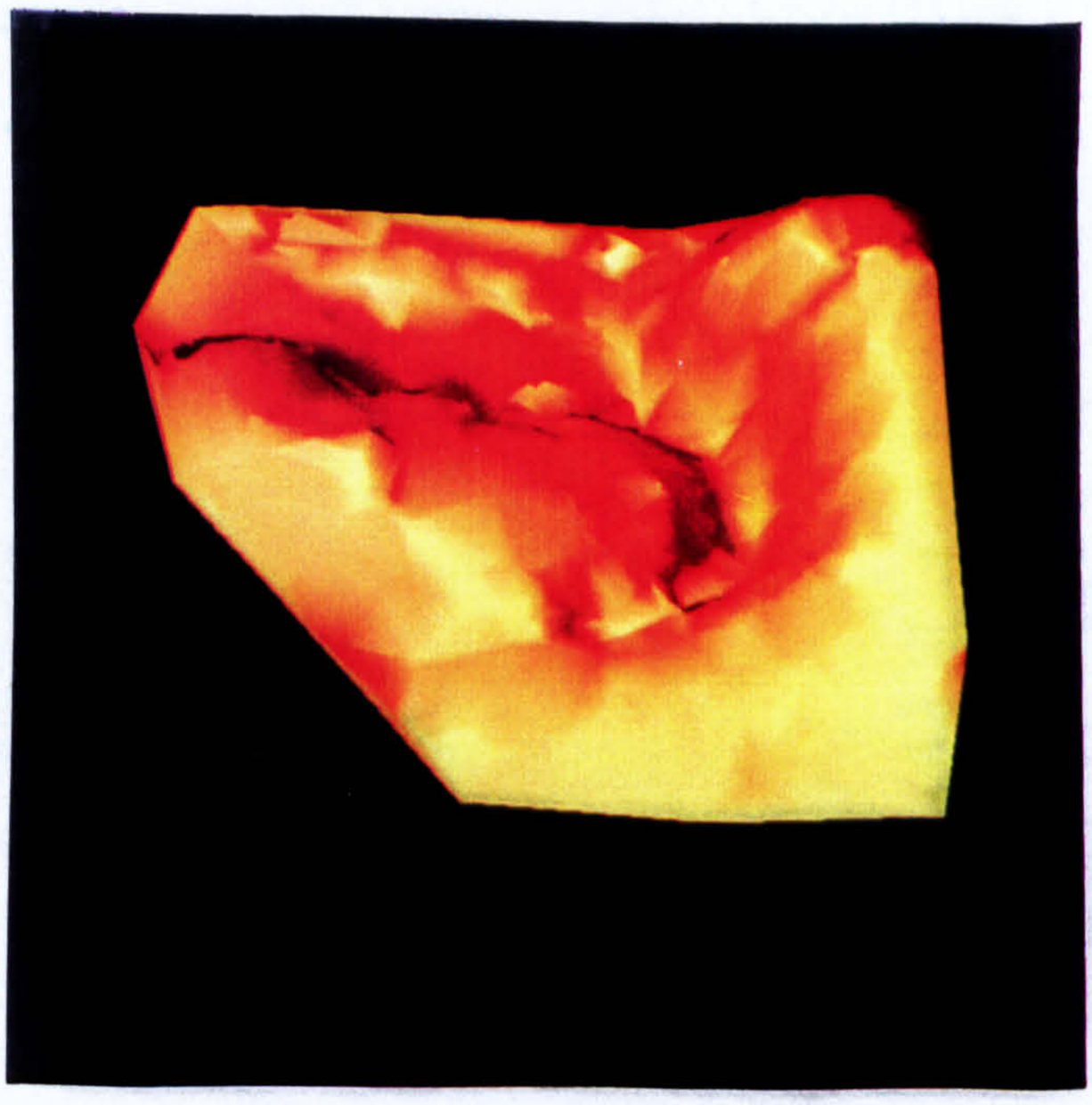


Plate 6.8 - Database 3, level 3 (vertical error = 0.5m, lateral error = 0.5m). *Shaded, perspective view.*

In order to further evaluate the multi-scale database approach a series of tests have been carried out to compare MTSD 1.0 with two alternative methods, namely, generalisation at run-time and multiple representation. These tests necessitated the creation of two test databases. The first, adopting a generalisation at run-time approach consists of tables in which the source-scale data set (that is, terrain and outcrop data) is stored. Subsequent scale-specific queries required application of the Douglas-Peucker algorithm and error-directed constrained triangulation algorithm, using the supplied lateral and vertical error tolerances. The multiple representation database approach, as with MTSD 1.0, involved application of these algorithms during the creation of the database. The multiple representation database differed from MTSD 1.0 in that at each level, full line and triangle descriptions were stored, as opposed to the line generalisation tree and CDP approach used in MTSD 1.0.

Method	Number of levels	Vertical Error (m)	Lateral Error (m)	Storage used (K-bytes)	Time taken to retrieve all triangles (s)	Time taken to retrieve all objects (s)
MTSD 1.0	3	30.0	5.0	428	14.0	11.0
		12.5	2.5		20.0	14.0
		1.0	1.0		29.0	17.0
Generalisation at run-time.	1	30.0	5.0	205	80.0	28.0
		12.5	2.5		101.0	35.0
		1.0	1.0		130.0	46.0
Multiple representation	3	30.0	5.0	601	12.0	7.0
		12.5	2.5		14.0	9.0
		1.0	1.0		15.0	10.5
MTSD 1.0	3	35.0	8.5	417	14.5	9.0
		10.5	2.5		23.0	12.5
		1.0	1.0		31.0	21.0
Generalisation at run-time.	1	35.0	8.5	205	91.5	31.0
		10.5	2.5		123.0	36.0
		1.0	1.0		150.0	44.0
Multiple representation	3	35.0	8.5	692	10.0	7.5
		10.5	2.5		12.5	11.0
		1.0	1.0		16.0	13.5
MTSD 1.0	3	40.0	10.0	457	12.0	7.0
		7.5	3.0		25.0	14.5
		0.5	0.5		38.5	17.5
Generalisation at run-time.	1	40.0	10.0	205	89.5	28.0
		7.5	3.0		146.5	34.5
		0.5	0.5		197.0	46.0
Multiple representation	3	40.0	10.0	748	9.0	11.5
		7.5	3.0		12.0	13.0
		0.5	0.5		16.5	17.0

Figure 6.4 - Results of comparison tests between MTSD 1.0, generalisation at run-time and multiple representation.

The comparison tests are made in terms of storage efficiency and query-response time. The results, shown in Figure 6.4, are to some extent compatible with the performance predictions made in Chapter 5. The response time to scale-specific queries is significantly slower for the generalisation at run-time approach than the other methods, which is as expected. The multiple representation method offers better response time than MTSD 1.0, but not perhaps as much as had been expected. When considering

storage efficiency, the generalisation at run-time method appears a clear winner, as predicted, while MTSD 1.0 is considerably more efficient than the multiple representation approach.

6.3 A Prototype ISAM Database Implementation - MTSD 2.0.

It was decided that a second implementation, to run on a UNIX-based SUN workstation, was necessary. There were a number of reasons for this decision, the most important being that the work was being carried out in conjunction with other projects, at both the University and BGS, which were using UNIX-based workstations. Other reasons included the better graphics capabilities offered by the University's SUN software in the form of PHIGS, an important factor in light of the long term move towards a 3-D system, and the performance improvements gained in a single-user RISC environment (as opposed to the multi-user environment encountered on the Vax).

```
void
get_values(point_id, level_used, x_value, y_value, z_value)
int point_id;
int *level_used;
double *x_value, *y_value, *z_value;
{
    struct Primary_Point_DB single_point;

    single_point.vert_id=point_id;
    if (btRead(fd_pp,&single_point) != GOOD)
            die();
    *level_used=single_point.level_used;
    *x_value=single_point.x_value;
    *y_value=single_point.y_value;
    *z_value=single_point.z_value;
    return;
}
```

Figure 6.5 - The MTSD 2.0 procedure for retrieving the values associated with a particular point.

The change from Vax to SUN computer necessitated a change from ORACLE to some other means of basic data storage, as no ORACLE software was available on the SUN. Two options were duly considered. The first entailed developing in-house file handling routines, equivalent to those used in the ORACLE system, using the standard C input/output functions as the basic building blocks. The second was to make use of a C ISAM library which was installed on the SUN. The former option had the advantage that all routines could be written with the particular application in mind. A disadvantage was that carrying out this task would be time consuming. The ISAM option had the advantage of having an extensive set of pre-written C file handling functions, many of which matched directly to an equivalent ORACLE function used in

MTSD 1.0. In view of this last fact, the ISAM option was preferred.

The ISAM implementation makes full advantage of the modular fashion in which the first implementation was written. The only libraries which needed extensive modification were the Data_Retrieval library (ISAM routines instead of embedded SQL) and the Output library (PHIGS replacing UNIRAS). For example, compare the ISAM coordinate retrieval procedure shown in Figure 6.5 with its ORACLE equivalent (Figure 6.2). The other libraries were transferred directly from the Vax and after only very minor alterations compiled and executed successfully.

6.3.1 Database Description.

The database design is almost identical to that of the relational system, with an ISAM file matching to each of the ORACLE tables. The information contained within each file is the same as that of its corresponding relational table. One slight difference is in the way the ISAM files handle the lists of point_ids, line_ids, polygon_ids, seq_nos, object_ids and tri_ids. The relational database takes advantage of ORACLE's ability to store variable length character strings, with limited data redundancy. This enables lists of any length (up to an ORACLE defined limit) to be stored in a single record field. The ISAM library does not provide this facility and therefore an alternative arrangement was needed. The solution was to employ a chaining mechanism. Here, the lists are stored in fixed length integer arrays, the length of which (for each file type) has to be defined before the database is initialised. Any list whose number of entries is less than the length of its appropriate fixed length is stored in a single array. Lists which contain a number of entries greater than or equal to the maximum allowed are stored in a chain of arrays, the first of which is the originally intended array. The second array is contained in a newly created record, the key of which is stored in the last element of the first array. The value (and type) of the key will depend on the type of file being processed. A simple approach is to assign negative values to those keys corresponding to overflow records. Other records are created and linked together until the whole list is stored.

6.3.2 Database Retrieval and Update.

The database retrieval operations included in the MTSD 2.0 system are the same as those included in MTSD 1.0. Queries, defined in terms of three parameters (data type, level of detail and area of interest) allow triangles and topographic objects to be retrieved at specific scales and for particular regions of interest. As was the case with MTSD 1.0, no database update operations are provided.

6.3.3 Testing MTSD 2.0.

The MTSD 2.0 system has been tested using the same test data as used in the testing of MTSD 1.0. Three databases have been created, details of which are given in Figure 6.6. The database creation times, which offer an improvement when compared to MTSD 1.0, are again satisfactory.

Database Name	Number of levels	Vertical Error(m)	Lateral Error (m)	Number of terrain points	Number of topographic points	Number of edges	Creation time (s)
1	3	30.0	5.0	168	216	227	168.0
		12.5	2.5	48	90	317	
		1.0	1.0	148	198	515	
2	3	35.0	8.5	158	157	161	189.0
		10.5	2.5	77	129	317	
		1.0	1.0	129	198	515	
3	3	40.0	10.0	150	138	150	203.0
		7.5	3.0	105	140	289	
		0.5	0.5	114	448	737	

Figure 6.6 - Database creation performance table for MTSD 2.0.

Figure 6.7 shows the results of comparison tests made between MTSD 2.0 and the two alternative representations, generalisation at run-time and multiple representation. The results are in line with those reported in 6.2.5. MTSD 2.0 performs better than generalisation at run-time with regard to response time to scale-specific queries, and requires significantly less storage space than multiple representation. On the other hand, MTSD 2.0 is slower to respond to queries than the multiple representation technique, and requires more storage space than generalisation at run-time.

Method	Number of levels	Vertical Error (m)	Lateral Error (m)	Storage used (K-bytes)	Time taken to retrieve all triangles (s)	Time taken to retrieve all objects (s)
MTSD 2.0	3	30.0	5.0	395	8.0	5.0
		12.5	2.5		12.0	6.0
		1.0	1.0		17.0	8.0
Generalisation at run-time.	1	30.0	5.0	183	50.0	12.0
		12.5	2.5		65.0	15.0
		1.0	1.0		81.0	19.0
Multiple representation	3	30.0	5.0	560	5.0	4.0
		12.5	2.5		5.5	4.0
		1.0	1.0		6.0	4.5
MTSD 2.0	3	35.0	8.5	386	7.5	4.5
		10.5	2.5		13.5	6.5
		1.0	1.0		16.0	8.5
Generalisation at run-time.	1	35.0	8.5	183	48.0	14.0
		10.5	2.5		63.0	16.0
		1.0	1.0		85.5	19.5
Multiple representation	3	35.0	8.5	554	4.5	4.0
		10.5	2.5		5.5	4.5
		1.0	1.0		6.0	5.0
MTSD 2.0	3	40.0	10.0	421	7.0	4.0
		7.5	3.0		13.0	7.0
		0.5	0.5		18.5	9.0
Generalisation at run-time.	1	40.0	10.0	183	40.0	14.5
		7.5	3.0		70.0	17.0
		0.5	0.5		98.0	21.5
Multiple representation	3	40.0	10.0	594	4.0	5.0
		7.5	3.0		5.0	5.5
		0.5	0.5		6.5	6.0

Figure 6.7- Results of comparison tests between MTSD 2.0, generalisation at run-time and multiple representation.

6.4 Summary and Conclusions.

This chapter has described two database system implementations, MTSD 1.0 and MTSD 2.0, which succeed in allowing terrain and topographic object data to be combined in a single database at multiple levels of detail. Points, lines and polygons are integrated with, and serve to constrain, a hierarchical triangulation (the CDP) which avoids data duplication. Multi-scale representation of line data is accommodated by adoption of the line generalisation tree method. Data is accessed via spatial grid indexes referencing topographic objects and the triangles that model the terrain surface.

The database implementations have been subjected to several tests involving a test data set obtained from BGS. In comparison tests with two alternative multiple scale representation techniques, both MTSD 1.0 and MTSD 2.0 perform in a manner to be expected of systems based on multi-scale data structures. The multi-scale databases outperform a generalisation at run-time approach with regards query response-time, at the expense of increased data storage. Conversely, the multi-scale databases require less storage than a multiple representation approach, but do not perform as effectively in response to database queries.

When comparing MTSD 1.0 and MTSD 2.0 to each other, it becomes evident that the ISAM implementation (MTSD 2.0) offers better performance (in terms of creation time, storage requirements and query response time) than its ORACLE counterpart. However, the author believes that with regards database creation time and query response time the reason for the difference in performance has little to do with the relative effectiveness of a relational approach as opposed to an ISAM approach. A more likely explanation is to consider the performance difference as a by-product of the different environments in which the two methods have been implemented (that is, a multi-user Vax-based database as opposed to a single-user SUN-based database). For example, a series of benchmark tests involving the construction of a number of main-memory based constrained Delaunay pyramids show the SUN to considerably outperform the Vax. This would indicate that any differences between ISAM database implementation and ORACLE database implementation is to some extent influenced by the difference in the machines on which they were implemented.

It could be argued that query response-times in both implementations are far from good. For example, a response time of 17 seconds when retrieving all triangles from level 3 of MTSD 2.0 Database 1 seems slow when taking into account how relatively small the test data set is. However, it should be noted that the emphasis of this research has been on the logical design rather than an optimal implementation of the multi-scale database. This chapter has shown that such a database is feasible and offers performance benefits when compared to its main alternatives.

Chapter 7

The Implicit TIN

7.1 Introduction.

This chapter gives details concerning a recently developed surface representation technique termed the Implicit TIN. Section 7.2 gives explanation as to why the Implicit TIN was developed, while details of the original implementation are given in Section 7.3. Several of the limitations of the original Implicit TIN are discussed in Section 7.4, which proceeds to describe a new, improved Implicit TIN scheme. This improved version is then used as the basis for an implicit multiresolution topographic surface database (I_MTSD), details of which are presented in Section 7.5. The performance of I_MTSD is discussed in Section 7.5, which includes comparison tests involving MTSD 2.0. Finally, conclusions and summary are given in Section 7.6

7.2 Motivation for the Implicit TIN.

Data storage schemes based on conventional TIN data structures incur a storage overhead due to the fact that they represent the topology of the triangulation explicitly. This is true of the multiresolution data storage scheme presented in Chapters 5 and 6. Storing topology explicitly results in the fact that although TINs typically use many fewer points than their main alternative representation, the regular rectangular grid, they do not usually occupy much less data storage. This is demonstrated by the following example, taken from Kidner and Jones [86]. Consider a vertex-based TIN, derived from a regular grid DTM, where 10% of the original grid points are needed to represent the surface satisfactorily. If it is assumed that each x, y and z coordinate of every TIN point can be represented in the same storage space as each TIN topologic pointer value, then it has been shown in Section 2.4.3.2 that an edge-based TIN, made up from N points, requires a storage space of approximately 10N. This storage cost, 10N, is equivalent to 100% of the original grid storage space, and therefore no storage saving has been achieved. A triangle-based TIN will require even greater storage equivalent to 150% of the original grid storage, that is, a net increase in storage use.

When the amount of data stored is not great this storage overhead does not prove a problem. However, the ever increasing demand of GIS users is resulting in the need for accurate terrain data at finer resolutions and at national levels, thus leading to very large quantities of data. Such vast volumes of data when stored in the conventional TIN format may exceed the storage limits of many systems.

7.3 The Original Implicit TIN.

The original Implicit TIN [86, 87] differs from a conventional TIN in that only vertices are stored. In other words, in an Implicit TIN the topological relationships defining the triangulation are not explicitly recorded. TIN topology is reconstructed by a procedure if and when it is required. Thus a suitable triangulation procedure is required, which can itself be thought of as part of the Implicit TIN.

The Implicit TIN provides a highly compact storage scheme for representing surfaces when compared to a conventional TIN. It is however apparent that this advantage is at the cost of having to reconstruct TIN topology when required. With this in mind it is noted that many GIS operations concerning terrain data, such as profiling, visibility analysis, earthwork calculations and communication network siting, only require subsets of the complete data coverage. Therefore with regards the Implicit TIN it follows that for any particular operation, only the topology of points relating to the operation need be reconstructed.

Kidner and Jones [86] propose a method for spatially specific reconstruction of TIN topology based on Delaunay triangulation. Elevation points are stored in a box-sort structure [70], which allows for fast access to required points. When an operation requires a surface reconstruction for a particular area of interest the grid cells covering this area can be directly accessed to determine all the vertices of the local TIN. A conventional Delaunay triangulation algorithm is then applied to these points resulting in an explicitly defined TIN.

An important characteristic of the Delaunay triangulation is its uniqueness for a given set of points. When triangulating a subset of the original points however, the triangulation at the boundary of the subset cannot be guaranteed to be the same as that obtained in a global triangulation. This is because neighbouring vertices outside the region of interest, which may affect the triangulation within the area of interest, are not taken into account. In an attempt to overcome this problem a provision is made to allow for a 'reasonable' margin of points around the area of interest by expanding the search window. This margin should be dependent upon the sparseness of the data within the terrain model.

7.4 An Improved and Constrained Implicit TIN.

The previously described implementation of the Implicit TIN is deficient in two ways. Firstly, in certain instances around the border of an area of interest, the derived triangulation will not conform to the uniqueness property of a Delaunay triangulation. This will sometimes occur despite the use of an expanded search window. Secondly, it makes no provision for the inclusion of linear constraints within a triangulation. In response to these inadequacies a modified version of the original was developed collaboratively by Jones, Kidner and Ware [88]. The modifications were exploited in a multi-scale database which will be discussed in the Section 7.5. The next section will serve as an introduction and concern itself with a single-scale Implicit TIN.

7.4.1 Database Design.

The single-scale Implicit TIN consists of an ISAM database, used for storing permanent data (Figure 7.1), plus the constrained Delaunay triangulation routines required for

generating temporary data, that is, TIN topology. The ISAM database is made up from 6 files. The Point File stores the x, y and z coordinate for each terrain point, each accessed via a unique identifier. The Object File stores details of surface constraining features. These features are defined in terms of their constituent polygons, lines and points which are stored in the Polygon, Line and Point Files respectively.

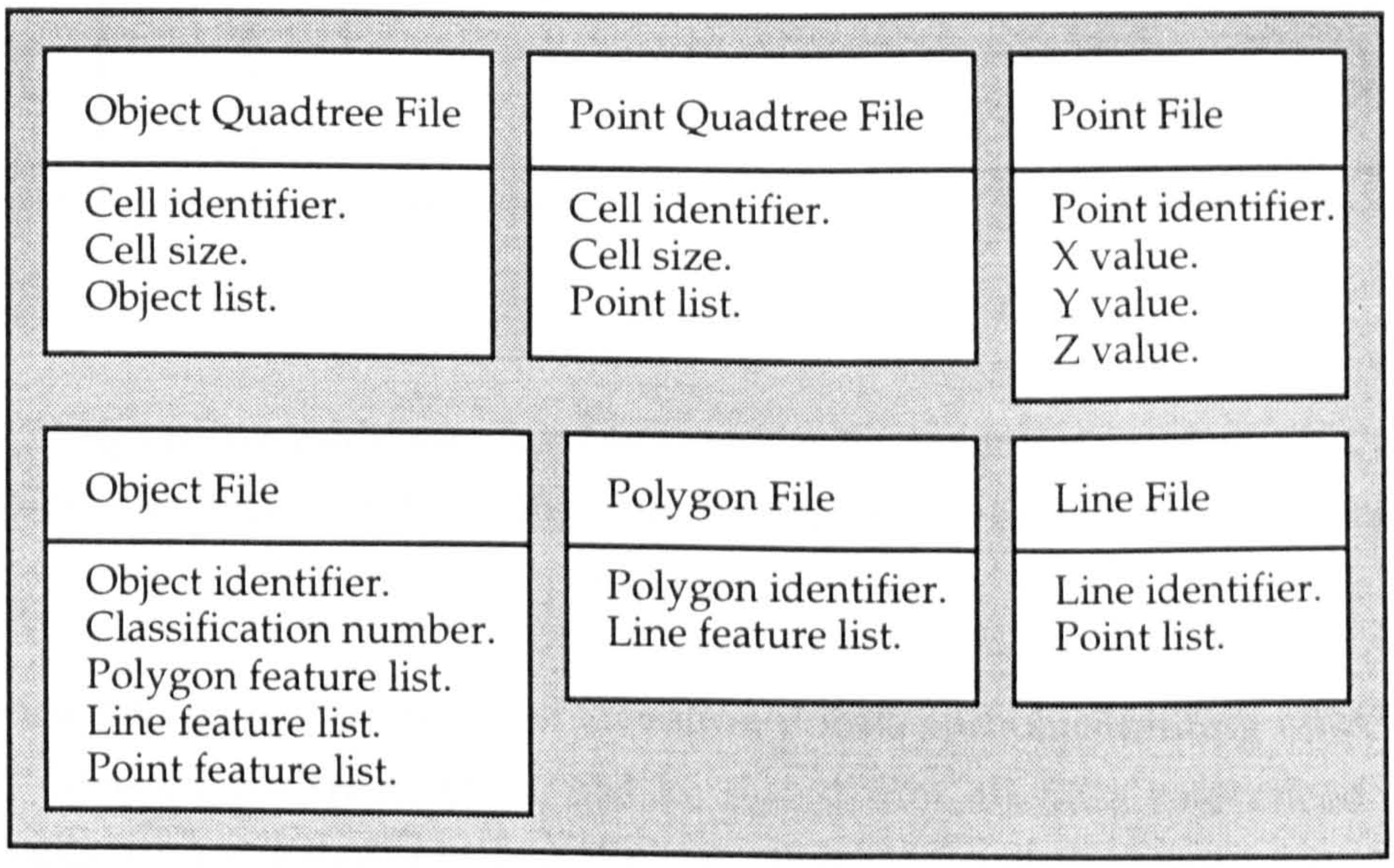
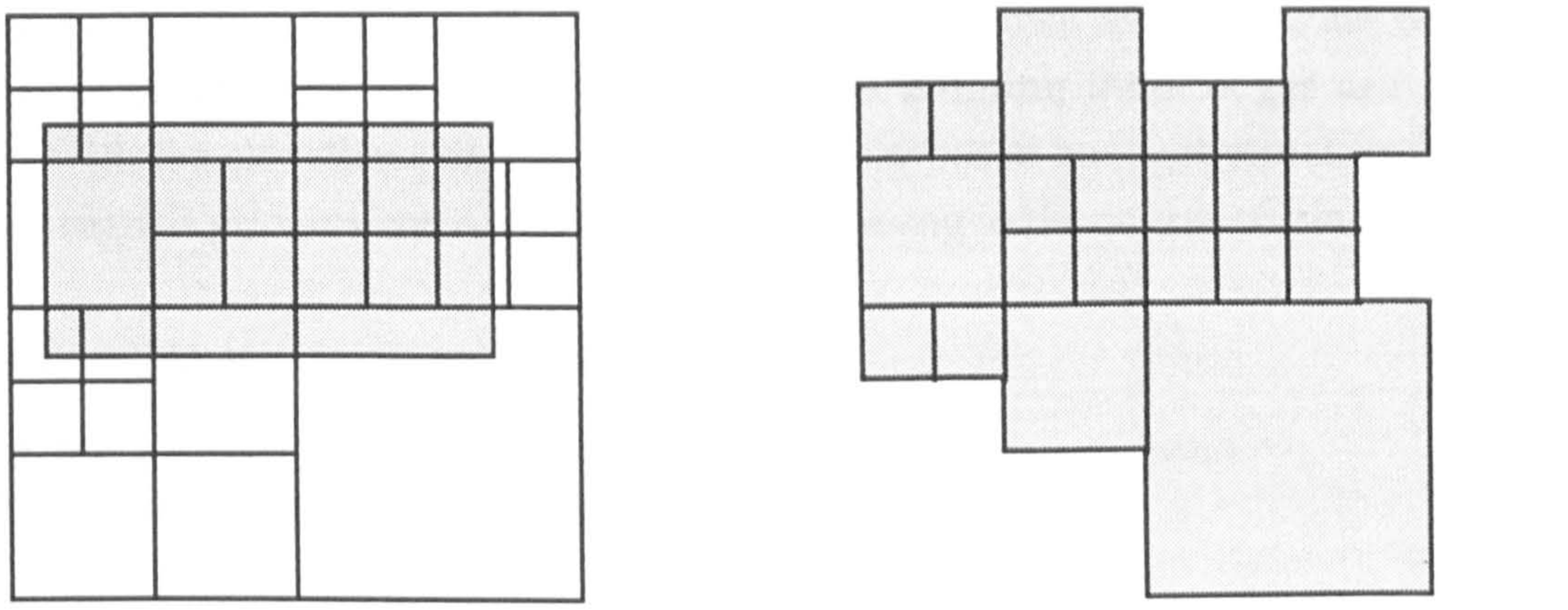


Figure 7.1 - Overview of the single-scale database.

Two spatial indexes, the Point Quadtree File and Object Quadtree File, are employed. Each Point Quadtree cell references all terrain points which intersect it, whereas each Object Quadtree cell references all objects which intersect it. This simple quadtree approach is preferred to the regular grid overlay method of Section 5.2.2 for reasons which will become evident in the Section 7.4.2. Quadtree cells are identified using the Morton code method. In this implementation, the bottom left hand coordinates of each cell are bit-interleaved, to form the Morton code. This code, coupled with a record of the cell size, provides a means for uniquely identifying the location and extent of each cell. The reason for maintaining separate object and point quadtrees is due to a distinction between real world objects with name and class attributes and the lower level point geometry used to describe the objects. Many points will only be used for describing the ground surface and will not be part of the boundary of objects mapped onto that surface.

7.4.2 The Implicit TIN Algorithm.

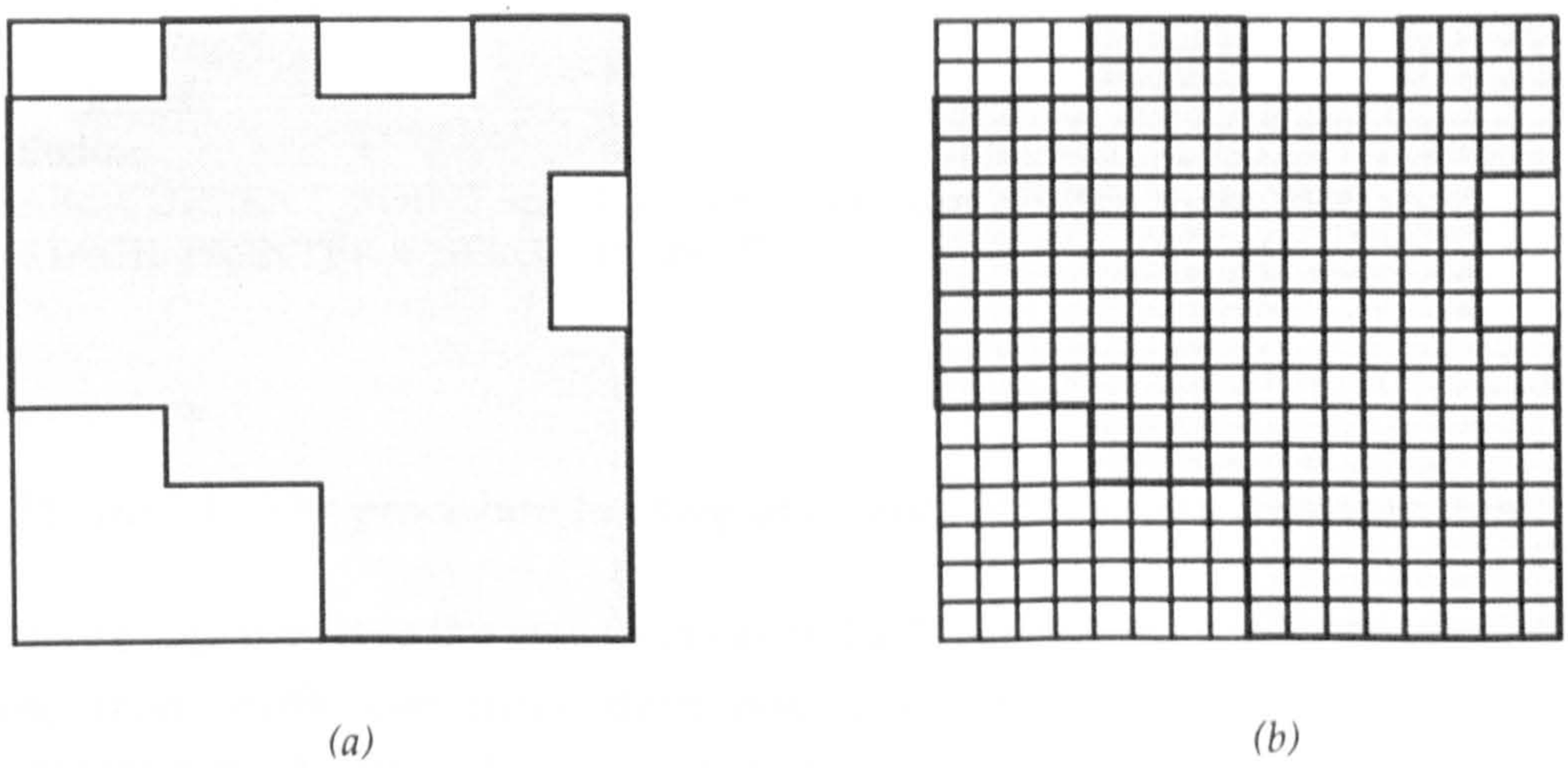
This section presents an algorithm which constructs a constrained Delaunay TIN for a given query window. The triangulation is obtained in three stages. Initially, the algorithm uses the query window to generate a list of quadtree addresses (Figure 7.2).



(a) *(b)*

Figure 7.2 - Quadtree addresses used to access relevant points and objects. *(a) The query region with respect to the database. (b) The quadtree cells (and data) accessed in the database.*

These are used to access the relevant elevation points and constraining objects via the Point and Object Quadtrees. It should be noted that all data referenced by these quadtree cells are retrieved, not just the data that intersect the query window. Data which lies beyond these quadtree cells will be retrieved if required during triangulation. The advantage of using the quadtree approach, as opposed to the regular grid overlay scheme adopted in the MTSD systems, lies in the fact that where data is sparse, quadtree cells will be large. This means that if the nearest data to the edge of the query window is in fact distant it may still be in one of the originally generated quadtree cells. This limits the need to re-access the database during the subsequent triangulation.



(a) *(b)*

Figure 7.3 - Elevation and edge points stored in box-sort structure. *(a) A bounding rectangle placed around quadtree cells. (b) Box-sort structure with referenced and empty cells.*

The object data, consisting of a combination of polygons, lines and points, are reduced where appropriate to a list of edges. The vertices defining these edges are stored, together with the elevation points, in a main-memory based box-sort structure (Figure 7.3). This regular grid structure provides spatial indexing in the course of triangulation.

```
Procedure DELAUNAY_TRIANGULATE

/* TNBS - an array used to hold the list of Thiessen neighbours of a point */
/* NTN - the number of Thiessen neighbours a point has */

Define query region.
Use region definition to generate the required quadtree addresses for
both object data quadtree and height data quadtree.
Read required data. Store objects as a sequential list of edges, each referencing two
vertices.
Store object vertices and height vertices in the box-sort data structure.
Put all vertices VERTS (NUMBER_OF_VERTICES)
on the TRIANGULATION_STACK.
Initialise the stack pointer (STACK_POINTER = 1).
Do While STACK_POINTER ≠ NUMBER_OF_VERTICES
    Let CURRENT_POINT = top point
    of TRIANGULATION_STACK(STACK_POINTER).
    FIND_THIESSEN_NEIGHBOURS(CURRENT_POINT, TNBS, NTN).
    For each pair of Thiessen neighbours (Na,Nb)
        If both neighbours are outside of the query region, then
            If external edge (Na,Nb) intersects the query region, then
                If Na has not already been triangulated, then
                    NUMBER_OF_VERTICES = NUMBER_OF_VERTICES + 1.
                    TRIANGULATION_STACK(NUMBER_OF_VERTICES) = Na.
                Endif.
                If Nb has not already been triangulated, then
                    NUMBER_OF_VERTICES = NUMBER_OF_VERTICES + 1.
                    TRIANGULATION_STACK(NUMBER_OF_VERTICES) = Nb.
                Endif.
            Endif.
        Endif.
    Endfor.
    Add CURRENT_POINT and Thiessen neighbours (TNBS) to the TIN.
    STACK_POINTER = STACK_POINTER + 1.
Enddo.

Endprocedure.
```

Figure 7.4 - The procedure to carry out Implicit Delaunay triangulation.

The second stage involves the construction of the Delaunay triangulation of all relevant vertices, from both elevation data and constraining objects (see Procedure DELAUNAY_TRIANGULATE, Figure 7.4). Points that lie inside the query region will always belong to the final triangulation, so initially these points are placed on a stack TRIANGULATION_STACK of points to be triangulated. The Thiessen neighbours of each point CURRENT_POINT on the stack are then found in the following way (see Figure 7.5). The nearest neighbour, NNB, of CURRENT_POINT is regarded as the first

Thiessen neighbour. The next Thiessen neighbour K to the right of the edge (CURRENT_POINT, NNB), a Delaunay edge, is then found and added to the list of Thiessen neighbours. The Thiessen neighbour to the right of the edge (CURRENT_POINT, K), also a Delaunay edge, is then found. The process is repeated until the latest neighbour K is equal to the original neighbour NNB. The search for Thiessen neighbours makes use of the box-sort data structure, such that only points within the vicinity of a Delaunay edge are tested. If the search for a Thiessen neighbour includes box-sort cells which are empty (that is, lie outside the generated quadtree region) or extends beyond the box-sort coverage, the required quadtree cells in the database intersected by the local search region are accessed and the necessary vertex information is retrieved (Figure 7.6).

```
Procedure FIND_THIESSEN_NEIGHBOURS(CURRENT_POINT, TNBS, NTN)

/* TNBS - an array used to hold the list of Thiessen neighbours of a point  */
/* NTN - the number of Thiessen neighbours a point has */
/* NNB - the nearest point (neighbour) to the CURRENT_POINT */

Initialise SEARCH_AREA (in terms of box-sort cells).
FOUND = 0.
Do While NOT FOUND
    Check SEARCH_AREA for nearest point (NNB) to CURRENT_POINT.
    If NNB was found, then
        FOUND = 1.
    Else
        Expand SEARCH_AREA.
        If SEARCH_AREA now extends outside the buffer limits, then
            Generate quadtree addresses for external region and read data into
            appropriate external cell.
        Endif.
    Endif.
Enddo.
NTN = 1.
TNBS[NTN] = NNB.
J = NNB.
FINISHED = 0.
Do While NOT FINISHED
    Initialise SEARCH_AREA.
    FOUND = 0.
    Do While NOT FOUND
        Check SEARCH_AREA for Thiessen neighbour (K)
        of edge (CURRENT_POINT, J).
        If K was found, then
            .FOUND = 1.
        Else
            Expand SEARCH_AREA.
            If SEARCH_AREA now extends outside the buffer limits, then
                Generate quadtree addresses for external region and read data into
                appropriate external cell.
            Endif.
        Endif.
    Enddo.
    If K ≠ NNB, then
        NTN = NTN + 1.
        TNBS[NTN] = K.
        J = K.
    Else
        FINISHED = 1.
    Endif.
Enddo.

Endprocedure.
```

Figure 7.5 - The procedure to find the Thiessen neighbours of a point.

When all Thiessen neighbours of CURRENT_POINT have been found, the point is removed from the stack and added, together with its list of neighbours, to a list of processed points. This list, when complete, forms a vertex-based TIN. The next point from the TRIANGULATION_STACK now becomes the CURRENT_POINT and its Thiessen neighbours are found in like fashion. The process is repeated until the TRIANGULATION_STACK is emptied.

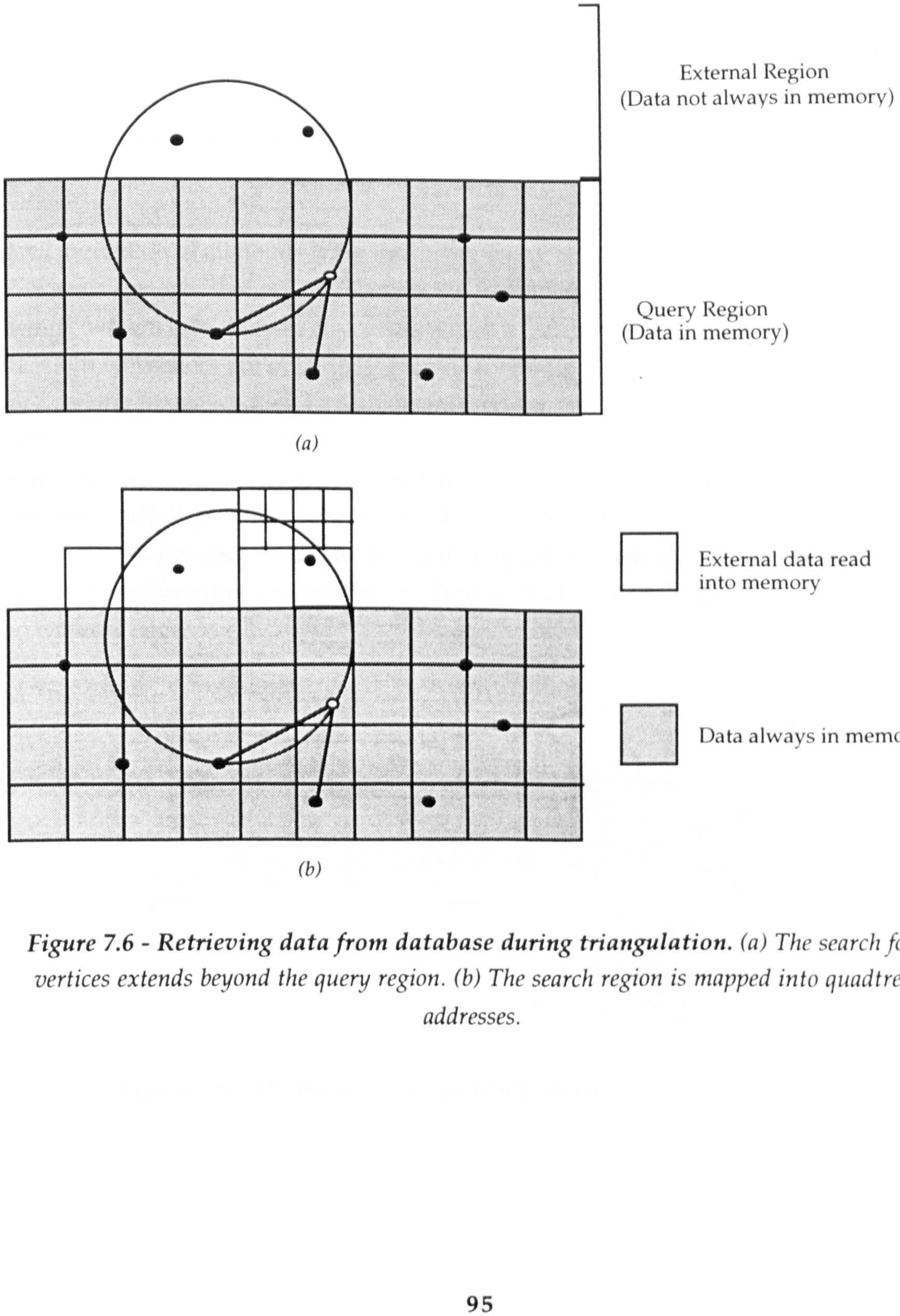


Figure 7.6 - Retrieving data from database during triangulation. *(a) The search for vertices extends beyond the query region. (b) The search region is mapped into quadtree addresses.*

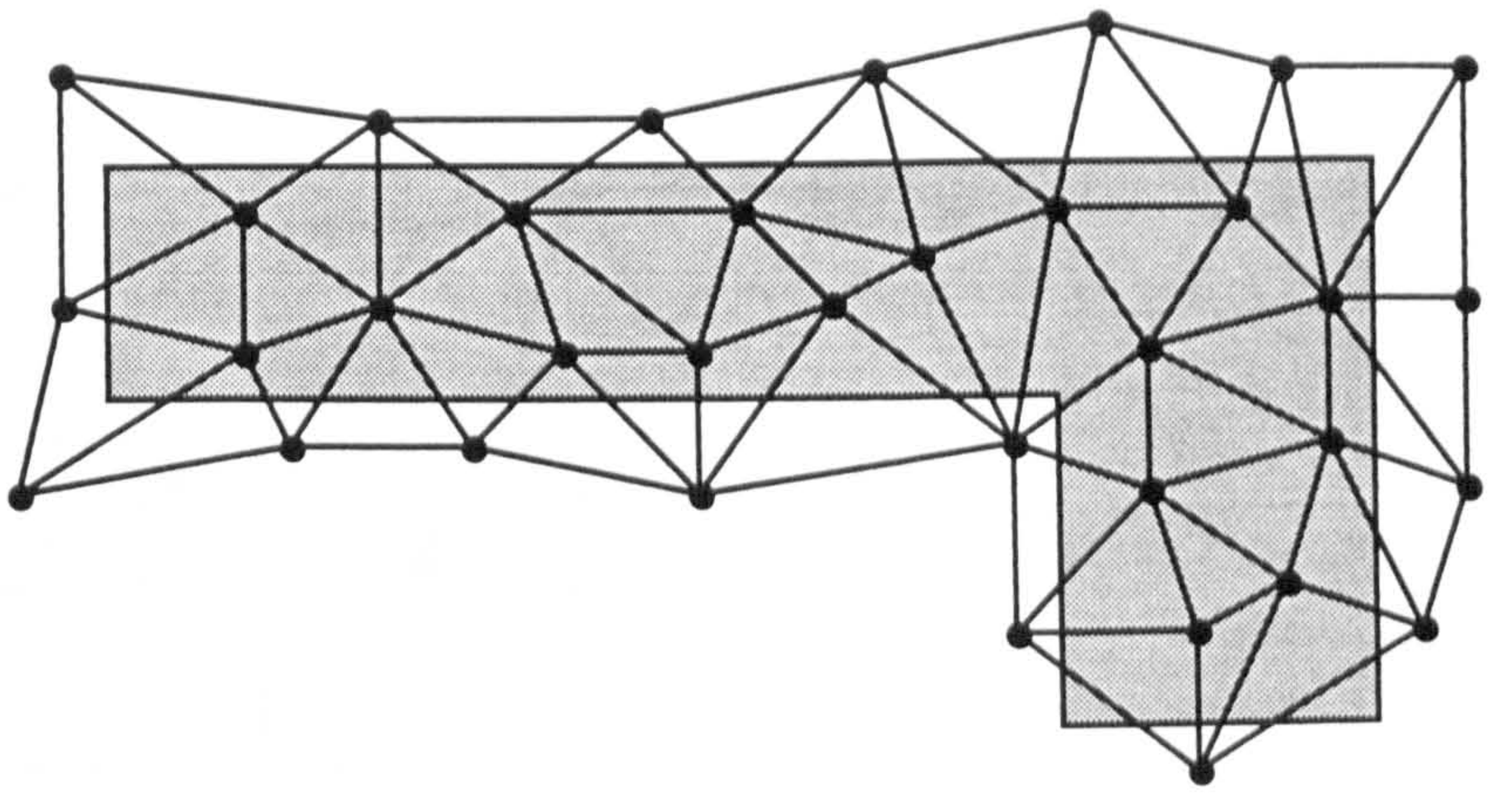


Figure 7.7 - The triangulation of all vertices within a query region. *Note that coverage is not complete in the bottom two corners.*

A triangulation of all points lying inside the query region will not guarantee a complete TIN coverage over that region. There is the possibility of parts of the query window not being covered, especially in its corners where a Delaunay edge might cross the window but both its vertices are outside (Figure 7.7). The algorithm caters for such occurrences by checking for such edges, termed external edges, and when found, adding the vertices belonging to the edges to the stack of vertices to be triangulated. Thus when the triangulation is complete, triangles will have been constructed on both sides of all such edges (Figure 7.8). This process introduces unrequired edges, which can either be retained or discarded. Such an edge is distinguished from other edges by the fact that one of its endpoints has no neighbour. Figure 7.9 illustrates the process of dealing with an external edge.

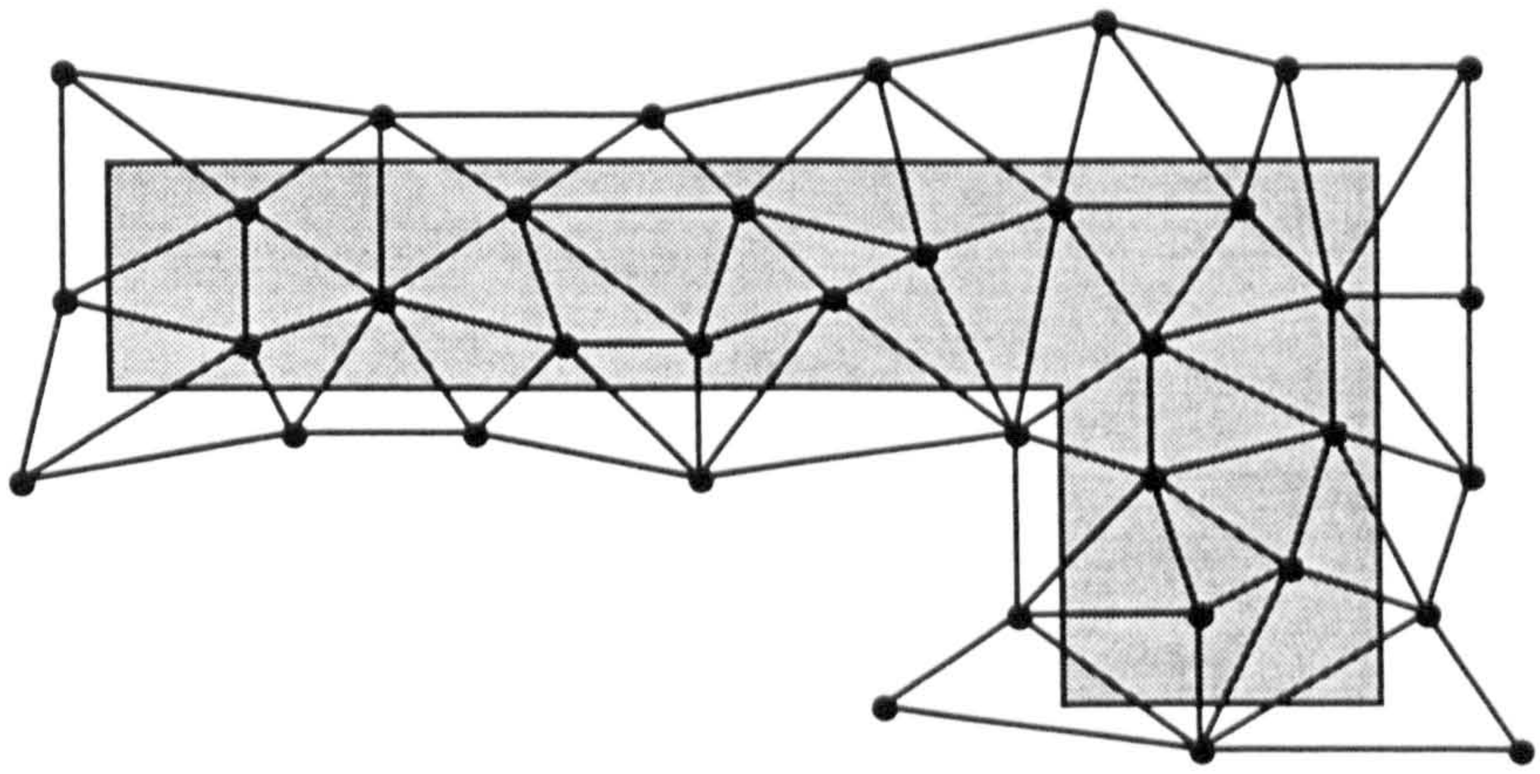


Figure 7.8 - The final Delaunay triangulation of the query region.

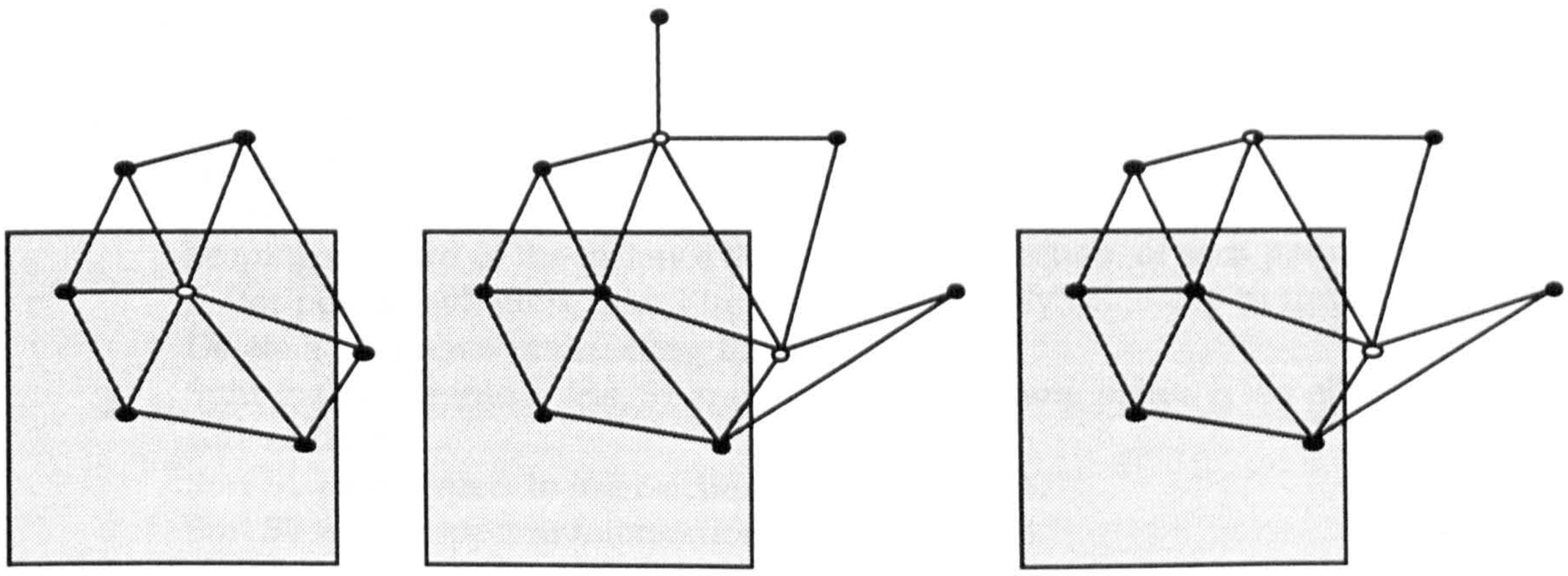


Figure 7.9 - Test for complete coverage and resolution of completeness by triangulation of external vertices. *(a) External edge intersection. (b) Triangulation of external vertices. (c) Removal of unrequired edges.*

The final stage in the triangulation process is to insert the linear constraints of all objects which lie within or intersect the current TIN (see Procedure CONSTRAIN_TRIANGULATION, Figure 7.10).

```
Procedure CONSTRAIN_TRIANGULATION

For each constraining edge A to B
    If edge not already in the TIN, then
        Identify all points which have 1 or more links that intersect the edge (A, B),
        keeping a record of the distance the point of intersection, of each point, is from
        A (for points with more than 1 intersection, it is only necessary to store 1 distance).
        Delete each points intersecting links.
        Split the points into 2 sets, S1 and S2, each containing points lying either
        side of (A, B).
        Sort S1 with respect to intersection distance from A.
        Sort S2 with respect to intersection distance from B.
        Add A to the list of neighbours of B.
        Add B to the list of neighbours of A.
        TRIANGULATE_POLYGON(A, B, S1).
        TRIANGULATE_POLYGON(B, A, S2).
    Endif.
Endfor.

Endprocedure.

Procedure TRIANGULATE_POLYGON(P1, P2, S).

Find all points not in current TIN but which belong to polygon of influence of
(P1, P2), store in V.
Search S and V for the point Q  with largest subtended angle to edge (P1, P2).
If Q not already in TIN, then
    Add Q to TIN.
Endif.
Add P1 to the list of neighbours of Q.
Add P2 to the list of neighbours of Q.
Add Q to the list of neighbours of P1.
Add Q to the list of neighbours of P2.
If Q is in S, then
    If edge (P1, Q) is not an edge in original TIN, then
        Create set S3, containing points in S lying between P1 and Q.
        TRIANGULATE_POLYGON(P1, Q, S3).
    Endif.
    If edge (P2, Q) is not an edge in original TIN, then
        Create set S4, containing points in S lying between Q and P2.
        TRIANGULATE_POLYGON(Q, P2, S4).
    Endif.
Else
    If edge (P1, Q) intersects original TIN, then
        TRIANGULATE_POLYGON(P1, Q, S).
    Endif.
    If edge (P2, Q) intersects original TIN, then
        TRIANGULATE_POLYGON(Q, P2, S).
    Endif.
Endif.

Endprocedure.
```

Figure 7.10 - The procedure to implicitly constrain a Delaunay triangulation. *Also included is a procedure to Delaunay triangulate a polygon.*

Each constraining edge (p_1, p_2) can have one of five possible states : (i) p_1 and p_2 are both vertices within the TIN and form a Delaunay edge; (ii) p_1 and p_2 are both vertices within the TIN but are not connected; (iii) either p_1 or p_2 is a TIN vertex whilst the other is external to the TIN; (iv) both p_1 and p_2 are external to the TIN; or (v) any of the cases (i) - (iv) but where the constraining edge passes through a concavity or a hole in the triangulation. Cases (i) and (ii) are the most likely, the probabilities of each depending upon the density at which the elevation and object data are sampled. In the first instance (i), the segment exists within the TIN and therefore no update is necessary. In the other cases the segment does not exist and therefore the TIN must be constrained (Figure 7.11).

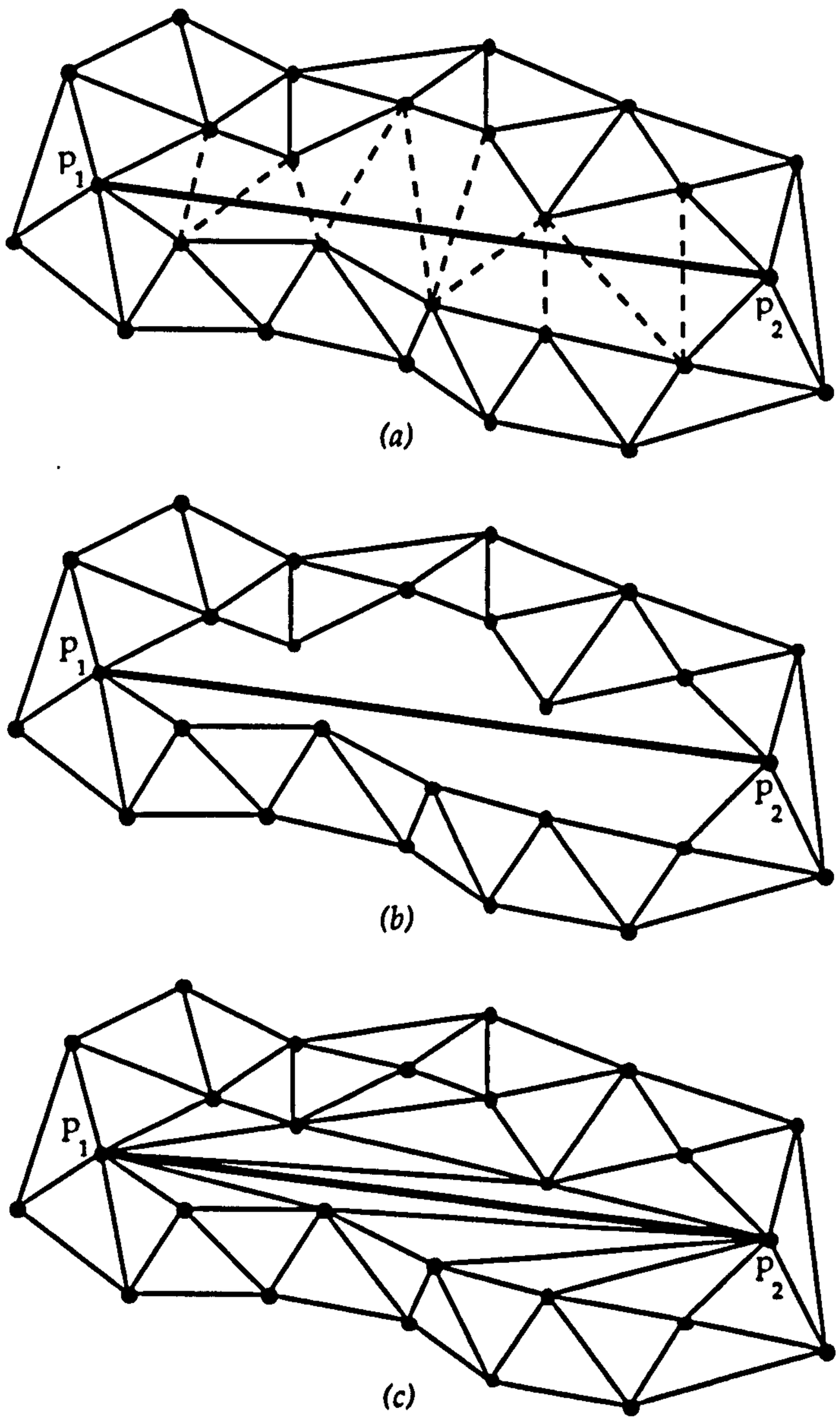


Figure 7.11 - Constrained edge insertion within a TIN. (a) Identify intersecting edges (dashed lines). (b) Delete edges. (c) Re-triangulate around constrained edge.

For a constraining segment (p_1,p_2) with both vertices within the TIN but not connected, that is case (ii), the procedure for inserting the edge can be broken down into four steps. The first consists of determining the current edges which are intersected by the constraint (Figure 7.11a). Step two proceeds to delete these edges (Figure 7.11b), while the third step involves re-triangulating around the new edge (Figure 7.11c). Finally, step four deals with updating the TIN data structure.

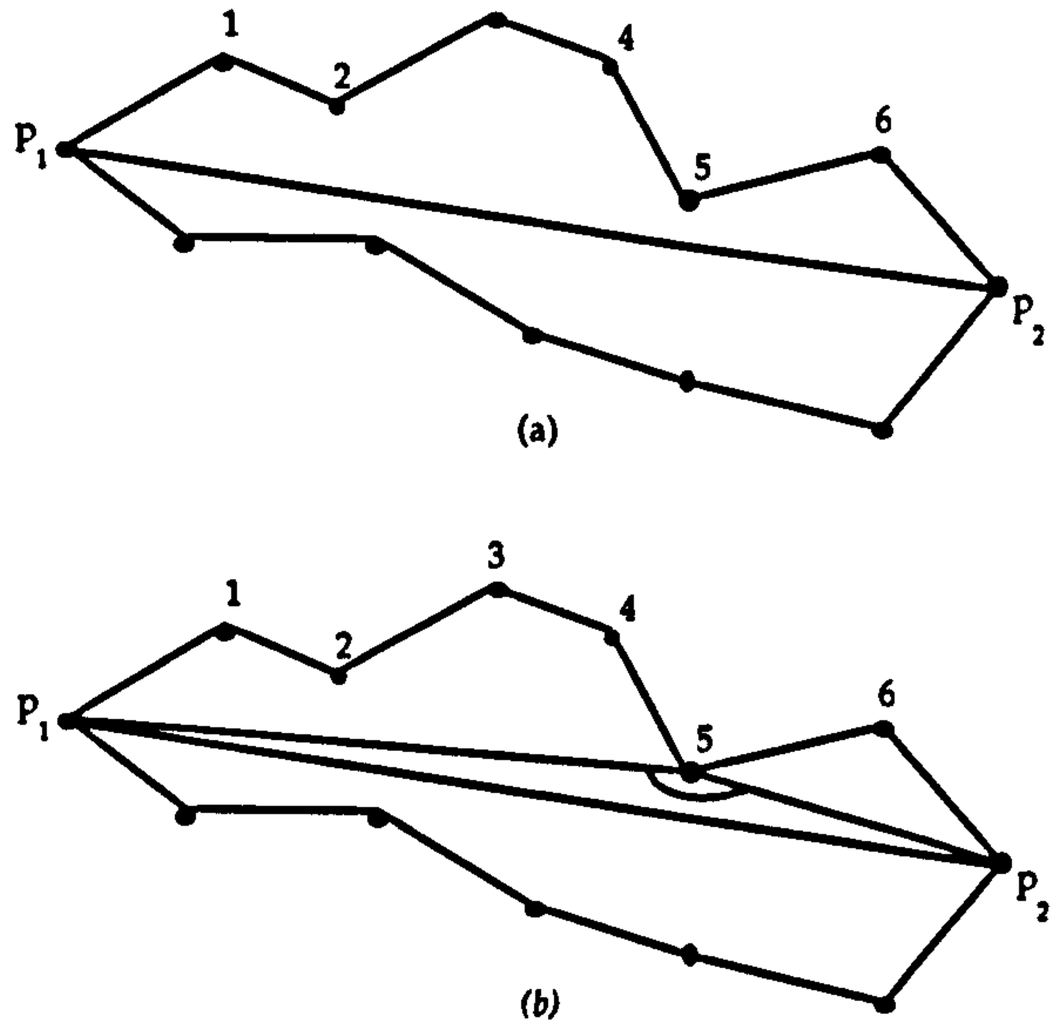


Figure 7.12 - Triangulation within a polygon around a constrained edge. *(a) The polygon, (p_1, 1, 2, 3, 4, 5, 6, p_2, p_1), to be triangulated. (b) Point 5 subtends largest angle to base edge. Thus two new polygons formed. (c) - (d) Subsequent sub-polygons treated recursively.*

The problem of re-triangulating around the constraining edge is reduced to that of separately triangulating the two polygons formed either side of the edge. These

polygons are sometimes referred to as the polygons of influence [22]. The triangulation of each polygon proceeds as follows. Consider the edge (p_1,p_2) to be the base edge of the polygon. The initial step is to find the vertex Q of the polygon, discounting the vertices p_1 and p_2, which subtends the largest angle to the base edge. For the upper polygon in Figure 7.12a this is vertex 5. This vertex is added to the list of neighbours of both p_1 and p_2, and similarly p_1 and p_2 become neighbours of vertex 5. Two sub-polygons have now been formed with base edge (p_1,5) and (5,p_2) respectively (Figure 7.12b). The two sub-polygons, and any subsequent sub-polygons, are dealt with, recursively, in the same way as the original polygon (Figures 7.12c - 7.12d). The recursion continues along a particular path until the latest new edge matches an edge in the original TIN.

For case (iii), where a constraining segment has one vertex in the TIN and the second outside, the procedure is similar, but the polygon of influence may now include vertices outside the original TIN. For example, consider the insertion of the highlighted segment (p_1,p_2) in Figure 7.13a. Here, as before, the search for the vertex Q with largest subtended angle involves examining the vertices making up the polygon of influence (discounting p_1 and p_2), which in this case includes the external point p_i (Figure 7.13b). The recursive procedure in this case continues along a particular path until either the latest edge matches an edge in the original TIN or the latest edge fails to intersect the original TIN (Figure 7.13c).

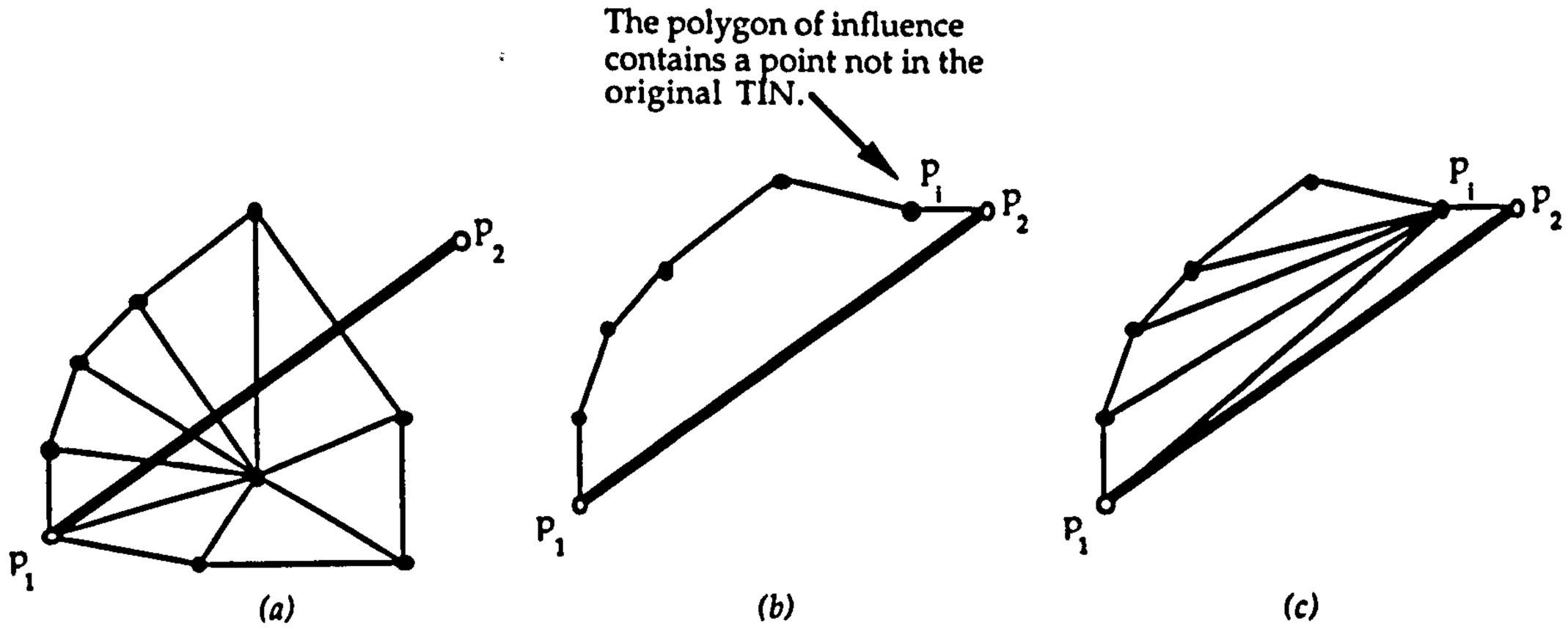


Figure 7.13 - Insertion of edge with one external vertex. (a) Vertex p_2 external to TIN. (b) Need to include all external vertices making up the polygon of influence, including external vertices, in search for Q. (c) Edge has been inserted.

The fourth possible situation, case (iv), is where both vertices of the constraining edge lie outside the original TIN. The procedure for inserting such an edge follows that of case (iii) as shown in Figure 7.14.

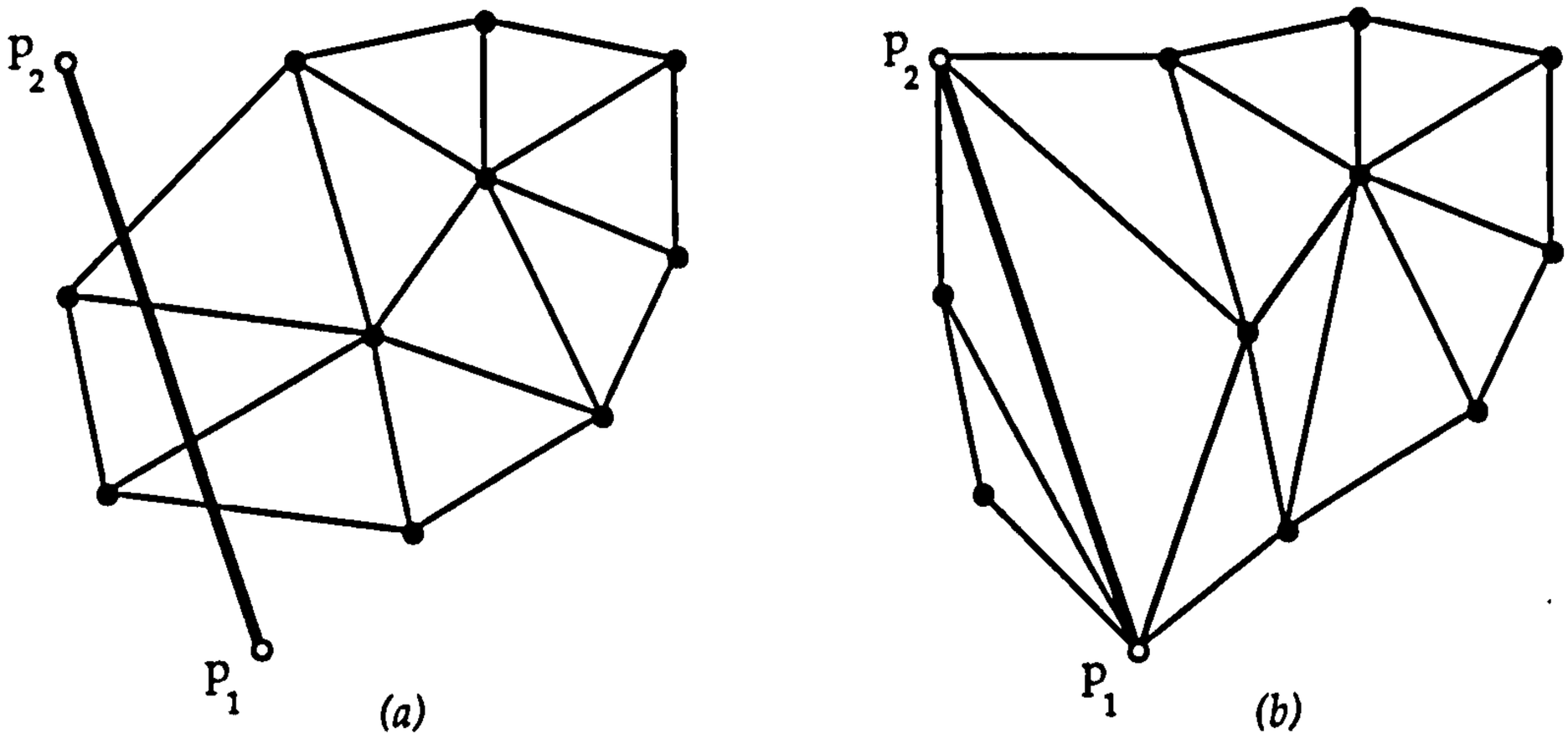


Figure 7.14 - Insertion of an edge with two external vertices. (a)Triangulation prior to insertion of constraint. (b) Triangulation after insertion.

In some cases the Implicit TIN algorithm will produce triangulations containing holes, due to triangles crossing concave regions of a query window. The algorithm has been designed to handle query windows that are themselves concave in shape or include a hole. Introducing a constraint which passes through such a hole or concavity is catered for by using the methods for case (iii) and case (iv), as shown in Figure 7.15.

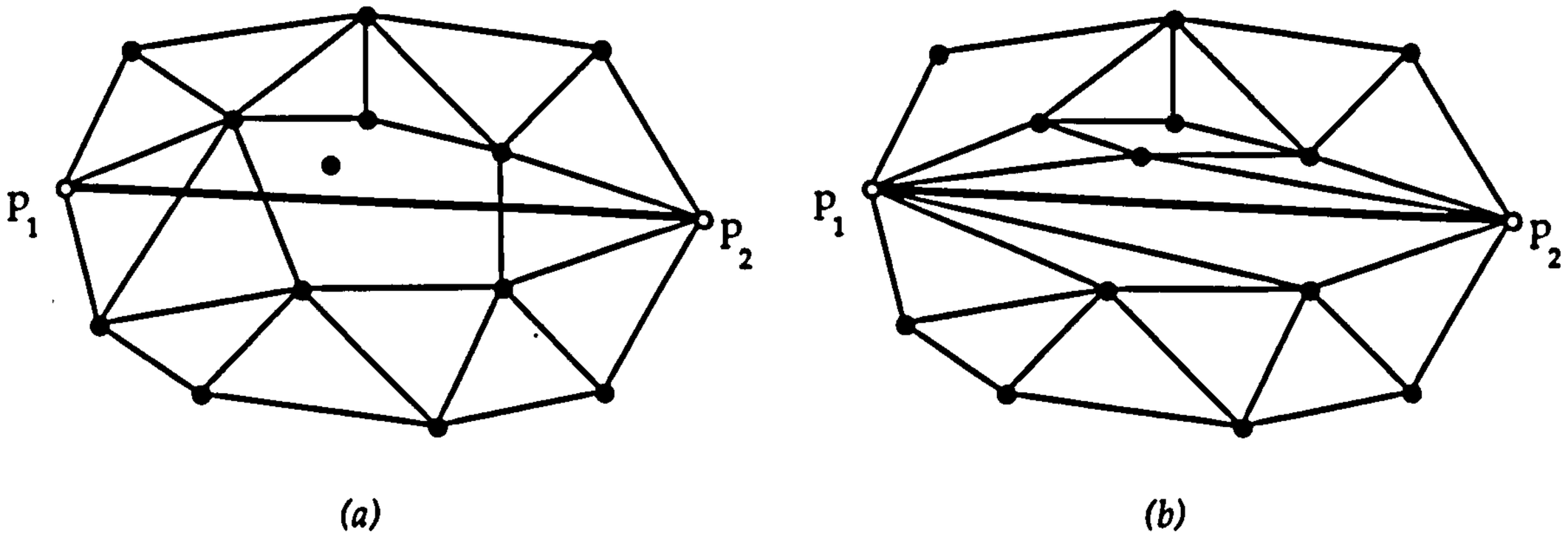


Figure 7.15 - Insertion of an edge through a hole in the triangulation. (a) Triangulation before insertion of constraint. (b) Triangulation after insertion of constraint. Notice that a part of the hole remains un-triangulated since it does not influence the query region.

7.4.3 Triangulating within a Restricted Region.

To construct TIN topology for any query the algorithm in the previous section requires an initial vertex to start the triangulation process. In most cases an arbitrary vertex from within the query window is chosen (the vertex which happens to be on the top of the TRIANGULATION_STACK). However, in certain circumstances, no vertices lie within the initial query window. This situation may arise if the query window is

narrow, or has no width as in the case of a profile. In such a case the initial vertex can be found in a number of ways. One method is to find the nearest neighbour of the centre of gravity of the points defining the query window. Another method is to search for a straight line segment (constraint) that crosses the query window and to select one of its bounding vertices. This would have to act as a preliminary method in that it will of course only work if there is an intersecting constraint. When an initial vertex is found, the algorithm proceeds to find its Thiessen neighbours. Each connecting edge (initial vertex to Thiessen neighbour) is then tested for intersection with the query window. If intersection does occur the neighbour is placed on the TRIANGULATION_STACK. If one or more intersections are found the remaining TIN can be constructed as in Procedure DELAUNAY_TRIANGULATE. If no intersecting edge is found another 'initial' vertex in the vicinity must be selected and the procedure repeated. The case where the query region is completely contained within a Delaunay triangle, that is, there are no intersecting edges, must also be catered for. This is achieved by finding the Thiessen neighbours of the vertex closest to the query region. One of the triangles thus formed must contain the query region.

7.4.4 Implicit TIN Flexibility.

It should be noted that storage saving is not the only advantage gained by using the Implicit TIN. The method also provides adaptability in allowing for selection of specific features for integration into a triangulation. Thus constraints on the terrain model are not predetermined, as they would be in an explicitly defined TIN.

The line features to be included in the database can be grouped into two categories. Firstly, there are those which can be described as structural. Such features, when included as constraints in a terrain model, will improve the accuracy of the model in describing the form of the terrain. Lines that describe phenomena such as ridges, valleys and breaks of slope fall into the category. Also included are any lines that describe physical objects, such as roads, rivers and the outlines of buildings. The second category of line features are those which are non-structural. These lines are distinguished by the fact that they do not add to the accuracy of the surface representation, but are used as constraints merely to facilitate visual display of the surface. A typical non-structural line could, for example, represent an administrative boundary (which may in some circumstances coincide with structural lines).

The significance of distinguishing between line features in this way is that it offers a measure of flexibility when producing a topographic surface. When structural lines are initially added to the database, their vertices are added to the terrain data and the lines labelled as necessary constraints. This will ensure that such lines will always be included as constraints when TIN topology is constructed. In contrast to this, non-structural lines can be labelled as not always necessary constraints, and as such are

only included in a TIN when a particular query requires their presence.

7.5 The Implicit TIN in a Multiresolution Environment.

The Implicit TIN algorithm from the previous section has been adapted to form part of an experimental multi-scale database (I_MTSD), details of which are given by Jones, Kidner and Ware [88]. This database can be regarded as the implicit equivalent to the MTSD 2.0 described in Chapter 6.

7.5.1 Database Design and Construction.

The database is partitioned into n levels, each representing the topographic surface at a different resolution. The top level (level 0) represents the coarsest resolution and the bottom level (level n-1) the finest resolution. The database is initially constructed in a way similar to the construction of the conventional MTSD. Points forming part of a topographic feature are assigned a level of significance using the Douglas-Peucker algorithm, while the terrain points are assigned their level of significance as a result of applying De Floriani's point insertion algorithm. The implicit version differs from the conventional in that when pyramid construction is complete (that is, all points have been assigned a level of significance), all TIN topology is discarded. This topology is re-constructed, where and when required, by applying the Implicit TIN algorithm. Figure 7.16 illustrates the main components of the database and will now be explained.

The Information File stores the lateral error (related to object resolution) and the vertical error (related to terrain resolution) associated with each level and a record of the number of levels in the database. The Point Files, with one file at each database level, store a point identifier and x, y and z coordinate for each vertex. The Point File at a particular level will only record information pertaining to points whose level of significance matches to that level. Also, vertices that define non-structural line features are assigned a null z-value. When such vertices are included in a TIN construction their z coordinates are interpolated from the terrain elevation data.

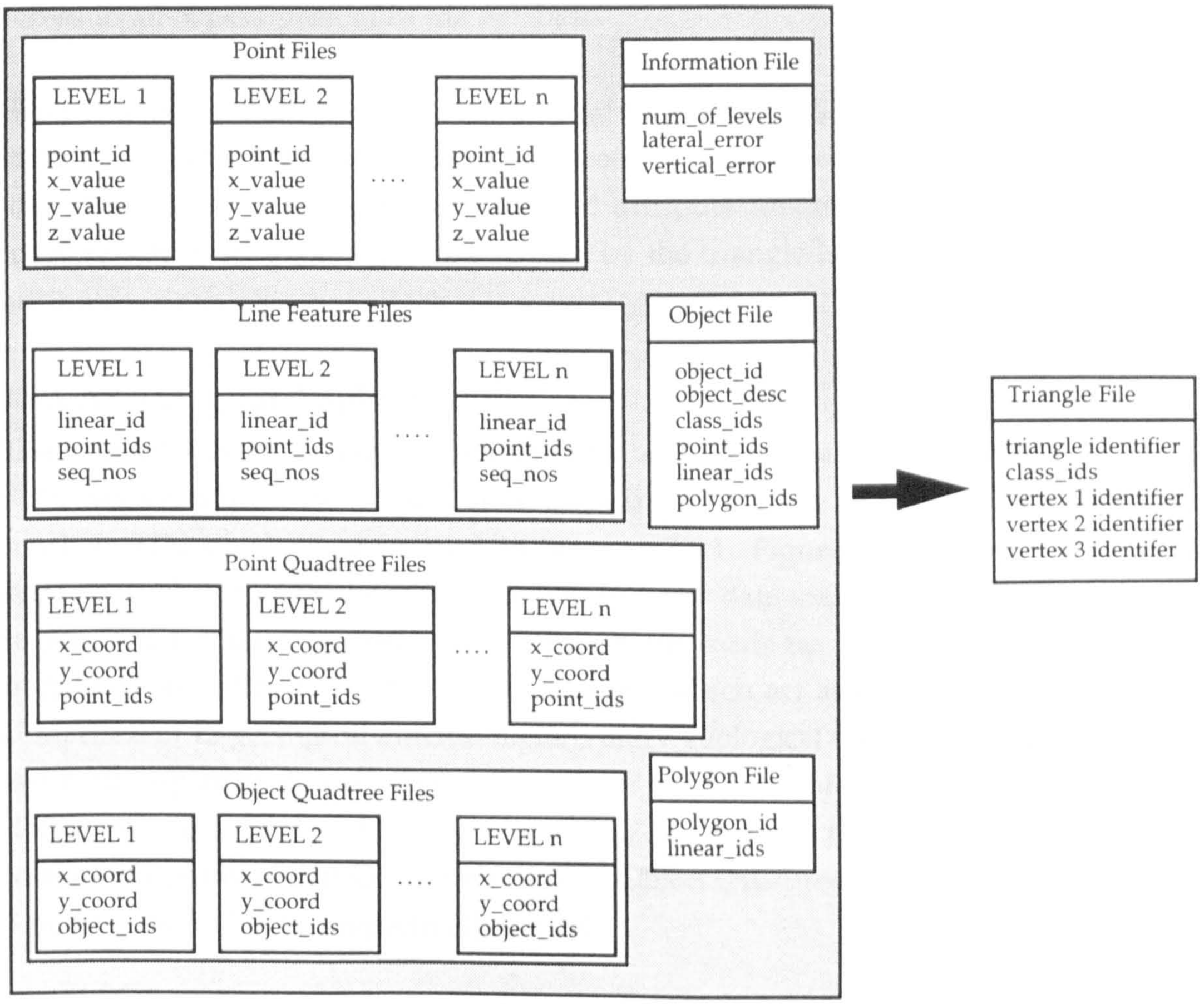


Figure 7.16 - Overview of the multi-scale database (I_MTSD).

The Object File contains a record for every object in the database. It is assumed that each object is present at every level, and has the same constituent parts at every level. An object record consists of an object identifier, an object description, an integer value which acts as a 'real world' classification code and lists of references to its component polygon, line and point features. Note that Point objects refer directly to the Point File where the coordinates are stored. The Polygon Feature File stores the description of each polygon feature present in the database. Each polygon is described by its polygon identifier and a list of the identifiers of the line features from which it is made up. The Line Feature Files, with one file per database level, stores for each line feature its identifier and the list of point identifier/sequence number pairs which make up the line. Each file only stores the identifiers/sequence numbers of vertices which are introduced at its level. Therefore, to construct a line feature at a given level, it is necessary to access all Line Feature Files from the highest level of occurrence down to the given level.

There are two quadtree files, a Point Quadtree File and an Object Quadtree File, at each level of the database. Each Point Quadtree File is used to provide spatial access to all points relevant to its particular level. Each Object Quadtree File provides spatial

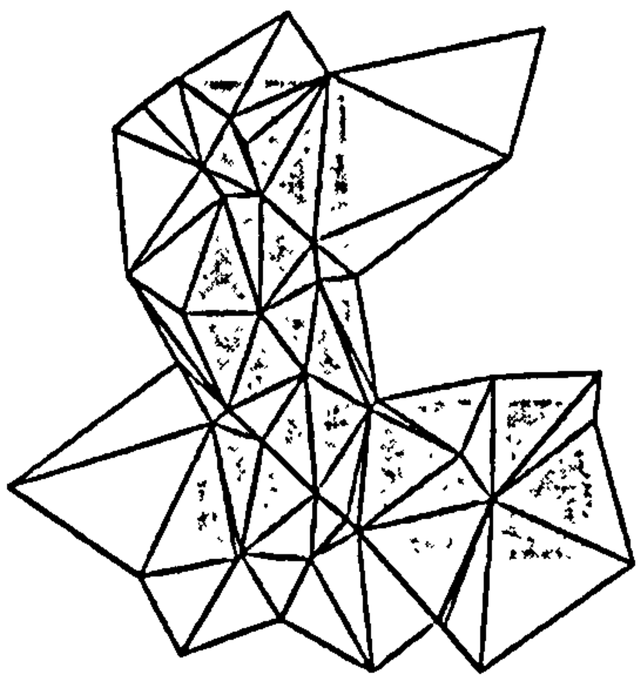
access to all objects present in the database.

Explicit triangulations, formed as a result of triangulating the Implicit TIN, are stored temporarily in the Triangle File. This file records for each triangle its identifier, the point identifiers of its constituent vertices and attribute identifiers associated with the triangle. Attribute identifiers are obtained by the triangle taking on the classification attributes of any object of which it is a part.

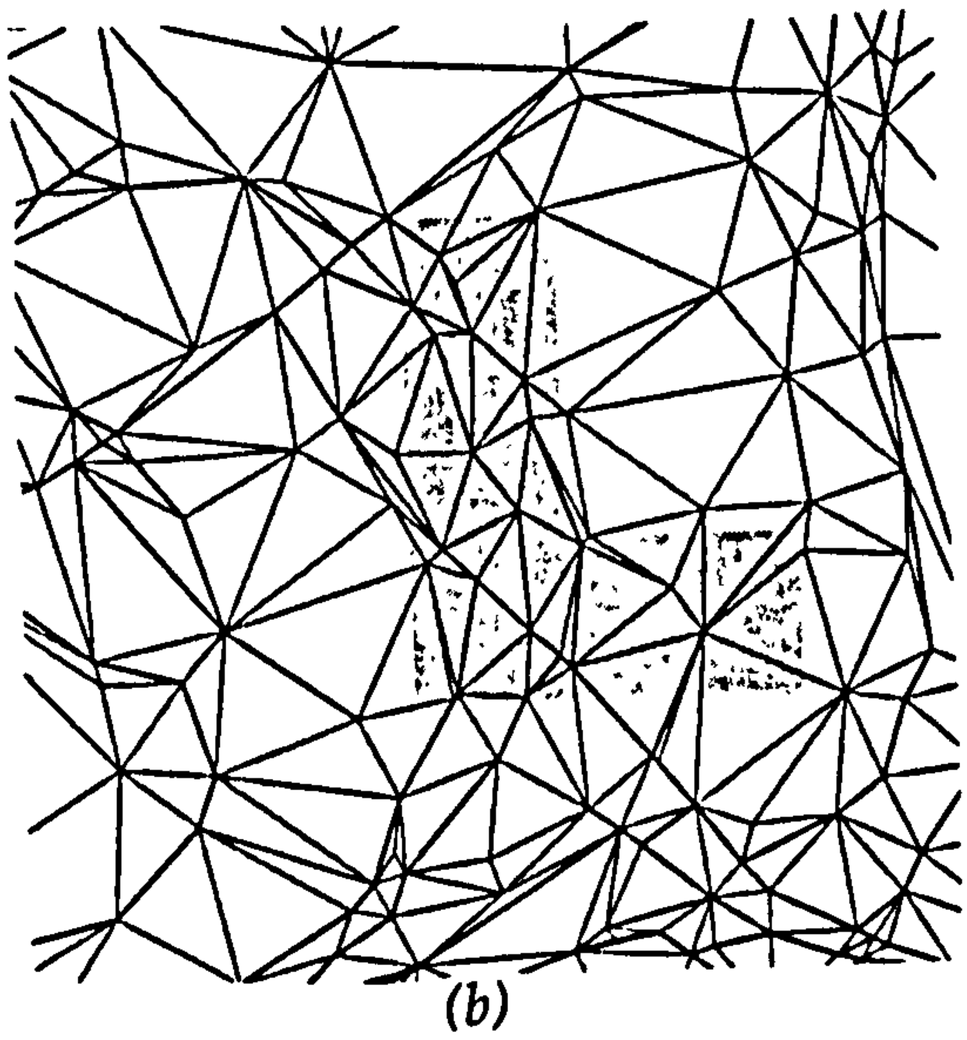
7.5.2 A Geological Application.

The Implicit TIN algorithm described in section 7.4.2 has been implemented in C on a SUN workstation. It forms part of an experimental ISAM version of the multi-scale Implicit TIN database described in section 7.5.1. Figure 7.17 shows the output produced when the database was applied to a test data set, using an L shaped query region. The test data consists of a terrain surface, made up from 612 points, and a set of geological outcrop features. These features, which act as constraining objects, are comprised of 13 geological outcrop regions and 7 geological faults. The outcrop regions are made up from 20 polygons, while there are a total of 143 line features used to define these polygons and the fault lines. The constraining features are described by a total of 967 points. Point Quadtree cells and Object Quadtree cells have a maximum of 5 points and 5 objects, respectively, per cell.

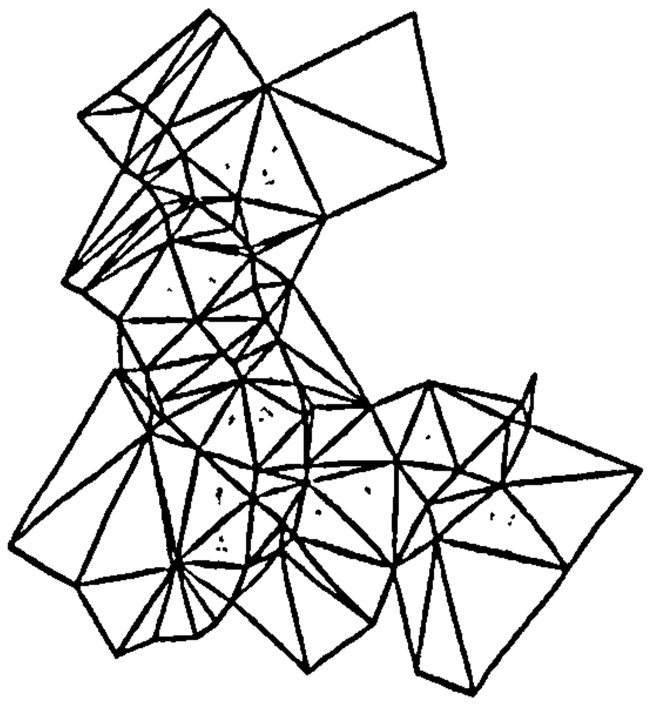
Two database levels are shown to demonstrate the differing amount of detail at each level. The first level (Figure 7.17a) was created with vertical and horizontal error tolerances of 10 metres. This resulted in 45 points being retrieved for the given query window, 14 for the terrain elevation and 31 for the linear constraints. The second, more detailed level (Figure 7.17c) was created with vertical and horizontal error tolerances of 5 metres, resulting in a total of 77 points (21 terrain and 56 linear constraints) being retrieved. For each level, a complete triangulation of the corresponding part of the database is also shown (created using the conventional constrained Delaunay triangulation algorithm of De Floriani and Puppo [22]). The less detailed section (Figure 7.17b) contains 587 points, while the more detailed section (Figure 7.17d) contains 891 points. Inspection of the triangles shown on Figure 7.17 confirms that the Implicit TIN produces the same topology as that of the conventional TIN.



(a)



(b)



(c)

(d)

Figure 7.17 - Output from I_MTSD. (a) and (c) The triangulation produced by the Implicit TIN algorithm when applied to two levels of the test database. (b) and (d) The triangulations produced using a conventional constrained Delaunay triangulation algorithm.

7.6 Performance of the I_MTSD.

The major advantage that an Implicit TIN system holds over a conventional, explicitly defined TIN system is the saving in storage space made as a result of not recording TIN topology. Indeed, in a comparison of different methods for storing digital elevation data Kidner [87] has been shown that the Implicit TIN is the most space efficient. When a single scale Implicit TIN database, of the type described in section 7.4.1, is compared to an equivalent explicit TIN database using triangle adjacency pointers, there is an approximate storage saving of

12N - 6B -12

where there are a total of N points, B of which are boundary points. The explicit scheme incurs an additional storage cost proportional to 3N to represent coordinates of the points. This additional cost becomes 4N if it is assumed that a unique identifier is also stored for each point. It is also necessary to consider the storage requirements for the geometric definitions of objects. If the object definitions are stored as lists of point identifiers, the approximate upper limit on the storage required approximates to 1N. Therefore, the storage required by an explicit TIN, in addition to triangulation topology pointers, is estimated to be 5N. This estimate can be regarded as a measure of the total storage cost for the implicit scheme as other than constraints, there is no triangulation topology. The number of points defining the boundary is usually much less than N and thus triangulation topology storage approximates to 12N. Therefore the relative size of the implicit and explicit TIN schemes is in the ratio 5/17.

If the Implicit TIN is considered in the context of a multi-scale database, as has been described in section 7.5.1, it follows that its approximate storage saving when compared to its explicit equivalent (see Chapters 5 and 6) is equal to

$$\sum_{i=0}^{m} 12N_i - 6B_i - 12.$$

Here there are (m+1) levels in the database and a total of N_i points in the triangulation at level i, B_i of which are boundary points. Using the same argument as described in the previous paragraph, the storage saving made when comparing any level in the implicit system to its equivalent in the explicit system is in the ratio 5/17. It follows that as the number of levels increase, the overheads for explicit scheme increase significantly. For example, consider a database with five levels, each of which involves a reduction in the number of points by two-thirds. In this case, the overhead would amount to about 50% of that of the most detailed level, that is, in proportion to 6N. Therefore the overall ratio of storage between implicit and explicit databases would be 5/23.

The usefulness of the Implicit TIN will depend, for many applications, on the ability to reconstruct the correct constrained Delaunay triangulation for a given query region

within a satisfactory length of time. The maximum time allowed to perform this task will relate to the specific needs of the application. The major time penalty introduced by the Implicit TIN system is that of having to reconstruct the constrained Delaunay triangulation from the main memory data. The reconstruction algorithm currently used has a worst case time complexity of $O(N^2)$, where N is the number of points to be triangulated. This represents an upper bound on time for any incremental Delaunay triangulation algorithm (constrained or unconstrained), although some parallel algorithms improve on this, with O(logN) reported by El Gindy [89].

A series of tests have been carried out which show the I_MTSD achieves satisfactory results with regards triangulation reconstruction time. For example, the response time for producing the triangulation shown in Figure 7.17c was approximately 21 seconds. It is believed that response times are as yet far from optimal and an improved implementation, based on the same logical design, is possible. For example, the introduction of parallel processing methods to triangulation can be expected to improve performance in the future [89, 90]. Figures 7.18 and 7.19 give the results of comparison tests made between I_MTSD and MTSD 2.0. These results indicate a storage saving of about 75% when employing the implicit approach. The resulting increase in query response time is seen to depend greatly on the size of the query region. For example, I_MTSD queries involving the full spatial extent of the database incur an increase in response time of approximately 30% when compared to the equivalent MTSD 2.0 query (Figure 7.18). In the case of the L-shaped query region the percentage response time increase when using the I_MTSD is not as great, ranging between 0% and about 20% (Figure 7.19).

Method	Number of levels	Vertical Error (m)	Lateral Error (m)	Storage used (K-bytes)	Time taken to retrieve all triangles (s)
MTSD 2.0	3	30.0	5.0	395	8.0
		12.5	2.5		12.0
		1.0	1.0		17.0
Implicit TIN	3	30.0	5.0	104	11.5
		12.5	2.5		16.0
		1.0	1.0		23.0

Figure 7.18 - Results of comparison tests between I_MTSD and MTSD 2.0. Here, queries involve the full spatial extent of the database.

Method	Number of levels	Vertical Error (m)	Lateral Error (m)	Storage used (K-bytes)	Time taken to retrieve all triangles intersecting L-shaped query window (s)
MTSD 2.0	3	30.0	5.0	395	6.0
		12.5	2.5		10.5
		1.0	1.0		16.0
Implicit TIN	3	30.0	5.0	104	6.5
		12.5	2.5		10.5
		1.0	1.0		22.0

Figure 7.19 - Results of comparison tests between I_MTSD and MTSD 2.0. Here, queries involve an L-shaped region.

7.7 Summary and Conclusions.

This chapter has introduced a new, and improved, version of the Implicit TIN data storage scheme. The most important aspect of this work has been the development of a novel constrained Delaunay triangulation algorithm, specifically designed for use in an Implicit TIN environment. The Implicit TIN has been shown to be a data storage scheme that can be used as a space efficient means of representing topographic surfaces in a multi-scale environment. The major benefits of the implicit approach are its storage efficiency and the flexibility it gives in allowing for the selection of specific line features, and the omission of others, for inclusion as constraints upon the model. Execution of the TIN reconstruction algorithm inevitably introduces a time overhead. Whether this overhead is acceptable will depend on the specific application.

Chapter 8

A Multi-Scale Geological Model

8.1 Introduction.

This chapter gives details of a new data model suited to the efficient multi-scale representation of 3-D geological data. This model can be regarded as a natural progression of the MTSM described in Chapter 5. Section 8.2 provides a brief introduction to 3-D GIS, often referred to as Geoscientific Information Systems (GSIS). In particular, it outlines the 3-D GIS project currently being carried out by the British Geological Survey (BGS), and gives a review of the general requirements of a GSIS. The spatial representation of 3-D data is discussed in Section 8.3. The subject is looked at firstly from an object representation point of view, with attention being given to the boundary representation and octree methods. Various techniques for providing efficient spatial access to collections of 3-D objects are then reviewed. Section 8.4, building on the design of the MTSM, describes a multi-scale 3-D data model. The model, which represents both the ground surface and geological subsurface horizons, is based on a series of constrained Delaunay pyramids. A conclusion and summary is given in Section 8.5.

8.2 An Introduction to Geoscientific Information Systems.

Recent years have seen the accumulation of large quantities of geological and geophysical information. This information comes in the form of raw data, such as well logs, seismic surveys, and gravity and magnetic studies, and interpreted data, such as contours, cross sections, grids of horizons and outcrop maps [91, 92, 93]. Large geological institutions, such as BGS, are currently converting this information into digital form. The availability of such digital data has created the possibility for the development of a 3-D GIS. Central to this system is a 3-D geological model, based on the interpretation of various data sources. It is hoped that the equivalent of currently existing published maps will in future be 'derived as projections of the model' [94].

8.2.1 The BGS Project.

In 1990 BGS commenced a research programme under the title 'Three-dimensional Integrated Geoscience Mapping'. The main objective of the programme was 'to develop the concept of a unified, 3-D representation, or map, of the geology based on surface mapping and borehole information, and constrained by joint modelling of multiple geophysical datasets' [95]. The 3-D geological map would be designed by a geologist on a graphics workstation, starting from a digital terrain model, field data and available borehole information. Traditional maps, cross sections and 3-D views (from various angles) could then be produced as an output of the 3-D map. From the outset of the project it was realised that often there is no direct geological data available. In such instances, geophysical methods, such as seismic reflections and potential field surveys, provide the only evidence of subsurface structure. However, a problem exists in that each geophysical technique pertains to a particular physical property (for

example, density or velocity), and when compared will often yield contradictory models of subsurface geology. With this in mind, the BGS project is also concerned with the design of software which will enable users to correlate and interpret all available data sets in terms of a single, unified 3-D model.

8.2.2 Basic Requirements of a GSIS.

The basic functions of a GSIS (see [96, 97]) are little different from those in a 2-D GIS, even though there is greater complexity involved in the design and implementation of the 3-D system. For example, such a system has to provide a process for creating a 3-D model from available data; have the ability to efficiently store and access the information contained within the model; include means by which stored information can be updated; provide functions for performing analysis of the model; and include procedures for displaying the model and results of analysis (Figure 8.1).

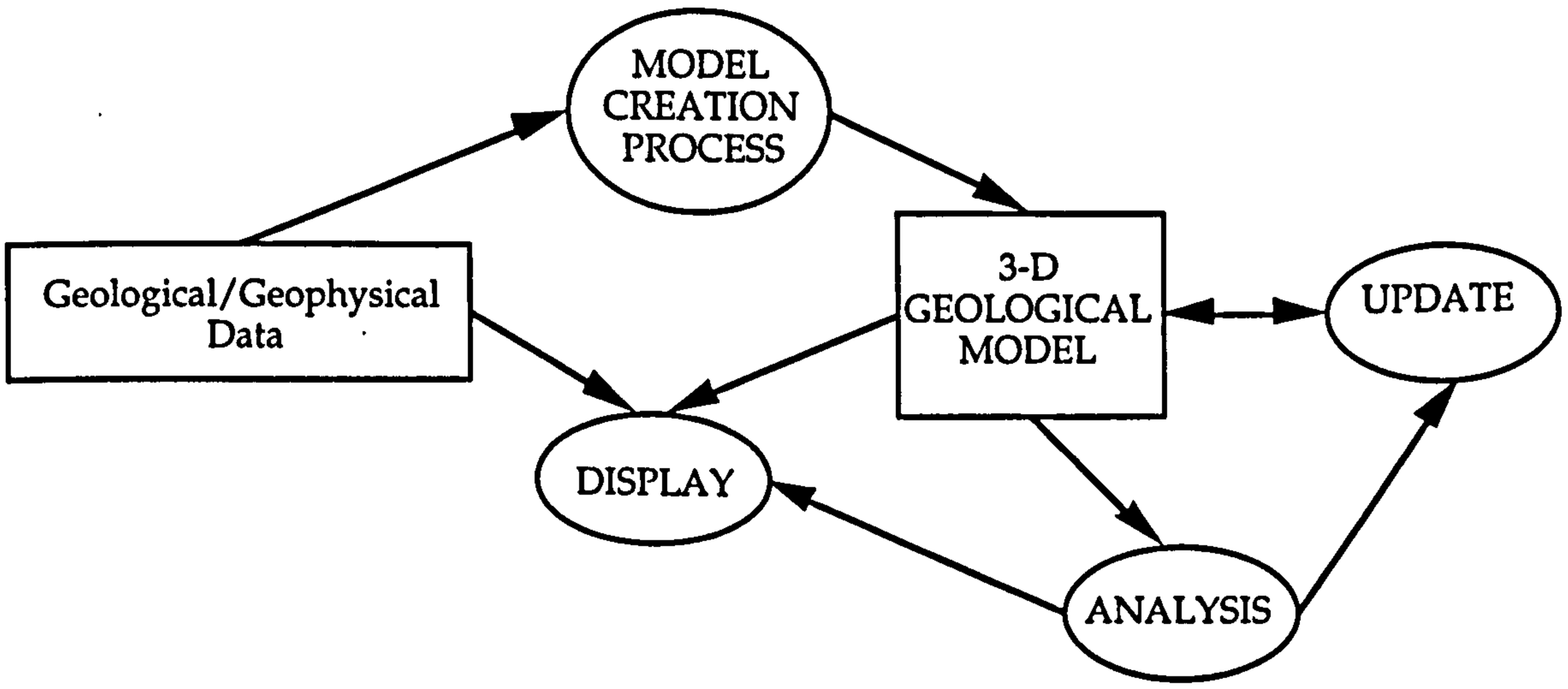


Figure 8.1 - The basic components of a 3-D GIS.

Of these five basic requirements, this thesis is primarily concerned with the first two, that is, model creation and model storage. Model update has been given some thought in Section 9.3, but not to any great extent. Also, it is felt that the development of new techniques for the display and analysis of the 3-D model provides scope for considerable research effort in itself (see, for example, various papers in [92]). However, 'the usefulness of the model will come from the ease with which it can be developed, modified and analysed' [93]. Therefore, since the 3-D model with which this work is concerned must eventually form part of an overall system, the implication is that for the purpose of update, display and analysis, any such model will have to be dynamic, provide efficient access to information and allow for high levels of detail to be included when describing objects.

An additional requirement of the data model being detailed in this chapter will be to

allow for the creation, storage, display and analysis of data at different levels of detail. This capability is 'clearly desirable, both for the purpose of mapping at a range of scales and for search and overlay procedures' [91].

For the purposes of this section, it is important to consider two fundamental differences that exist between a conventional GIS and its geological equivalent, a GSIS. The first difference lies in the dimensionality of the data being stored. GIS are primarily two-dimensional and are interested in the main with x and y coordinates. When the third dimension is involved it is usually only included as an attribute of the 2-D data. Consequently, the databases, data structures, analysis functions and display/output facilities provided by GIS are suited only to 2-D data. For example, even a GIS facility which at first glance may appear to offer three-dimensional capabilities, such as a Delaunay TIN model, is primarily concerned with the x and y coordinates of a point, and can only deal with surfaces which are single-valued with respect to the xy-plane. Conversely, a GSIS needs to cater for three-dimensional data. The world of the geologist is a 3-D world, where the z coordinate is as important as either the x or y coordinate, and as such the geologist's data is 3-D. Therefore, the databases, data structures, model creation and analysis operations and display/output facilities provided by a GSIS need to be able to deal with 3-D data.

A second difference lies in the fact that when dealing with subsurface geological data, it is very often the case that only incomplete, and sometimes conflicting, information is available [97, 98]. This situation is mainly due to the difficulties, and associated high financial costs, in collecting subsurface information. The problem of limited subsurface data is compounded by the extremely complex nature of certain subsurface structures. This incomplete subsurface (and hence not visible) data can be contrasted with the highly visible, and hence relatively easy to sample, nature of 2-D geographic data.

As has been stated, geological information is made available from a number of sources. In order to create the best model it may be argued that a sensible approach to model creation is to consider all available data during the creation process. Dabek et al [99] add weight to this argument by recognising that 'the correlation and joint 3-D modelling of all available datasets is necessary to achieve the best possible interpretation of geological structure in any area'. This thinking appears to be the motivation behind the approach adopted by Unger et al [100] when a model of a portion of the Earth's crust is formed by bringing together information from six different data sets, namely, surface outcrop maps, migrated seismic reflection profiles, seismic refraction data, gravity models and magnetic models.

8.3 The Spatial Modelling of 3-D Objects.

Before proceeding, it is necessary at this stage to formally define the types of object

which are to be facilitated by the 3-D data model. It seems reasonable to assume that to be of greatest advantage to potential users (that is, geologists), the GSIS must be able to include any object type that is of interest to the user. Rhind [97] divides these objects into two categories, namely, those which are designed and those which are revealed. Designed objects include objects such as toxic waste disposal sites, quarries and buildings. In these cases the shape of the object is the product of workmanship based upon an original, man-made design. In contrast, revealed objects include all objects which have not been designed, their shape therefore having to be deduced from secondary evidence. Raper [92] has divided this group of object into two sub-groups. The first consists of all objects which can be regarded as sampling-limited. Discrete spatial entities, such as a perched aquifer, which may be progressively better defined by increased sampling, are included in this sub-group. The second sub-group consists of objects which can be regarded as definition-limited, that is, those which are identified by specification of particular threshold values of aspatial attributes. For example, the shape of an object defining a stratigraphic unit, as defined by the frequency of a particular micro-fossil, will change as the frequency threshold changes.

8.3.1 Modelling 3-D Objects.

Requicha [101] in a review of solid modelling techniques identifies six schemes which produce unambiguous definitions of solid objects. They are primitive instancing, constructive solid geometry, sweep representations, spatial occupancy enumeration, cell decomposition and boundary representations. Of these, only the latter three appear to be relevant to geological modelling [91]. Raper [92] gives an overview of the two approaches to a 'full 3-D spatial structuring of geo-objects'. These can be regarded as the 3-D equivalents of tessellated (raster) and vector structures. The tessellation solutions mentioned are the simple voxel models [102], the octree [96] and the polytree [103]. Vector data structure schemes included are those which use topological relations to define 3-D boundary representations for indexing geometric data (see 'simplicial complexes' of Carlson [104], 'structured vector fields' of Burns [105]); the 3-D definition of iso-surfaces by the 3-D interpolation between points (see IVM system [106]); and the spatial clustering of vectors defining the geological objects by a geometrical attribute in a geoscience database (see "Geokernal" [107]). Jones [91] presents a review of the conventional digital methods of representing geological structures. They are regular grids, surface patches, triangulations and block models. Also included in Jones [91] is detail of two other, at the time more novel, approaches, namely the octree and polytree. Two methods, suggested by several authors in the literature [91, 96, 108], seem to be of particular interest. They are the octree, an adaption of the spatial occupancy enumeration and cell decomposition methods, and boundary representations, where a solid is defined by the geometry of its bounding surface.

8.3.1.1 Boundary Representations.

A boundary representation represents a solid by dividing its boundary into a finite number of faces or patches. Each face is explicitly defined by its bounding edges which in turn are defined by their vertices. Free-form surfaces can be modelled by using spline functions to fit patches through the vertices of each polygonal face [101, 91, 96]. A boundary representation example is given in Figure 8.2.

This scheme is widely used in computer graphics due to its ability to cater for effective visualisation of complex objects (shading, hidden line and surface removal, light source simulation). Boundary representations can represent objects of any shape, but the validity of these objects is difficult to verify. With geological applications in mind, another problem presented is the difficulty in searching a given region in 3-D space. The technique is suitable for computing volumetric properties but Boolean operations are complex (quadratic complexity compared to linear complexity of the octree [109]). However, the exact geometry of the model does provide for accurate analysis. A TIN can be regarded as an example of a boundary representation scheme and seems a convenient technique for representing subsurface boundaries at multiple scale.

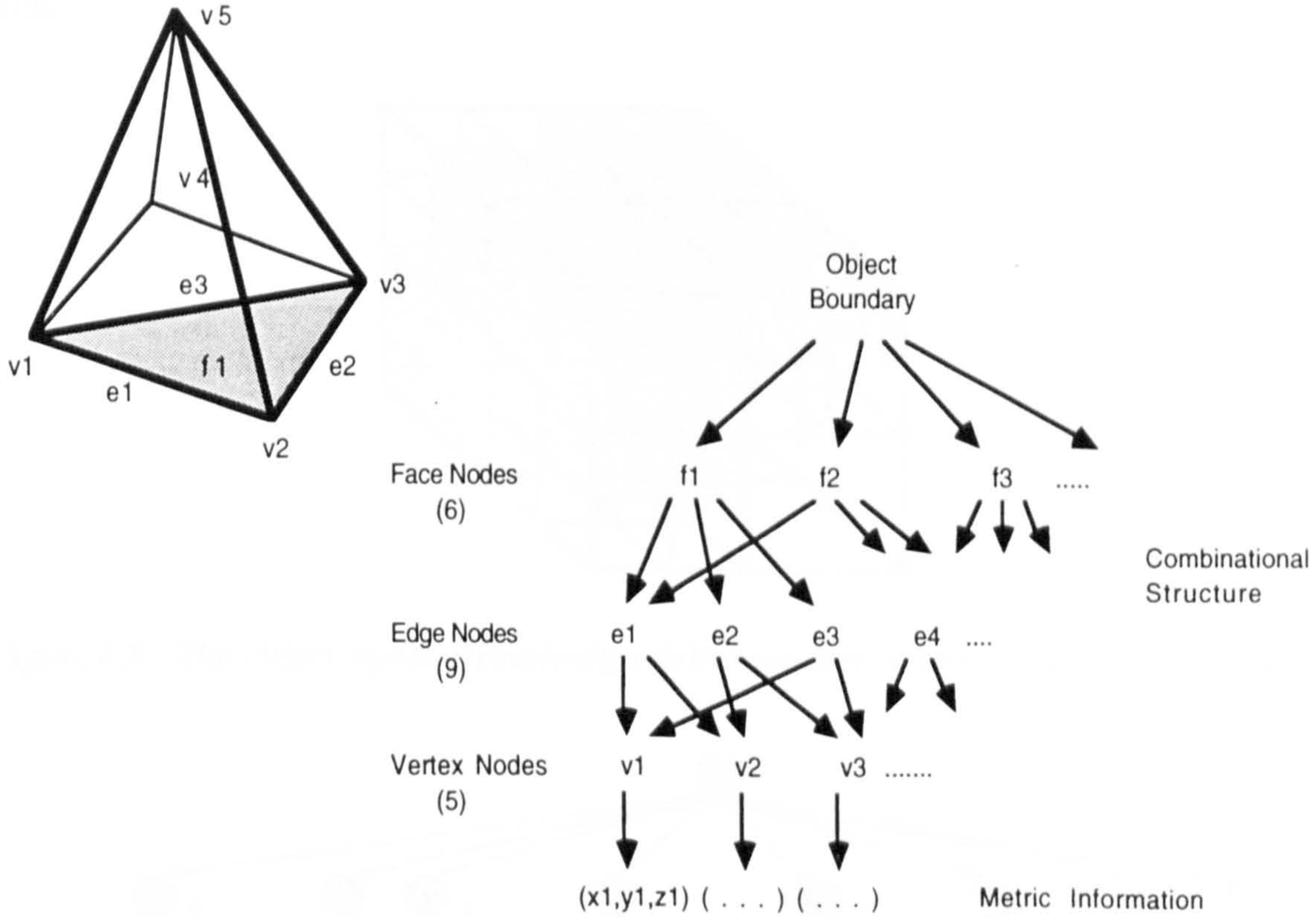


Figure 8.2 - A boundary representation for a rectangular pyramid. *(From Requicha [101]).*

8.3.1.2 The Octree.

The octree data structure can be regarded as the 3-D equivalent of the quadtree. It is built by the recursive subdivision of the object space (a cubic volume) into eight equally sized sub-cubes, or octants. Subdivision stops when a criterion of uniformity is met (Figure 8.3). The octree is stored explicitly as a tree data structure (Figure 8.4) where every non-terminal node, or GREY cell (except for those of minimal size), will have a pointer to each of its eight child nodes. An object can be reconstructed by traversing the tree and assembling all terminal nodes in the process.

Octrees have several distinct advantages over other solid modelling techniques (see [103], [91], [96]). Firstly, they can represent any arbitrarily shaped objects (convex, concave, interior holes) to the precision of the smallest cell. Secondly, Boolean operations are less complex than those for boundary representations. A third advantage is that geometrical properties, such as surface area, volume, centre of mass and interference can be calculated at different levels of precision. Also, they provide the basis for spatial sorting operations thus increasing search efficiency, they ensure that space is uniquely defined which is desirable since space cannot be occupied by two or more objects, and they can be used to represent the interior of non-homogeneous solids.

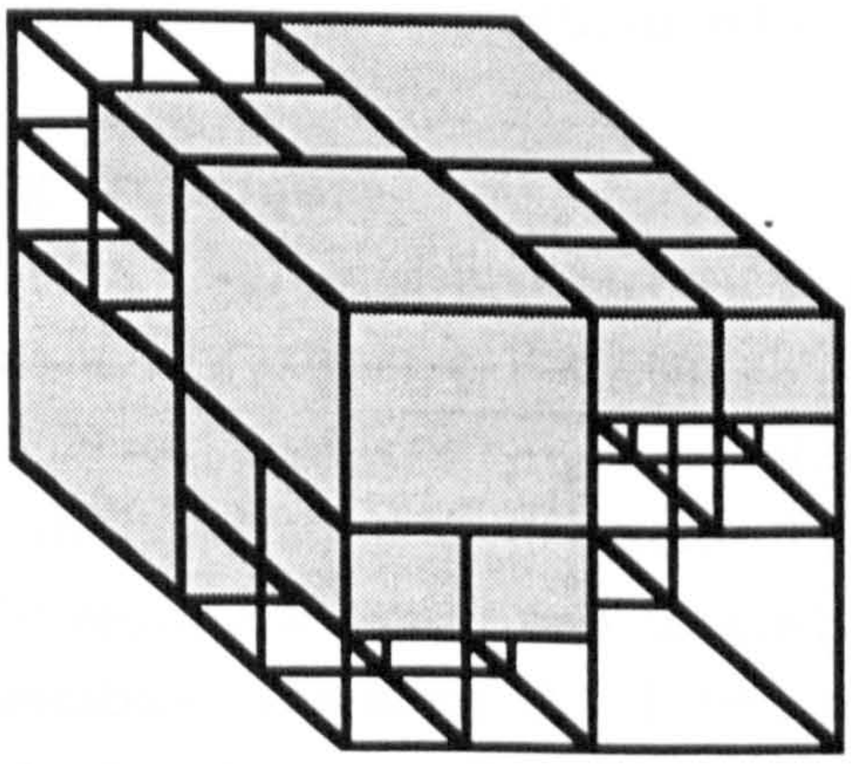


Figure 8.3 - The object-space recursively subdivided into octants to form an octree.

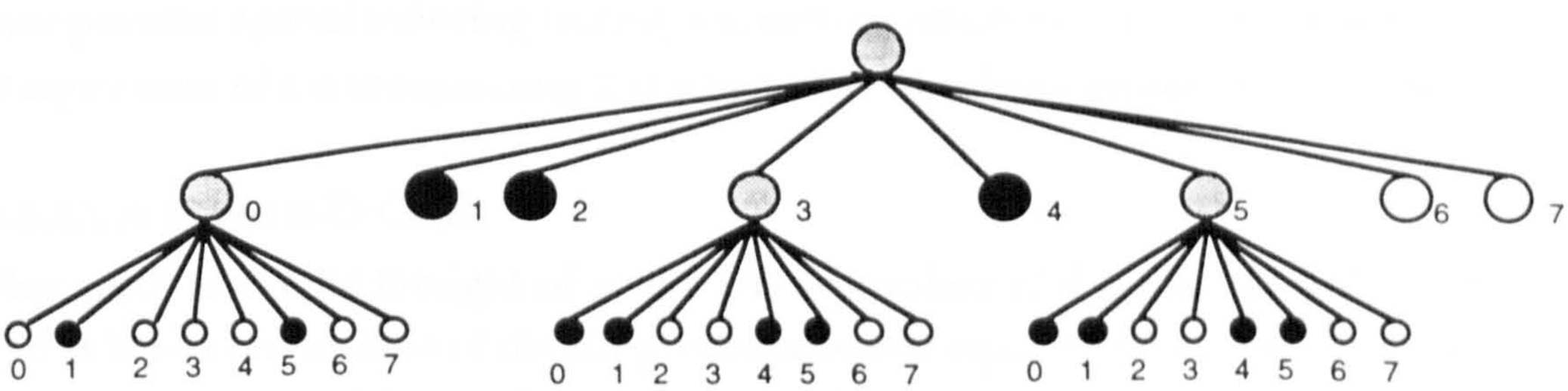


Figure 8.4 - The explicit tree corresponding to Figure 8.3.

However, octrees do incur some disadvantages (again see [103], [91], [96]). Firstly, the boundary surface of the object being modelled is represented by a set of square facets between empty and full cells. Therefore octrees provide only an approximation of the original surface, the accuracy of which is dependent on the maximum number of cells that can be stored. A second disadvantage is due to the storage of eight pointers for every non-terminal node which introduces considerable storage overheads.

The linear octree [110] is a data structure which represents all the basic properties of a regular octree, but with considerable savings in space. It achieves this by only storing terminal nodes, also referred to as leaf nodes. Each leaf node is assigned an unique storage address which is directly related to the nodes position in the octree. The address for each leaf node is generated using numbering systems known as tesseral addresses [37]. Bak and Mill [96], Gargantini [111], Jones [91] and Abel and Smith [72] give examples of such systems.

The regular and linear octrees described thus far are deficient in the fact that they represent only an approximation of the original object. However, there is no reason why nodes in such structures cannot store precise definitions of vertices, edges and faces. Similar assertions regarding quadtrees resulted in structures such as the PM trees of Samet and Webber [34]. The polytree [103] and the exact octree [109, 112], adaptions of the normal octree types, store exact representations of the object being modelled.

8.3.2 3-D Spatial Indexing Techniques.

As is the case with 2-D GIS, a GSIS must be expected to provide spatially specific access to data which may be areally extensive. For example, it may be that in the future an establishment such as BGS will possess a single, seamless 3-D geological database covering the whole of the UK. It is reasonable to assume that certain queries delivered to this database will only require a small, and spatially specific, portion of the information held by the database. Therefore, it will be necessary to provide spatial indexing of some sort on the 3-D database. Thus individual geological objects, which have been defined using methods of the sort described in the Section 8.3.1, will be retrieved on the basis of their location in xyz-space. This section will deal briefly with four possible spatial indexing techniques, each of which can be regarded as being the 3-D equivalent of a corresponding 2-D scheme previously described (see Chapter 3).

8.3.2.1 A Fixed 3-D Grid.

This method can be thought of as the 3-D equivalent of the fixed grid (Section 3.3.1) and is based on the idea of dividing xyz-space into equal-sized cubes. Thus a spatially extensive volume of data is divided into smaller, equal-sized sub-regions. One possible way of implementing this scheme would be for each cube to maintain a reference to each object which intersected that cube. A second possibility would be for each cell to

correspond to an area of storage (or memory) in which the intersecting objects are themselves stored.

8.3.2.2 The Octree.

The octree data structure has already been described in the context of its usefulness as a means of storing individual solid objects. For the purposes of a large GSIS database, a further application of the octree would be to use it purely as a spatial indexing scheme, with each octant keeping a record of any object which intersected it. Thus a 'universal' octree would provide fast spatially specific access to geological objects. Note that these objects could themselves be defined using the octree approach described in Section 8.3.1.2. This scheme can be regarded as the 3-D equivalent of the quadtree scheme described in Section 7.4.1.

8.3.2.3 The R-tree.

The description of the R-tree given in Chapter 3 restricted itself to a 2-D implementation. However, it should be noted that the R-tree is in fact a k-dimensional spatial indexing scheme and can therefore be used to index 3-D data. In this case each leaf node contains one or more record entries of the form

(J, object_id)

such that J is the smallest 3-D rectangle that spatially contains the data object pointed to by the identifier object_id. Non-leaf nodes contain entries of the form

(J, child_id)

where child_id points to a node in the next lower level of the R-tree and J is the bounding 3-D rectangle of all objects pointed to by the lower node entries. Initialising and accessing the 3-D implementation of an R-tree is analogous to that of a 2-D implementation, and as such will not be discussed here.

8.3.2.4 The Grid File.

As was the case with the R-tree, the grid file was originally designed as a k-dimensional spatial access data structure. Therefore, with regards a 3-D application, the grid file can be described as being based on the principle of dividing space into 3-D rectangles, which are not necessarily equal sized. The Grid Directory again consists of two parts, the first a dynamic 3-D (not 2-D) array containing one entry per 3-D grid block. The second part of the Grid Directory now consists of three (not two) linear scales, which define the size of 3-D cell in the x, y and z directions. Thus 3-D spatial searches are supported as a result of an initial search of the linear scales.

8.4 A Multi-Scale 3-D Model.

It is the intention here to develop the 2-D MTSM design into a 3-D multi-scale data model. This is seen as a natural development, since the MTSD appears to have certain characteristics which lend themselves to the modelling of 3-D geological structures. For example, the constrained Delaunay pyramid (CDP) seems to be a technique well suited to the multi-scale representation of subsurface horizons.

8.4.1 A Review of the 2-D Data Model

The MTSM provides efficient multi-scale storage of, and access to, what is essentially 2-D information. This information comprises of ground surface data and topographic object data (made up from polygons, lines and points) representing features which lie on the ground surface. These data are combined and processed to form a multi-scale data model. Efficient access to this model is provided by introducing spatial access data structures (in the form of fixed grids) to each level of the model. During model construction, levels of scale significance are assigned to individual data entities by applying a suitable generalisation algorithm (that is, either the Douglas-Peucker algorithm or De Floriani's error-directed point insertion algorithm).

8.4.2 Extending the 2-D Design into 3-D.

When designing the data structure for the multi-scale 3-D data model it is essential to have a clear understanding of what is being modelled. An ideal data model should be capable of including both revealed (sample-limited and definition-limited) and designed objects. Revealed sample-limited objects and designed objects are similar in that they are each, at all times, defined by a definite, discernible boundary. Definition-limited objects differ in that they can be regarded as being secondary objects, in which a definite boundary only comes into existence as a result of a particular query (and the processes resulting from the query) being applied to what can be regarded, in this instance, as primary data. It therefore follows that for the purposes of data model design, geological objects fall into one of two categories, which shall be termed hard-object and soft-object. Hard-objects correspond to revealed sample-limited and designed objects, and can be best catered for using a boundary representation approach. A soft-object can be thought of as a collection of related information from which definition-limited objects are derived. Soft-objects, in which data is characterised by the gradual variation of a particular property, can be represented using the octree method.

With the shift from 2-D to 3-D it is inevitable that more complex modelling routines will be required. This is due, in the most part, to the greater complexity involved when dealing with 3-D data, particularly of the geological kind. A particular problem is the difficulty in modelling multi-valued surfaces. Several solutions to the multi-valued

surface problem have been proposed [113, 114, 115]. However, these solutions are limited in that they either require dense data sampling or the provision of an initial, albeit coarse, input of surface topology. A further cause of greater complexity is the disparate nature of the many sources of geological information. Therefore the model must provide means by which all data sources can be integrated and processed to produce a correct interpretation of the geology it is seeking to represent.

The means by which geological information is generalised, and subsequently represented, also needs to be addressed. Generalisation of geological information can be divided into two categories, which will be termed micro-generalisation and macro-generalisation. Micro-generalisation concerns itself with individual objects and how they may be represented at various levels of generalisation. This could include, for example, the generalisation of a hard-object, defined using a boundary representation made up from triangles, by applying an adaption of the surface generalisation technique adopted by the MTSD systems. A second aspect of geological generalisation concerns itself with how collections of objects relate to each other at various scales. This type of generalisation, referred to here as macro-generalisation, is the process by which individual objects are themselves eliminated from a particular level of generalisation, or undergo some special type of merging with surrounding objects. Such generalisation is usually associated with a specifically adopted geological convention. For example, a series of horizons representing a sandstone formation at source-scale may be replaced by a single horizon at a smaller scale representation. Rules governing such behaviour are usually specific to particular geological agencies, and even within individual agencies conventions may differ from model to model (data site to data site). It is necessary to include in the multi-scale model provision for both micro and macro generalisation.

Spatial indexing can be provided by applying one of the methods described in Section 8.3.2. It is suggested that in keeping with the design of the MTSM that either the 3-D fixed grid or octree method be adopted. A comparison of the relative advantages and disadvantages of these two approaches would suggest that the octree method would offer the better overall performance. It is also necessary to decide on which information is to be spatially referenced. It is perhaps obvious that individual objects need to be indexed in this way, but less obvious perhaps is the way in which 'object parts' might benefit from such indexing. For example, an object representing a subsurface horizon might be made up from a collection of triangulation patches, which in turn are made up from a number of triangles. In this instance it may prove beneficial to spatially index surface patches, in addition to just objects, as it could be that the subsurface horizon itself is spatially extensive. Therefore it seems that a 3-D model in which individual objects may themselves be spatially extensive would benefit from an additional object part spatial index. A query would now firstly involve identifying all relevant objects,

then secondly, retrieving only relevant parts of these objects. Spatial indexing would also need to be separated into levels, as in the MTSM, since it is quite possible that different objects and object parts will be present at different levels.

8.4.3 A Prototype Multi-Scale 3-D Model.

A prototype multi-scale 3-D geological data model (MGM) is now presented, implementation details of which are given in Chapter 9. The prototype model, illustrated in Figure 8.5, provides efficient multi-scale storage of, and access to, the ground surface and subsurface horizons. Before the model is discussed in detail, it is first of all necessary to discuss the type of data that is to be modelled.

8.4.3.1 A Description of Data to be Modelled.

Three types of data have been made available from BGS, namely, ground surface data, outcrop object data and borehole (well) log data. The ground surface data is in the form of a list of irregularly distributed 3-D coordinates. The outcrop data consists of a list of outcrop objects, a list of polygon features, a list of line features and a list of point features. Each object is made up from a list of constituent parts, an object type identifier, an object description and a unique object identifier. Outcrop objects are of two types, that is, either region or fault. Region outcrop objects consist of a list of constituent polygon identifiers, fault outcrop objects consist of a list of constituent line identifiers. Polygon features are made up from a polygon identifier and a list of constituent line identifiers. A line feature consists of an unique identifier, a list of its constituent points and two integer values indicating which two region outcrop objects lie to its left and right. Each point feature is made up from an x and y coordinate, plus a unique identifier. The borehole data is made up from a number of borehole records. Each borehole record is made up of a header record, which stores the x, y and z location of the borehole at the ground surface, and a series of subsurface horizon depth measurements. Also included is a file containing a record of the order in which the subsurface horizons appear in the geological column. Specific subsurface horizons found in the data include the Lincolnshire Limestone (LLL) Formation, the Grantham (GRF) Formation and the Northamptonshire Sands (NS) Formation. In this thesis, when a particular horizon is referred to the convention adopted is that the horizon represents the base of the formation. For example, the LLL horizon refers to the base of the Lincolnshire Limestone Formation.

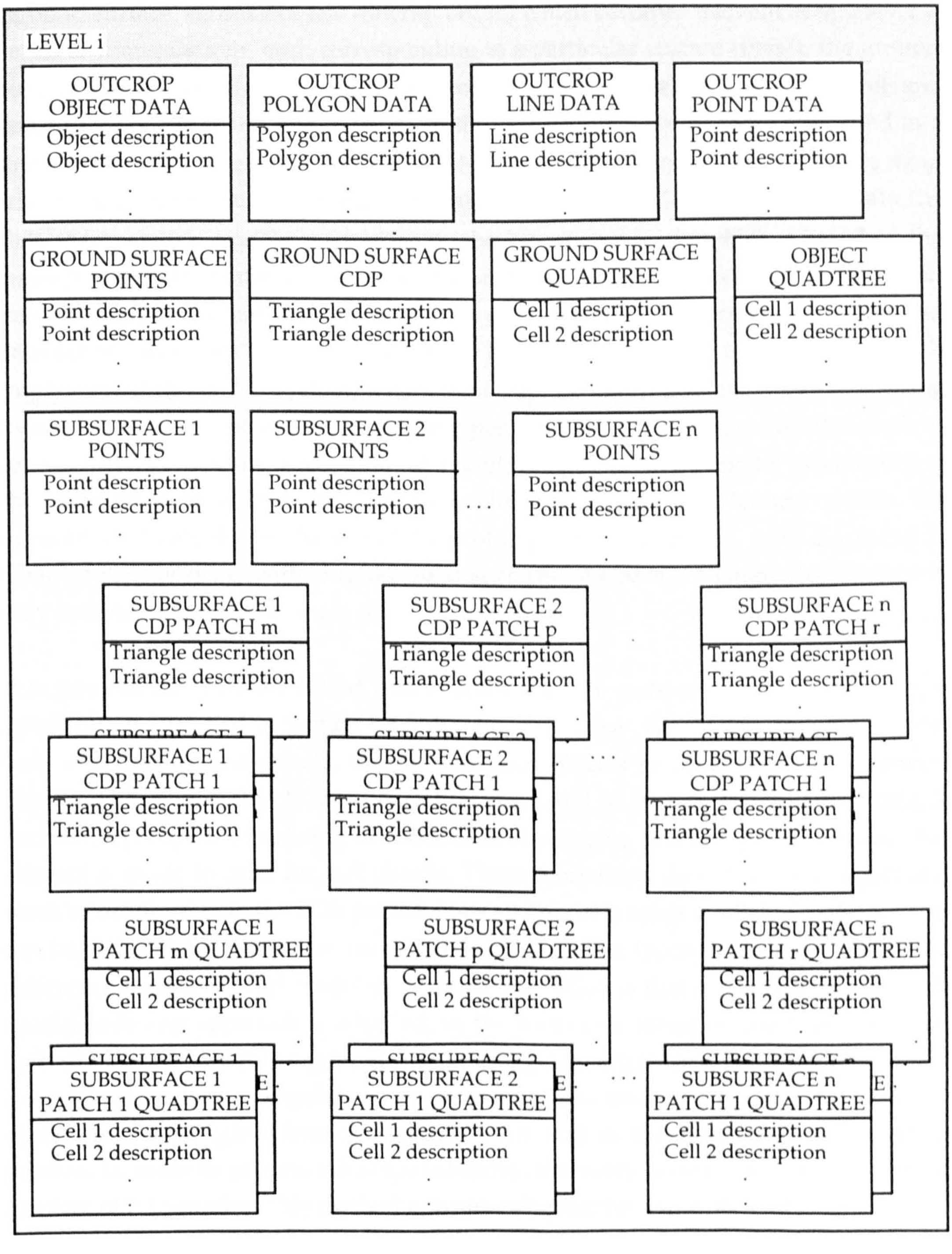


Figure 8.5 - A single level in the multi-scale geological model.

8.4.3.2 Model Description.

The prototype data model is capable of representing the ground surface and subsurface horizons at multiple scales. The model is divided into levels, each corresponding to a different resolution. Each level in the model will include details of the point data

(ground surface, subsurface and outcrop object) which becomes relevant at that level, a series of triangulations, each corresponding to a particular surface (that it, the ground surface or a subsurface horizon), the outcrop objects relating to the level and constituent polygon and line features. In the case of line features these are stored in a line generalisation tree format. Subsurface triangulations are made up from one or more triangulation patches. This particular structure is included in order to facilitate the data segmentation model construction method (Section 9.5.1.1), which is based on the concept of a multi-valued triangulated surface being made up from a number of single-valued triangulated patches. The structure of each ground surface triangulation and each set of subsurface triangulation patches follows closely to the triangulation design implemented in the 2-D system, where data duplication is minimised by introducing internal, boundary and external triangle types. Any known geological feature, such as an outcrop region or the intersection of a fault, is included in the model by embedding the isolated points, lines or polygons which represent that feature within the appropriate triangulation. Note that the prototype implementation, to be discussed in Chapter 9, only deals with single-valued surfaces. As such a triangulated surface is only ever made up from a single patch.

It is pointed out that the MGM differs from the full multi-scale 3-D model design specification (outlined in Section 8.4.3) in two main areas. Firstly, the prototype model only deals with hard-objects, that is, designed objects or revealed sample-limited objects. These hard-objects are themselves restricted to surface triangulation objects and outcrop objects (consisting of collections of polygon, line and point features). No attempt is made to cater for soft-objects. These limitations do not severely effect this work's contribution to the BGS project since all data sets made available to this project can be adequately catered for using hard-objects of the types described. The second difference between a full model design and the MGM is that a less complicated 2-D spatial indexing approach is adopted, in the form of a series of quadtrees. At each level of the model there is a ground surface triangle quadtree for referencing triangles in the ground surface triangulation, an outcrop object quadtree referencing outcrop objects and a triangle-referencing quadtree for each of the subsurface triangulation patches. In order to process a 3-D spatial query, the query must be broken down into a number of 2-D queries. This method appears adequate for use with the BGS data.

8.4.4 Model Creation.

Having described the prototype multi-scale 3-D model, it is now possible to give details as to how the model can be created. It should be noted that the methods described here assume that all surfaces are single-valued with regard to the xy-plane. The creation process is firstly described, for the sake of simplicity, in the context of a single-scale environment. A method for applying this process to the creation of a multi-scale model is then described. There are three main stages in the single-scale model

creation process. These are the triangulation of the ground surface data, triangulation of the subsurface horizon data and the inclusion of fault data.

8.4.4.1 Ground Surface Triangulation.

The first stage in the model creation process is the production of a ground surface triangulation. The surface is defined by the set of irregularly distributed terrain data and the collection of geological outcrop objects (outcrop regions and faults), which act as constraints upon the surface. The surface triangulation is created by applying a constrained Delaunay triangulation algorithm to the data. Initially, all terrain points and points forming part of geological outcrop objects are grouped together and Delaunay triangulated. This is followed by the process of inserting the line features, from which the geological outcrop objects are made, into the triangulation as a series of constraining line segments.

8.4.4.2 Triangulation of the Subsurface.

The second stage of model creation is that of subsurface triangulation. Before describing this process it is necessary to clarify what subsurface information is available. The data being modelled in the MGM includes two sources of subsurface information. The first, and perhaps most obvious source, is that obtained from the borehole logs. Each log consists of a header record, which stores the x, y and z location of the borehole at the ground surface, and a series of subsurface horizon depth measurements. These measurements record the vertical depth of each subsurface horizon relative to the ground surface. By combining the depth measurements with the ground surface level (recorded in the header record) it is possible to produce a series of 3-D points, each of which lies on a particular subsurface horizon. For the purposes of triangulation it is necessary to collate individual borehole information (that is, each 3-D point) on the basis of on which subsurface horizon it lies. This is achieved by collectively processing the borehole logs in such a way as to group together, in a single file, all 3-D points which relate to a particular horizon. Therefore each subsurface horizon will have associated with it a subsurface elevation file which contains a collection of irregularly distributed 3-D coordinates which describe that horizon.

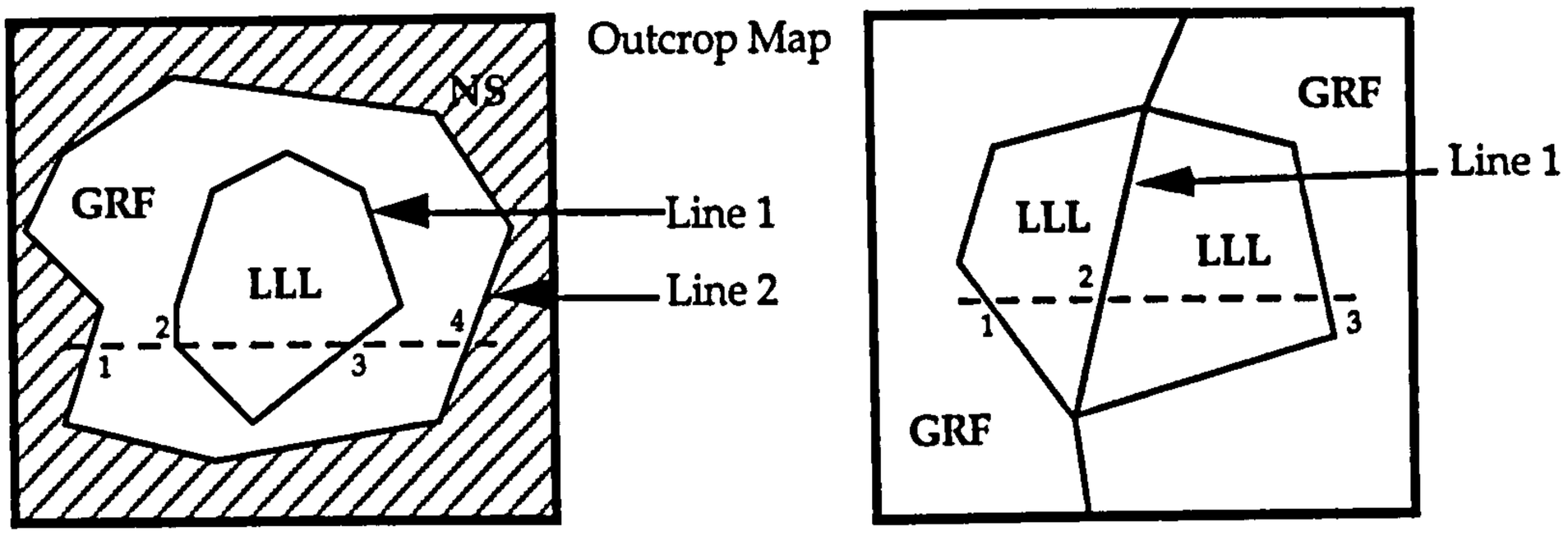


Figure 8.6 - Assigning outcrop boundaries to their correct subsurface horizon. (a) Line 1 assigned to Lincolnshire Limestone Formation, line 2 to Grantham Formation. (b) Line 1 is identified as a fault.

A second source of subsurface horizon information is provided by the geological outcrop boundaries. Each line feature from which a boundary is made-up may possibly be associated with a particular subsurface in the sense that it describes a series of points at which that subsurface outcrops at the ground surface. Hence the points defining the line can be regarded as forming part of the subsurface in question. Pairing an outcrop boundary to its correct subsurface is achieved as follows. Each line feature has associated with it two pointers, indicating the outcrop objects lying adjacent to it (that is, to its left and right). By examining these pointers and associated objects it is possible to derive the subsurface to which the line feature belongs. For example, if the two regions adjacent to the line feature are of different types, say LLL and GRF, then the line is assigned to the subsurface which appears higher in the geological sequence, in this case the LLL subsurface (Figure 8.6a). Note that a record of the geological sequence associated with the data set being modelled forms part of the source data set. Alternatively, if the line feature in question points to adjacent regions of the same type, LLL and LLL for example, then the explanation is that the line feature represents a fault, or part of a fault, and therefore is not assigned to a subsurface horizon (Figure 8.6b). The inclusion of faults within the model is dealt with at a later stage.

It is now possible to proceed with the triangulation of a particular subsurface horizon. The process, follows very closely to that of triangulating the ground surface data, but with an additional, final stage. Initially all points from the subsurface elevation file of the horizon to be triangulated are grouped together with the points defining the line features assigned to that subsurface. A Delaunay triangulation algorithm is then

applied to these points. Next, the line features are added into the triangulation as a series of constraints, producing a constrained Delaunay triangulation. It should be noted that these edges will each have a matching edge in the ground surface triangulation. The Delaunay triangulation algorithm assumes that the surface being triangulated is a 2-D plenum, that is, there are no internal holes or gaps within the triangulation. This is always the case when considering the ground surface, but not when considering a subsurface horizon. Therefore, it is sometimes the case that unwanted triangles are produced (Figure 8.7). The final stage is to delete these unwanted triangles from the triangulation. The unwanted triangles occur in areas where in reality the subsurface does not exist.

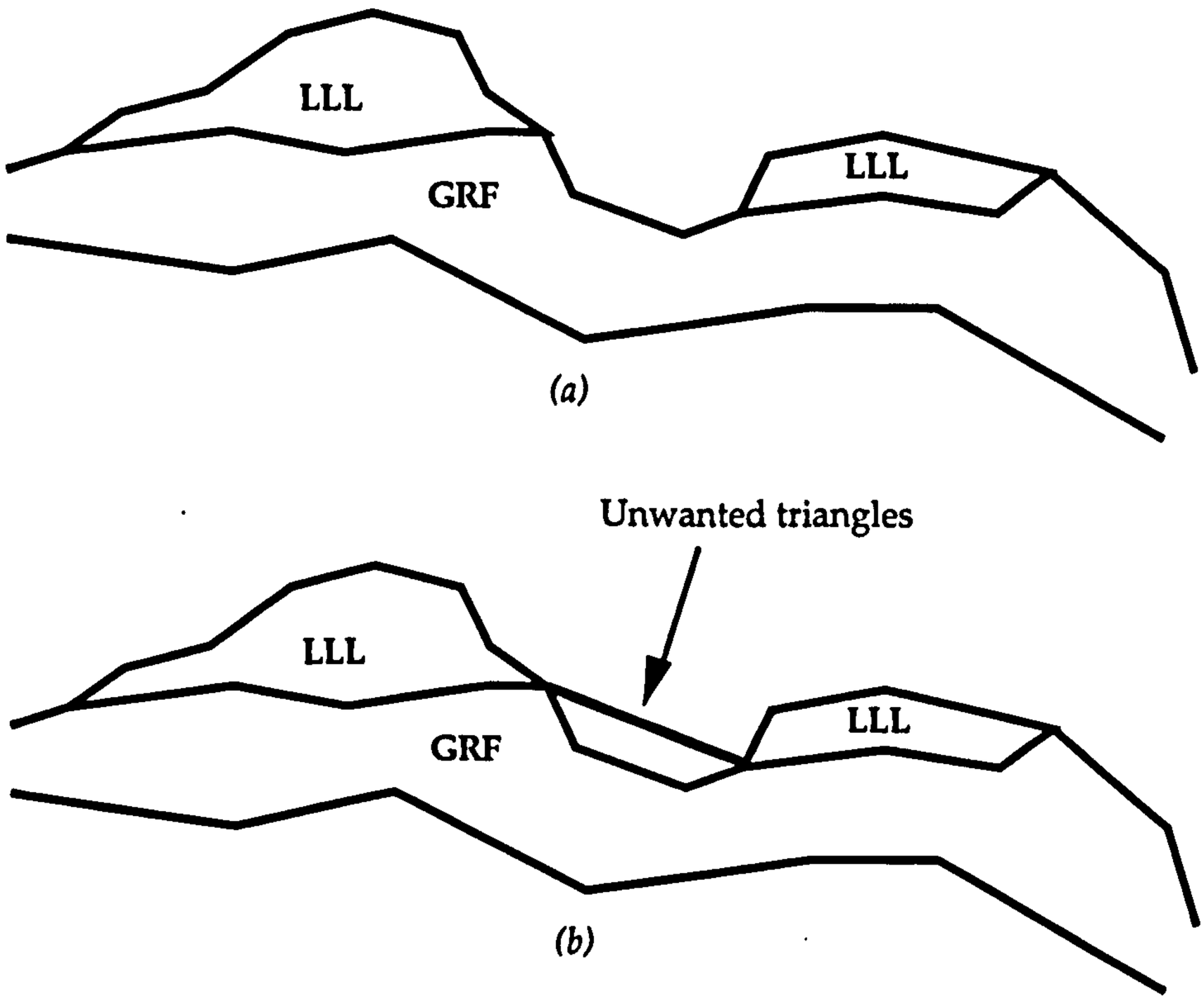


Figure 8.7 - Unwanted triangles created during triangulation of the subsurface. (a) A cross section of the correct geology. (b) The corresponding model cross-section where the LLL horizon has unwanted triangles.

In order to delete the unwanted triangles it is first of all necessary to identify them. When dealing with single-valued surfaces, as is the case here, this can be achieved in one of two ways. The first involves comparing each subsurface triangle with the ground surface triangulation. If a subsurface triangle is found to lie above the ground surface then a contradiction to what can be regarded as possible reality has occurred (that it, the subsurface cannot lie above the ground surface). Therefore, such a triangle can be marked as unwanted. The second method, which can only be applied when the subsurface is known to be single-valued, adopts the approach of comparing each

subsurface triangle with the geological outcrop regions obtained from the outcrop map. If the triangle lies in an outcrop region which is lower in the geological sequence than the subsurface the triangle belongs to, then it follows that the subsurface in question is not present at that location, and hence the triangle can be marked as unwanted (Figure 8.7). This second approach is the one adopted in the implementation described in Chapter 9. When all triangles have been processed, and marked accordingly, all those which are unwanted can be deleted from the triangulation. The process of subsurface triangulation is repeated for each of the horizons that are to be included in the model.

A further possibility for the prototype model, although not implemented in Chapter 9, is the formation of enclosed volumes by joining adjacent surfaces along common borders. Surfaces with no common border can be catered for by introducing dummy surfaces to connect upper and lower boundaries. This can be achieved using methods such as the shortest span technique of Christiansen and Sederberg [116]. Complex geological objects could also be represented as collections of relationships between constituent volumes and surfaces.

8.4.4.3 Including Faults in the Model.

Fault lines, obtained from the outcrop map, are already present as constraining objects within the ground surface triangulation. In order to provide a true representation of subsurface horizons it is also necessary, where appropriate, to include projections of these fault lines in the subsurface triangulations. The situation is that a fault has been identified, and recorded, on the ground surface. This fault may be assumed to affect certain of the subsurface horizons, the way in which it does depending on its depth, angle of dip and throw (Figure 8.8).

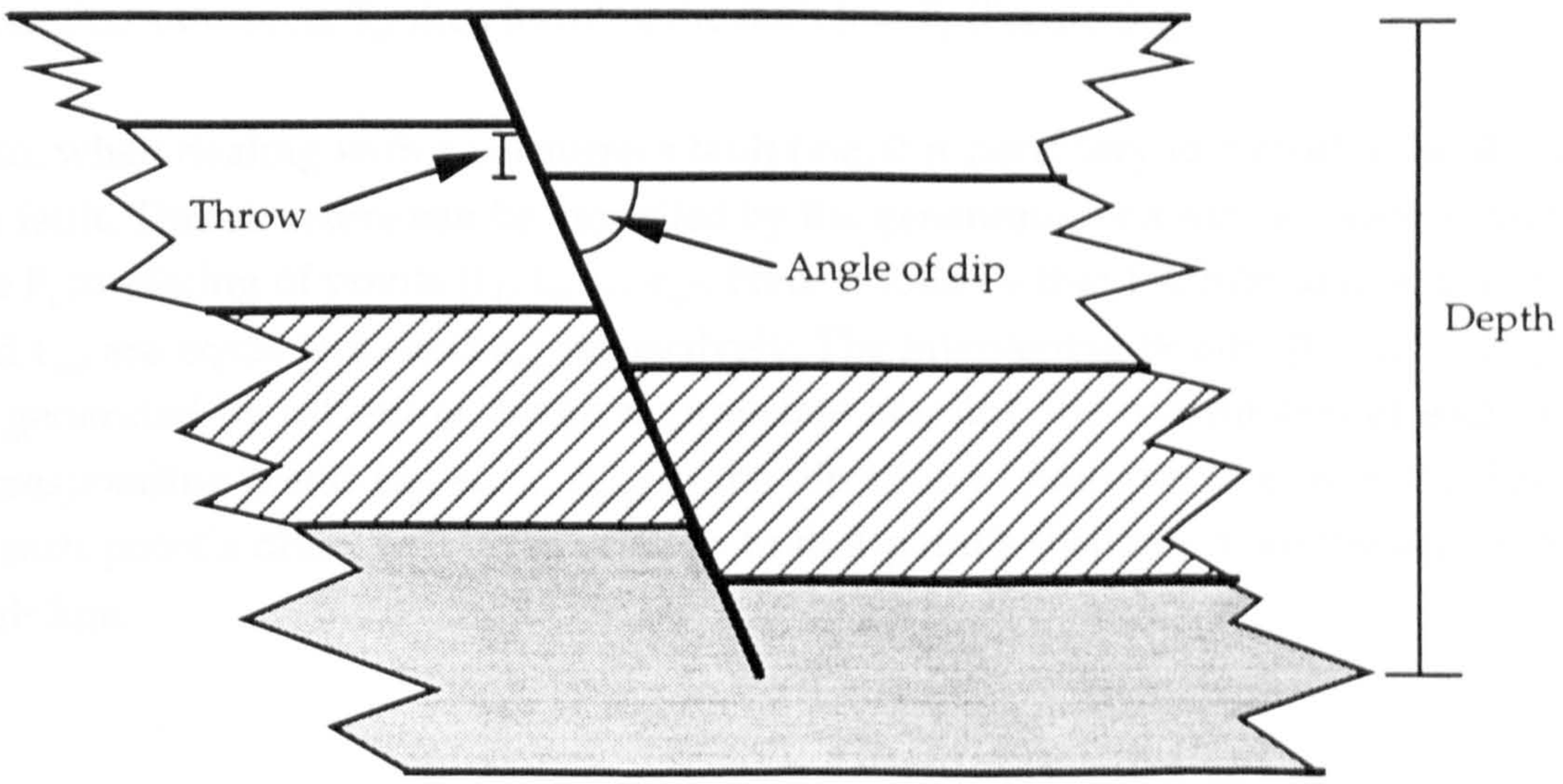


Figure 8.8 - The dip, throw and depth of a fault.

Consider a single fault line F_g in the ground surface triangulation T_g, and how it may affect a single subsurface horizon T_s. F_g is defined by a series of m 3-D points ($g_1, g_2, \ldots, g_m$). The first stage in the fault modelling process is to ascertain if the fault does actually affect the subsurface horizon in question.

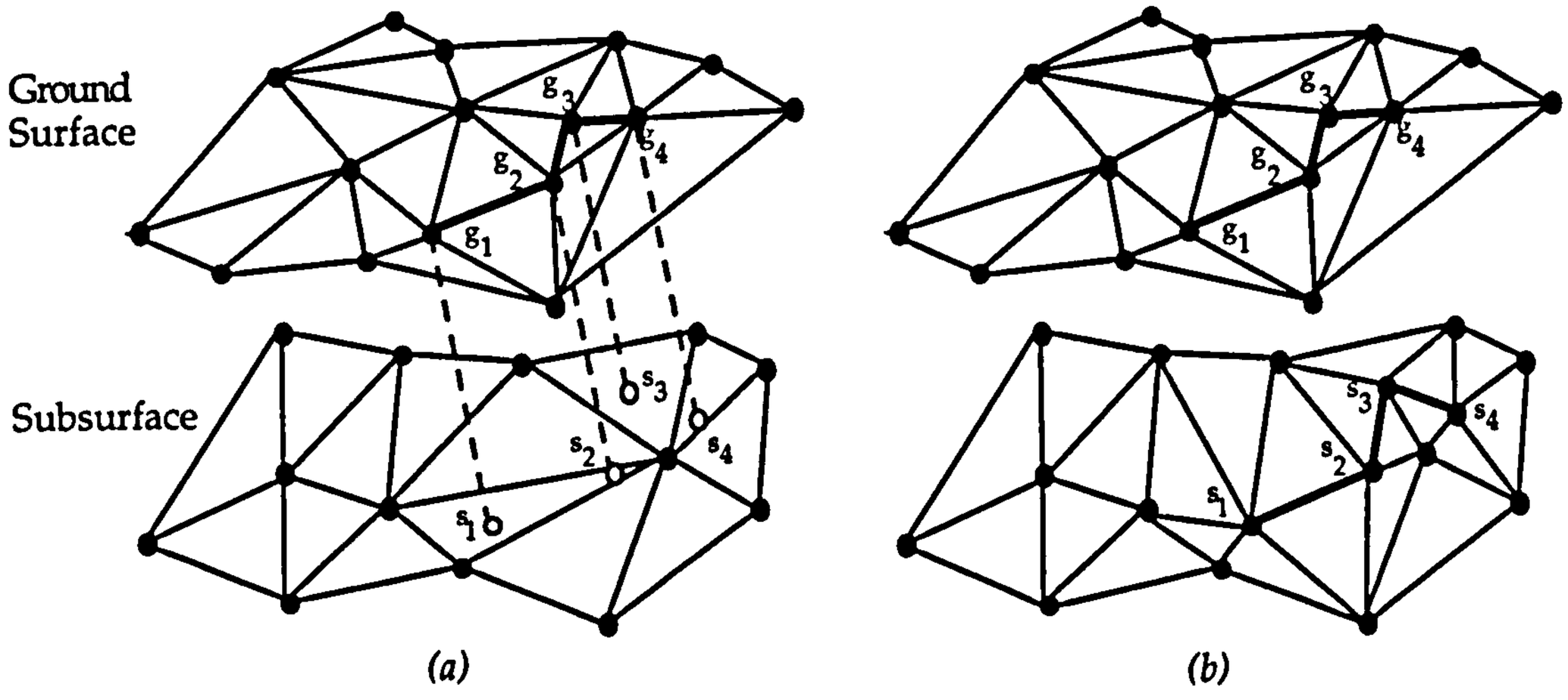


Figure 8.9 - Projecting a fault onto a subsurface. *(a) The ground surface fault is projected along path parallel to its angle of dip, generating points on subsurface. (b) The new points and fault line are added to subsurface triangulation.*

This is achieved by comparing the estimated depth d_f of the fault with the depth d_h of the horizon. If $d_f >= d_h$ then the fault can be regarded as being present in T_s. The second step, when necessary, is to generate a subsurface fault line F_s, consisting of m points ($s_1, s_2, \ldots, s_m$) which lies on T_s. This is achieved by projecting each point g_i of F_g along a path parallel to the angle of dip of F_g, recording the point s_i where the path of projection intersects T_s, thus forming the fault line F_s (Figure 8.9).

Also, when dealing with a subsurface fault line, it is necessary to consider the throw of the fault. This structure can be modelled by the generation of a second, additional fault line F_t consisting of points ($t_1, t_2, \ldots, t_m$). Here it follows that the first and last points, t_1 and t_m, are equal to s_1 and s_m, respectively. The intervening points, ($t_2, t_3, \ldots, t_{m-1}$) can be generated by offsetting, by an appropriate amount, the coordinates of each of the corresponding points ($s_2, s_3, \ldots, s_{m-1}$) in the direction of fault dip (Figure 8.10). The size of each point's offset will be in relation to that point's distance from the centre of the fault line.

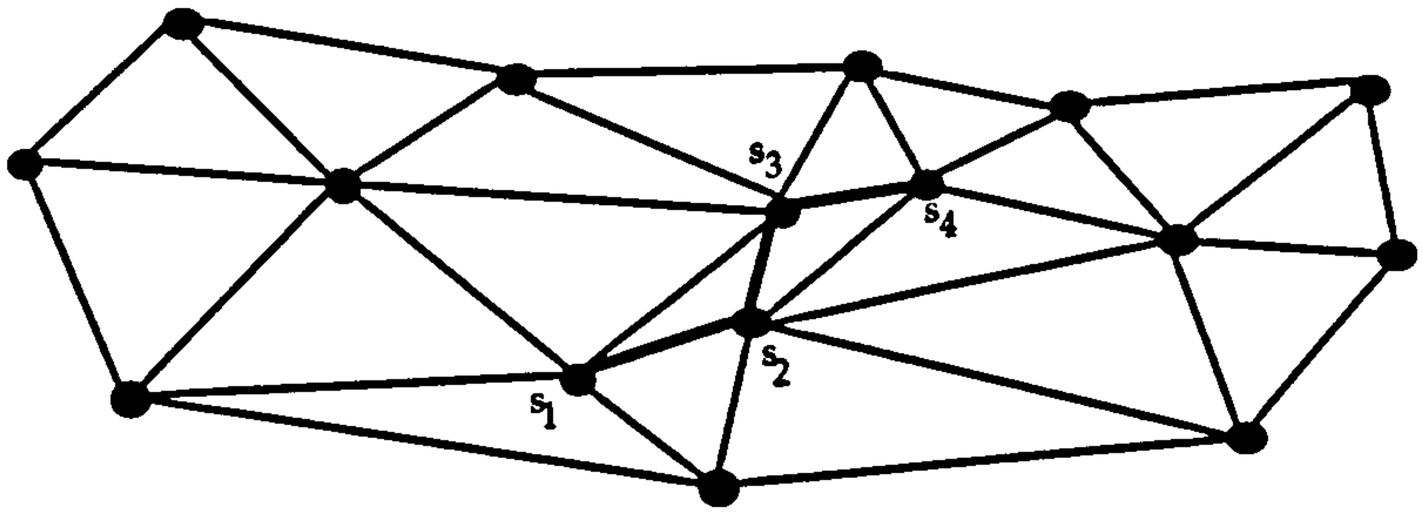


Figure 8.10 - Modelling the throw of a fault. (a) Triangulation before throw is inserted. (b) Triangulation after throw is inserted.

For this technique to be of use, the assumption is made that the depth, angle of dip and throw information pertaining to each fault is known. In the case of the BGS Grantham data these values have been estimated (following consultation with BGS geologists). However in some cases, particularly when it is the computer doing the modelling in the first instance (as opposed to a geologist using a computer to model his/her model), such information will not be available. Chapter 10 includes details of recent research which may be of help in automatically generating fault information data.

When all subsurface horizons have had the appropriate fault lines inserted, it is possible to further enhance the model by generating fault surfaces. This is achieved by triangulating between corresponding fault lines in adjacent surfaces, thus forming a set of complete fault representations, in the form of fault surface triangulations. This process has not yet been implemented in the system described in the next chapter.

8.4.4.4 Creating the Multi-Scale Model.

Each of the previously described 3-D model creation stages can be adapted to cater for the creation of a multi-scale 3-D model. The process of creating the model follows closely to that of building the MTSM, as described in Section 5.3. Consider a set of points S_g describing the ground surface, a set of objects O (and constituent polygon, line and point features) which act as constraints on the ground surface, and a series of sets of points, S_{s1}, S_{s2}....., S_{sn}, each describing a particular subsurface horizon. Now consider the steps involved in creating a k level multi-scale 3-D model from this data.

Each level i has two error tolerances, E_{vi} and E_{li}, associated with it, relating to vertical error and lateral error respectively.

The first stage is to simplify the outcrop objects O into k levels of generalisation. The generalisation is achieved, as in the MTSM, by applying the Douglas-Peucker algorithm, with error tolerances of E_{l1}, E_{l2},....E_{lk} at progressive levels, to each of the line features which make up the individual outcrop objects. Data duplication is minimised by storing each of the resulting generalised line features in a line generalisation tree. The next stage is to construct a k-level CDP, referred to here as CDP_g, from the points S_g and generalised objects O. This is achieved by applying the adapted CDP algorithm, described in Section 5.3, with a combination of the error tolerances E_{vi} and E_{li} governing which points are included at a particular level i. CDP_g will serve as the multi-scale representation of the ground surface. The third stage in the creation of the 3-D multi-scale model is to create a series of CDPs, CDP_{s1}, CDP_{s2},...., CDP_{sn}, corresponding to each of the n subsurface horizons. For a particular subsurface i, this involves, firstly, identifying which of the outcrop objects O_i of O are associated with that subsurface. This is achieved as described in Section 8.5.3.2. The adapted CDP algorithm is then applied to S_{si} and O_i, thus creating CDP_{si}. Note that if a particular horizon was made up from more than one triangulation patch there would have to be a CDP for each patch. In its application to subsurfaces, a minor alteration in how the CDP algorithm is applied is in the insistence that when objects are included in a subsurface pyramid, CDP_{si}, then the level at which their constituent points appear in the pyramid is governed by the level at which each point appears in CDP_g. This ensures that there is consistency between constraining edges within CDP_g and CDP_{si}. Unwanted triangles are deleted as described in Section 8.5.3.2. The final stage in the creation of the multi-scale 3-D model is the extrapolation of ground surface fault lines into the subsurface. This is achieved by applying the method described in 8.5.3.3 to each of the model's generalisation levels.

8.5 Summary and Conclusions.

This chapter has been concerned with the design of a multi-scale 3-D data model suited to geological applications. It has given a brief introduction to the subject of GSIS, particularly in the context of the 3-D Integrated Geoscience Mapping project currently being carried out by BGS. A review of the conventional methods used for representing and spatially referencing 3-D objects have been described, with the conclusion being that the boundary representation and octree techniques are well suited to representing geological structures. The main body of work in this chapter is contained in Section 8.4, which concerns itself with extending the MTSM (Chapter 5) into 3-D. Section 8.4.2 provides description of a proposed 'ideal' multi-scale 3-D geological data model, which includes two new geological data types, namely, hard-objects and soft-objects. Section 8.4.3 describes in detail a prototype multi-scale 3-D geological model (MGM)

and associated model construction algorithms. The MGM provides multi-scale storage of, and access to, the ground surface and subsurface horizons. This model, and associated construction algorithms, are believed to offer a novel, and very useful, means by which subsurface geology can be represented. The MGM has been used as the basis for a prototype multi-scale GSIS, details of which are given in Chapter 9.

Chapter 9

A Multi-Scale Geological Database

9.1 Introduction.

Chapter 8 included a description of a prototype data model (MGM) suited to the efficient multi-scale storage of 3-D geological data. The methods required to construct this model from source data were also discussed. The current chapter provides details of an ISAM database implementation of the MGM. Section 9.2 gives a description of the database and outlines the stages involved during database creation. Database retrieval and update is discussed in Section 9.3, the type and format of queries supported by the prototype system being explained. Details of system testing, with accompanying results, are then reported in Section 9.4. Section 9.5 devotes itself to a discussion of some of the limitations of the prototype database system, with special attention given to its inability to cope with multi-valued surfaces. Several methods are proposed as to how this particular limitation can be removed. Finally, Section 9.6 provides a chapter summary and conclusion.

9.2 Database Description and Creation.

A prototype database system, termed the multiresolution geological database (MGD), which is based on the MGM design, has been implemented in C on a SUN workstation. Advantage is taken of the reusability of many of the routines used in the MTSD systems. Data storage is again provided by use of the ISAM file handling library. Note that the MGD is limited to working with surfaces which are single-valued with regards to the xy-plane. There are two reasons for this restriction, the first being the fact that dealing with single-valued surfaces is much simpler than dealing with multi-valued surfaces. While the author admits that this reason was the primary factor for opting for a single-valued approach, a second reason was also taken into consideration. This concerned the fact that the subsurface data obtained from BGS was known beforehand to represent single-valued surfaces (this information was obtained from BGS). If this fact was not known, it might well have been the case that more effort would have been directed towards finding solutions to the creation of a multi-valued surface model. As it is, some thought has been given to this problem, a report of which is given in Section 9.5.1. An overview of the ISAM database architecture is given in Figure 9.1.

9.2.1 Primary Files.

The first stage in the database creation process is to load the BGS data (that is, the ground surface data, outcrop data and subsurface data), currently stored in flat (one data item per line), sequential (that is, not indexed) files, into the ISAM Primary Files (Figure 9.2). The ground surface data file, which consists of a list of 3-D coordinates, is loaded into the Primary Ground Surface Points File. When doing so each point is assigned an unique identifier (point_id).

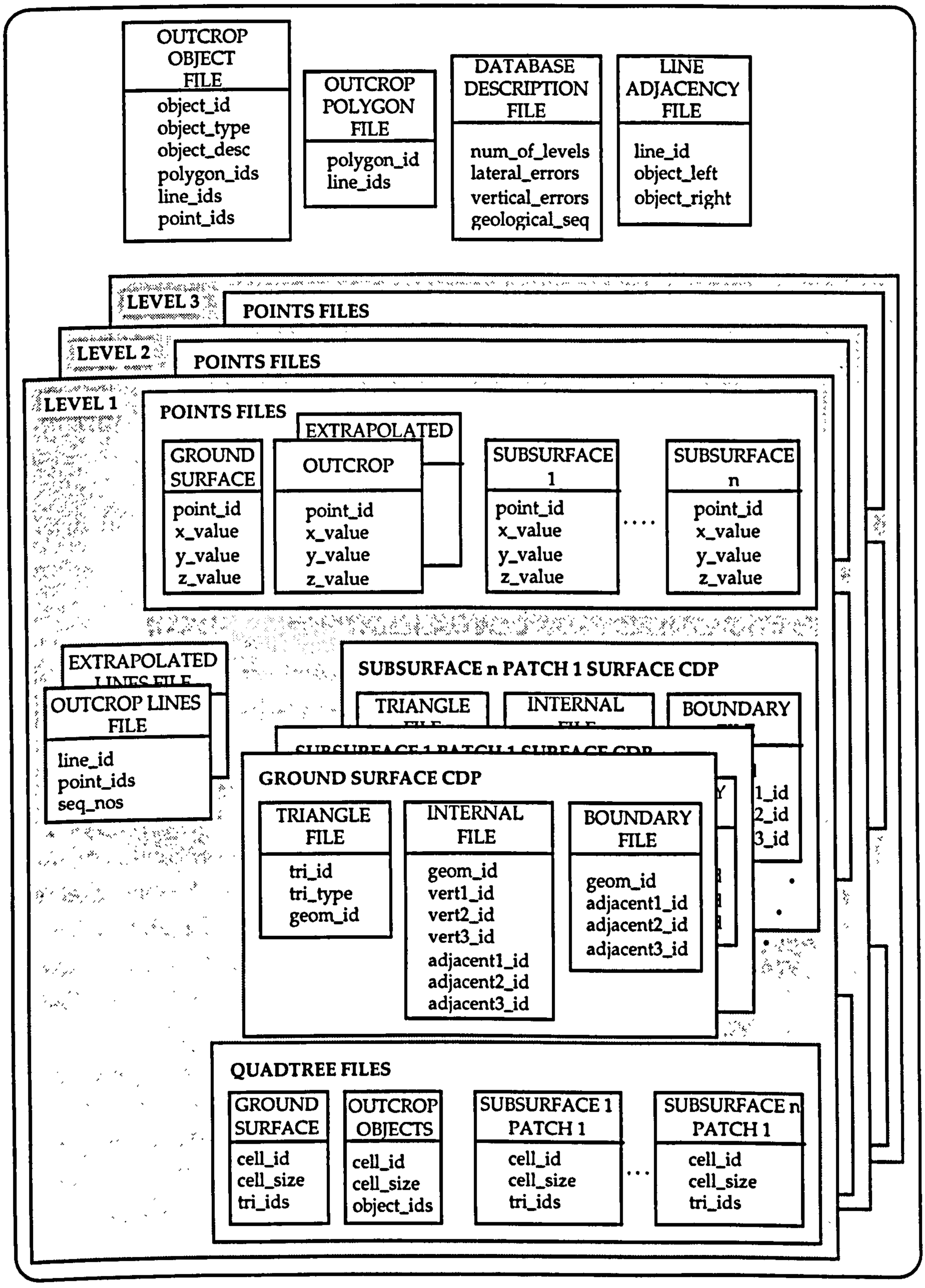


Figure 9.1 - The multiresolution geological database (3 levels).

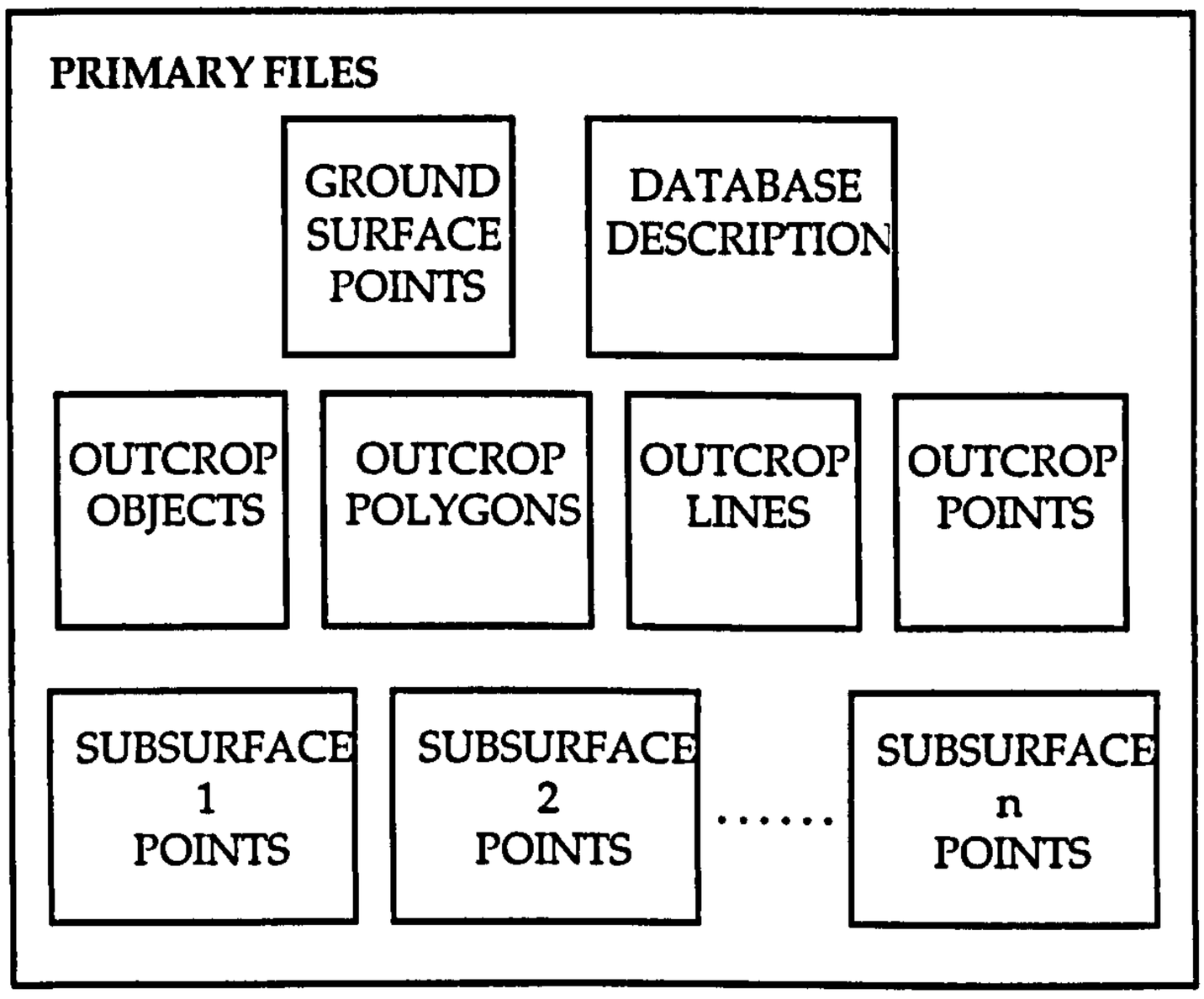


Figure 9.2 - The Primary Files.

The original BGS outcrop data is contained in four files, namely the object file, the polygon feature file, the line feature file and the point feature file. The object file consists of a series of object definitions. Each object definition is made up of an unique identifier, an object description (a character string), an object type identifier (region or fault) and a list of references to its constituent parts, details of which are held in either the polygon feature file, line feature file or point feature file. Each part reference is made up of two integer values, the first indicating what type of part is being referenced (that is, polygon, line or point) and the second giving the unique identifier of that part. The object data is loaded into the Primary Outcrop Objects File, each object reference of which consists of an object identifier (object_id), an object type identifier (object_type), an object description (object_desc) and lists of the polygon (polygon_ids), line (line_ids) and point (point_ids) features which make up that object. The polygon feature file is made up of a list of polygon definitions, each of which consists of an unique polygon identifier and references to the component line features (definitions of which are currently held in the line feature file). The polygon data is loaded into the Primary Outcrop Polygons File, each record of which stores a polygon identifier (polygon_id) and the list of line features (line_ids) from which the polygon is made up. In a similar fashion, the line feature file contains a series of line definitions, each made up of an unique line identifier and a list of references to constituent points. Each line definition also records the object identifiers of the two adjacent outcrop region objects. The line definitions are loaded into the Primary Outcrop Lines File, each line being defined by a line identifier (line_id), a list of component points (point_ids) and its adjacent outcrop region references (object_left and object_right). The point

feature file contains a list of 2-D points, each of which has an associated unique point identifier. This information is loaded into the Primary Outcrop Points File, each point being assigned a NULL z value during the process. It is noted that the lists stored in these ISAM files (polygon_ids, line_ids, point_ids) employ the chaining mechanism as described in Section 6.3.1.

Each of the n subsurface horizons to be represented in the model has a subsurface elevation file associated with it (see Section 8.5.3.2), the format of which is the same as that of the ground surface data file (that is, a list of 3-D coordinates). The data from each of these files is loaded into a corresponding Primary Subsurface Points File. During this process each point is assigned an unique point identifier (point_id).

During database construction, and its subsequent usage, it will often be necessary to retrieve information about a particular point, that is its coordinate values, given its unique identifier (point_id). However, at this stage there is no way of knowing in which primary file this information is held, that is to say, the point could be stored in the Primary Ground Surface Points File, the Primary Outcrop Points File or any one of the Primary Subsurface Points Files. Retrieval of information using this set-up will be prone to inefficiency due to the possible involvement of primary files which do not contain the required information. In order to overcome this problem, certain constraints are placed on the unique identifiers assigned to the points of a particular primary file. The method adopted is to insist that all points held in a particular primary file are given identifiers which lie within a given range, this range not being allowed to overlap with the ranges of other primary files. As an example, it might be decided that all points held in the Primary Ground Surface Points File are assigned identifiers in the range 1 - 1000, those in the Primary Outcrop Points File in the range 1001 - 2000, those stored in the Primary Subsurface 1 Points File 2001 - 2200, and so on. In this way it is possible to directly deduce the primary file in which the details pertaining to a particular point_id are stored.

It is also necessary at this stage to provide information for the Primary Database Description File. This file contains information concerning the number of levels of generalisation to be created (num_of_levels), and the lateral error (lateral_error) and vertical_error (vertical_error) associated with each of these levels. This file also stores the order in which subsurface horizons appear in the geological sequence.

9.2.2 Quadtree Initialisation.

The second stage of the database creation process is to initialise the quadtree files. Spatial indexing is provided on outcrop objects, ground surface triangles and the triangles of each subsurface triangulation patch. The spatial indexing is also separated into levels of generalisation. Therefore at each level of the database there is a Ground

Surface Quadtree File, an Outcrop Object Quadtree File, and a Subsurface Quadtree File for each subsurface triangulation patch (1 patch per subsurface). In each of the quadtree files a record consists of a cell identifier (cell_id), the size of the cell (cell_size) and a list of references to data items which intersect the cell. In the case of a Surface Quadtree these references will be to triangles (tri_ids), whereas the Object Quadtree File will reference outcrop objects (object_ids). The cell identifier is taken to be the Morton code of the bottom left-hand coordinate of the cell. This identifier is coupled with the cell size to record the location and extent of the cell. The maximum number of data item references per cell is set, arbitrarily, to 10, for all quadtree files.

At this stage of database creation no surface triangles for either the ground surface horizon or any of the subsurface horizons will have been created. Therefore, each of the corresponding quadtree files (at each level) are initialised as empty (that is, one cell, with no triangle references, covering the full areal extent of the data). Each of the Object Quadtree Files is also initialised to empty. This is because, at this stage, no object generalisation has taken place and thus the spatial extent of individual objects at specific scales is not know.

9.2.3 Generalisation of Outcrop Objects.

Stage three of the database creation process involves the generalisation of the outcrop objects held in the Primary Outcrop Objects File, and the storage of the resulting simplified objects. The MGD assumes that all outcrop objects, polygon features and line features are present at every level of the database. This is due to the absence of generalisation functions in which objects, polygons and lines are created and deleted. In the case of outcrop objects and polygon features the MGD also assumes that their constituent part descriptions do not change between levels. Therefore, there is need to store outcrop object and outcrop polygon descriptions only once, and this is catered for by the Outcrop Objects File and the Outcrop Polygons File (Figure 9.1). Each record in the Outcrop Objects File corresponds to a single outcrop object and consists of an object identifier (object_id), an object type identifier (object_type), an object description (object_desc) and lists of the polygon (polygon_ids), line (line_ids) and point (point_ids) features which make up that object. Each Outcrop Polygons File record refers to a particular polygon and consists of a polygon identifier (polygon_id) and the list of line features (line_ids) which make up the polygon.

Object generalisation in some measure is achieved by applying the Douglas-Peucker algorithm to each of the line features (held in the Primary Outcrop Lines File) from which an object is made up. For each line feature, the Douglas-Peucker algorithm is applied for each of the k levels of generalisation, with the appropriate error tolerance, obtained from the Database Description File (an exact copy of the Primary Database Description File), being applied in each case. The results obtained from the

generalisation of a single line feature are initially held in a main memory line generalisation tree. Permanent storage of these results is provided by the Outcrop Lines Files, of which there is one for each level in the database. Each Outcrop Lines File record can be thought of as a level in a line generalisation tree. An individual line feature (line_id) is described by a list of constituent points (point_ids), and a list of sequence numbers (seq_nos) indicating the position of each point in the original line description. The adjacent outcrop regions pointers for each line are stored in the Line Adjacency File, each record of which is made up of a line identifier (line_id) and references to the outcrop objects which lie to that line's left (object_left) and right (object_right). After all lines have been generalised it is possible to insert object references into each of the object quadtrees.

During line generalisation it becomes evident that some points are not required by the database, that is, they are not selected by the Douglas-Peucker algorithm at any level of significance. When a point is selected, it is stored in the Outcrop Points File at the appropriate level of the database, and deleted from the Primary Outcrop Points File. Each Outcrop Points File record refers to a single point and consists of the point's unique identifier (point_id) and its x, y and z coordinates (x_value, y_value and z_value).

9.2.4 Creation of Constrained Delaunay Pyramids.

The final stage in the database creation process is to create the constrained Delaunay pyramids needed for each surface represented in the model, that is, the ground surface and each subsurface. A method for constructing each of these pyramids has been described in Section 8.4.4. The database arrangement for storing a single pyramid is the same as that for the pyramid storage in the MTSD (see Section 6.2.1). Consider the storage of the ground surface pyramid, CDP_g. Three database files are required per level to store this pyramid, namely, the Ground Triangle File, the Ground Internal File and the Ground Boundary File. Each Ground Triangle File holds details of the triangles which exist at its particular level in the pyramid. Each record in the file corresponds to a triangle and consists of a triangle identification number (tri_id), a flag (tri_type) indicating the triangle type (0, 1 or 2 corresponding to internal, boundary or external) and, where required, a pointer (geom_id) to the appropriate record in either the Ground Internal or Ground Boundary File. If a triangle exists at more than one level, in the form of a boundary or external triangle in the lower level, it will have the same tri_id at each level. The Ground Internal File holds the full geometry and adjacency information for internal triangles. The Ground Boundary File holds the adjacency information for boundary triangles, the vertices of which are found by obtaining details from a higher level Ground Triangles File. There is no need to have a Ground External File since adjacency and geometry information of external triangles is found by retrieving details from a higher level. In a similar fashion, each of the subsurface pyramids require three

files per database level. For a particular subsurface, i, these files will be Subsurface i Patch 1 Triangle File, Subsurface i Patch 1 Internal File and Subsurface i Patch 1 Boundary File. During the creation of pyramids, points are included at a particular level according to either their contribution to the form of a surface or their level assigned by the Douglas-Peucker algorithm. Whenever a point is chosen to be included in a pyramid it is added to the appropriate points file and deleted from its primary file. After each pyramid is created the appropriate quadtree files are updated.

During the creation of the subsurface pyramids fault objects are extrapolated onto subsurface triangulations, creating extrapolated fault objects (Section 8.4.4.3). These objects are always made up of extrapolated line features, which are themselves made up from extrapolated points. The extrapolated line features are stored in the Extrapolated Lines Files, of which there is one for each level in the database. The format of these files is the same as that of the Outcrop Lines Files. Extrapolated points are stored in the Extrapolated Points Files, the format of which corresponds to that of the Outcrop Points Files.

9.3 Database Retrieval and Update.

A number of basic database retrieval operations are included as part of the prototype system. These operations allow for the retrieval of the ground surface, outcrop objects and each of the subsurface horizons at different levels of detail and for particular areas of interest. Three parameters are needed to define a specific query. The first parameter indicates what type of information is to be retrieved. The information type is described in terms of an integer code value, with -1 referring to outcrop objects, 0 to the ground surface, and the values 1, 2,....., n corresponding to each of the n subsurface horizons. In the case of the BGS data the values 1, 2 and 3 are assigned to the LLL Formation, the GRF Formation and the NS Formation respectively. The second query parameter specifies at which level of detail the information is required and is defined by an integer value corresponding to the required level. The third parameter defines the area of interest over which information is required. This area is defined in terms of a bounding rectangle which is described by two x,y coordinate pairs, the first corresponding to the bottom left hand corner of the bounding rectangle, the second to the top right hand corner. As an example, consider a query to retrieve subsurface horizon GRF at detail level 3, the area of interest being a 1000m square region with bottom left hand corner coordinates (40000, 20000). The corresponding MGD query would be (2, 3, 40000, 20000, 41000, 21000).

No database update operations are provided by the prototype system. However, it is pointed out that the underlying data structures on which the database is based (that is, the CDP, the line generalisation tree and the quadtree) in their original form allow for update. It therefore follows that the MGD will lend itself to the future inclusion of

update operations. Note also that the point and edge insertion algorithms used in the creation of the CDPs are themselves dynamic and could readily be adapted to facilitate the insertion of new data.

9.4 Testing the MGD System.

The MGD system has been used to model terrain, outcrop and borehole data supplied by BGS. The test data lies within the same 2km x 2km region as that described in Section 6.2.4. The terrain consists of 380 points, the distribution of which is shown in Plate 9.1. The outcrop data, illustrated in Plate 9.2, consists of 20 objects made up from a total of 20 polygons and 143 lines. The objects represent geological outcrop regions and geological faults. There are 81 boreholes situated within the test site (Plate 9.3), which provide evidence of 3 subsurface horizons, namely, the LLL (Lincolnshire Limestone) Formation, the GRF (Grantham) Formation and the NS (Northamptonshire Sands) Formation. When processed as described in Section 8.4.4.2 the borehole data gives rise to 3 subsurface elevation files, corresponding to each subsurface horizon. The LLL Formation is described by 54 points, the GRF Formation by 80 points and the NS Formation by 78 points.

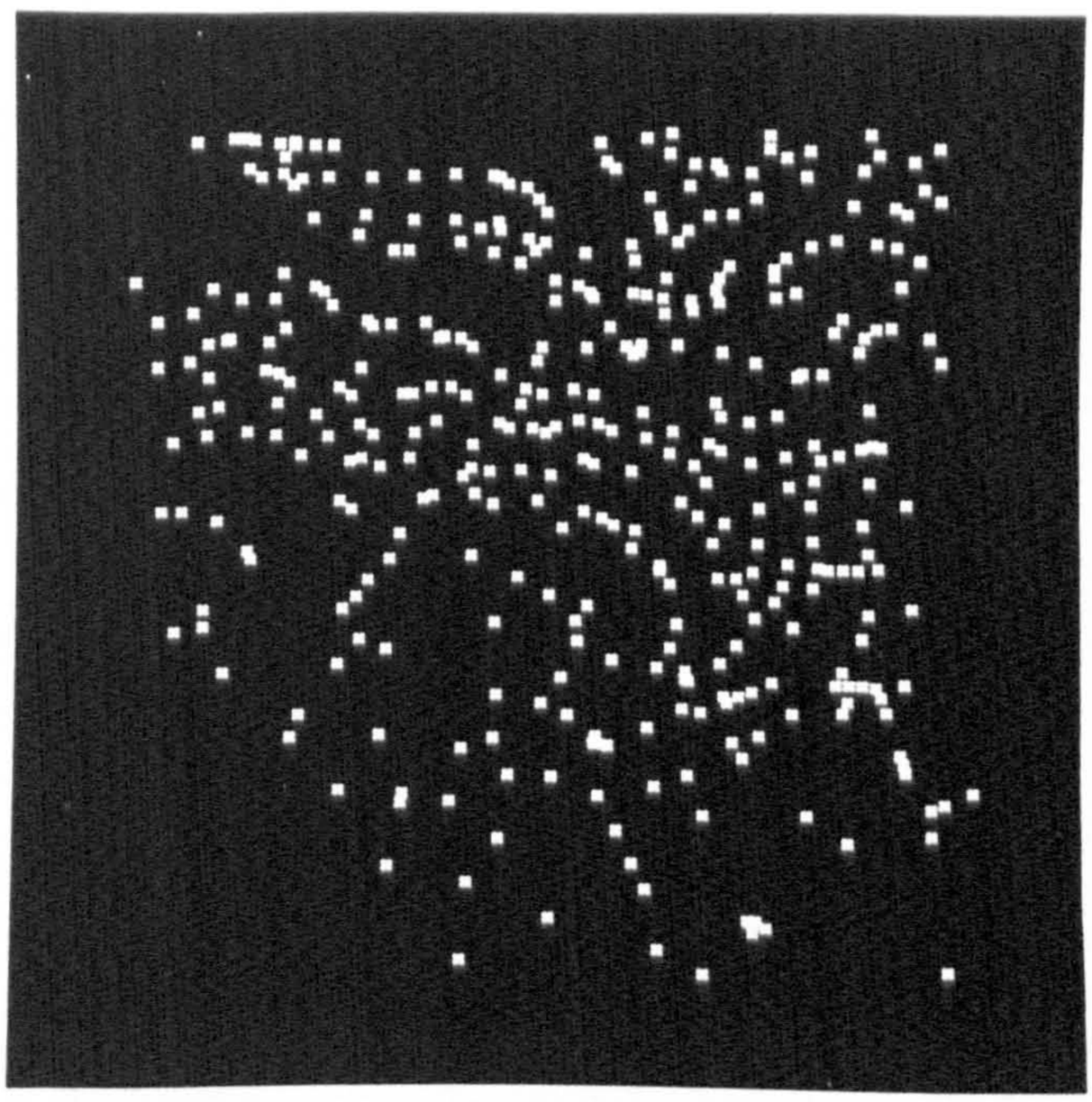


Plate 9.1 - The distribution of terrain data (380 points).

Plate 9.2 - The outcrop data, consisting of 20 objects (13 outcrop regions and 7 faults).

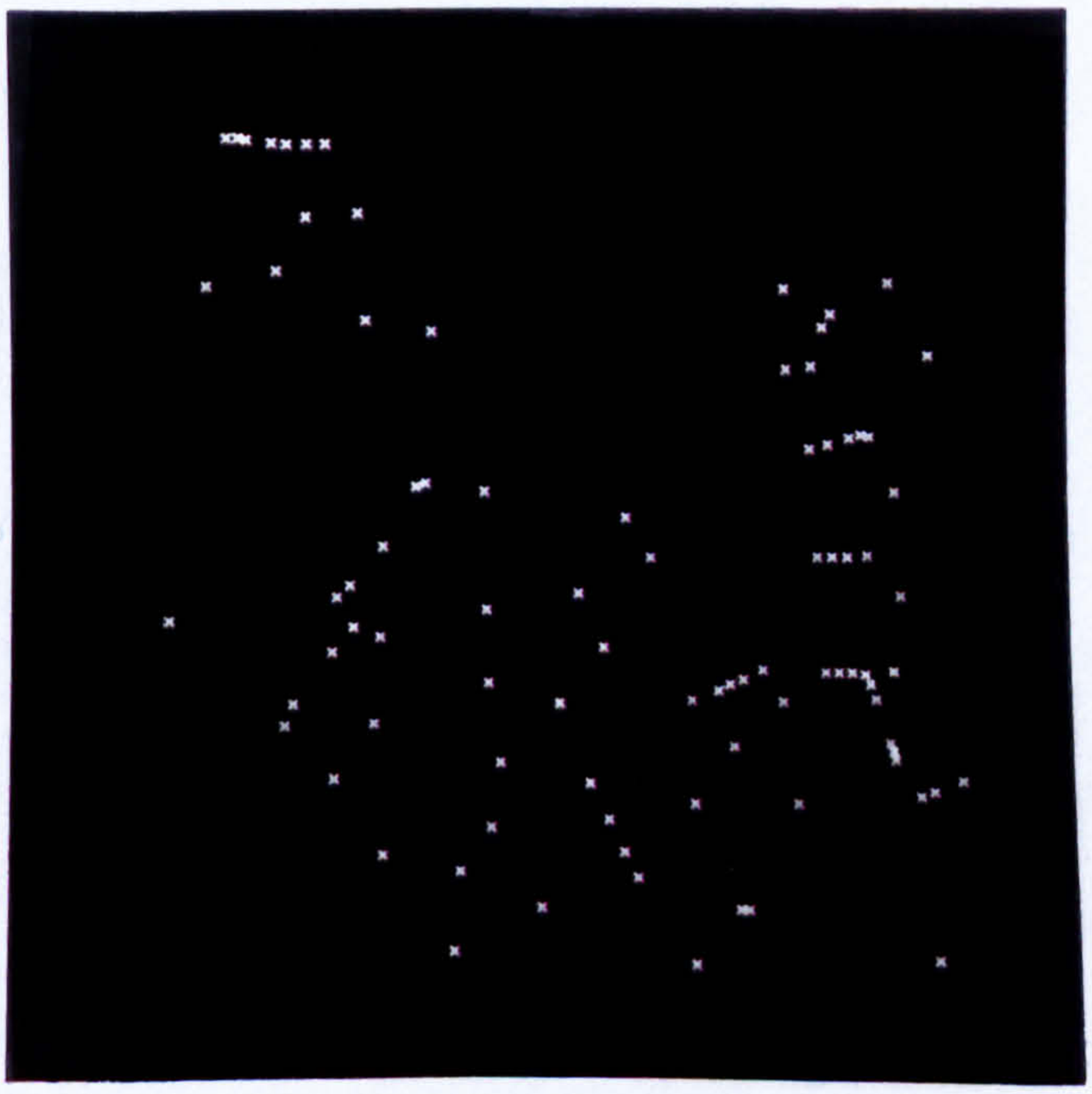


Plate 9.3 - The distribution of boreholes (81).

A series of test databases have been created with various error tolerances applied in each case. The results of database creation performance tests and database comparison tests, similar to those described in Chapter 6, are shown in Figure 9.3 and Figure 9.4 respectively. Database creation time appears satisfactory, as was the case with the 2-D MTSD implementations described in Chapters 6 and 7. The comparison tests again highlight the relative merits and demerits of the multi-scale approach. Storage savings gained when compared to multiple representation are at the cost of an increase in query response time, while when compared to a generalisation at run-time approach, reduced response time is countered by an increase in storage.

Database Name	Number of levels	Vertical Error(m)	Lateral Error (m)	Number of terrain points	Number of topographic points	Number of edges	Number of borehole points per subsurface			Creation time (s)
							LLL	GRF	NS	
1	3	30 0	5 0	168	216	227	49	71	70	288.0
		12.5	2 5	48	90	317	50	71	71	
		1 0	1 0	148	198	515	54	75	74	
2	3	35 0	8 5	158	157	161	49	71	69	316.0
		10.5	2 5	77	129	317	50	73	71	
		1 0	1 0	129	198	515	54	75	74	
3	3	40 0	10 0	150	138	150	48	70	68	341.0
		7.5	3.0	105	140	289	52	72	71	
		0.5	0.5	114	448	737	54	79	76	

Figure 9.3 - Database creation performance results for MGD.

Method	Number of levels	Vertical Error (m)	Lateral Error (m)	Storage used (K-bytes)	Time taken to retrieve all triangles (s)	Time taken to retrieve all objects (s)	Time taken to retrieve all LLL triangles (s)	Time taken to retrieve all GRF triangles (s)	Time taken to retrieve all NS triangles (s)
MGD	3	30 0	5.0	548	16 0	12 0	10 0	10 0	10 0
		12 5	2.5		21 0	16 0	10.0	11.0	11.0
		1 0	1.0		30 0	18 0	11 0	11 0	12 0
Generalisation at run-time.	1	30 0	5.0	290	81 0	32 0	70.0	74 0	72.0
		12 5	2.5		103 0	38 0	76 0	76 0	75 0
		1 0	1.0		132 0	49 0	78.0	81.0	82.0
Multiple representation	3	30 0	5 0	813	13 0	9 0	8 0	9.0	10 0
		12 5	2.5		17 0	10 0	8.0	9.0	10.0
		1.0	1.0		18.0	12.5	9.0	10.0	11.0
MGD	3	35 0	8 5	521	16 0	10 0	10 0	11 0	10 0
		10.5	2 5		24 0	14 0	11 0	11 0	11.0
		1 0	1 0		31 0	24 0	11 0	12 0	11 0
Generalisation at run-time.	1	35.0	8 5	290	94 0	33 0	70 0	71 0	76.0
		10 5	2.5		125 0	38 0	77.0	77 0	79 0
		1.0	1 0		157 0	47.0	82 0	84 0	83.0
Multiple representation	3	35 0	8 5	793	12 0	8 0	9 0	9 0	10 0
		10.5	2 5		15 0	12 0	9 0	9 0	11.0
		1.0	1.0		18.0	15.5	10.0	10.0	11.0
MGD	3	40 0	10 0	568	13 0	9 0	9 0	10 0	11 0
		7 5	3 0		24 0	18 0	11.0	10 0	11 0
		0 5	0 5		39 0	20 0	13 0	12 0	12 0
Generalisation at run-time.	1	40 0	10.0	290	94 0	31 0	71.0	73 0	75 0
		7.5	3 0		141 0	36 0	78 0	74 0	79 0
		0.5	0.5		204 0	50 0	85.0	86 0	89 0
Multiple representation	3	40 0	10 0	821	11 0	11 0	8 0	8 0	9 0
		7 5	3.0		15 0	13 5	8.0	10 0	10 0
		0.5	0.5		18.0	19.5	11.0	12.0	12.0

Figure 9.4 - Results of comparison tests between MGD, generalisation at run-time and multiple representation.

Database 3 (levels 1 and 3) is illustrated in Plates 9.4 - 9.23.

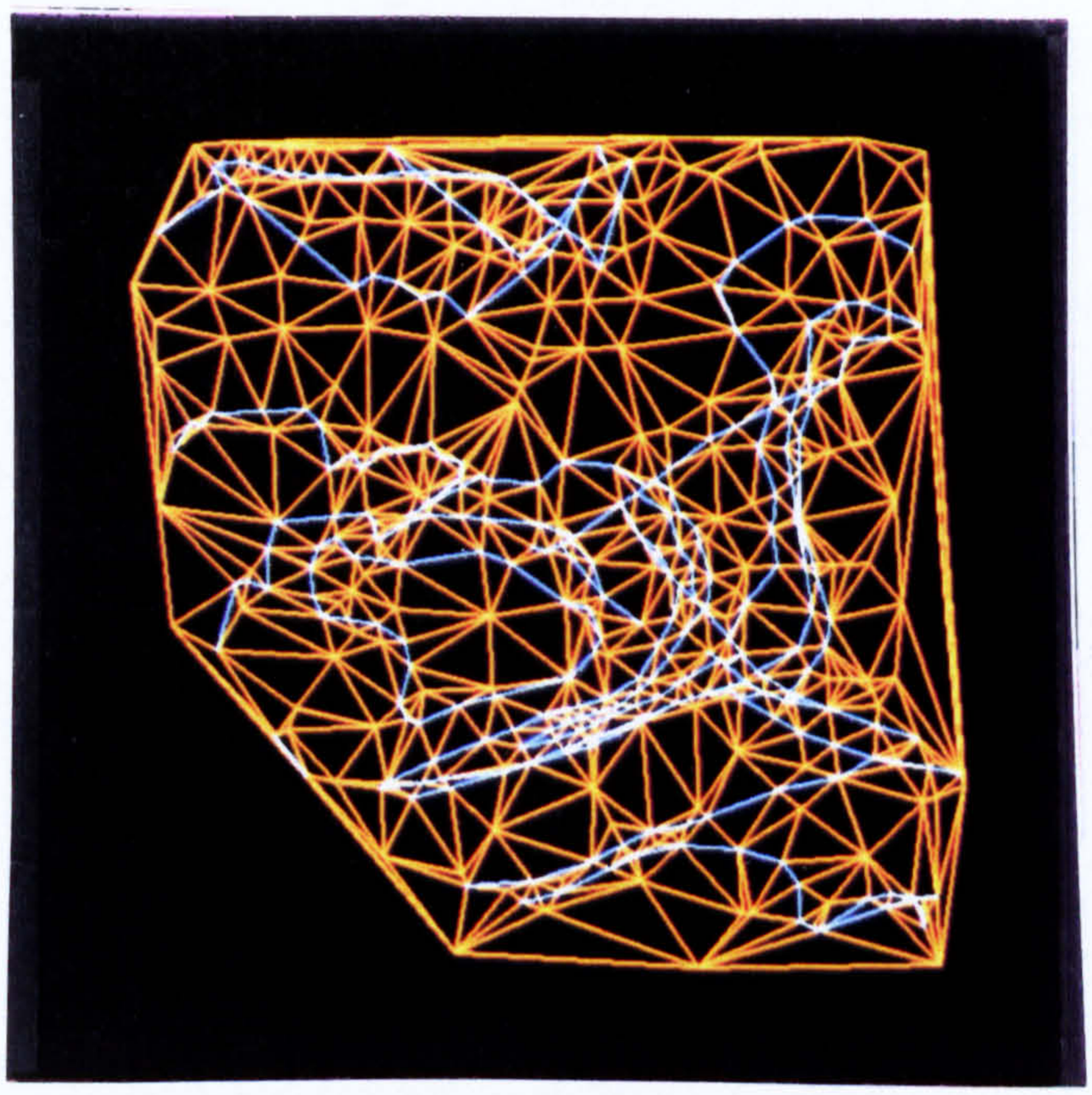


Plate 9.4 - Database 3, level 1 ground surface triangulation. *Plan view.*

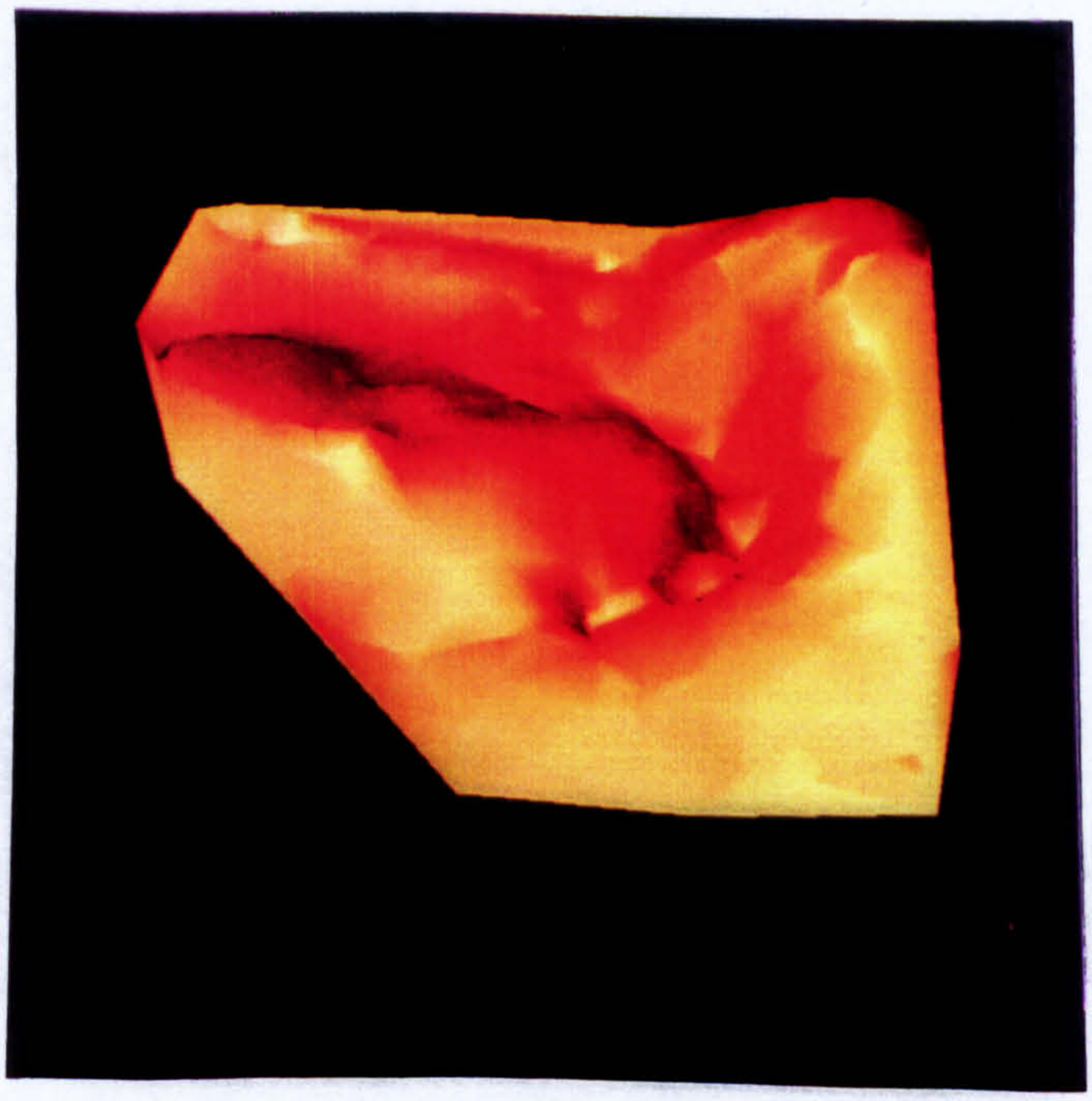


Plate 9.5 - Database 3, level 1 ground surface triangulation. *Shaded, perspective view.*

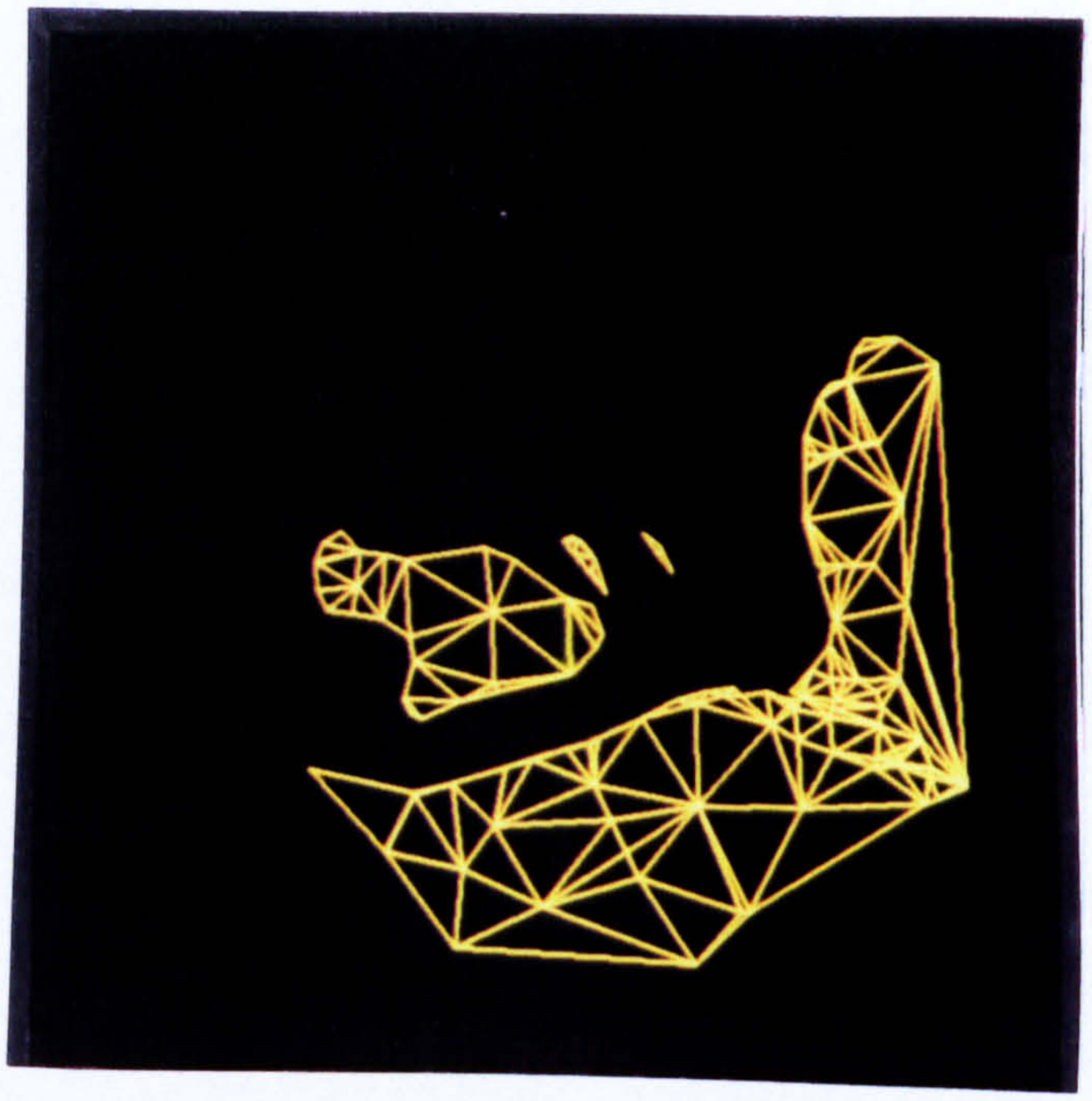


Plate 9.6 - Database 3, level 1 LLL triangulation. *Plan view.*

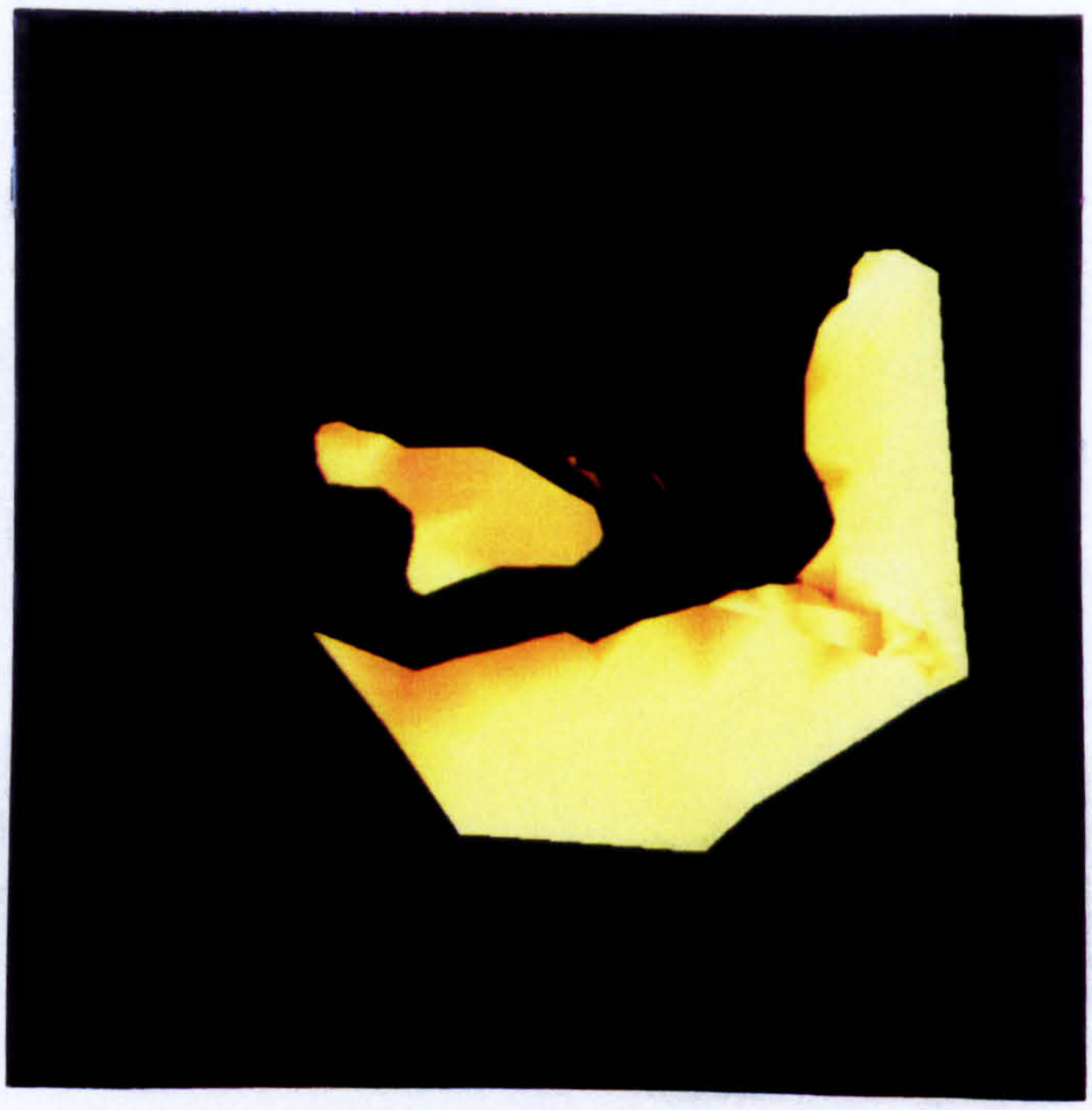


Plate 9.7 - Database 3, level 1 LLL triangulation. *Shaded, perspective view.*

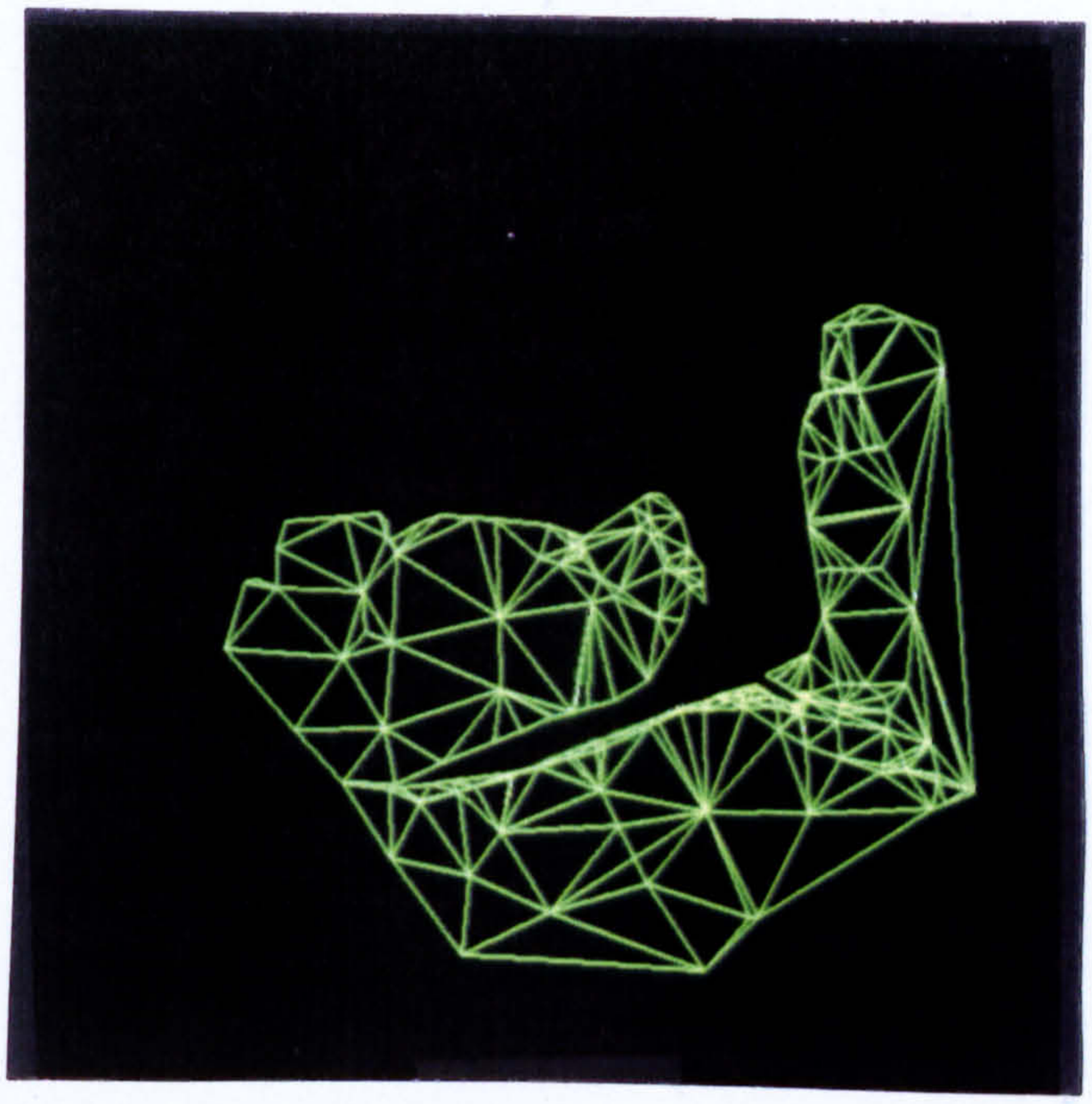


Plate 9.8 - Database 3, level 1 GRF triangulation. *Plan view.*

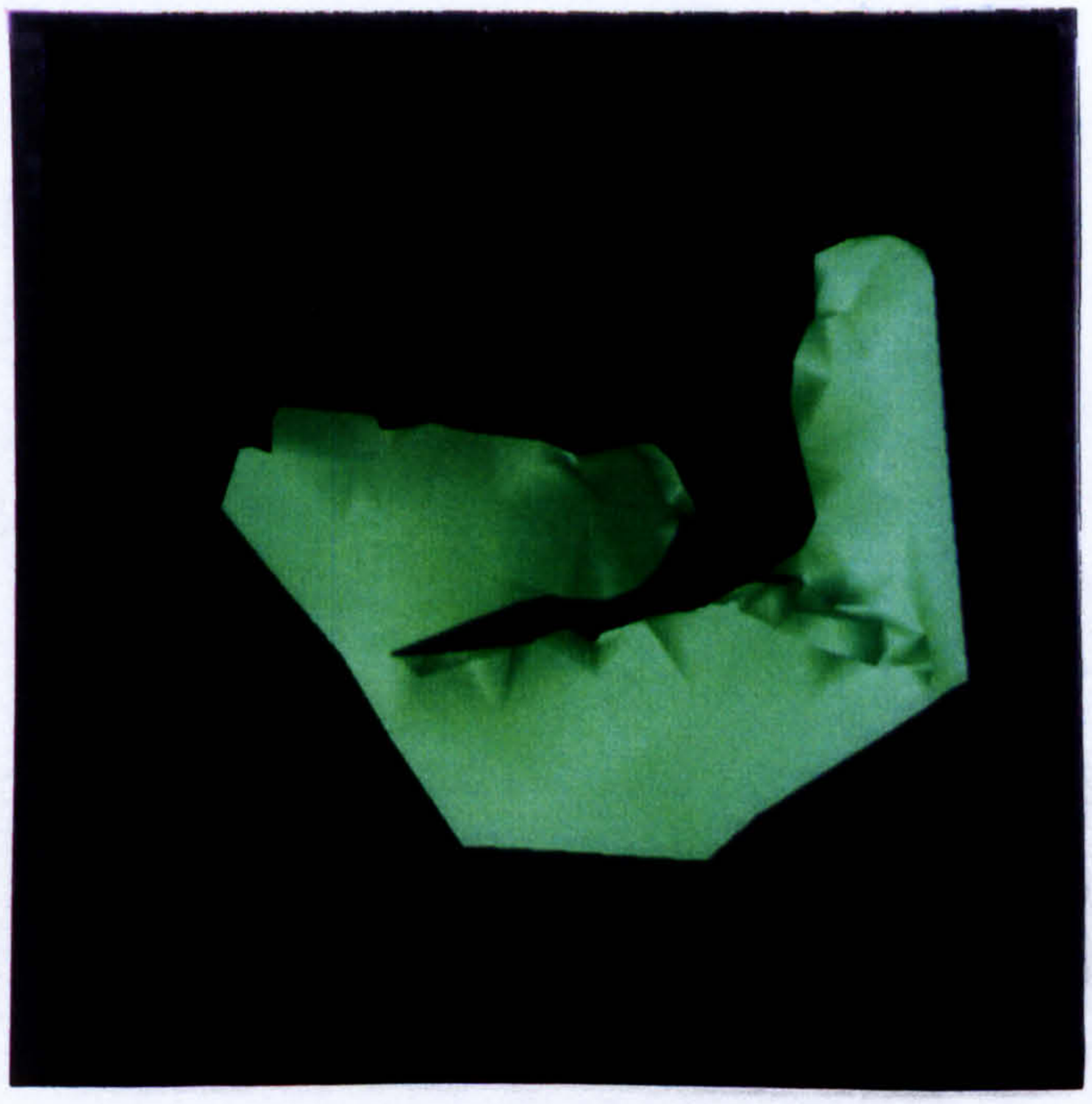


Plate 9.9 - Database 3, level 1 GRF triangulation. *Shaded, perspective view.*

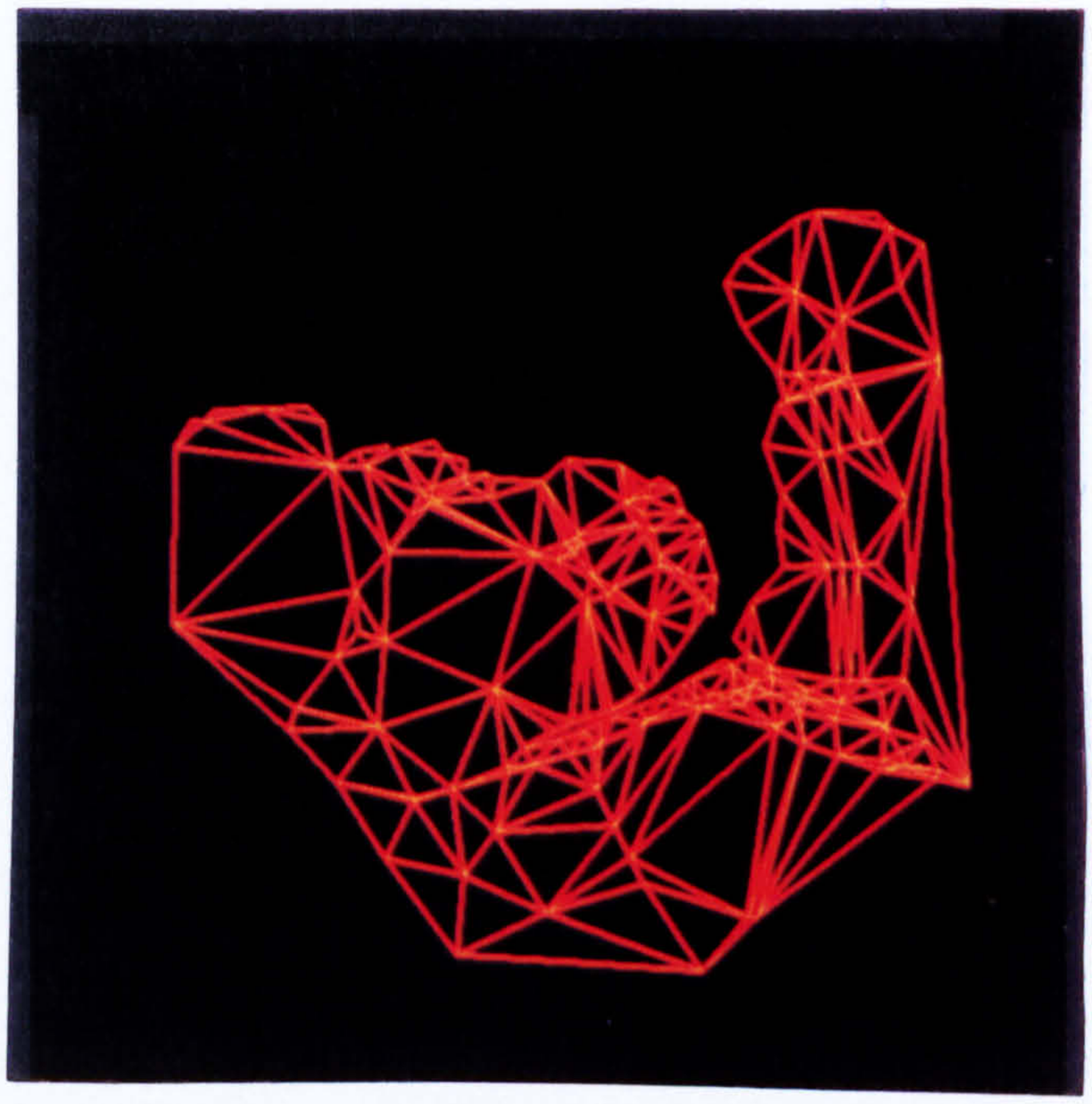


Plate 9.10 - Database 3, level 1 NS triangulation. *Plan view.*

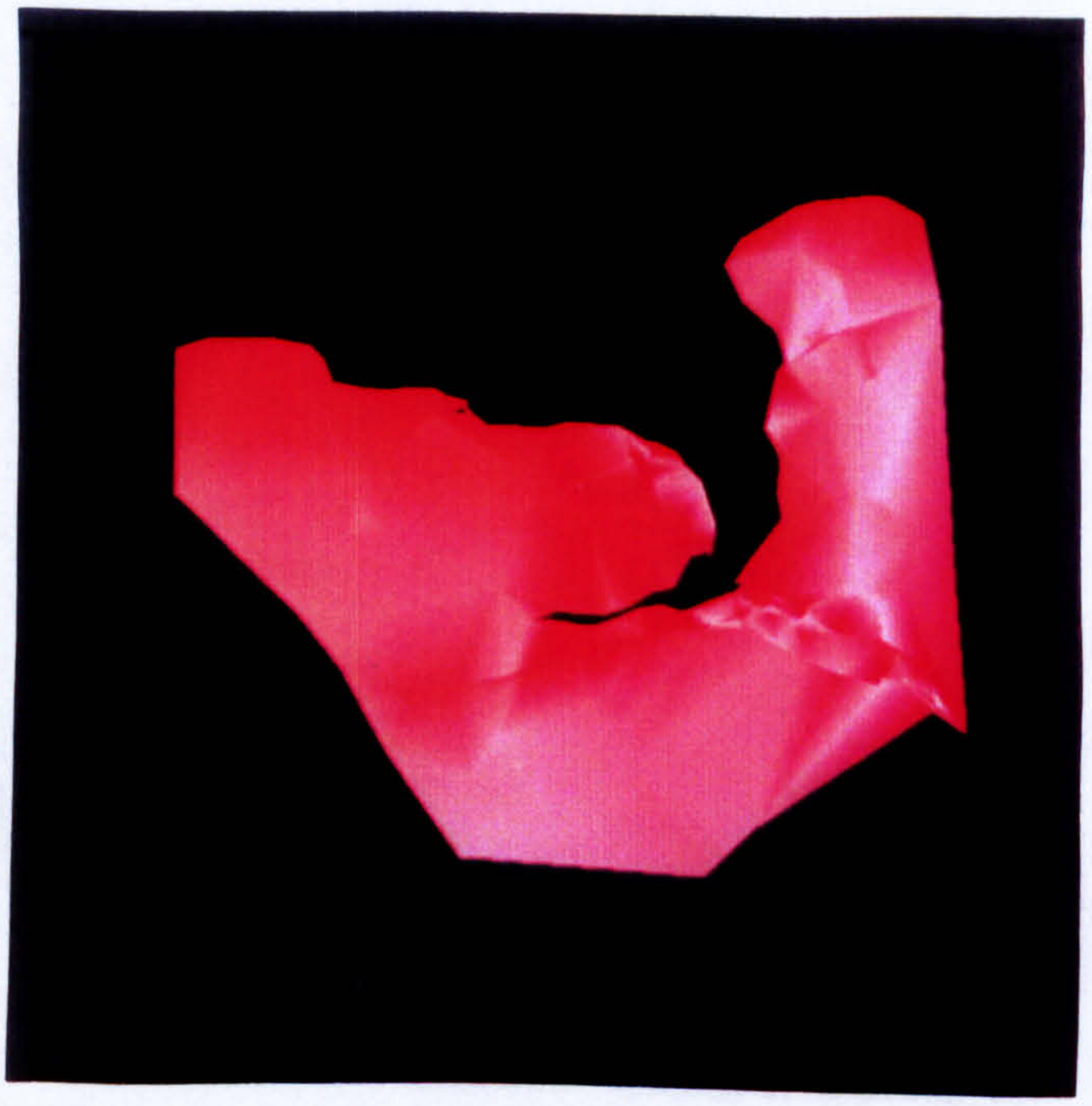


Plate 9.11 - Database 3, level 1 NS triangulation. *Shaded, perspective view.*

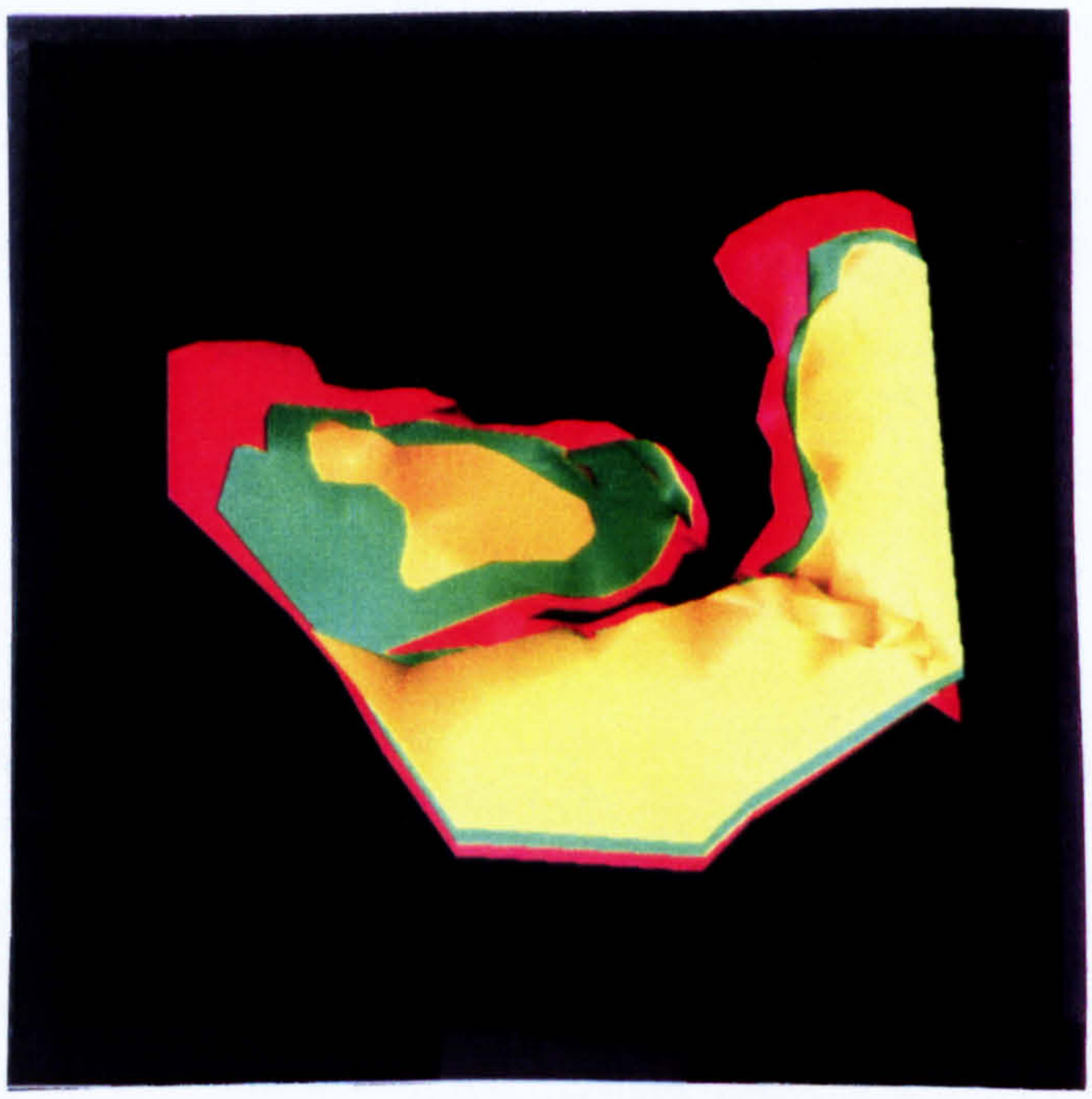


Plate 9.12 - Database 3, level 1 subsurface triangulations.

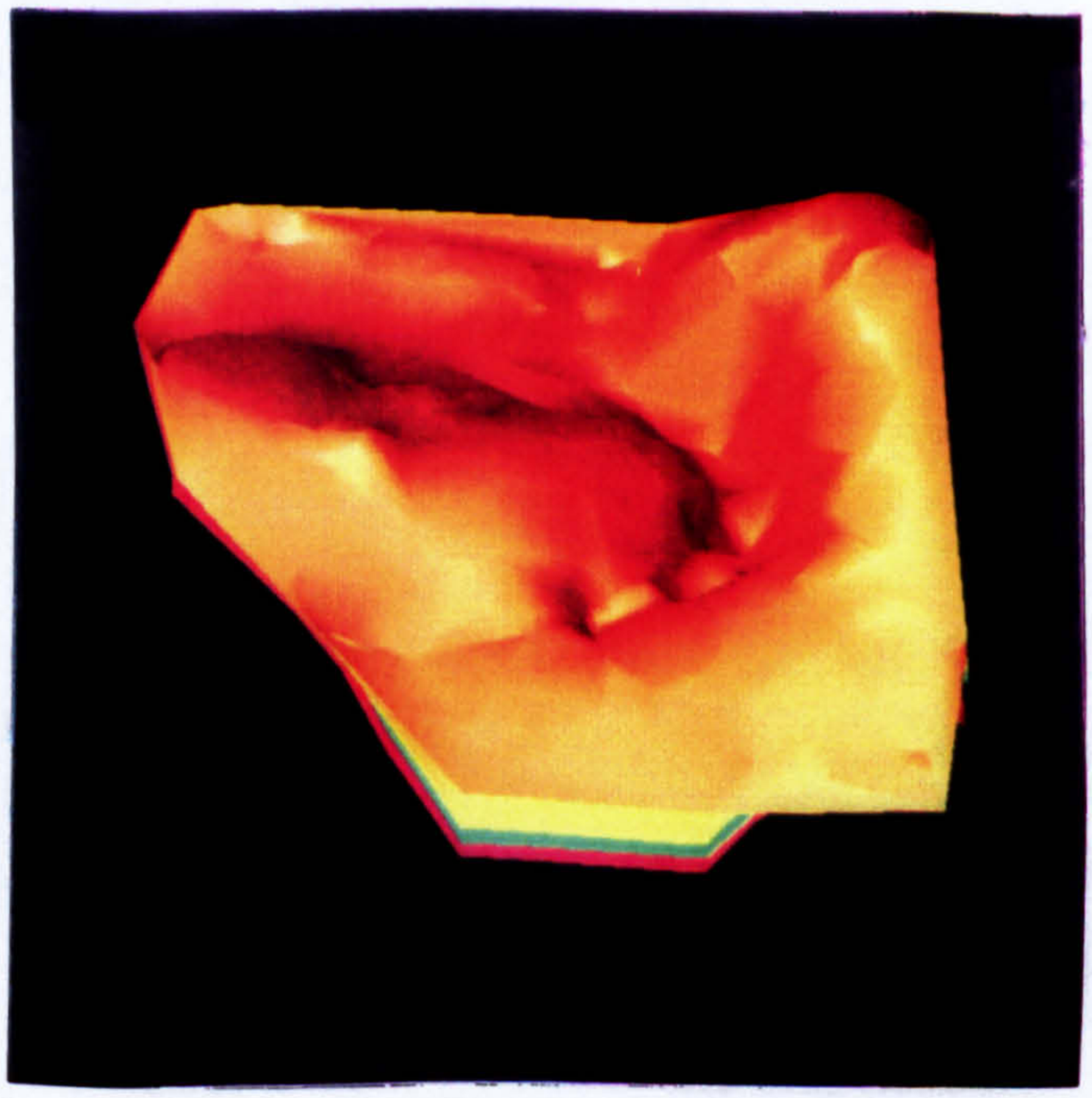


Plate 9.13 - Database 3, level 1 all triangulations.

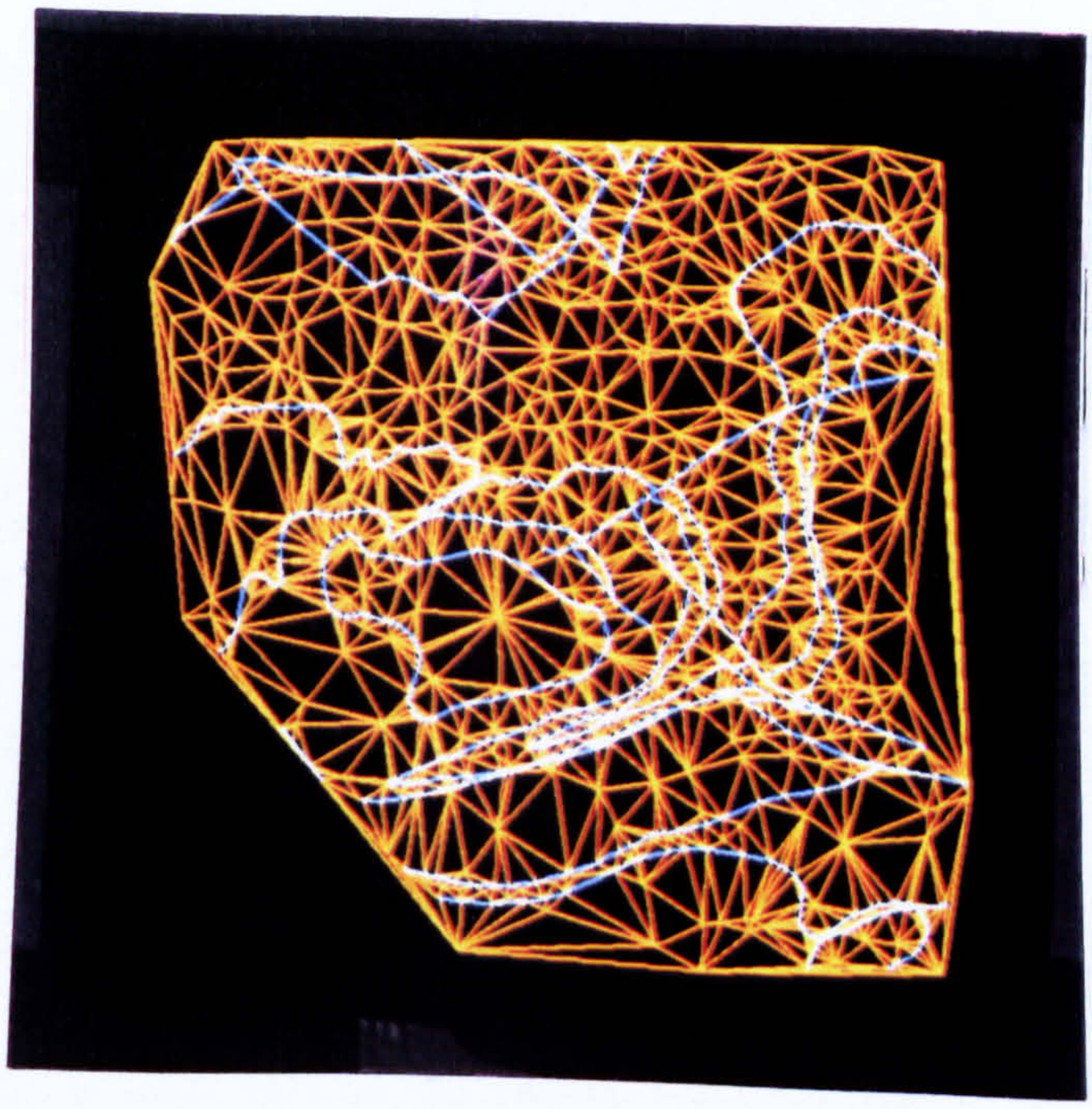


Plate 9.14 - Database 3, level 3 ground surface triangulation. *Plan view.*

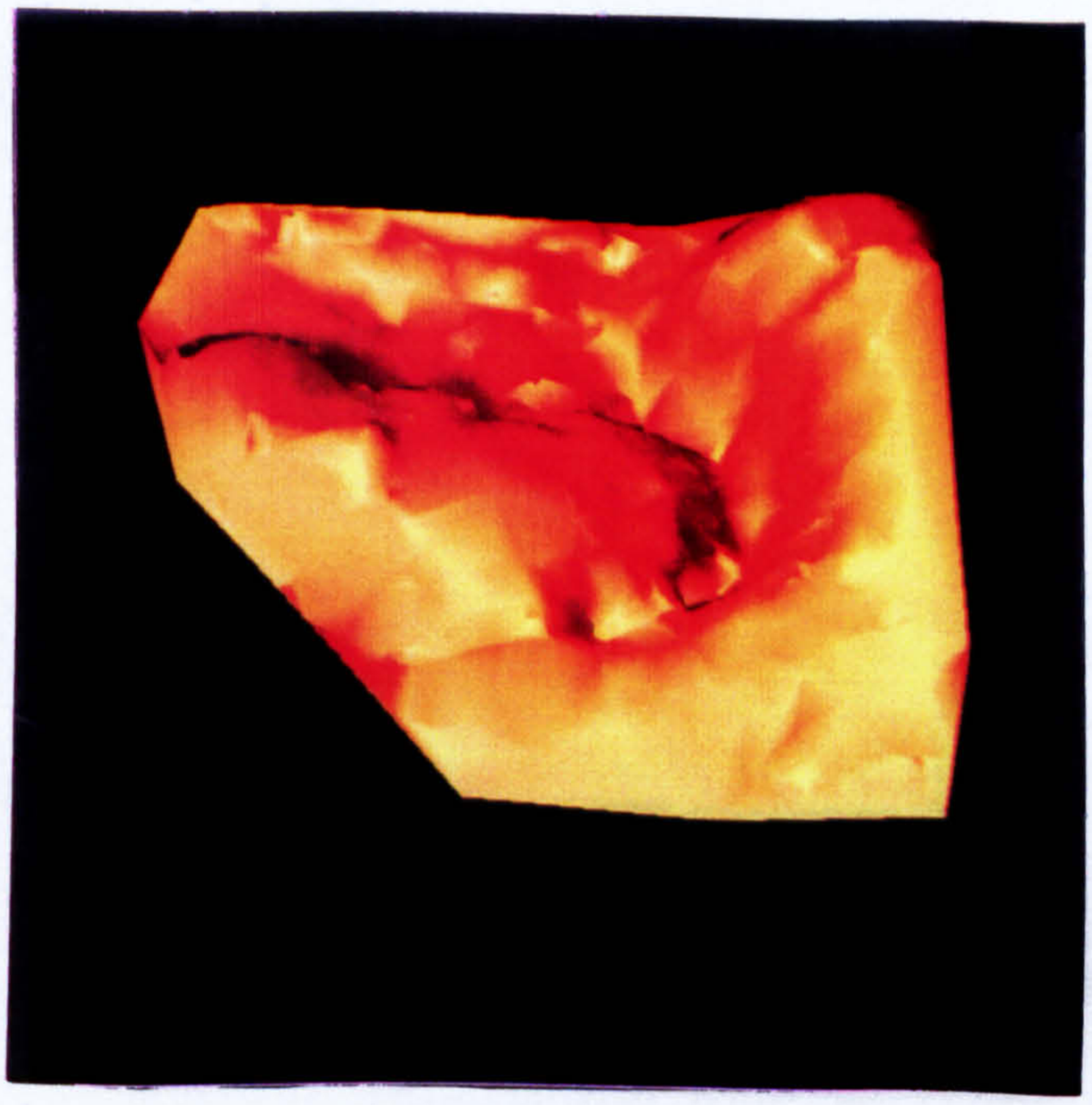


Plate 9.15 - Database 3, level 3 ground surface triangulation. *Shaded, perspective view.*

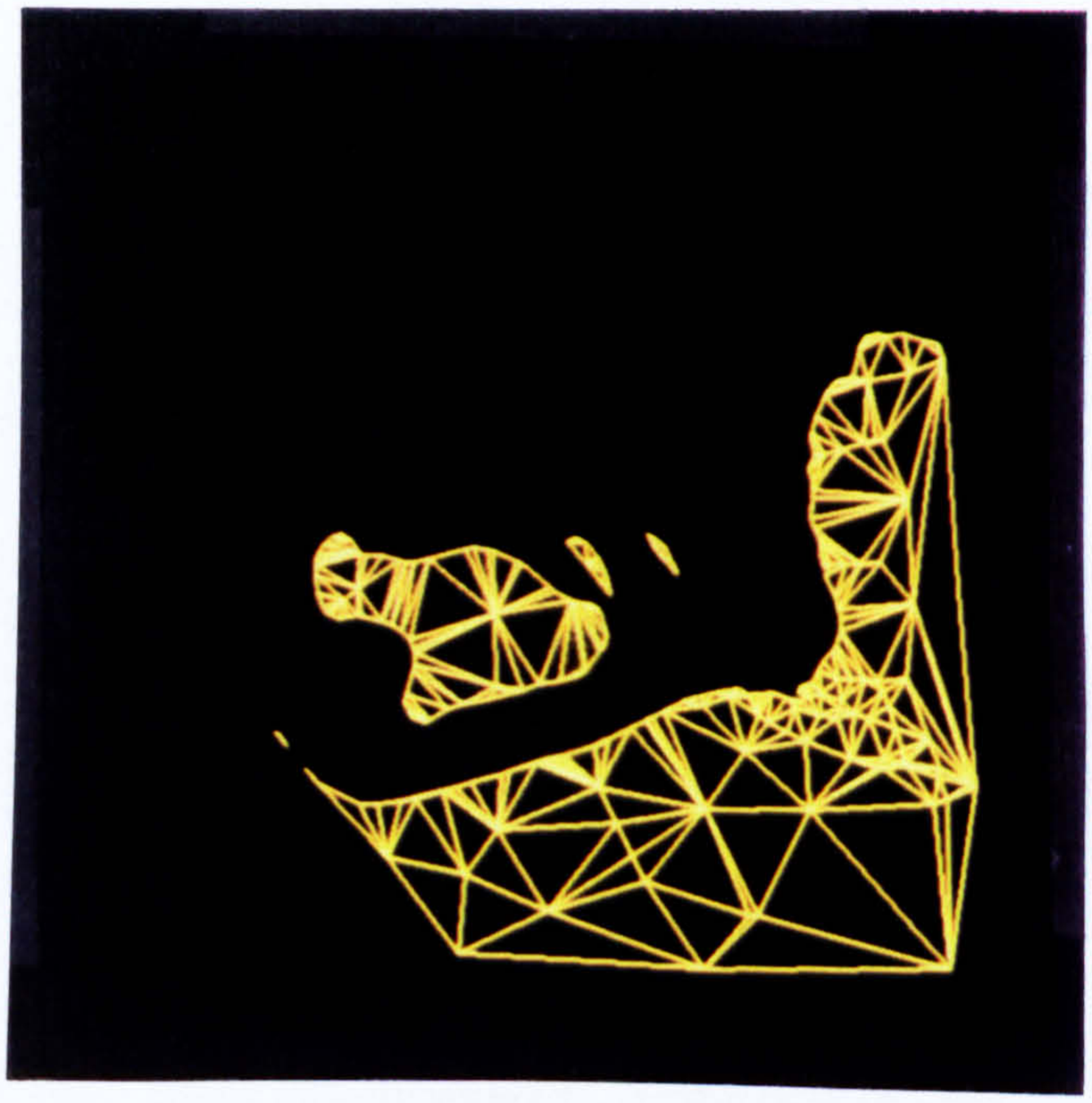


Plate 9.16 - Database 3, level 3 LLL triangulation. *Plan view.*

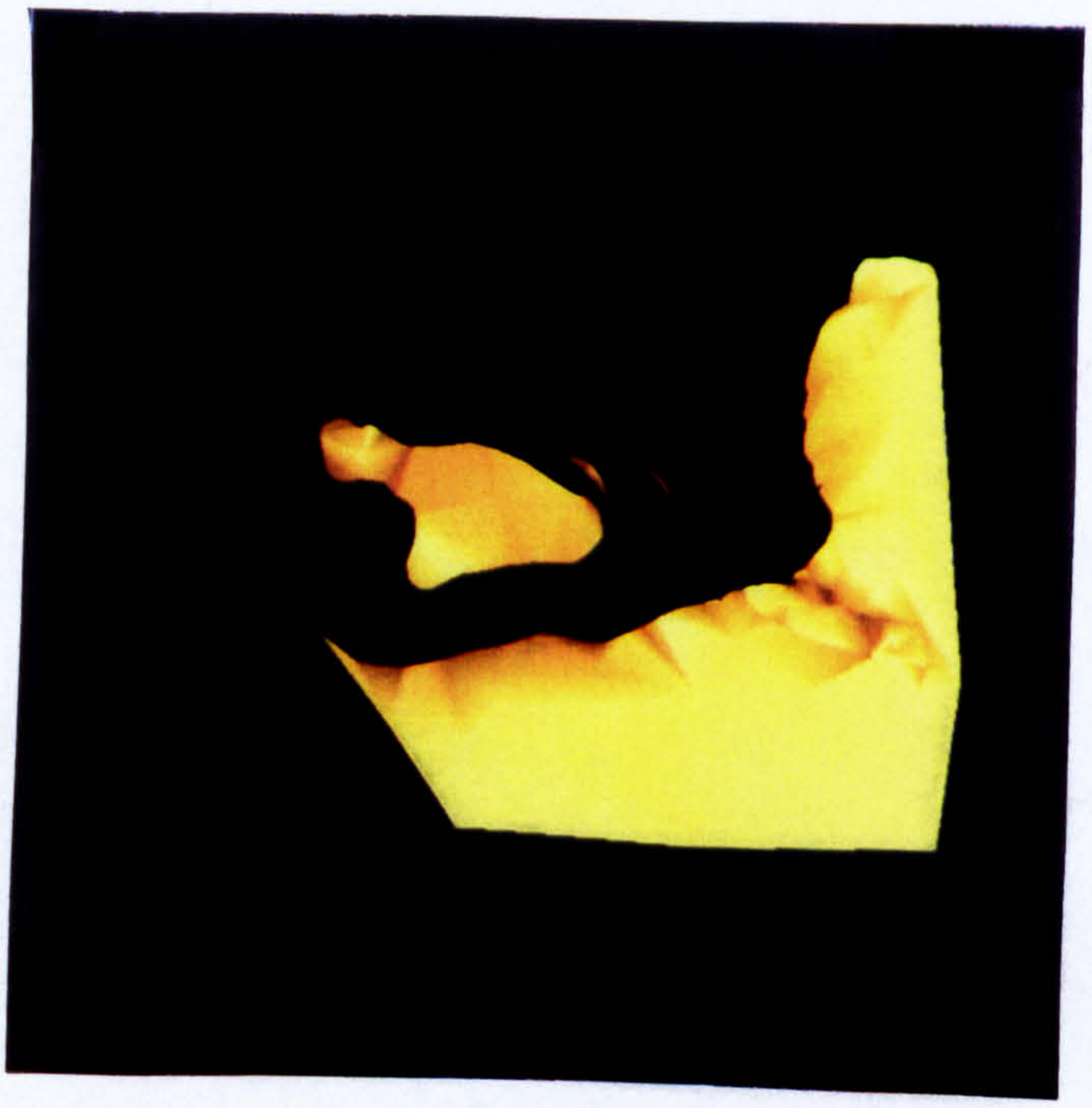


Plate 9.17 - Database 3, level 3 LLL triangulation. *Shaded, perspective view.*

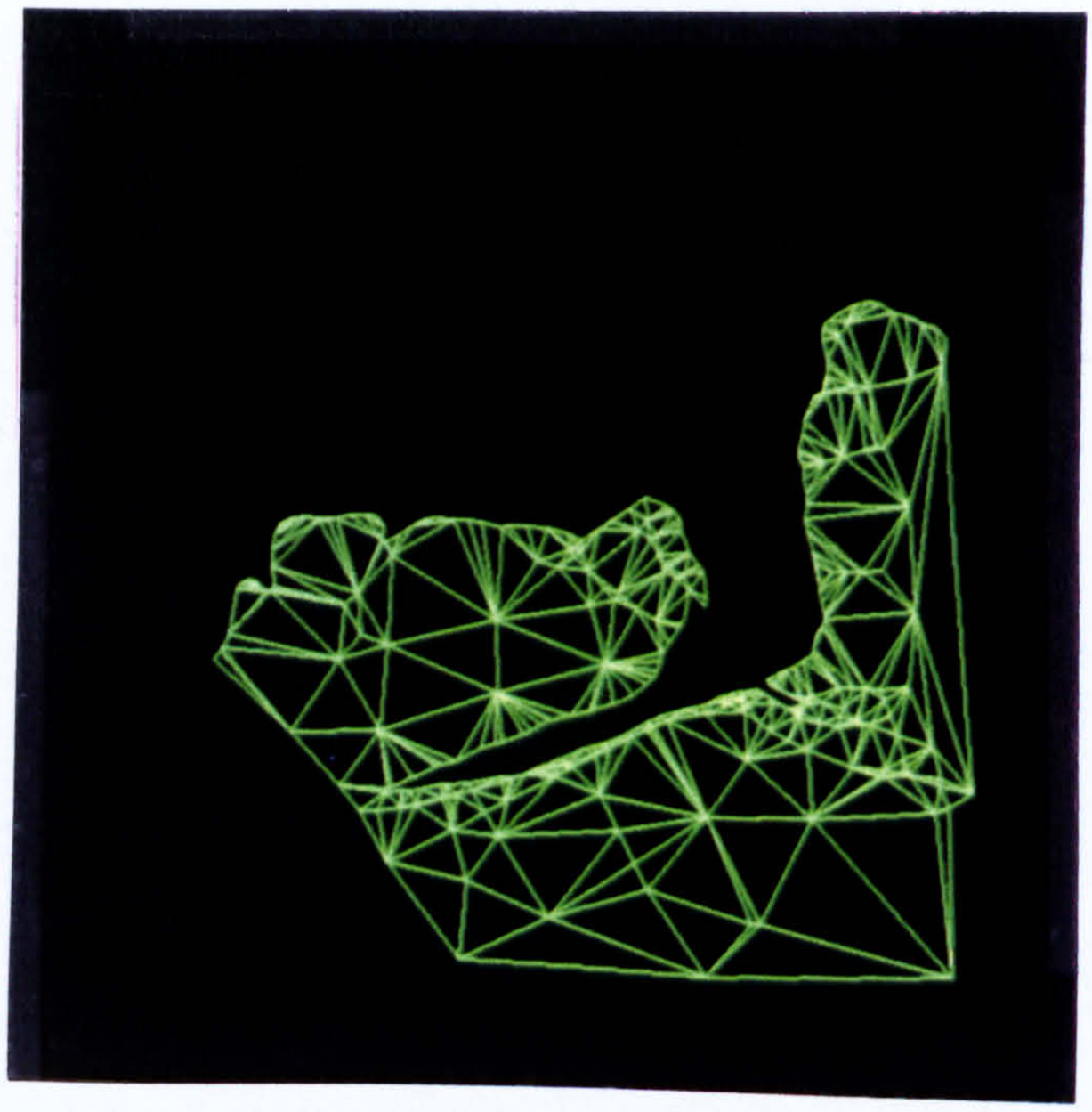


Plate 9.18 - Database 3, level 3 GRF triangulation. *Plan view.*

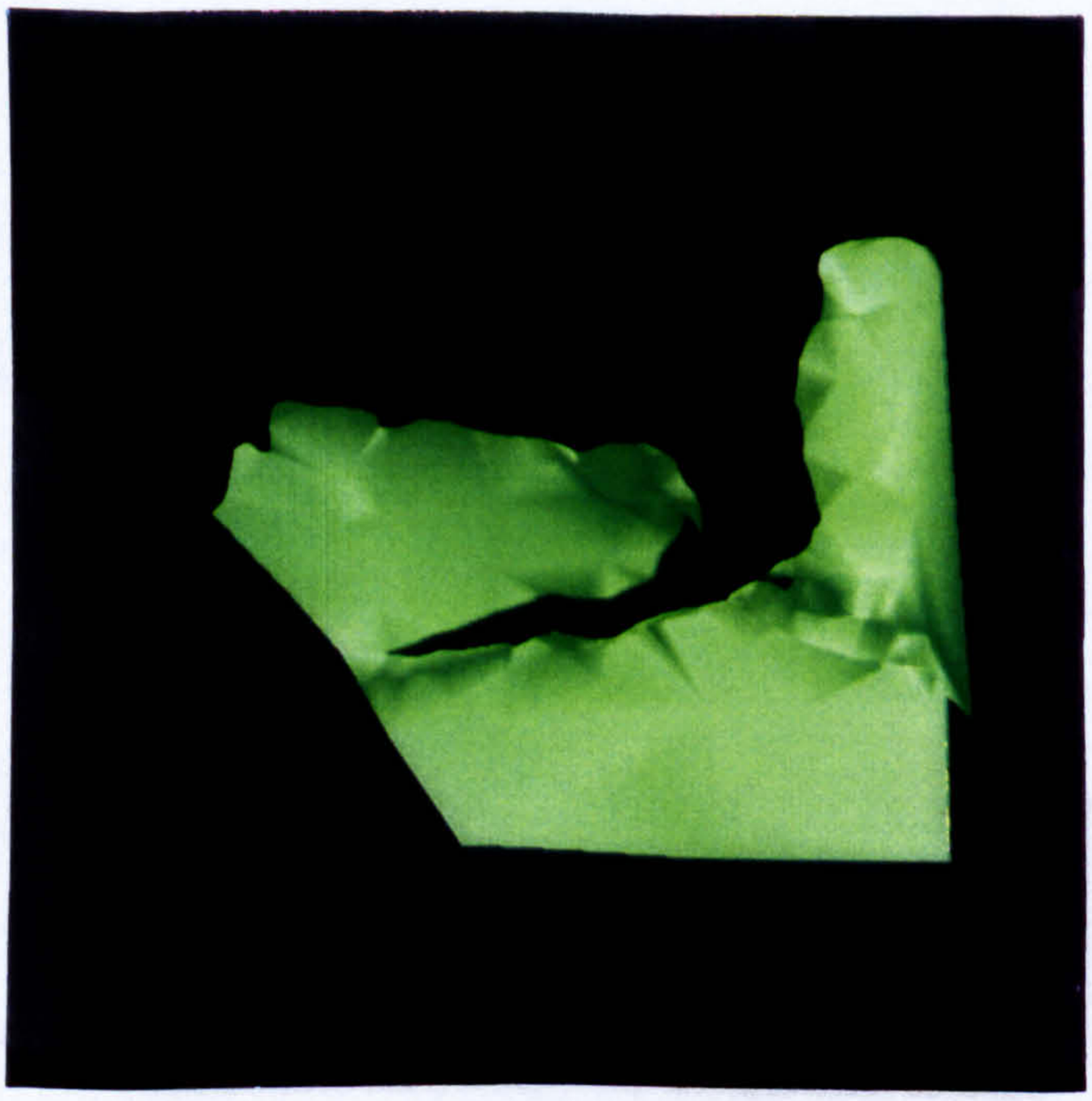


Plate 9.19 - Database 3, level 3 GRF triangulation. *Shaded, perspective view.*

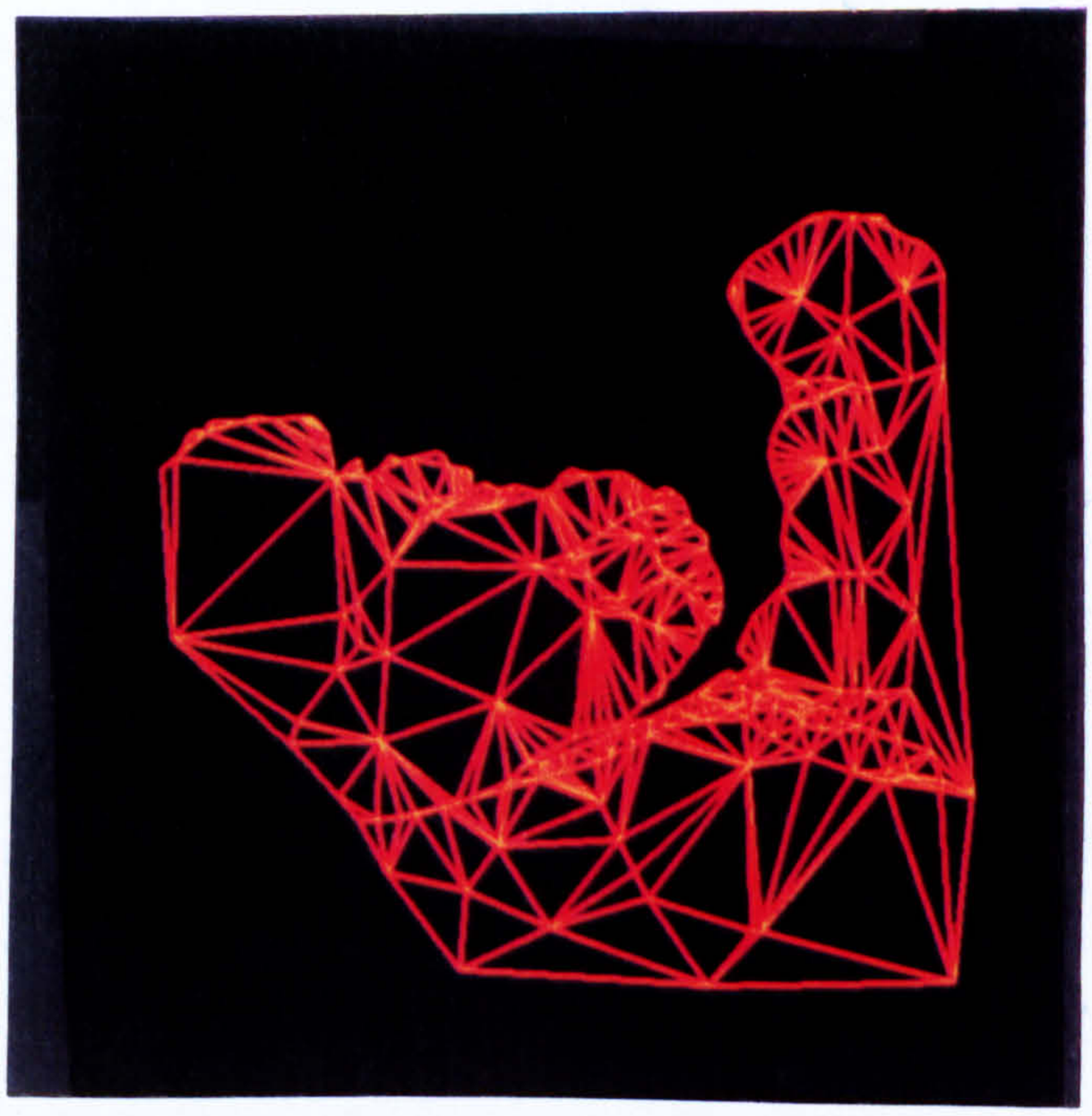


Plate 9.20 - Database 3, level 3 NS triangulation. *Plan view.*

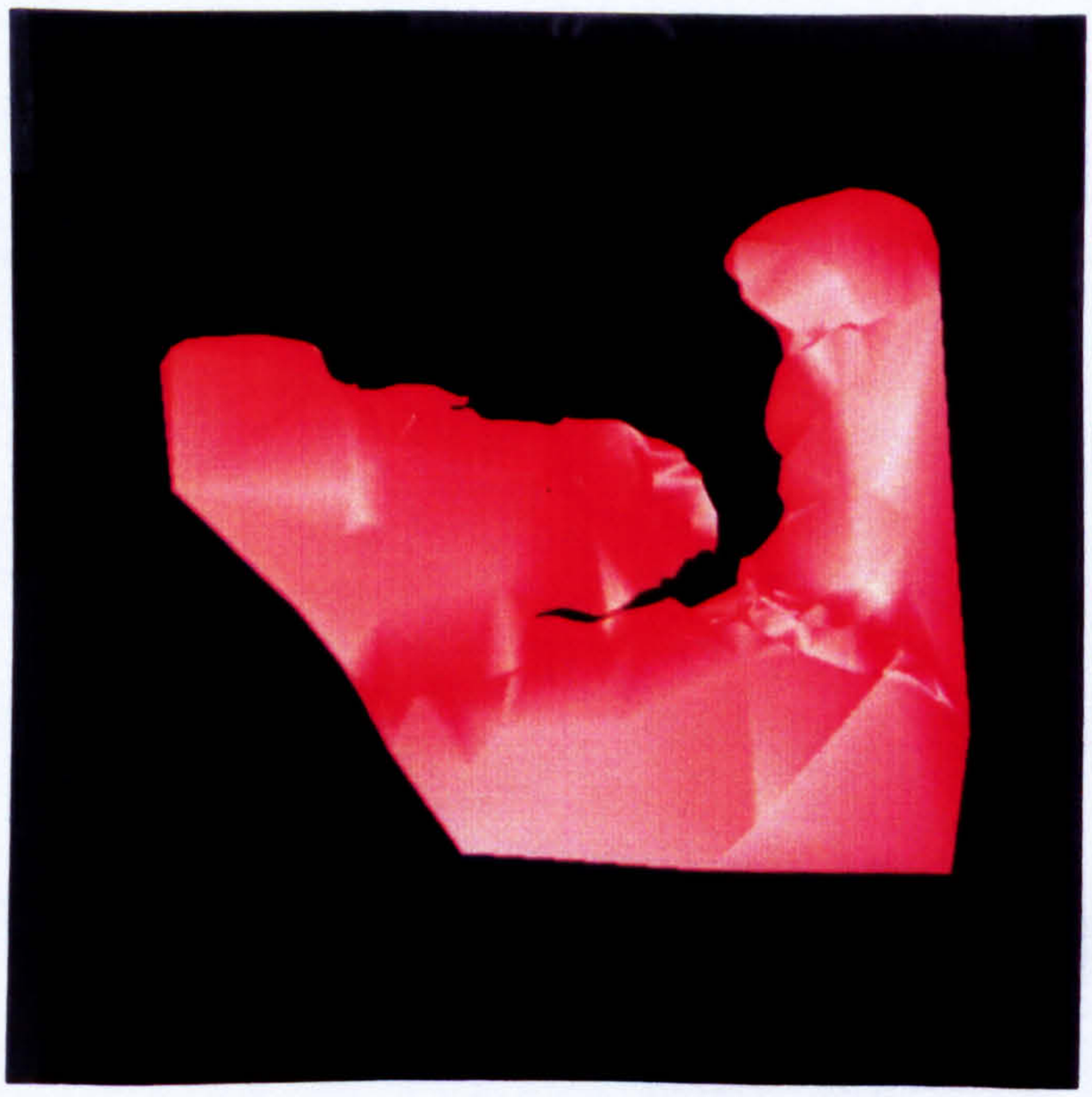


Plate 9.21 - Database 3, level 3 NS triangulation. *Shaded, perspective view.*

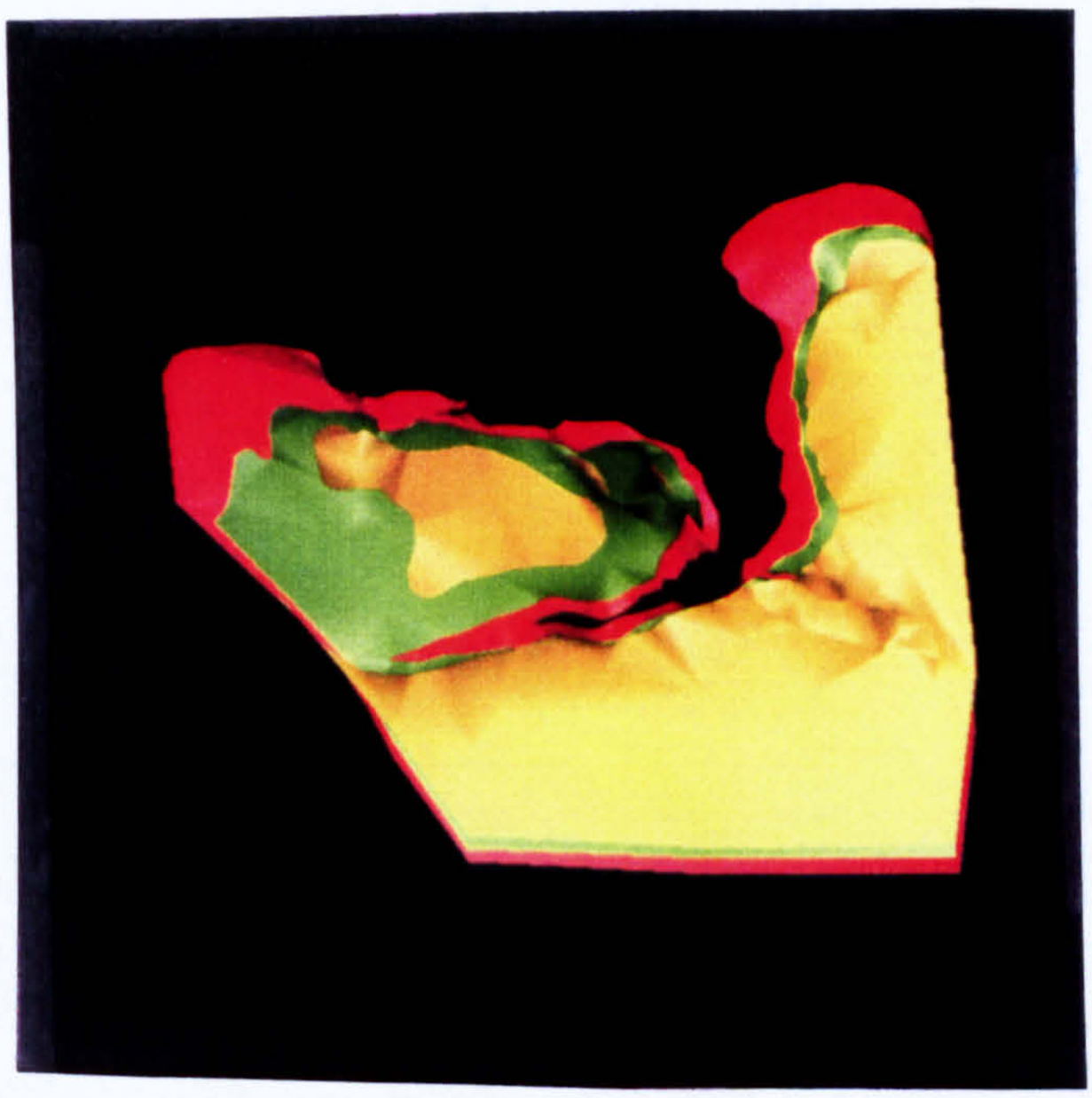


Plate 9.22 - Database 3, level 3 subsurface triangulations.

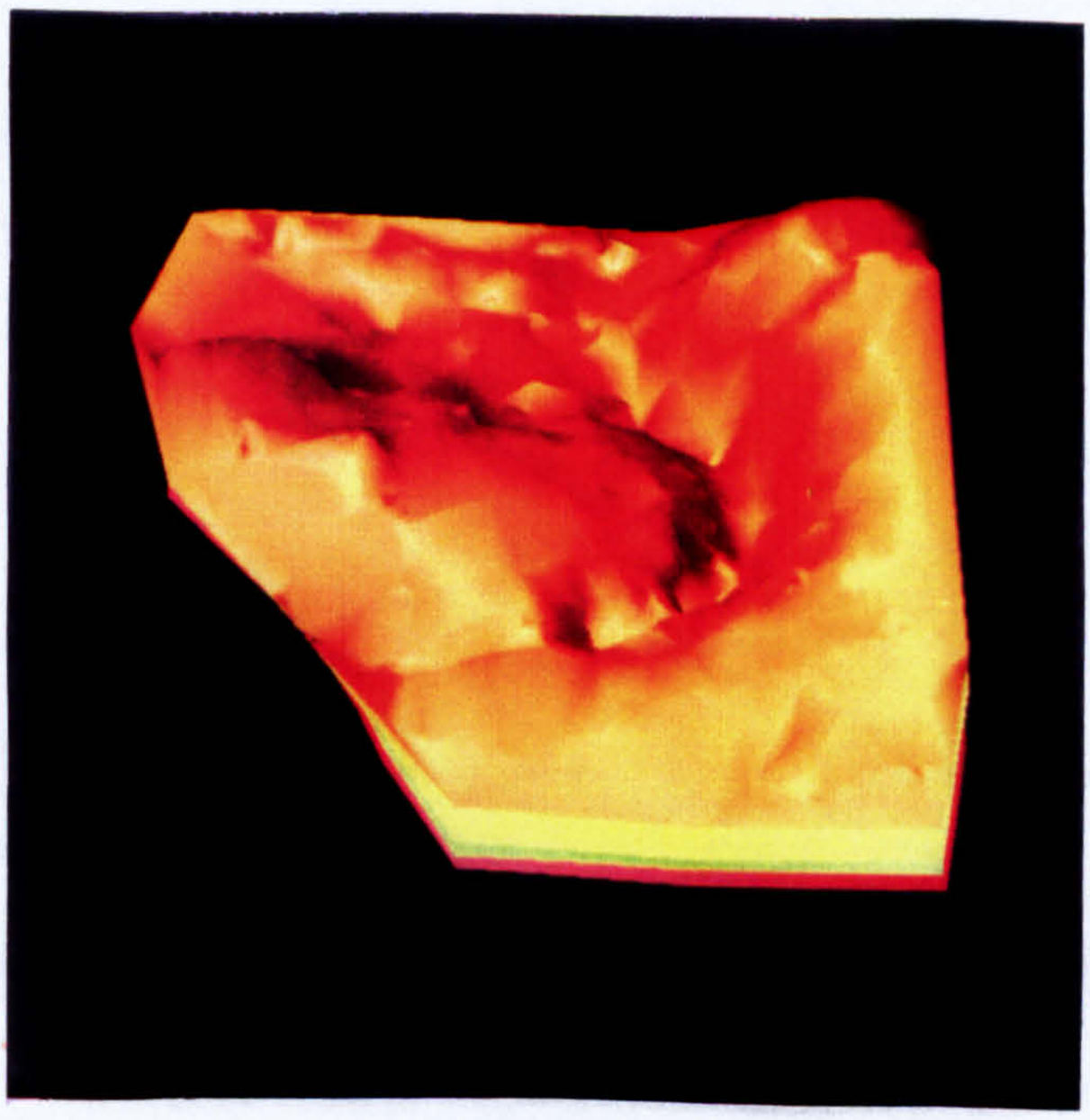


Plate 9.23 - Database 3, level 3 all triangulations.

9.5 Problems and Discussion.

The prototype GSIS described in Section 9.2 is not without its limitations, the most notable of which is its inability to cope with multi-valued surfaces. This particular shortcoming is one which had been foreseen, but for the particular data sets modelled in this thesis was not thought to present a problem. However, some thought has been given as to how multi-valued surfaces can be catered for in the future, a report of which is given here. Other problems concerning the prototype system only came to light during the testing of the system. These mainly concerned invalid intersections between adjacent surface triangulations. Solutions to these problems have been found, and implemented, and are also described here.

9.5.1 Coping with Multi-Valued Surfaces.

As indicated at the start of Section 9.2, a limitation of the prototype system is its inability to cope with multi-valued surfaces. This problem does not prove a hindrance for ground surface models, or indeed for some subsurface models such as those obtained from the BGS data. For example, consider a set of points S_g representing a portion of the Earth's surface. Each point of S_g consists of an x, y and z coordinate. For the purpose of constructing the Delaunay triangulation (2-D Delaunay tessellation) of S_g it is usual to consider only the x and y values of each point and neglect the z value. This is analogous to only considering the x,y value of the intersection each point makes with the xy-plane when projected in a vertical direction towards that plane. The z value is selected for omission from considerations, as opposed to either the x value or y value, on account of the Earth's surface being much less likely to be multi-valued in the xy-plane than in the yz-plane or xz-plane.

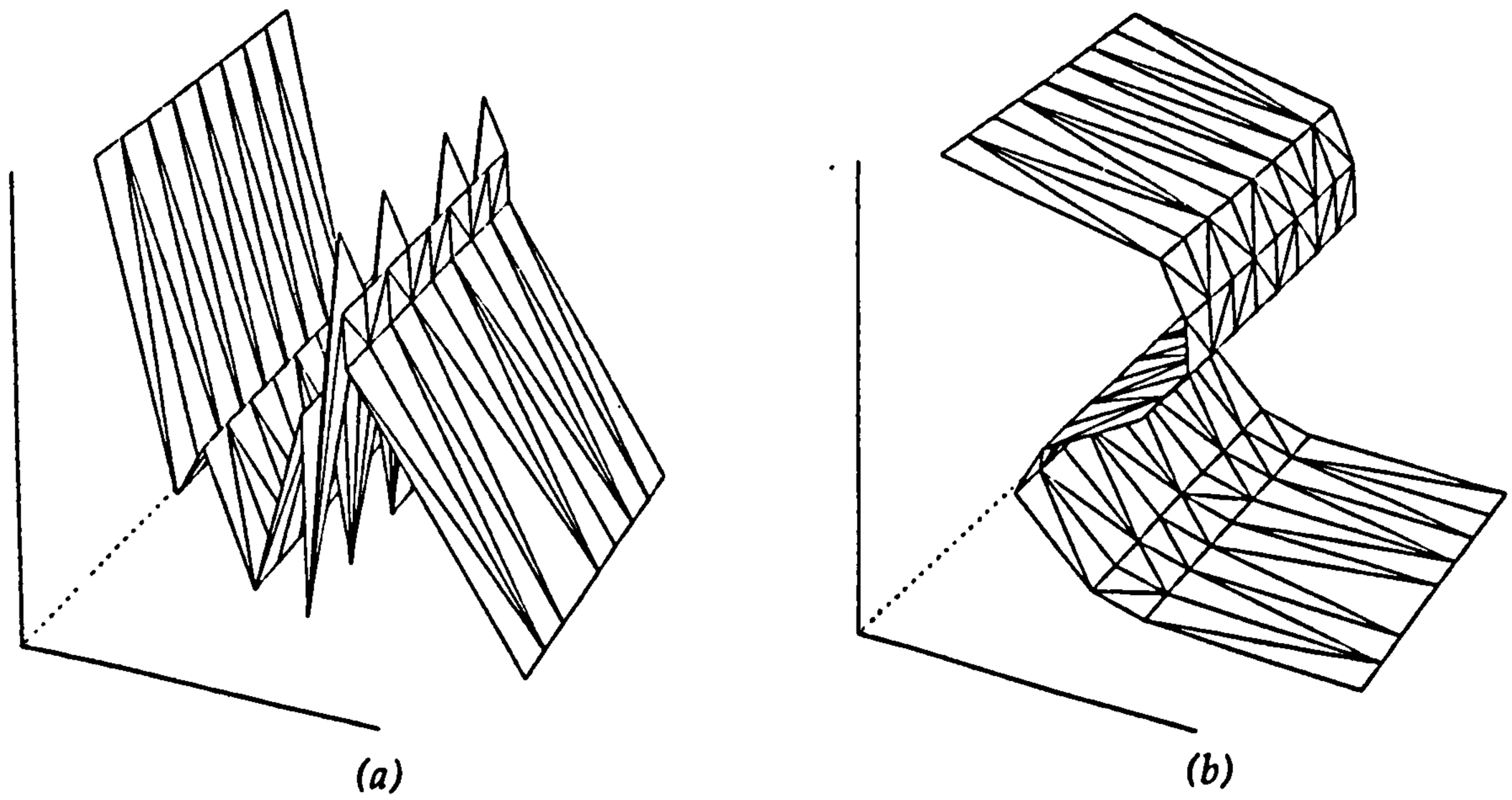


Figure 9.5 - The effect of triangulating on planes. (a) A set of points triangulated on the xy-plane. (b) The correct surface, obtained by triangulating on the yz-plane.

This fact would not necessarily be true for a set of points S_s representing a geological boundary in the subsurface. Here the surface is more likely to be multi-valued in the xy-plane than for S_g. This is due to the complex faulting and folding which takes place in the subsurface. In general, such surfaces may be of any orientation and may be overfolded, at least relative to the horizontal plane. As an example, Figure 9.5a shows the surface which results from the Delaunay triangulation in the xy-plane of a set of points, which is in fact quite different from the true surface, shown in Figure 9.5b. This is due to the fact that the true surface is multi-valued in the xy-plane. Therefore, an alternative method for constructing a surface triangulation of S_s has to be found.

9.5.1.1 Data Segmentation.

With a prospective method in mind, attention is drawn to work described by Boissonnat [113], in which procedures for the Delaunay triangulation, in 2-D and 3-D, of arbitrarily oriented surfaces are presented. In the 2-D case, collections of vertices are projected onto local planes on which triangulation takes place. Vertices which are triangulated together are assumed to belong to a neighbourhood of vertices which represent a surface which is single-valued relative to the local plane, which itself is derived by a least square fit. This method is limited in that it requires that vertex neighbourhoods be defined in advance. If data sampling is dense then this is not a problem since it can be assumed that any vertices lying within a reasonable distance of another vertex is in the neighbourhood of that vertex. When working with subsurface geological data, this assumption would not always hold true, since data may be sparsely distributed, and at the same time be representing complex structures. However, if some other means by which neighbourhoods could be defined was available, then the method would seem to be suited to the task in hand. One possible way in which this might be achieved is by dividing the vertices into subsets according to localised measurements of gradient (the trend of the surface). Another technique could involve a geologist interactively splitting up the data via a suitably designed 3-D graphics interface. Once all neighbourhoods are defined, they are each triangulated separately as normal, except that projection is onto a local plane of appropriate orientation. Triangulations, having been constructed, could then be be 'zipped' across common borders to form a complete surface. Algorithms for 'zipping' triangulations together in the xy-plane are given in the literature [11, 12]. These algorithms could be adapted to cater for arbitrary planes.

It may also be the case that the best plane of projection is not always the least square plane. In such cases, the availability of other information might assist in deciding upon the best plane. An experimental program has been implemented which uses dip information to decide on what this plane may be. For example, given sufficient dip information, the program correctly identifies the yz-plane to be more appropriate than the xy-plane as the plane of projections for the data given in Figures 9.5a and 9.5b.

Projecting onto the yz-plane does in fact produce the correct surface triangulation.

9.5.1.2 Computing the 3-D Delaunay Tessellation.

A second approach to solving the multi-valued surface problem is to consider algorithms which compute the 3-D Delaunay tessellation, such as those published by Riedinger et al [117], Tanemura et al [118], Devijver and Dekesel [119] and Boissonnat [113]. For the problem being discussed here, the approach would be to use the 3-D tessellation only as a temporary structure, from which the bounding surface triangulation could be extracted. The extraction of the surface would be assisted by taking into consideration any known constraints, such as dip, on the data. The paper by Devijver and Dekesel describes a dynamic procedure where points are inserted one at a time into an existing tessellation which is repeatedly updated. This method is equivalent to the 2-D algorithms already implemented and could therefore be well suited to a multiresolution model. Boissonnant suggests a 3-D volumetric approach where the surface represented by a set of vertices is found by determining the boundary of a 3-D tessellation. Initially the volumetric tessellation occupies the entire convex hull of the vertices, after which tetrahedra occupying what are assumed to be concavities of the surface are progressively eliminated. A limitation of this method is that the resulting triangulated surface can only be expected to correspond topologically with the real surface if the original vertices have been obtained by a relatively regular and dense sampling of the object surface.

9.5.2 3-D Spatial Conflict.

During the testing of the 3-D multi-scale database it became apparent that in certain instances topological errors, or spatial conflict, had occurred. These errors involved invalid intersections between adjacent surfaces. For example, each of the subsurface triangulations intersected the ground surface triangulation in a number of places. In some cases a subsurface triangulation was found to intersect one or more other subsurface triangulations. Through experimentation it was found that the spatial conflict was caused in one of three ways. An explanation of each cause of error, and solutions, is now presented.

9.5.2.1 Conflict due to Data Set Discrepancies.

This error type came to light as a result of observing triangulated surfaces of the type shown in Figure 9.6a. This error, in the first instance, is considered to have occurred as a result of a sampling discrepancy existing between the ground surface data and the subsurface elevation data (obtained from the borehole data). The discrepancy causes spatial conflict between the two surface triangulations. A partial solution to this problem is to ensure that the data sets in question are as well matched as possible. This has been achieved by introducing the top of borehole height measurements

(included in the original borehole data) as additional elevation data in the ground surface data file (Figure 9.6b). This has the effect of offering a better match between data sets, and subsequently, an improved model.

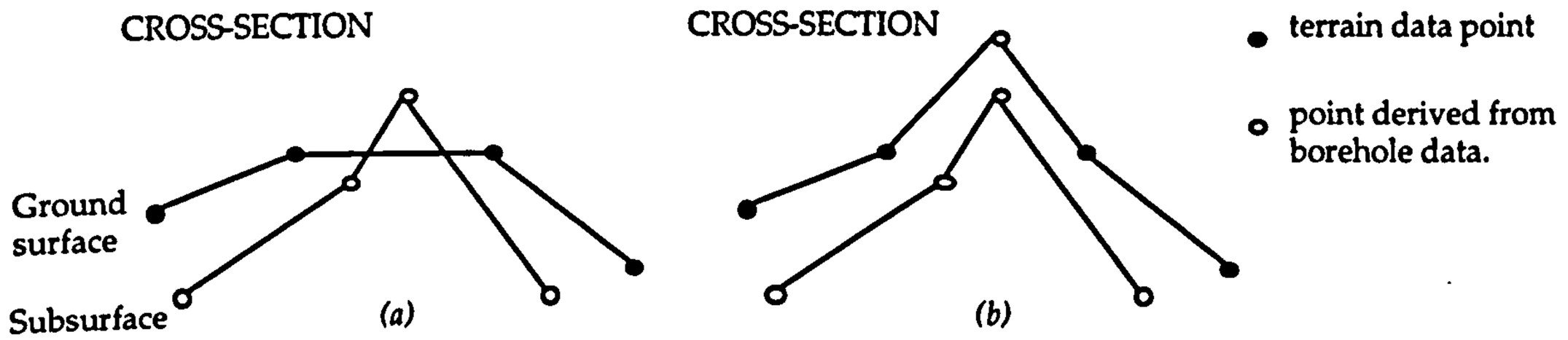


Figure 9.6 - Spatial conflict due to a data discrepancy. (a) The subsurface incorrectly intersects the ground surface. (b) The top of borehole measurements are introduced in the terrain data, in this case resulting in spatial integrity.

A further problem is encountered due to discrepancies between the outcrop data, ground surface data and subsurface data. Consider the cross section shown in Figure 9.7a which contains no conflict. However, if the ground elevation point i was not included in the ground surface data set, which will sometimes be the case with real data, the cross section will appear as shown in Figure 9.7b. Here the ground surface triangulation has become co-incident with the subsurface triangulation. It can be deduced that such conflict occurs when ground surface triangles are made up entirely of points which belong to the same region outcrop object. A first pass attempt at resolving this situation is to check if there are any currently unused subsurface points (held in the appropriate primary subsurface points file) which lie within one of the offending ground surface triangles. If a suitable subsurface point is found it is added to the subsurface triangulation. Also, to ensure consistency between triangulations, the top of borehole height measurement is added as an elevation point to the ground surface triangulation. If no suitable subsurface point is found the next step is to check if any suitable currently unused ground elevation point (held in the Primary Ground Surface Points File) is available. If an unused ground elevation point is found which lies within an offending triangle, it is only inserted if it lies above the current ground surface triangulation. A point which lies below the triangulation, if inserted, will result in intersection between its triangulation on the currently co-incident subsurface triangulation.

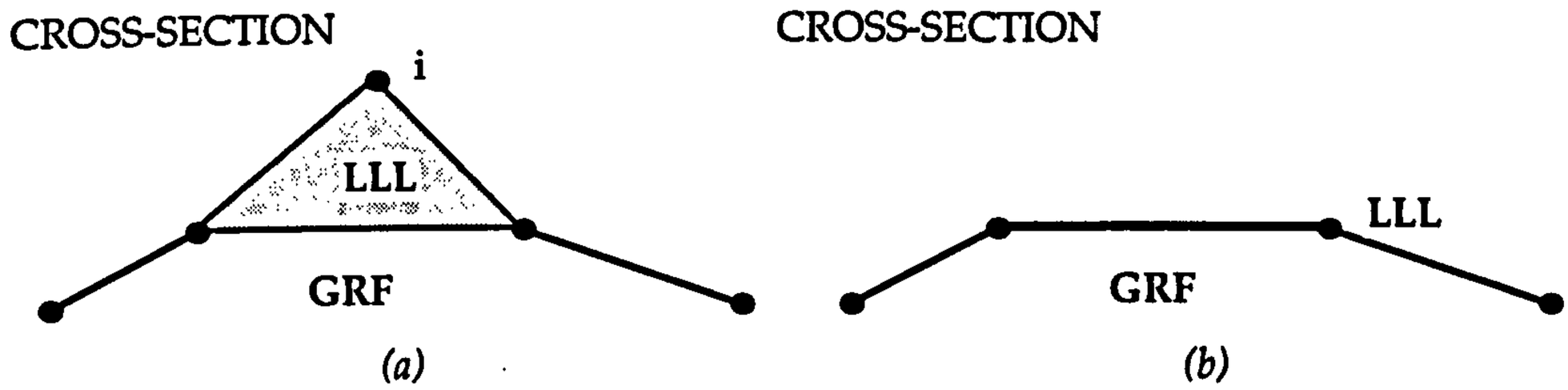


Figure 9.7 - A second error due to a data discrepancy. (a) Surfaces contain no conflict. (b) Point i is removed resulting in co-incidence of the base of the LLL Formation and the ground surface.

If no suitable unused point is available the solution is to create and insert a dummy point into either the ground surface triangulation or subsurface triangulation in such a way as to remove the conflict. Which of these is the better option will depend on the form of each of the surfaces. For example, in Figure 9.7b it would appear sensible to create a dummy point in the ground surface triangulation in such a way as to slightly raise the ground surface elevation in its vicinity. In Figure 9.8 a better approach would seem to be to insert a dummy point in the subsurface triangulation, slightly lowering the subsurface elevation in its vicinity. Automation of this choice making process would involve examination of all triangles, and their respective gradients, within the vicinity of the place where spatial conflict has occurred. This process has not been implemented in the current version of the MGD. At present the dummy point is always inserted in the ground surface triangulation.

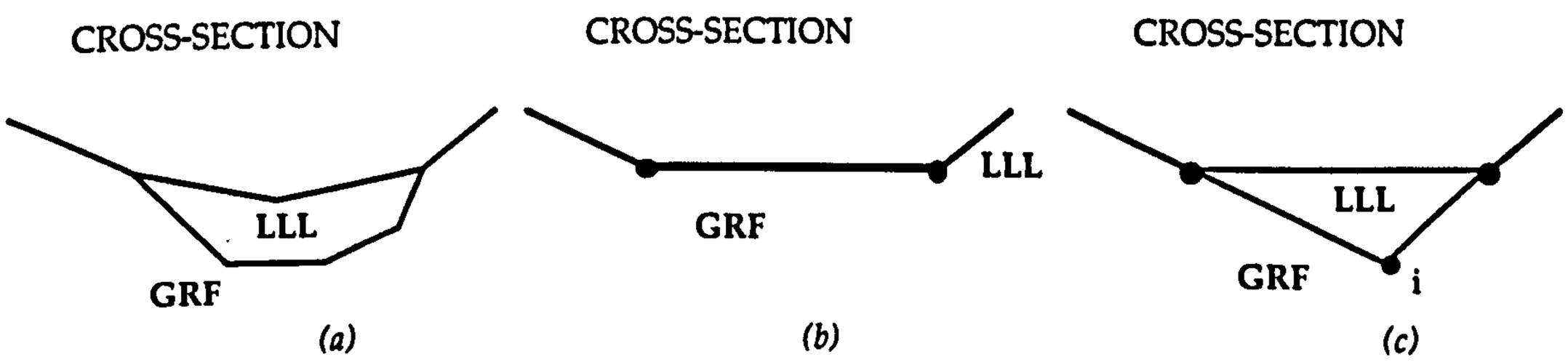


Figure 9.8 - Insertion of dummy point to restore integrity. (a) The correct surface cross-section. (b) Surfaces in conflict. (c) The solution is to add a dummy point i into subsurface.

9.5.2.2 Conflict due to Generalisation.

The problem of spatial conflict caused by application of the Douglas-Peucker algorithm to line data has been addressed in Appendix 1. During testing of the 3-D system it was observed that spatial conflict also occurred as a direct result of surface generalisation. The conflict is in the form of intersections between surface triangulations (Figure 9.9a). This problem can be remedied by re-inserting currently unused points in areas of conflict in way analogous to the 2-D solutions proposed in Appendix 1. However, this

is not the method used in the MGD system at present.

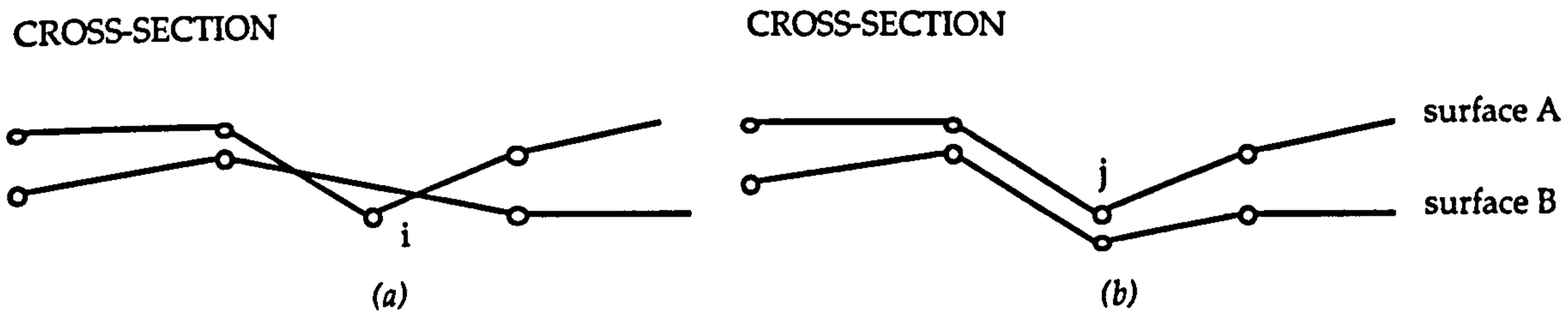


Figure 9.9 - Conflict due to generalisation. (a) The inclusion of point i in surface A causes conflict with surface B. (b) The solution is to include point j in surface B, point j being derived from same borehole as point i.

A simpler approach is adopted in which a condition is placed on certain of the points selected for a particular surface. The points to which the condition is applied are those which have been derived from borehole data (that is, subsurface elevation data and tops of borehole ground surface data). The condition insists that if any such point is selected for use in a particular surface triangulation, then all points derived from the same borehole as that point must also be added to the appropriate previously created surface triangulation and be included in any yet to be created surface triangulations, thus ensuring consistency between adjacent surfaces (Figure 9.9b). This method, while not being optimal with regards the number of points included in the model, is effective in removing spatial conflicts caused by surface generalisation.

9.5.2.3 Conflict due to Incorrect Insertion of Constraints.

Despite application of the conflict resolution techniques described in Sections 9.5.2.1 and 9.5.2.2, conflict still occurred between the ground surface triangulation and certain subsurface triangulations. The reason for this problem was diagnosed as being an incorrect assumption made about the topographic properties of the geological outcrop region objects. The assumption made is that the edges from which these objects are made have special influence on the topography of a surface, in much the same way as the edges defining a ridge influence ground surface topography. Therefore, in the algorithms used to date, the topographic surface is forced to conform to edge topography when edges are inserted as constraints. In other words, the edge data assume greater significance than the surface data. Figure 9.10 shows how this assumption can affect ground surface and subsurface spatial integrity. It is suggested here that the edges from which a geological outcrop boundary are made up to do not possess such topographic significance and therefore, when inserted into the surface triangulation, they should be forced to conform to surface topography.

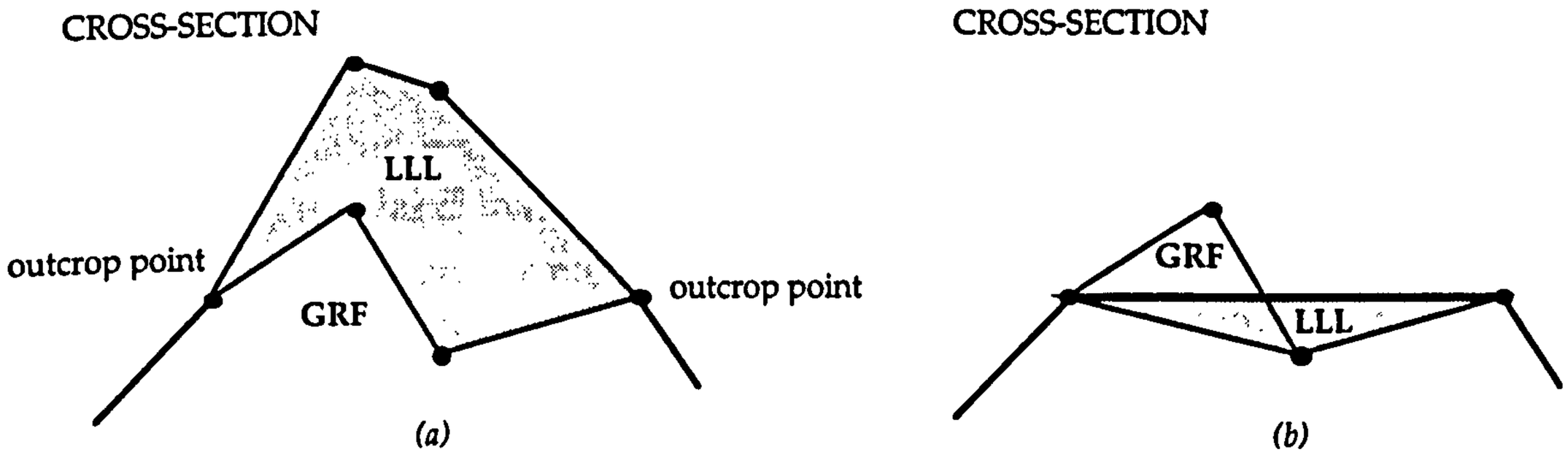


Figure 9.10 - The effect of forcing the ground surface to conform to geological outcrop region object constraints. (a) The correct cross section. (b) Conflict resulting from the insertion of constraint.

A constraining algorithm which adopts a more surface conforming approach is given by Kraak and Gazdzicki [69]. The method addresses the problem of inserting an edge, defined by its end points p_1 and p_2, into an existing triangulation. It is assumed that the points are already part of the triangulation. The algorithm begins by testing if the edge already exists. If so, the algorithm terminates. If the edge does not exist the next step is to generate a dummy point p_d with x and y coordinates taken as the mid-way point between p_1 and p_2. The z value of p_d is interpolated using the current surface triangulation. The algorithm is now repeated, in turn for edge (p_1, p_d) and edge (p_d, p_2). The recursive procedure repeats itself until eventually p_1 and p_2 are connected by a series of one or more co-linear edges.

This surface conforming algorithm now forms part of the MGD triangulation library and is used as part of the model creation process. When triangulating the ground surface, any outcrop polygon edge inserted as a constraint is inserted as a surface conforming edge. Dummy points, created during this process are stored in the Outcrop Points File. They are distinguished from other points by being assigned negative point identifiers. It is also necessary to update the line descriptions in the Outcrop Lines File. It is should be noted that a dummy point is only relevant to the level at which it was created. Therefore when reconstructing a line at a particular level of significance, all points which have negative identifiers and are at a higher level in the database should be ignored. The triangulation of subsurfaces reverts to the edge conforming constraining algorithm. This is to ensure that there is an exact match between the meeting points of the ground surface and subsurface.

The Kraak and Gazdzicki algorithm, while appearing to solve the problems caused by the test data sets, does not insert edges which conform absolutely to the triangulated surface. It does however produce a constrained triangulation which more closely conforms to the original triangulation than the previously used algorithm (see Section

2.4.4). An algorithm which inserts constraints which conform precisely to the surface has also been implemented. The algorithm is illustrated in Figure 9.11. Consider an edge (p_a, p_b) to be inserted into an existing triangulation T. It is assumed that p_a and p_b already belong to T and that the edge (p_a, p_b) does not exist. It follows that the proposed edge makes one or more intersections with existing triangle edges. Let the number of intersections be n and dummy points $p_1 \ldots p_n$ be created. The height value of each dummy point is interpolated from the current triangulation. It is now possible to insert the series of edges (p_a, p_1), (p_1, p_2),....., (p_{n-1}, p_n) and (p_n, p_b) into the triangulation, thus forming a series of co-linear , surface conforming edges between p_a and p_b. A disadvantage incurred by this method, when compared to the Kraak algorithm, is an increase in the number of dummy points.

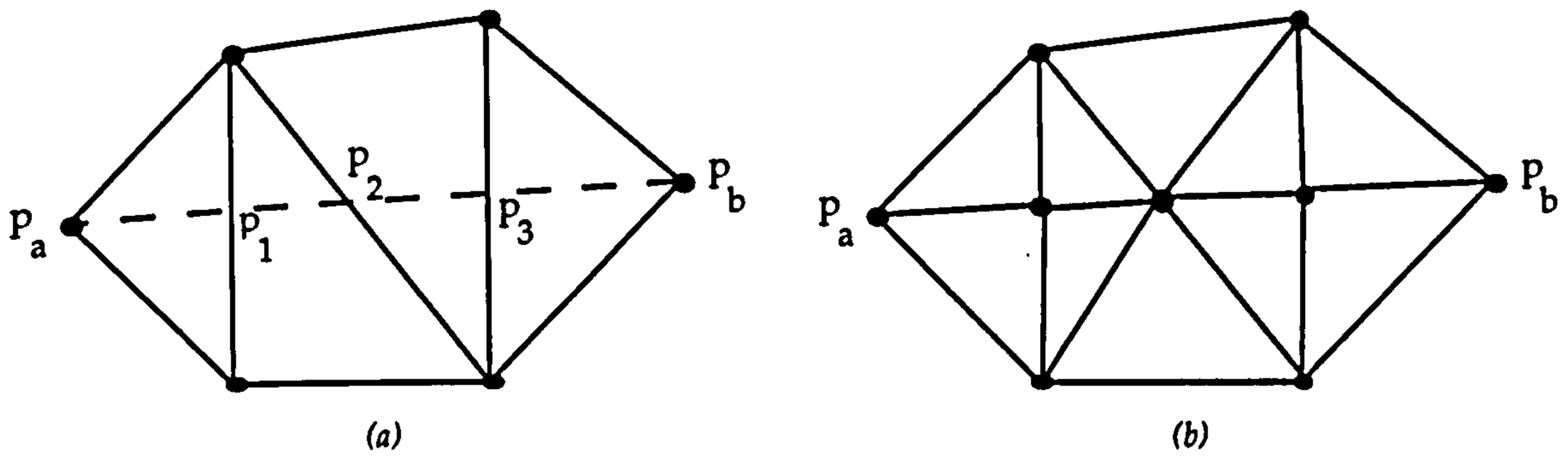


Figure 9.11 - The insertion of a surface conforming constraint. (a) Triangulation before insertion. (b) Triangulation after insertion.

9.6 Summary and Conclusions.

This chapter has described a successful implementation of the MGM. Three types of data, namely, terrain data, outcrop data and borehole data, have been combined to form a multi-scale 3-D triangulation-based geological model, which is stored efficiently in an ISAM database. The use of outcrop objects as constraints in both ground surface and subsurface triangulations, thus forming an accurate definition of where each subsurface intersects the ground surface, appears to offer a novel, and accurate, means by which geology can be represented. This technique is of equal benefit when applied in either a single-scale or multi-scale environment. The multi-scale aspect of the model also offers new and useful tools for the geologist. Some of the issues concerning the modelling of multi-valued surfaces have also been addressed. It is thought that a method based on the segmentation of data into locally single-valued patches offers the best prospect for future success with regards obtaining a solution to this problem.

Chapter 10

Thesis Summary and Conclusions

10.1 Introduction.

This thesis has described the research undertaken in order to develop and implement a number of data storage schemes suited to the efficient multi-scale representation of integrated spatial data. In this concluding chapter, Section 10.2 begins with a summary of the research, outlining both the benefits derived and conclusions reached. Section 10.3 then provides suggestions as to how the work might be progressed in the future. Final remarks are made in Section 10.4.

10.2 Project Summary and Achievements.

The overall aim of the work reported in this thesis can be summarised as the design and implementation of data storage schemes, or data models, suited to the efficient multi-scale representation of integrated spatial data. In addition to model design, the fulfilment of this aim has involved the development of methods by which the models can be created. These intentions were borne out of an observation that current GIS, and their geological counterpart GSIS, are poor in terms of offering data modelling facilities which include the integration of data of different types, and that they cater for data at only a single-scale.

In Chapter 1 the author has attempted to demonstrate the limitations of the various approaches adopted by current systems and has given reasons as to why the use of integrated multi-scale methods will be of benefit. In particular, the thesis has concentrated on finding solutions to two specific problems. The first, relating to GIS, has involved providing an efficient means of storing a combination of terrain data and topographic feature data at multiple scales. This data has been assumed to be spatially extensive. The second problem concerned the efficient multi-scale representation of 3-D geological data, consisting of terrain, outcrop and borehole data. Again, the assumption is that this data is spatially extensive. Solutions to both problems, in the form of a number of data model designs and database implementations, have been found and have been reported in the thesis. Details of publications relating to this work are given in Appendix 3.

10.2.1 Combining Terrain and Topographic Data at Multiple Scale.

Initially the work concerned itself with the integration and multi-scale representation of the two geographic data types, namely, topographic feature data and terrain data. Details regarding this part of the project are found in Chapters 2 to 7. It has been noted that GIS are currently restricted to the separate representation of topographic data and terrain data, while the representation of data at various scales can only be accommodated by adopting a multiple version approach. By comparison, an integrated multi-scale representation was thought from the outset of the project to have several distinct advantages, and be of benefit to a variety of disciplines. These

advantages have been outlined in Chapter 1. Chapters 2, 3 and 4 have provided an overview of currently available data structures suited to the storage of the data types being considered. These chapters have revealed that while some current data structures meet certain of the requirements of the proposed data storage scheme, none meet all the requirements. Following the review it was decided that the best approach to meeting the full requirements of the proposed scheme would be to adapt and merge those current data structures which appeared most useful in such a way as to form the integrated multi-scale data model. This resulted in three such data structures, namely, the line generalisation tree, the constrained Delaunay pyramid (CDP) and the fixed grid, being adopted for use as the basis for what has been termed the Multiresolution Topographic Surface Model (MTSM), details of which are given in Chapter 5.

The MTSM is a new data storage scheme which offers efficient multi-scale storage of, and access to, topographic data and terrain data. Data integration is achieved by embedding the line features making up topographic objects as constraints within a CDP. Line definitions are stored in a series of line generalisation trees. Efficient spatial access is provided by means of an adaption of the fixed grid method. The MTSM is created by means of a construction algorithm which is based on a combination of the Douglas-Peucker line simplification algorithm and De Floriani's error-directed constrained Delaunay triangulation algorithm. Chapter 5 also provides a brief argument in justification of the use here of the Douglas-Peucker algorithm. This is included in answer to a series of comments in recent papers which have shed doubt on the suitability of the Douglas-Peucker algorithm to line generalisation.

Two prototype database systems, MTSD 1.0 and MTSD 2.0, based on the MTSM design have been implemented. Details regarding these implementations are found in Chapter 6. The first implementation, MTSD 1.0, adopts the relational database ORACLE as its primary data storage medium, while MTSD 2.0 makes use of an ISAM file handling library. The performance of both systems has been evaluated in a series of tests. These tests have served to demonstrate the viability of the MTSM and its accompanying construction algorithm. System testing has also emphasised the relative advantages and disadvantages of adopting a multi-scale approach as opposed to either generalisation at run-time or multiple representation. The generalisation at run-time method has been shown to offer data storage benefits when compared to MTSM, but at the expense of an increase in data retrieval time. Conversely, the multiple version approach provides a quicker response to queries than MTSM, but suffers in that it incurs an increase in data storage.

Chapter 7 begins by describing a new version of the Implicit TIN data storage scheme. This new version offers improved performance when compared to the old in that it allows for the insertion of constraining segments and guarantees to produce the same

triangles as would be produced by a conventional constrained Delaunay triangulation algorithm. Included as part of the new Implicit TIN is a new constrained Delaunay triangulation algorithm, specifically designed for use in the Implicit TIN environment. Advantage is taken of these improvements to the Implicit TIN when it is applied to an Implicit version of the MTSD. The Implicit system, termed I_MTSD, offers considerable savings in storage when compared to MTSD 2.0, at the cost of only a slight increase in query response time. A further advantage of the Implicit TIN approach is that it offers flexibility with regard to the choice of which constraining objects are included in a particular triangulation.

10.2.2 Combining Geological Data Types at Multiple Scale.

Chapters 8 and 9 are concerned with extending the previously described work into 3-D and applying it to geological data. These chapters are related to the 3-D GIS project currently being carried out at BGS. Chapter 8, after giving a brief introduction to the topic of 3-D GIS, or GSIS, proposes a design for the ideal GSIS data model. The proposal suggests that a GSIS data model should be able to accommodate objects derived from various sources of data, allow for the multi-scale representation of these objects and provide spatial indexing on objects. It is also necessary to include algorithms for creating the model. The proposal suggests that two fundamental object types need to be accommodated, namely, hard-objects and soft-objects. It is put forward that hard-objects can adequately be catered for using a boundary representation method, while the octree method would appear suited to storing soft-objects. The model creation algorithms would need to provide facilities for object generalisation, integration of data from different sources and be able to cope with sparse data representing complex objects. It is also proposed that spatial indexing can be provided for by adopting an octree approach.

Chapter 8 concludes with the description of a prototype multi-scale 3-D model design, the MGM, which integrates topographic data, terrain data and subsurface elevation data. The MGM, an extension of the MTSM, uses a series of CDPs to provide multi-scale storage of, and access to, ground surface and subsurface horizons. A particularly useful, and novel, aspect of this work is the well defined relationship which exists between the subsurface horizons and the ground surface. This is made possible by the inclusion of geological outcrop objects as constraints in both the ground surface and subsurface pyramids, thus creating common edges at those places where subsurface horizons intersect the ground surface. A method for model creation is also described and includes new techniques by which faults lines, present in the ground surface, can be automatically extrapolated into the subsurface, thus producing a more accurate model.

Chapter 9 reports the details of an ISAM implementation of the prototype 3-D model.

The implementation, termed the MGD, has been successfully tested using data obtained from BGS. As was the case with the MTSD systems, comparison tests have shown that the MGD system performs as would be expected when compared to equivalent generalisation at run time and multiple version database systems. The multiple version approach offers a better response to queries than MGD, at the expense of an increase in storage costs. The MGD has been shown to provide faster query response times than a generalisation at run time system, but this is countered by an increase in storage costs. Note that the MGD system is limited in that surfaces can only be single-valued with regards the xy-plane. However, this has not proven to be a problem when modelling the BGS data set. Nevertheless, Chapter 9 has included some suggestions as to how multi-valued surfaces may be supported in the future.

10.3 Future Work.

What is apparent at the conclusion of this research project is the wide scope for future development of the work. Due to time constraints, many avenues of potential interest that have been uncovered during the project have not been fully pursued. With this in mind, this section gives suggestions for further work. The list of suggestions does not include all the ideas that have arisen but confines itself to those areas which the author thinks warrant most immediate attention.

10.3.1 Scale and Generalisation.

It is noted that in each of the new data models described in this thesis, points are selected for inclusion at a particular level depending on either their importance in describing a surface or their significance in describing a particular topographic object. Thus terrain data and subsurface elevation data points are selected according to their vertical displacements relative to the surface to which they belong, while topographic object points may also be selected on the basis of their lateral displacement. In the former case, point selection is achieved as the result of applying an error-directed point insertion triangulation algorithm, which can be regarded as the 2.5-D equivalent of the Douglas-Peucker algorithm, while in the latter case points are selected by applying a conventional Douglas-Peucker algorithm. It would seem that a logical progression, particularly in the context of the geological data model, would be to develop a 3-D version of the Douglas-Peucker (or any better 2-D generalisation) algorithm, in which points are selected on the basis of their overall contribution to the shape of the surface or object to which they belong.

The issues concerning the automated generalisation of 2-D map data are currently receiving much attention in the literature [44, 45, 46, 47, 48]. It is apparent that much work is still to be done before solutions to all the problems associated with this task can be said to have been solved. The multi-scale models described in this thesis each

provide efficient storage of, and access to, generalised versions of source data. The generalised versions are geometric subsets of the original data and therefore data duplication, at least for line feature and surface representation, is avoided. Note that with regards topographic data this thesis has restricted itself to automated line simplification. It is hoped that in the future the MTSD and MGD systems will incorporate additional topographic data generalisation capabilities, such as, selection, exaggeration, combination, displacement and symbolisation, relating to polygon and point features, as well as to lines. Work is currently being undertaken at the University of Glamorgan, as part of another PhD project, which is seeking to develop functions which perform certain of these tasks [121, 48]. The generalisation techniques adopted involve the map data undergoing generalisation operators that are applied to constraints within a TIN model. There is a definite connection between the two projects and obvious scope for inclusion of the more advanced generalisation operations within the multi-scale database systems described in this thesis. However, it must be noted that when applying operations such as exaggeration, displacement and symbolisation to assist in large changes in scale a multi-scale approach is inappropriate. This is because the derived, smaller scale version is no longer, in a geometric sense, a subset of the larger scale version. This does not however rule out the use of multi-scale data structures for separately representing both the smaller and larger scale versions across their different scale ranges. Such an approach is adopted in the multi-scale database suggested by Jones [122].

Throughout the thesis the terms 'scale', 'resolution', 'generalisation level', and other similar expressions, have been used interchangeably and have assumed the same meaning, that is, a measure of the amount of detail present in a particular set of data. This measure, as far as the data models presented in Chapters 5 to 9 are concerned, has been in terms of two error tolerance values, namely, vertical error and lateral error. A criticism of the work might be that no attempt has been made to relate these error tolerances to what professional geographers and geologists regard as scale. The author believes that if such relationships could be incorporated into the generalisation algorithms used in the model creation processes they would add considerably to the usefulness of the models. With regards the generalisation of topographic line data, attention is drawn to work reported by Abraham [52] regarding the establishment of a direct relationship between scale and the error-tolerance value supplied to the Douglas-Peucker algorithm. The approach used was to initially establish a link between error tolerance and the number of points selected by the Douglas-Peucker algorithm. This link was then used in conjunction with an adapted version of Topfer's Radical Law [123], which predicts the number of data objects at a derived scale given the number of objects at the source scale, to provide a relationship between Douglas-Peucker error tolerance and scale. Note that this approach to topographic generalisation is limited in that it views generalisation as purely a geometric process

whereas in reality it is a highly conceptual process. However, the MTSD and MGD systems described in this thesis, in their present form, would benefit by the inclusion of this relationship (and, perhaps, a vertical error tolerance equivalent) since they are restricted to geometrical generalisation.

An area of automated generalisation which does not appear to have received much attention in the literature is that which concerns geological data. There appears to be scope for a great deal of future research to be carried out in this area. Note that much of the generalisation reported in the literature is specifically geared towards 2-D geographic data. This work cannot readily be applied to the generalisation of 2-D geological data (that is, outcrop data) since conventions differ between the generalisation of geographic map data and the generalisation of geological map data. For example, the generalisation functions required to generalise a collection of line data representing a network of roads will be different from those required to generalise a collection of lines representing a series of faults, such as the en echelon faults shown in Figure 10.1. The author hopes that some of the issues relating to the generalisation of geological map data will be addressed in the near future.

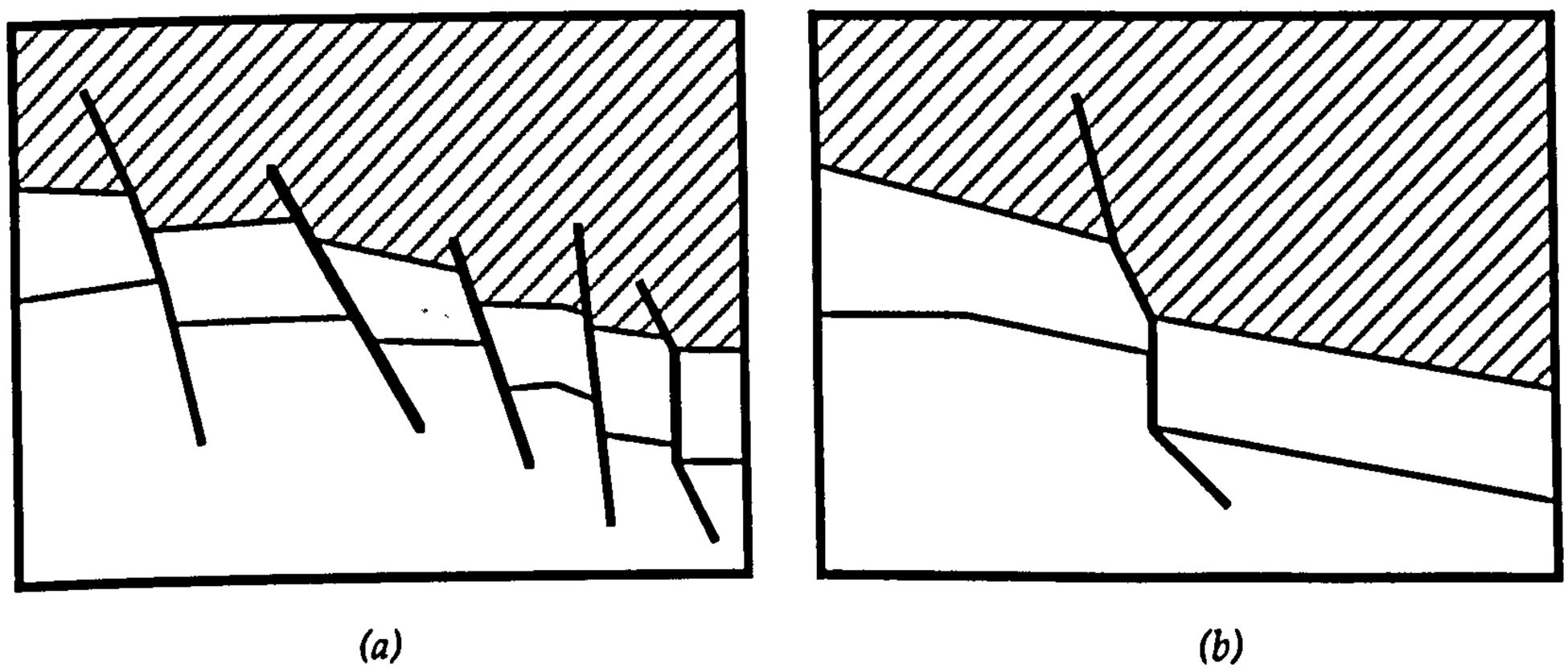


Figure 10.1 - The generalisation of a geological map which includes en echelon faults. Such generalisation is only applicable to geological data. *(a) Map prior to generalisation. (b) Map after generalisation.*

An aspect of geological generalisation which could be built into the MGD system with relative ease is that which concerns the classification of subsurface horizons according to their relevance to a particular scale. Geologists regard certain subsurface horizons as being of greater importance than others. A useful tool to geologists would be the ability to retrieve horizons according to their semantic importance. This could be accommodated in the MGD system by introducing a subsurface information file which records the range of database levels over which each subsurface horizon is significant. Queries involving a subsurface horizon at a particular level of detail would now need

to be validated, by checking if the required detail level lies within the specified range of levels, before being processed further. This mechanism has not been incorporated into the current version of the MGD system due to a lack of appropriate test data. It is the intention to include the facility in future versions.

10.3.2 The Modelling of 3-D objects.

A great deal of work remains to be done with regards the creation of the ideal GSIS data model. Perhaps the greatest limitation of the MGD presented in Chapter 9 is its inability to cope with multi-valued surfaces. Several techniques have been suggested, the further investigation of which would appear to be a priority for any future work. The data segmentation method is believed to offer the best chances of success. Other limitations of the MGD, such as the absence of 3-D spatial indexing and the exclusion of soft-object types, are a direct result of time constraints on the project. It is the intention to include these features in future implementations.

Looking further ahead, it is suggested that a solution to the full automation of the model creation process would benefit from the introduction of Artificial Intelligence techniques. This suggestion appears to be vindicated by recently published literature concerned with the use of knowledge-bases in the management and analysis of 3-D geological data [115, 124, 125]. A knowledge-base concerned with assisting in the model creation process might be expected to contain information such as physical geological observations concerning the area being modelled (such as dip measurements), any knowledge the geologist may have about the data being modelled (that it represents a salt dome, for example) and stereotypical surface shapes (to which the data in question can be matched). This knowledge will be examined during model creation and a suggestion as to how modelling can best proceed will be produced. For example, knowing that a particular data set represents a salt-dome, perhaps due to a pattern match, could lead to the suggestion of generating a vertical plane of symmetry through the data, triangulating the two sub-sets thus formed (by projecting each, in turn, onto the plane of symmetry) and then 'zipping' the two triangulations together to form the complete surface.

The method by which faults are extrapolated into the subsurface is also limited in the MGD system in that they require a prior knowledge of dip, throw and depth information. A means by which this information could be derived from source data would enhance the fault modelling capabilities of any future system. Mention is made of work reported by Walsh and Watterson [126, 127] in which methods for estimating the throw and dip of certain types of fault are described. It is also true that the characteristics of a fault are related to the type of rock in which it occurs. Another observation is that when no other information is available, geologists often apply standard characteristics when modelling a fault. For example, faults are usually

assumed to have a dip of about 70° in the absence of any conflicting evidence. Note also that when sampling is dense enough, subsurface elevation data itself can be used to identify the presence of faults. It can be imagined that by combining these various methods and sources of information in a knowledge-based modelling system a more fully automated, and improved, method of extrapolating fault data into the subsurface could be achieved.

10.4 Final Remarks.

There can be no doubt that spatial information systems will continue to have a great impact in both geography and geology. This thesis has based itself on the belief that such systems will benefit from data models which integrate all available data, and provide multi-scale representation of this data. It is hoped that this thesis has brought attention to these potential benefits and has gone a little way to providing data storage schemes, and associated model construction methods, that incorporate such benefits.

References

References

[1] **Maguire, D.J. 1991** "An overview and definition of GIS", Geographical Information Systems (book), Vol. 1, Longman Scientific and Technical publishers, pp. 9 - 20.

[2] **Turner, A.K. 1991** "Three-Dimensional Modelling with Geoscientific Information Systems : preface", Three-Dimensional Modelling with Geoscientific Information Systems (book), Kluwer Academic publishers, pp. 3 - 5.

[3] **Peucker, T. and Chrisman, N. 1975** "Cartographic data structures", American Cartographer, Vol. 2, No. 2, pp. 55 - 69.

[4] **Van Kuilenburg, 1981** "Segment encoding for the production of low-cost interpretation maps", ISSS Working Group on Soil Information Systems Newsletter, No 8, September 1981.

[5] **Bondriault, G. 1987** "Topology in the TIGER file", Proceedings of AutoCarto 8, March 1987, pp. 258 - 269.

[6] **Kinnear, C. 1987** "The TIGER structure", Proceedings of AutoCarto 8, March 1987, pp. 249 - 257.

[7] **Peucker, T.K. 1978** "Data structures for digital terrain models : discussion and comparison", Harvard Papers on Geographic Information Systems, First International Advances Study Symposium of on Topological Data Structures for Geographic Information Systems, Vol. 5.

[8] **Peucker, T.K. et al 1978** "The triangulated irregular network", Proceedings of the Digital Terrain Models (DTM) Symposium, May 1978, pp. 516 - 540.

[9] **Gold, C.M. 1979** "Triangulation-based terrain modelling - where are we now ?", Proceedings of AutoCarto 4, Vol. 2, pp. 104 - 111.

[10] **Preparata, F.P. and Shamos, M.I. 1985,** Computational Geometry : An Introduction (book), Springer-Verlag (New York) publishers.

[11] **Lee, D.T. and Shacter, B.J. 1980** "Two algorithms for constructing a Delaunay triangulation", International Journal of Computer and Information Sciences, Vol. 9, No. 3, pp. 219 - 242.

[12] **De Floriani, L. 1987** "Surface representations based on triangular grids", Visual Computer, Vol. 3, No. 1, pp. 27 - 50.

[13] **Woo, T.C. 1985** "A combinational analysis of boundary data structure schemata", IEEE Computer Graphics and Applications, Vol. 5, No. 3, pp. 19 - 27.

[14] **Watson, D.F. 1981** "Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes", The Computer Journal, Vol. 24, pp. 167 - 171.

[15] **De Floriani, L. 1989** "A pyramidal data structure for triangle-based surface description", IEEE Computer Graphics and Applications, March 1989, pp. 67 - 78.

[16] **Larkin, B.J. 1991** "An ANSI C program to determine in expected linear time the vertices of the convex hull of a set of planar points", Computers and Geosciences, Vol. 17, No. 3, pp. 431 - 443.

[17] **Devijver, P.A. and Maybank, S. 1982** "Computation of the Delaunay triangulation of a convex polygon with a minimum space complexity constraint", Proceedings of the 6th International Conference on Pattern Recognition, Munich (Germany), pp. 420 - 422.

[18] **Christiansen, A. 1987** "Fitting a triangulation to contour lines", Proceedings of Auto-Carto 8, Baltimore (USA), March 1987, pp. 57 - 67.

[19] **Chew, L.P. 1987** "Constrained Delaunay triangulation", Proceedings of the 3rd ACM Symposium on Computational Geometry, June 1987, pp. 216 - 222.

[20] **Wang, C.A. and Schubert, L. 1987** "An optimal algorithm for constructing the Delaunay triangulation of a set of line segments", Proceedings of the 3rd ACM Symposium on Computational Geometry, June 1987, pp. 223 - 232.

[21] **Heller, M. 1990** "Triangulation algorithms for adaptive terrain modelling", Proceedings of the 4th International Symposium on Spatial Data Handling, Zurich (Switzerland), Vol. 1, pp. 163 - 174.

[22] **De Floriani, L. and Puppo, E. 1988** "Constrained Delaunay triangulation for multiresolution surface description", IEEE Computer Society Reprint (reprinted from Proceedings of the 9th IEEE International Conference on Pattern Recognition, November 1988).

[23] **Samet, H. 1989,** The Design and Analysis of Spatial Data Structures (book), Addison-Wesley publishing company.

[24] **Samet, H. 1989,** Applications of Spatial Data Structures (book), Addison-Wesley publishing company.

[25] **Faloutsos, C., Sellis, T. and Roussopoulos, N. 1987** "Analysis of object oriented spatial access methods", Proceedings of the SIGMOD Conference, May 1987, San Francisco (USA), pp. 426 - 439.

[26] **Smith, T.R. and Gao, P. 1990** "Experimental performance evaluations on spatial access methods", Proceedings of the 4th International Symposium on Spatial Data Handling, Zurich (Switzerland), Vol. 2, pp. 991 - 1002.

[27] **Bentley, J.L. and Friedman, J.H. 1979** "Data structures for range searching", ACM Computing Surveys, Vol. 11, No. 4, pp. 397 - 409.

[28] **Franklin, W.R. and Akman, V. 1988** "An adaptive grid for polyhedral visibility in object space: an implementation", Computer Journal, Vol. 31, No. 1, pp. 56 - 60.

[29] **Klinger, A. 1971** "Patterns and search statistics", Optimising Methods in Statistics (book), Academic Press (New York) publishers, pp. 303 - 307.

[30] **Samet, H. 1984** "The quadtree and related hierarchical data structures", ACM Computing Surveys, Vol. 16, No. 2, pp. 187 - 260.

[31] **Klinger, A. and Dyer, C.R. 1976** "Experiments in picture representation using regular decomposition", Computer Graphics and Image Processing, Vol. 5, No. 1, pp. 68 - 105.

[32] **Schneier, M. 1981** "Two hierarchical linear feature representations: edge pyramids and edge quadtrees", Computer Graphics and Image Processing, Vol. 17, No. 3, pp. 211 - 224.

[33] **Martin, J.J. 1982** "Organisation of geographical data with quadtrees and least square approximation", Proceedings of the IEEE Conference on Pattern Recognition and Image Processing, June 1982, pp. 458 - 463.

[34] **Samet, H. and Webber, R.E. 1985** "Storing a collection of polygons using quadtrees", ACM Transactions on Graphics, Vol. 4, No. 3, pp. 182 - 222.

[35] **Nelson, R.C. and Samet, H. 1986** "A consistent hierarchical representation of vector data", Computer Graphics, Vol. 20, No. 4, pp. 197 - 206.

[36] **Gargantini, I. 1982** "An effective way to represent quadtrees", Communications of the ACM, Vol. 25, No. 12, pp. 905 - 910.

[37] **Diaz, B. and Bell, S.B.M. (editors) 1986,** Spatial Data Processing Using Tesseral Methods (Natural Environmental Research Council, UK).

[38] **Morton, G.M. 1966** "A computer oriented geodetic database and a new technique in file sequencing", IBM Ltd, Ottawa (Canada).

[39] **Guttman, A. 1984** "R-trees: a dynamic index structure for spatial searching", Proceedings of the SIGMOD Conference, June 1984, pp. 47 - 57.

[40] **Comer, D. 1979** "The ubiquitous B-tree", ACM Computing Surveys, Vol. 11, No. 2, pp. 121 - 137.

[41] **Stonebraker, M., Sellis, T. and Hanson, E. 1986** "An analysis of rule indexing implementations in database systems", Proceedings of 1st International Conference on Expert Database Systems, Charleston (USA), pp. 353 - 364.

[42] **Nievergelt, J., Hinterberger, H. and Sevcik, K.C. 1984** "The grid file: an adaptable, symmetric multikey file structure", ACM Transactions on Database Systems, Vol. 9, No. 1, pp. 38 - 71.

[43] **Monmonier, M.S. 1982,** Computer-assisted cartography: principles and prospects (book), Prentice-Hall publishers, New Jersey (USA).

[44] **Brassel, K.E. and Weibel, R. 1988** "A review and conceptual framework of automated map generalisation", International Journal of Geographic Information Systems, Vol. 2, No. 3, pp. 229 - 244.

[45] **Beard, K.M. 1991** "Theory of the cartographic line revisited: implications for automated generalisation", Cartographica, Vol. 18, No. 4, pp. 32 - 58.

[46] **Buttenfield, B.P. and McMaster, R.B. 1991,** Map generalisation: making rules for knowledge representation (book), Longman (UK) publishers.

[47] **Muller, J. and Wang, Z. 1993** "Area patch generalisation: a competitive approach", The Cartographic Journal, Vol. 29, No. 3, pp. 137 - 144.

[48] **Bundy, G.Ll., Jones, C.B. and Furse, E.** "Holistic generalisation of large scale cartographic data", European Science Foundation Scientific Programme on Geographical Information Systems Data Integration and Database Design (Specialist Meeting on Generalisation), Compeiegne (France), December 1993. Proceeding to be published in 1994.

[49] **McMaster, R.B. 1983** "A mathematical evaluation of simplification algorithms", Proceedings of AutoCarto 6, Vol. 2, pp. 267 - 276.

[50] **McMaster, R.B. 1987** "Automated line generalisation", Cartographica, Vol. 24, No. 2, pp. 74 - 111.

[51] **Zoraster, S., Davis, D. and Hugus, M. 1984** "Manual and automated line generalisation and feature displacement - appendices", Engineering Topographic Laboratories, Fort Belvoir, Virginia (USA), June 1984.

[52] **Abraham, I.M. 1988** "Automated cartographic line generalisation and scale-independent databases", Ph.D Thesis, Department of Computer Studies, The University of Glamorgan (UK).

[53] **Tobler, W.R. 1964** "An experiment in the computer generalisation of maps", Technical Report, Department of Geography, The University of Michigan (USA).

[54] **Harris, K.M. 1981** "Algorithms for line simplification and smoothing and their applicability to geographical data", Research Paper, University of Kansas (USA), January 1981.

[55] **Douglas, D.H. and Peucker, T.A. 1973** "Algorithms for the reduction of the number of points required to represent a digitised line or its caricature", Canadian Cartographer, Vol. 10, No. 2, pp. 112 - 122.

[56] **Harris, K.M. 1981** "Use of the Douglas-Peucker line generalisation algorithm for characteristic point selection", Research Paper, Geography 980, University of Kansas (USA), May 1981.

[57] **Jones, C.B. 1984** "A tree data structure for cartographic line generalisation", Proceedings of Eurocarto 3, Graz (Austria).

[58] **Jones, C.B. and Abraham, I.M. 1986** "Design considerations for a scale-independent cartographic database", Proceedings of the 2nd International Symposium on Spatial Data Handling, Seattle (USA), pp. 384 - 398.

[59] **Jones, C.B. and Abraham, I.M. 1987** "Line generalisation in a global cartographic database", Cartographica, Vol. 24, No. 3, pp. 32 - 45.

[60] **Ballard, D.H. 1981** "Strip trees: a hierarchical representation for curves", Communications of the ACM, Vol. 24, pp. 310 - 398.

[61] **van Oosterom, P. and van den Boss, J. 1989** "An object-oriented approach to the design of geographic information systems", Proceedings of the 1st Symposium of Large Spatial Databases", July 1989, pp. 255 - 269.

[62] **van Oosterom, P. 1990** "Reactive data structures for geographic information systems", Ph.D Thesis, Department of Computer Science, Lieden University (Netherlands).

[63] **Schmitt, F.J.M. and Gholzadeh, B. 1985** "Adaptive polyhedral approximation of digitised surfaces", Proceedings of SPIE - Computer Vision for Robots 595, Cannes (France), December 1985, pp. 101 - 108.

[64] **Chen, Z.T. and Tobler, W.R. 1986** "Quadtree representations of digital terrain", Proceedings of AutoCarto London, Vol. 1, pp. 475 - 484.

[65] **De Floriani, L. et al 1984** "A hierarchical structure for surface approximation", Computer Graphics, Vol. 8, pp. 183 - 193.

[66] **Gomez, D. and Guzman, A. 1979** "Digital model for three-dimensional surface representation", Geo-Processing, Vol. 1, pp. 53 - 70.

[67] **Barrera, R. and Vazques, A.M. 1984** "A hierarchical method for representing relief", Proceedings of Pecora 9 Symposium on Spatial Information Technologies for Remote Sensing Today and Tomorrow, South Dakota (USA), pp. 87 - 92.

[68] **Scarlatos, L. and Pavlidis, T. 1991** "Adaptive hierarchical triangulation", Proceedings of AutoCarto 10, pp. 234 - 246.

[69] **Kraak, M.J. and Gazdzicki, J. 1991** "Triangulation based modelling of spatial objects in relation to the terrain surface", Proceedings of the 2nd European Conference on Geographical Information Systems, Brussels (Belgium), pp. 564 - 572.

[70] **McCullagh, M.J. and Ross, C.G. 1980** "Delaunay triangulation of a random data set for isarithmic mapping", The Cartographic Journal, Vol. 17, pp. 93 - 99.

[71] **Franklin, W.R. 1983** "Adaptive grids for geometric operations", Proceedings of Auto-Carto 6, pp. 230 - 239.

[72] **Abel, D.J. and Smith, J.L. 1983** "A data structure and algorithm based on a linear key for rectangular retrieval", Computer Vision, Graphics and Image Processing, Vol. 2, No. 24, pp. 1 - 13.

[73] **van Oosterom, P. 1991** "The Reactive-tree - a storage structure for a seamless, scaleless geographic database", Proceedings of Auto-Carto 10, Baltimore (USA), pp. 393 - 407.

[74] **Becker, B., Six, H.W. and Widmayer, P. 1991** "Spatial priority search: an access technique for scaleless maps", Proceedings of SIGMOD 1991, pp. 128 - 137.

[75] **Hutflesz, A., Six, H.W. and Widmayer, P. 1990** "The R-file: an efficient access structure for proximity queries", Proceedings of the IEEE 6th International Conference on Data Engineering, pp. 372 - 379.

[76] **Visvalingam, M. and Whyatt, J. 1990** "The Douglas-Peucker algorithm for line simplification: reevaluation through visualisation", Computer Graphics Forum, Vol. 9, No. 3, pp. 213 - 228.

[77] **Li, Z. and Openshaw, S. 1993** "A natural principle for the objective generalisation of digital maps", Cartography and Geographic Information Systems, Vol. 20, No. 1, pp. 19 - 29.

[78] **Li, Z. 1993** "Some observations on the issue of line generalisation", The Cartographic Journal, June 1991, pp. 68 - 71.

[79] **Wang, Z. and Muller, J. 1993** "Complex coastline generalisation", Cartography and Geographic Information Systems, Vol. 20, No. 2, pp. 96 - 106.

[80] **Monmonier, M. 1986** "Towards a practical model of cartographic generalisation", Proceedings of Auto-Carto London, pp. 257 - 266.

[81] **Thapa, K. 1988** "Automated line generalisation using zero crossings", Photogrammetric Engineering and Remote Sensing, Vol. 54, No. 4, pp. 511 - 517.

[82] **Schwarz, C.R. 1991** "Comment and discussion: the removal of spatial conflicts in line generalisation", Cartography and Geographic Information Systems, Vol. 18, No. 4.

[83] **Muller, J. 1990** "The removal of spatial conflicts in line generalisation", Cartography and Geographic Information Systems, Vol. 17, No. 2, pp. 141 - 149.

[84] **Dutton, G.H. 1981** "Fractal enhancement of cartographic line detail", American Cartographer, Vol. 8, No. 1, pp. 23 - 40.

[85] **Ware, J.M. and Jones, C.B. 1992** "A multiresolution topographic surface database", International Journal of Geographical Information Systems, Vol. 6, No. 6, pp. 479 - 496.

[86] **Kidner, D.B. and Jones, C.B. 1991** "Implicit triangulations for large terrain databases", Proceedings of the 2nd European Conference on Geographical Information Systems, Vol.1, Brussels (Belgium), pp. 537 - 546.

[87] **Kidner, D.B. 1991** "Digital terrain models for radio path loss calculations", Ph.D Thesis, Department of Computer Studies, The University of Glamorgan (UK).

[88] **Jones, C.B., Kidner, D.B. and Ware, J.M. 1994** "The Implicit TIN and multi-scale spatial databases", The Computer Journal, Vol. 37, No. 1, pp. 43 - 57.

[89] **El Gindy, H. 1990** "Optimal parallel algorithms for updating planar triangulations", Proceedings of the 4th International Symposium on Spatial Data Handling, Vol. 1, pp. 200 - 208.

[90] **Ware, J.A. and Kidner, D.B. 1991** "Parallel implementation of the Delaunay triangulation within a transputer environment", Proceedings of the 2nd European Conference on Geographical Information Systems, Vol. 2, Brussels (Belgium), pp. 1199 - 1208.

[91] **Jones, C.B. (1989)** "Data structures for three-dimensional spatial information systems in geology", International Journal of Geographic Information Systems, Vol.3, No.1, pp. 15 - 31.

[92] **Raper, J.F. (1989)** "The three-dimensional geoscientific mapping and modelling system: a conceptual design", Three-Dimensional Applications in Geographic Information Systems (book), Taylor and Francis publishers, pp. 11 - 19.

[93] **Youngmann, C. (1989)** "Spatial data structures for modelling subsurface features", Three-Dimensional Applications in Geographic Information Systems (book), Taylor and Francis publishers, pp. 129 - 136.

[94] **Loudon, T.V. (1986)** "Digital spatial models and geological maps", Proceedings of Auto-Carto London, Vol.2, pp. 60 - 65.

[95] **Lee, M.K. et al (1990)** "Three-dimensional integrated geoscience mapping progress report number 1", British Geological Survey Project E77HAR12, Note 90/17.

[96] **Bak, P.R.G. and Mill, A.J.B. (1989)** "Three-dimensional representation in a geoscientific resource management system for the minerals industry", Three-Dimensional Applications in Geographic Information Systems (book), Taylor and Francis publishers.

[97] **Rhind, D.W. (1989)** "Spatial data handling in the geosciences", Three-Dimensional Modeling with Geoscientific Information Systems (book), Kluwer Academic publishers, pp. 13 - 27.

[98] **Kelk, B. (1989)** "Three-dimensional modelling with geoscientific information systems: the problem", Three-Dimensional Modelling with Geoscientific Information Systems (book), Kluwer Academic publishers, pp. 29 - 37.

[99] **Dabek, Z.K. et al (1988)** "Development of advanced interactive computer modelling techniques for multicomponent three-dimensional interpretation of geophysical data", British Geological Survey Technical Report WK/88/2.

[100] **Unger, J.D. et al (1989)** "Creating a three-dimensional transect of the Earth's crust from craton to ocean basin across the N. Appalachian Orogen", Three-Dimensional Applications in Geographic Information Systems (book), Taylor and Francis publishers, pp. 137 - 148.

[101] **Requicha, A.A.G. (1980)** "Representations for rigid solids: theory, methods and systems", ACM Computing Surveys, Vol. 12, pp. 437.

[102] **Jones, T.A.** (1988) "Modelling geology in three-dimensions", Geobyte, February 1988, pp. 14 - 20.

[103] **Carlbom, I. and Chakravatry, I. 1985** "A hierarchical data structure for representing the spatial decomposition of three-dimensional objects", IEEE Computer Graphics and Applications, April 1985, pp. 24 - 31.

[104] **Carlson, E. 1987** "Three-dimensional conceptual modelling of subsurface structures", Technical Papers of the ASPRS-ACSM Annual Convention, Vol. 4, Baltimore (USA).

[105] **Burns, K.L. 1988** "Lithologic topology and structural vector fields applied to subsurface prediction in geology", GIS/LIS 1988 (ACM-ASPRS), San Antonio (USA).

[106] **Smith, D.R. and Paradis, A.R. 1989** "Three-dimensional GIS for the earth sciences", Three-Dimensional Applications in Geographic Information Systems (book), Taylor and Francis publishers, pp. 149 - 154.

[107] **Schek, H.J. and Waterfeld, W. 1986** "A database kernal system for geoscientific applications", Proceedings of the Second International Symposium on Spatial Data Handling, Seattle (USA), pp. 273 - 288.

[108] **Kavouras, M. and Masry, S.E. 1987** "An information system for geosciences : design considerations", Proceedings of Auto-Carto 8, Baltimore (USA), March 1987, pp. 336 - 345.

[109] **Navazo, I., Ayala, D. and Brunet, P. 1986** "A geometric modeller based on the exact octree representation of polyhedra", Computer Graphics Forum, Vol. 5, pp. 91 - 104.

[110] **Gargantini, I. 1982** "Linear octrees for fast processing of three-dimensional objects", Computer Graphics and Image Processing, Vol. 20, pp. 365 - 374.

[111] **Gargantini, I., Walsh, T.R. and Wu, O.L. 1986** "Viewing transformations of voxel-based objects via linear octrees", IEEE Computer Graphics and Applications, October 1986, pp. 12 - 21.

[112] **Ayala, D. et al 1985** "Object representation by means of non-minimal division quadtrees and octrees", ACM Transactions on Graphics, Vol. 4, No. 1, pp. 41 - 59.

[113] **Boissonnat, J-D. 1984** "Geometric structures for three-dimensional shape recognition", ACM Transactions on Graphics, Vol. 3, No. 4, pp. 266 - 286.

[114] **Fisher, T. and Wales, R.Q. 1991** "Three-dimensional solid modelling of geo-objects using non-uniform rational B-splines (NURBS)", Three-Dimensional Modelling with Geoscientific Information Systems (book), Kluwer Academic publishers, pp. 85 - 105.

[115] **Mallet, J-L. 1991** "GOCAD : a computer aided design program for geological applications", Three-Dimensional Modelling with Geoscientific Information Systems (book), Kluwer Academic publishers, pp. 123 - 141.

[116] **Christiansen, H.N. and Sederberg, T.W. 1978** "Conversion of complex contour line definitions into polygonal element mosaics", ACM Computer Graphics, Vol. 12, No. 3, pp. 187 - 192.

[117] **Riedinger, R. et al 1988** "About the the Delaunay-Voronoi tessellation", Journal of Computational Physics, Vol. 74, pp. 61 - 72.

[118] **Tanemura, M., Ogawa, T. and Ogita, N. 1983** "A new algorithm for three-dimensional Voronoi tessellation", Journal of Computational Physics, Vol. 51, pp. 191 - 207.

[119] **Devijver, P.A. and Dekesel, M. 1983** "Computing multi-dimensional Delaunay tessellations", Pattern Recognition Letters 1, July 1983, pp. 311 - 316.

[120] **Bundy, G.Ll., Jones, C.B. and Furse, E. 1994** "A topological structure for the generalisation of large scale cartographic data", to be presented at Geographical Information Systems Research UK (GISRUK) 2nd National Conference.

[121] **Jones, C.B., Ware, J.M. and Bundy, G.Ll. 1992** "Multi-scale spatial modelling with triangulated surfaces", Proceedings of the 5th International Symposium on Spatial Data Handling, Charleston (USA), Vol. 2, pp. 612 - 621.

[122] **Jones, C.B. 1991** "Database architecture for multi-scale Geographical Information Systems", Proceedings of Auto-Carto 10, Baltimore (USA), pp. 1-14.

[123] **Topfer, F. and Pillewizer, W. 1966** "The principles of selection", Cartographic Journal, Vol. 3, pp. 10 - 16.

[124] **Morris, K. 1991** "Using knowledge-base rules to map the three-dimensional nature of geological features", Photogrammetric Engineering and Remote Sensing, Vol. 57, No. 9, pp. 1209 - 1216.

[125] **Fisher, P.F. (guest editor) 1990** "Artificial intelligence applications in geoscience", Computers and Geosciences (Special Issue), Vol. 16, No. 6.

[126] **Walsh, J.J. and Watterson, J. 1988** "Dips of normal faults in British coal measures and other sedimentary sequences", Journal of the Geological Society (London), Vol. 145, pp. 859 - 873.

[127] **Walsh, J.J. and Watterson, J. 1990** "New methods of fault projection for coalmine planning", Proceedings of the Yorkshire Geological Society, Vol. 48, pp. 209 - 219.

Appendix 1

Line Generalisation and Spatial Conflict

A1.1 The Introduction of Spatial Conflict as a Result of Line Generalisation.

A problem common to many generalisation operations is that of a loss of topological integrity as a result of performing the operation. The Douglas–Peucker algorithm is not immune to this problem, particularly when a large reduction in the number of points is involved.

A common error which causes spatial conflict is that of self–crossing, where a line is forced to intersect itself as a result of removing one or more of its points (Figure A1.1). A more unusual error is when two portions of the same line become co–incident (Figure A1.2).

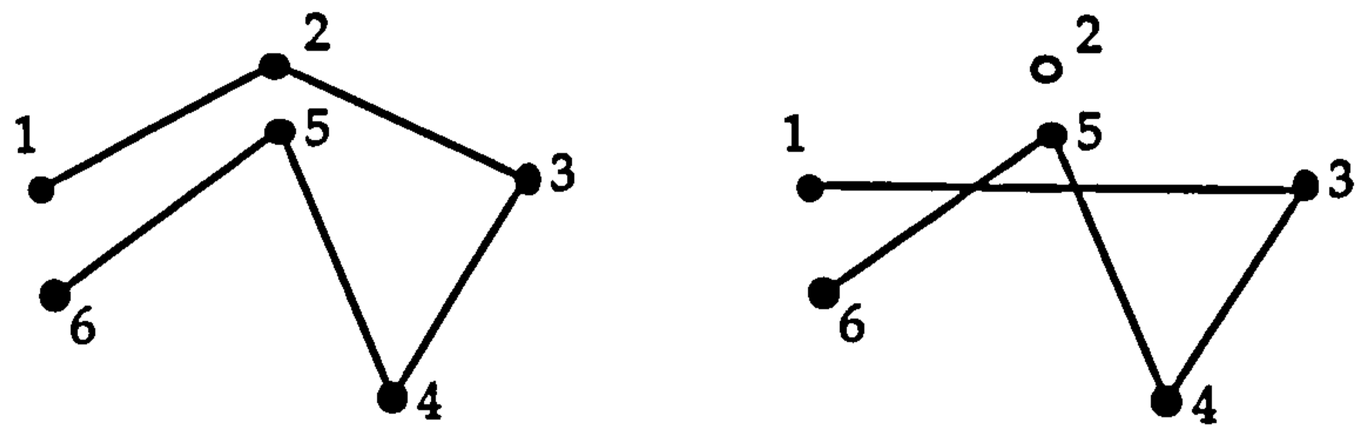


Figure A1.1 - Self-crossing as a result of generalisation.

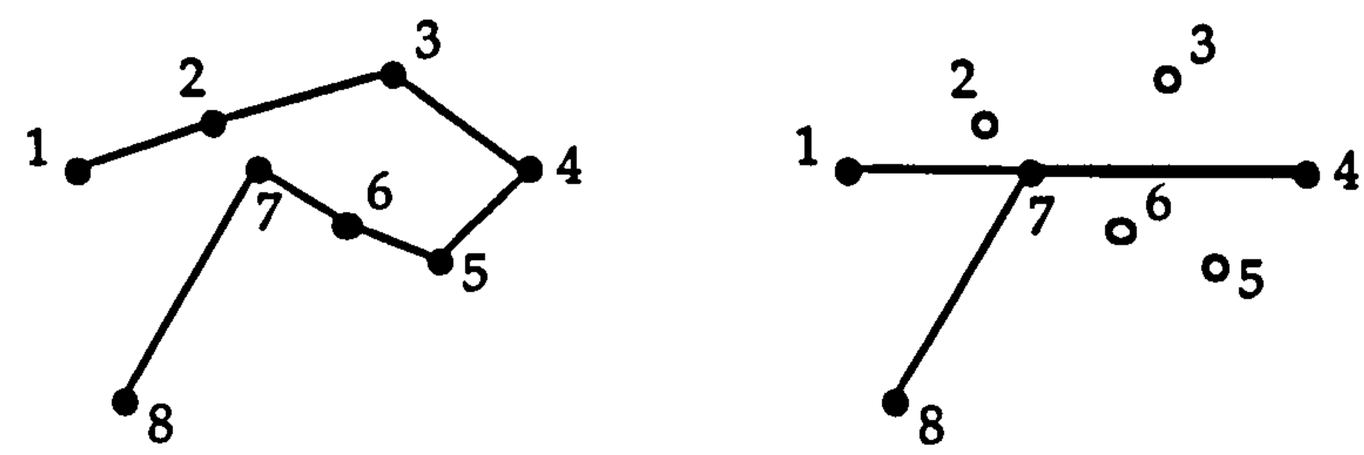
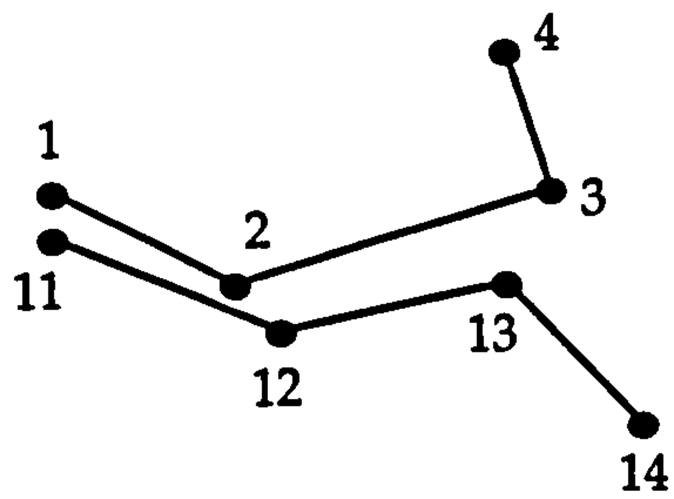


Figure A1.2 - Co-incidence as a result of generalisation.

When dealing with collections of lines there is also the risk of introducing invalid intersection of neighbouring lines (Figure A1.3) or neighbouring lines becoming co–incident (Figure A1.4). These errors are termed neighbour intersection and neighbour co-incidence errors respectively. If the lines being generalised represent a collection of polygons, neighbour intersection can result in overlapping polygons (Figure A1.5), while neighbour co-incidence can result in flat polygons (Figure A1.6).

Note that in this thesis generalisation is considered in a purely geometrical sense. No attempt at an automated solution to the more conceptual aspects of generalisation, such as feature selection, exaggeration, displacement and symbolisation, is being put forward. However, conceptual errors can occur as a result of applying the Douglas–Peucker algorithm.



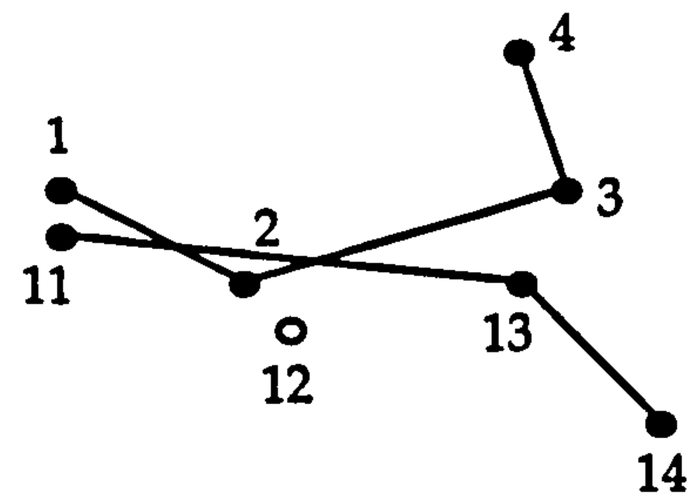
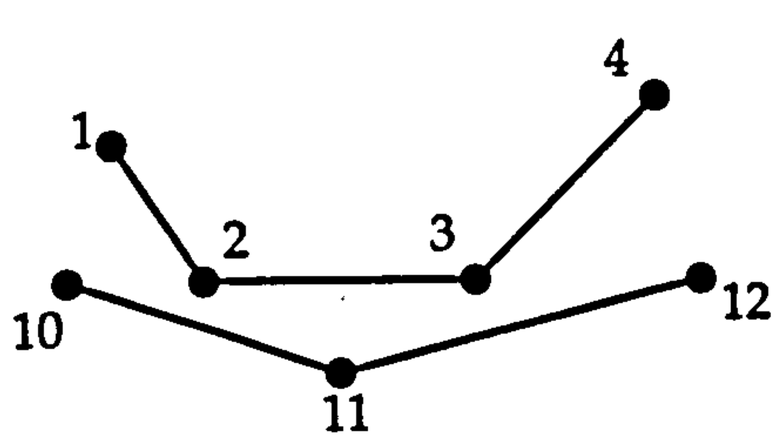


Figure A1.3 - Neighbouring lines intersecting following simplification.



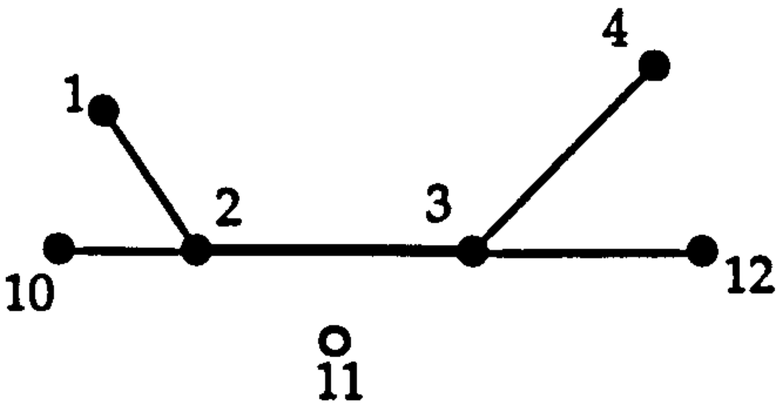
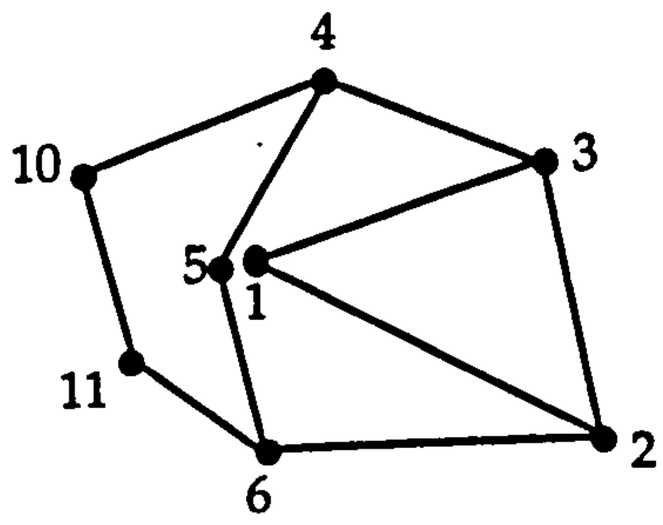


Figure A1.4 - Neighbouring lines becoming co-incident.



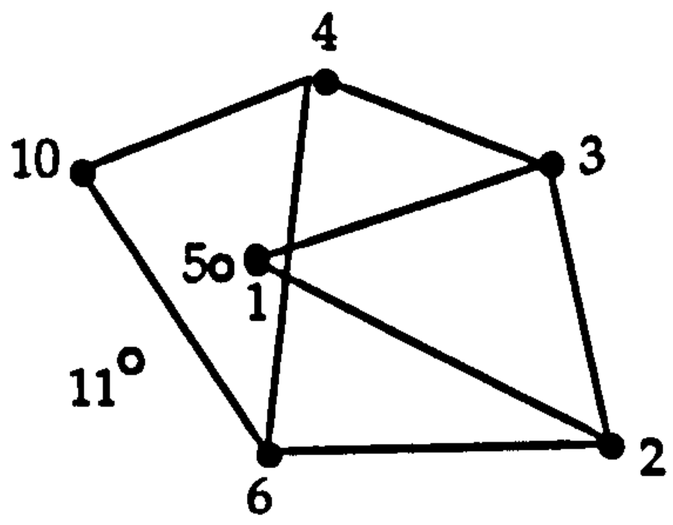
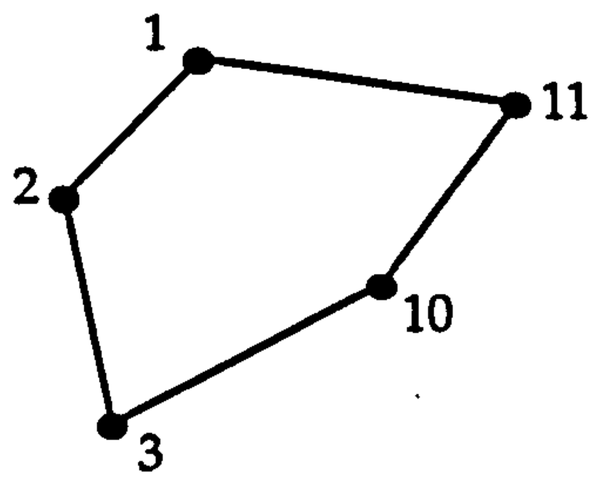


Figure A1.5 - Overlapping polygons.



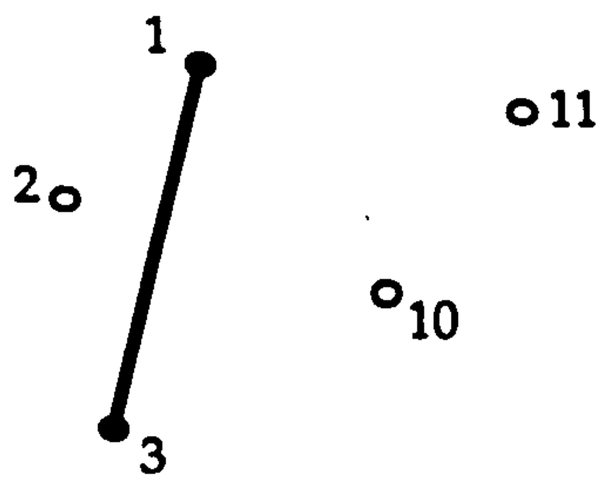


Figure A1.6 - A flat polygon.

One such example is that of introducing spikes. Such an error occurs when a line point nearly, but not quite, interferes with another line point or part of the same or other line, as a result of the generalisation (Figure A1.7). With such an error, the topological integrity of the line is maintained, but the resulting line will be visually displeasing. Muller [83] has produced an elegant solution to this particular problem. It will not be

addressed further here, particularly in view of the fact that the concern of this work is solely the topological integrity of lines and polygons.

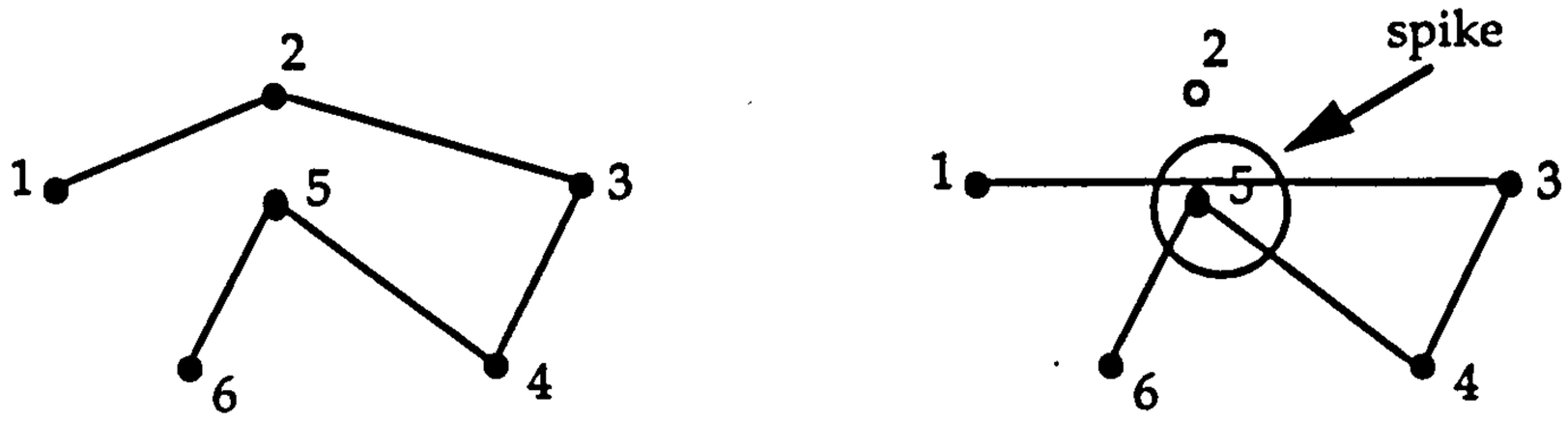


Figure A1.7 - A spike introduced as the result of simplification.

A1.2 Detecting Spatial Conflict.

Errors of the type previously described occur either as a result of two or more line segments intersecting each other or two or more line segments becoming co–incident. Detecting error is therefore equivalent to detecting either or both of these occurrences. It is assumed that the source data is clean, that is, it contains no intersecting or co–incident line segments.

A line can be checked for self–crossing by testing each of its constituent line segments, in turn, for intersection with any other of its constituent line segments. In similar fashion, a line with co–incident line segments can be detected by testing each of its line segments, in turn, for co–incidence with any other of its line segments. The detection of intersection or co–incidence of neighbouring lines can be carried out in a similar manner, checking the line segments of each line against all line segments of every neighbouring line. If no spatial indexing is employed, it is necessary to check each line segment of every line against each line segment of every other line, since each line must be regarded as the neighbour of every other line. This can lead to unsatisfactory processing times, even for relatively small data sets. However, if a spatial index is used, the number of neighbours per line, and hence the number of intersection and co–incidence tests needing to be performed, can be greatly reduced. This can be brought about by defining a spatial window based on the line segment currently being processed. It now follows that only those lines which intersect this window need be regarded as neighbours of the line segment.

A1.3 Re–establishing Topological Integrity.

It has been shown that the removal of points from a line or collection of lines can result in a loss of topological integrity. Methods have also been described for detecting such errors. There now follows a discussion as to how these errors can be resolved.

In each of the types of error described, the error has occurred as a result of removing

one or more points from the original line or lines. The error correction technique adopted here is therefore to replace one or more of the discarded points back into its original line, in such a way as to restore topological integrity. An alternative approach would be to retain only those points selected by the Douglas Peucker algorithm, but distorting the x and y coordinates of certain of these points in a fashion such as to result in a topologically correct data set. Muller [83] uses a method similar to this. The former method is more applicable to this work as it is important to retain consistency between the levels of the multi-scale database.

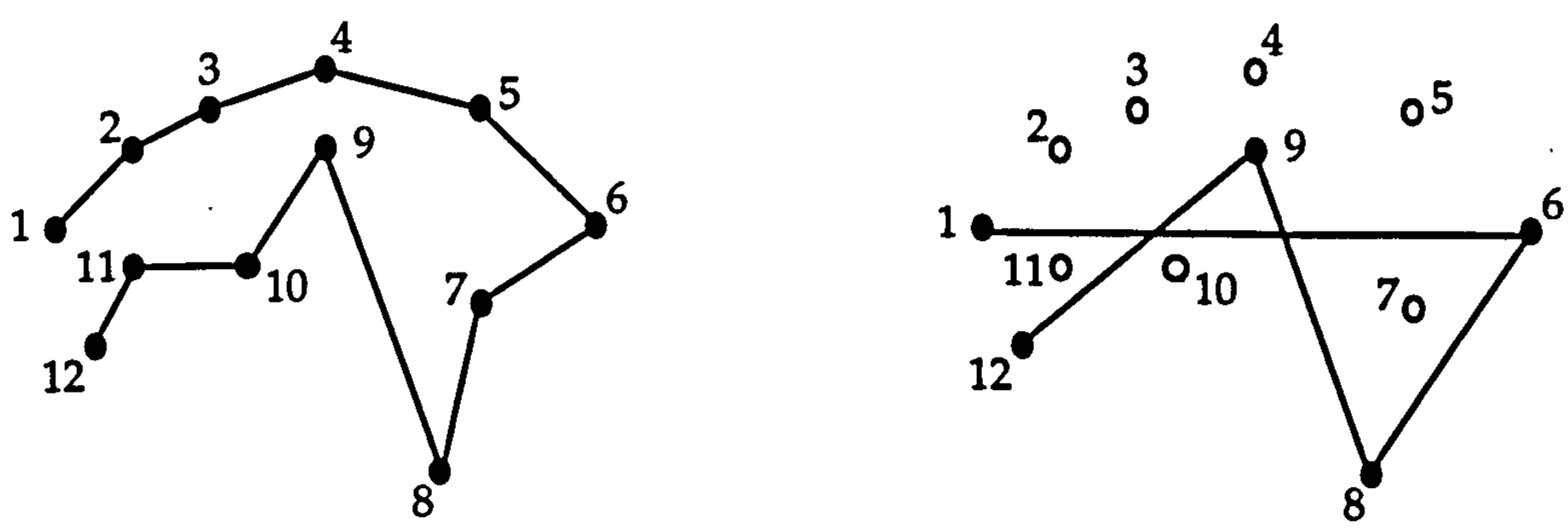


Figure A1.8 - Generalisation has resulted in self-crossing.

The self-crossing scenario will be dealt with first, and in detail. The other three situations can each be dealt with in a similar way to this. The situation is that a line L, consisting say of n points and (n-1) line segments, has been generalised, resulting in a generalised line L_g, consisting of (n-m) points and (n-m-1) line segments (Figure A1.8). Each of the (n-m-1) line segments, defined (a, b), may, or may not, have discarded points associated with it. For each segment, the associated discarded points, D, will be those points which lie between the endpoints, a and b, in the original line description. It is found that two of the line segments, S_1 and S_2, of the generalised line intersect each other, resulting in a loss of topological integrity. A method for resolving this situation is therefore needed.

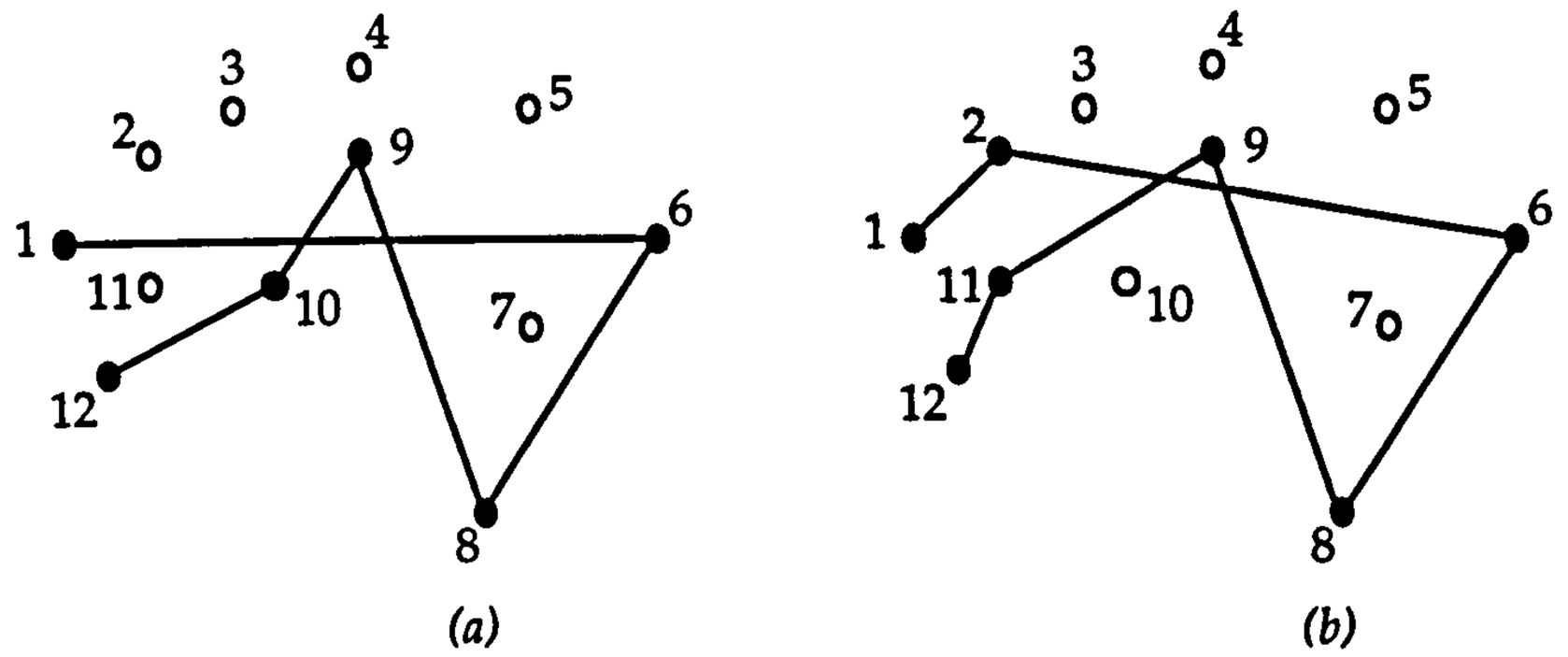
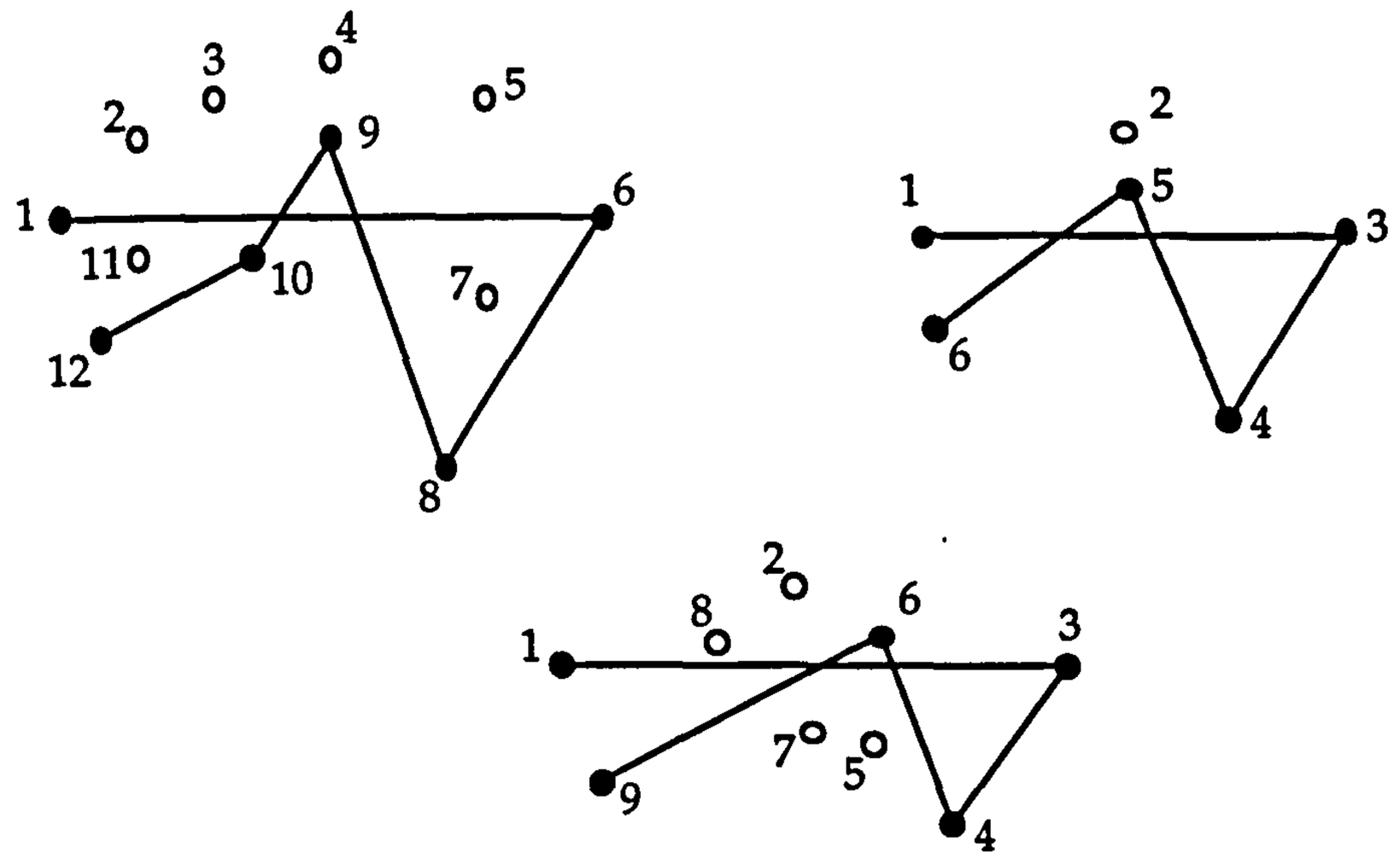


Figure A1.9 - Restoring spatial integrity by arbitrarily choosing points for reinsertion. (a) Self-crossing after simplification. (b) Point 2 added between point 1 and point 6, does not resolve spatial conflict. (c) Point 10 added between point 9 and point 11, does not resolve spatial conflict. (d) Point 4 added between point 2 and point 6, spatial integrity restored.

The simplest approach is to reintroduce all discarded points, D_{s1} and D_{s2}, resulting in a number of new, non-intersecting line segments. A second approach could be to reintroduce a single point, chosen arbitrarily from D_{s1} or D_{s2}, thus replacing one of the original line segments with two new segments, S_3 and S_4, each with their own list of associated points. S_3 and S_4 are now each checked in turn for intersection with the remaining original line segment. If an intersection is found, the process of choosing and inserting a point is repeated and continues in a recursive manner until no intersection is found (Figure A1.9). This is the approach adopted in the line generalisation algorithms used in this thesis.

Note that any new line segments introduced as a result of applying either of these processes may intersect other line segments of L_g, other than those derived from S_1 and S_2, and will have to be checked accordingly. This process of introducing new error into the line representation is referred to as error propagation.



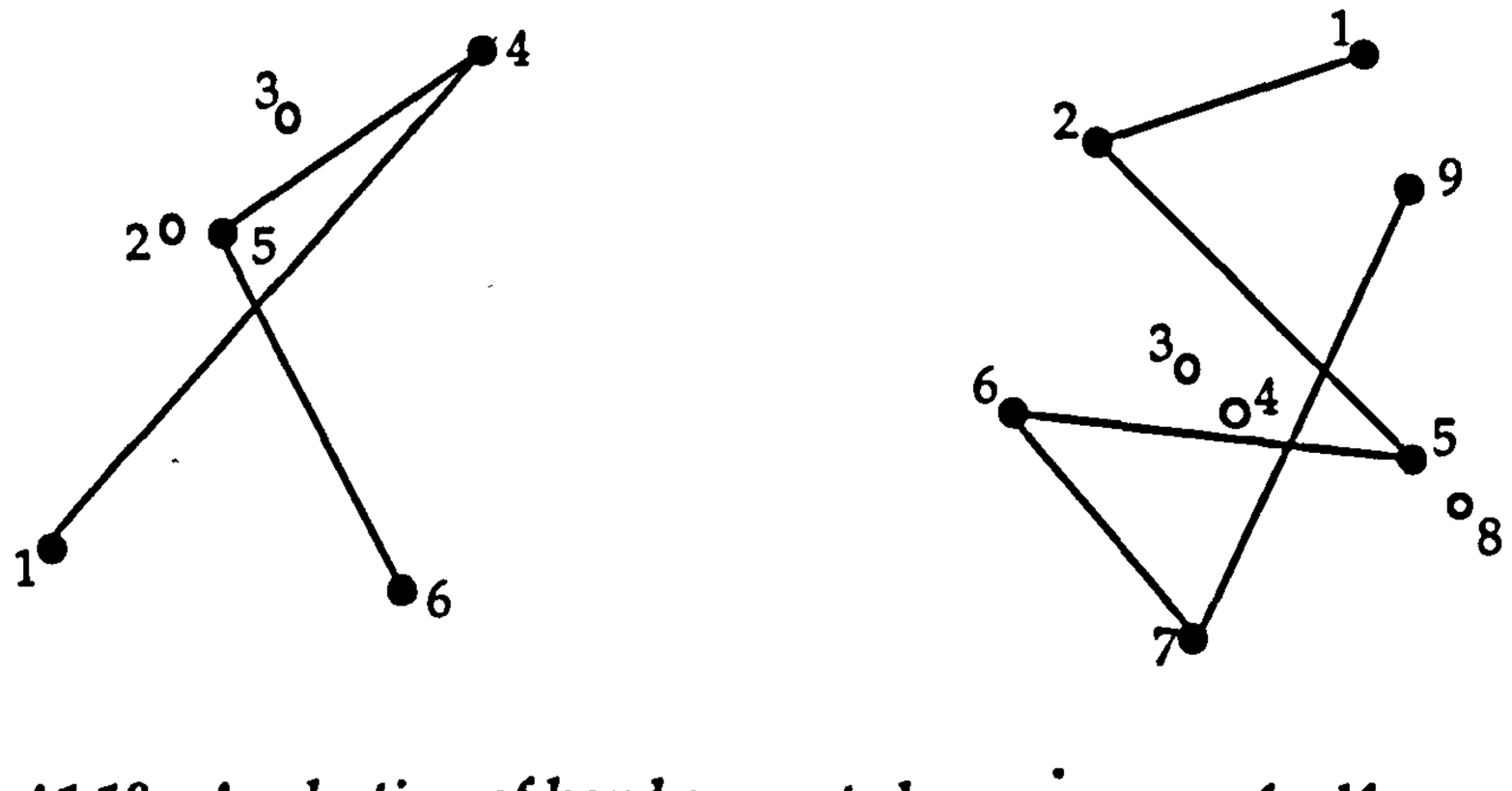


Figure A1.10 - A selection of hand generated occurrences of self-crossing.

Each of these methods will solve the problem of spatial conflict, but would seem to be far from optimal with regards the number of discarded points which are reintroduced. A more sensible approach could be to modify the second method, by applying certain criteria to govern the selection of the point that is reintroduced at each stage. This might result in restoring the spatial integrity of the line with fewer points having to be reintroduced. There now follows a number of suggestions as to how a point might be selected for reinsertion. These suggestions have not as yet been tested.

One obvious criteria would be to reintroduce that point with the greatest scale significance (as decided upon by the Douglas Peucker algorithm). An exhaustive technique would be to predict the effect of reintroducing each of the points. This could be achieved by temporarily reinserting each point, in turn, into the line, assessing its effect, then removing it. The point which gives the best result can then permanently reinserted. The result of inserting a point is quantified in terms of whether or not it solves the problem of intersection, the points significance in describing the line and the number of propagation errors introduced as a result. This approach would appear to

be very efficient in terms of number of points reinserted, but might suffer in being time consuming.

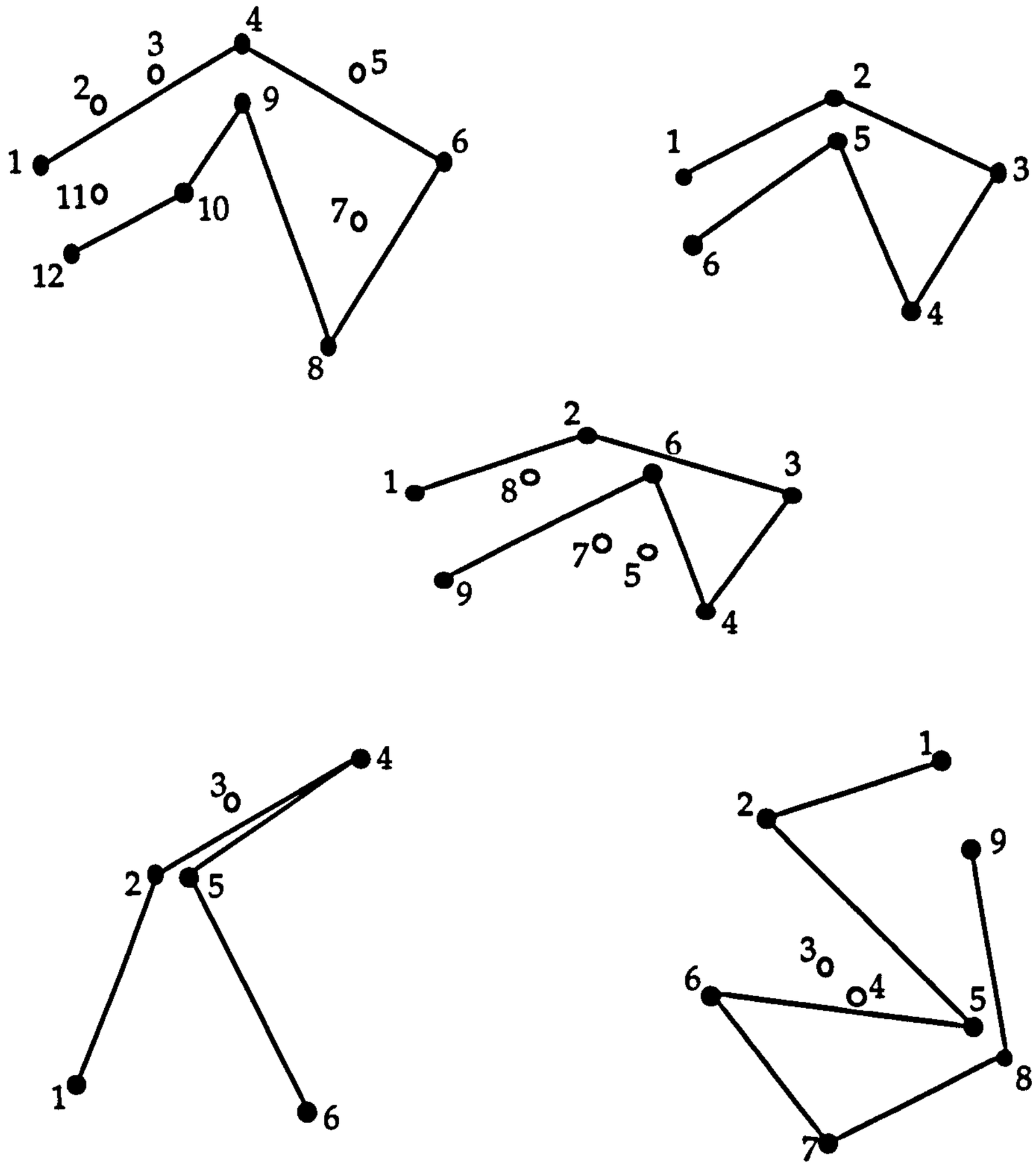


Figure A1.11 - Spatial integrity restored in each case by guessing at the best point to insert.

A compromise might be reached by introducing a rule by which the best point to reinsert can be guessed at. From studying a number of hand generated situations (Figure A1.10), it is noticed that in most cases when two line segments intersect, one of these segments, S, will also intersect at least one other segment. It would appear that the best approach to reinserting points would be to reinsert a point into the line segment S first. It also follows that this point will be the point from D_s with greatest scale significance. If both line segments intersect no other segment, a point is chosen purely according to scale significance. In practise, each line segment may intersect any number of line segments. The policy adopted is to reintroduce a point to that segment which intersects most other line segments. If each line segment intersects the same number of other line segments, the point is chosen according to scale significance. This method is illustrated in Figure A1.11.

This technique can easily be adapted to cater for the solution of self-coincidence, overlapping polygons and flat polygons. A single algorithm, RESTORE_SPATIAL_INTEGRITY, is presented in Figure A1.12. The algorithm describes a process by which a set of line data (whether the lines represent polygons or not does not matter) is examined for spatial conflict and has its spatial integrity restored whenever a conflict is found to have occurred.

```
Procedure RESTORE_SPATIAL_INTEGRITY

For each line L1
    Get neighbours.
    For each neighbour L2
        Put all line segments of L1 onto EDGE_STACK_1.
        Put all line segments of L2 onto EDGE_STACK_2.
        Do while EDGE_STACK_1 not empty
            Take line segment E1 (A1,A2) from top of EDGE_STACK_1.
            Do while EDGE_STACK_2 not empty
                Take line segment E2 (B1,B2) from top of EDGE_STACK_2.
                If (E1 ≠ E2), then
                    If (edges_intersect(E1,E2) or (edges_coincident(E1,E2)), then
                        Calculate I1, the number of edges of L2 intersected by or
                        coincident with E1.
                        Calculate I2, the number of edges of L1 intersected by or
                        coincident with E2.
                        If (I1 > I2), then
                            Get D1, the currently unused point with greatest
                            significance.
                            Replace edge E1 in L1 with edges N1 (A1,D1) and N2
                            (D2,A2).
                        Else
                            If (I2 > I1), then
                                Get D2, the currently unused point with greatest
                                significance.
                                Replace edge E2 in L2 with edges N1 (B1,D2) and N2
                                (D2,B2).
                            Else
                                Get D1.
                                Get D2.
                                If (distance D1 > distance D2), then
                                    Replace edge E1 in L1 with edges N1 (A1,D1) and
                                    N2 (D1,A2).
                                Else
                                    Replace edge E2 in L2 with edges N1 (B1,D2) and
                                    N2 (D2,B2).
                                Endif.
                            Endif.
                        Endif.
                    Endif.
                Endif.
            Endwhile.
        Endwhile.
    Endfor.

Endprocedure.
```

Figure A.12 - A procedure to restore spatial integrity.

Appendix 2

Core Library Functions

This appendix does not list all the functions held in the core libraries. Those listed here have been chosen because of their importance to database creation and query processing.

A2.1 Data_Retrieval Library.

This library contains low-level read/write functions which interact directly with the ORACLE or ISAM database. For each of the functions listed, which read data from the database, there is a corresponding write function in the actual implementation.

function get_values (point_id, *level_used, x_value, y_value, z_value*)
Returns a record from the Oracle Point Table (or ISAM Point File) for a given point_id.

function get_line_record (line_id, level, *point_ids, seq_nos*)
Returns a record from the appropriate Line Feature Table for a given (line_id, level) pair.

function get_polygon_record (polygon_id, *line_ids*)
Returns a record form the Polygon Table for a given polygon_id.

function get_object_record (object_id, *object_desc, class_ids, point_ids, line_ids, polygon_ids*)
Returns a record from the Object Table for a given object_id.

function get_triangle_record (tri_id, level, *geom_id*)
Returns a record from the appropriate Triangle Table for a given (tri_id, level) pair.

function get_internal_record (geom_id, level, *vert1_id, vert2_id, vert3_id, adjacent1_id, adjacent2_id, adjacent3_id*)
Returns a record from the appropriate Internal Table for a given (geom_id, level) pair.

function get_boundary_record (geom_id, level, *adjacent1_id, adjacent2_id, adjacent3_id*)
Returns a record from the appropriate Boundary Table for a given (geom_id, level) pair.

function get_info_details (*object_grid_x, object_grid_y, triangle_grid_x, triangle_grid_y, lateral_error, vertical_error*)
Returns details from the Information Table.

function get_triangle_grid_record (x_coord, y_coord, level, *triangle_ids*)
Returns a record from the appropriate Triangle Grid Table for a given coordinate pair and level.

function get_object_grid_record (x_coord, y_coord, level, *object_ids*)
Returns a record from the appropriate Object Grid Table for a given coordinate pair and level.

A2.2 Data_Transfer Library.

This library contains high-level read/wite functions which interact with the Data_Retrieval library.

funtion get_triangle_details (tri_id, level, *v1, v2, v3, a1, a2, a3*)
Returns the geometry and adjaceny information for a given tri_id at a specified level of detail.

function get_line_details (line_id, level, *point_ids*)
Returns a list of point_ids which reference the points that define a line at a particular level of detail.

function get_objects (x1, y1, x2, y2, level, *object_list*)
Returns a list of object_ids which reference all objects that intersect a query window (a rectangle defined by x1, y1, x2, y2) at a specified level of detail.

function get_triangles (x1, y1, x2, y2, level, *triangle_list*)
Returns a list of triangle_ids which reference all triangles that intersect a query window (a rectangle defined by x1, y1, x2, y2) at a specified level of detail.

A2.3 Geometry_Library.

This library contains the geometrical routines used during database creation and query processing.

function point_in_triangle (point_id, tri_id, *flag*)
Sets flag to 0 if point outside triangle, 1 if point is inside triangle, 2 if point lies on an edge of triangle or 3 if point coincides with a triangle vertex.

function point_in_polygon (point_id, polygon_id, *flag*)
Sets flag to 0 if point outside polygon, 1 if point is inside polygon, 2 if point lies on an edge of polygon or 3 if point coincides with a polygon vertex.

function line_segment_intersect (x1, y1, x2, y2, x3, y3, x4, y4, *flag*, *x*, *y*)
Tests if line segment (x1, y1), (x2, y2) intersects line segment (x3, y3), (x4, y4). Flag set to 1 if segments intersect, 0 otherwise. Also returns (x, y) value of point of intersection (when appropriate).

function calculate_centroid (tri_id, *x*, *y*)
Calculate the centroid of a triangle.

function calculate_circumcentre (tri_id, *x*, *y*)
Calculate the circumcentre of a triangle.

A2.4 Triangulation Library.

This library contains a number of the basic functions required to perform a constrained Delaunay triangulation of a set of points and constraining edges.

function calculate_convex_hull (point_ids, *convex_hull*)
Calulates the convex hull of a set of points.

function triangulate_convex_polygon (convex_polygon, *triangle_ids*)
Delaunay triangulates a convex polygon.

function triangulate_polygon (polygon, *triangle_ids*)
Delaunay triangulates a polygon.

function insert_point (point_id, triangle_ids)
Inserts a point into an existing triangulation.

function insert_edge (v1, v2, triangle_ids)
Inserts an edge (v1, v2) into an existing triangulation.

A2.5 Output Library.

This library contains a number of high-level computer graphics output primitives which interact with UNIRAS (MTSD 1.0) or PHIGS (MTSD 2.0 and MGD).

function draw_object_empty (object_id)
Draws an object.

function draw_object_shaded (object_id, colour)
Draws a shaded object.

function draw_polygon_empty (polygon_id)
Draws a polygon.

function draw_polygon_shaded (polygon_id, colour)
Draws a shaded polygon.

function draw_line(line_id)
Draws a line.

function plot_point (point_id)
Plots a point.

function draw_triangle_empty (triangle_id)
Draws a triangle.

function draw_triangle_shaded (triangle_id, colour)
Draws a shaded triangle.

Appendix 3

Published Papers

These papers make up the published work relating to the thesis. It should be noted that the results in the thesis represent a more up to date account of the research than the work presented in these papers. In the instance of any inconsistency between the work documented in the papers and that documented in the thesis, the thesis should be taken as being correct.

MULTISCALE SPATIAL MODELLING WITH TRIANGULATED SURFACES

Christopher B. Jones, J. Mark Ware and Geraint Ll. Bundy

Department of Computer Studies
The Polytechnic of Wales
Pontypridd
Mid Glamorgan, CF37 1DL, UK
email: cbjones@uk.ac.pow.genvax

Abstract

Triangulated surfaces provide a means of integrating linear and polygonal features within topographic and geological models. The approach is versatile in that it can be applied to planar 2D maps, terrain surfaces and fully 3D geological models. Hierarchical triangulated data structures can represent surfaces at multiple resolutions and can be combined with a hierarchical representation of the point, line and polygon features that constrain the triangulation. Triangulations can also be used to assist in the detection and resolution of spatial conflicts resulting from generalisation operators that involve enlargement and symbolisation.

Introduction

Despite the very widespread use of GIS technology, facilities for automatically generating spatial models and maps at different scales remain remarkably poor. Satisfactory execution of automatic scale change, and the process of generalisation which it entails, depends partly upon the incorporation within GIS of the human knowledge involved in generalisation (Weibel, 1991). It also depends upon the development of computer representations and computer techniques which can emulate the visual processing which is implicit in the human knowledge concerned with generalisation. This paper examines the use of triangulated surfaces, primarily as a framework for building multiple scale topographic databases, but also, more speculatively, as a representation which has potential for assisting generalisation procedures which can benefit from a continuous representation of the map space.

In the context of geographical information systems, the benefits of triangulations for representing terrain surfaces in the form of triangulated irregular networks (TIN) have long been recognised (e.g. Peucker et al, 1978). In particular they preserve spatial data at their original locational accuracy, unlike raster or grid models which often result in a loss of

locational resolution. The structure of a triangulated surface can also be used to represent local spatial relationships, through vertex, edge and triangle adjacency data. Their broader potential for providing topologically consistent representations of maps, in the form of cell graphs or simplicial complexes has been discussed by Frank and Kuhn (1986). This latter view of triangulations appears relevant to the solution of map generalisation problems in that it is desirable to retain the topological integrity of map data while applying simplification and symbolisation operators which are liable to jeopardise it due to changes in shape and size of objects. An example of a technique which uses triangulation explicitly to generalise polygons, by generating their skeleton, has been described by Chithambaram et al (1991). Frank and Kuhn drew attention to the value of triangulations in representing linear features as chains of edges, in addition to their conventional use in representing terrain defined by point elevations. They also referred to the benefits of creating hierarchical triangulations in which objects could be described at different levels of aggregation of their components.

There have been several research efforts with the objective of creating multiresolution hierarchical triangulations. Examples can be found in Gomez and Guzman (1979), De Floriani (1989), De Floriani and Puppo (1988), and Scarlatos and Pavlidis (1991). Hierarchical triangulated data structures are particularly relevant to representing the aspects of generalisation which involve gradual changes in spatial form. Provided the changes can be predetermined and can be described by a progressive reduction of the original geometric components, multiresolution hierarchical data structures may provide efficient access to specified levels of detail by traversing them to the appropriate depth.

The hierarchical triangulation schemes referred to can be applied to terrain surfaces in which the level of detail is defined with respect to a vertical error corresponding to the difference between a simplified surface and the original detailed surface. Comparable schemes for multiresolution representation of linear features have also been developed but the error refers to lateral or horizontal distances. Examples are strip trees (Ballard, 1981) and the multiscale line tree (Jones and Abraham, 1986, 1987).

The following sections of this paper are concerned with the development of a multiresolution database which integrates a hierarchical triangulation scheme based on the constrained Delaunay pyramid (De Floriani and Puppo 1988, De Floriani 1989) with a hierarchical line representation based on the multiscale line tree. We then discuss the issues which arise in extending this topographic surface modelling scheme to a multiresolution three dimensional geological model. Finally we consider briefly how a two dimensional triangulation can be used to assist

generalisation procedures involving object symbolisation and displacement.

A Multiresolution Topographic Surface Database

The objective in designing a multiresolution topographic surface database was to provide a spatial model which combined point, linear and areal topographic features with a terrain surface, in a manner which enabled efficient retrieval of different scale representations. In particular it was regarded as desirable to minimise duplication of data such that spatial data common to different levels of detail were only stored once. In combining ideas from the constrained Delaunay pyramid and the multiscale line tree, the approach adopted here categorises vertices according to their scale significance. The terrain surface and the topographic features embedded within it are then represented at varying levels of detail by describing triangulations and surface features in terms of references to the component vertices, which are stored separately.

The Delaunay pyramid, in its original form, consists of a hierarchy of triangulation definitions, each level of which contains progressively greater detail. There are three types of triangle which can occur in a particular level of triangulation - internal, boundary and external. An internal triangle consists of references to three vertices and three adjacent triangles. A boundary triangle consists of three references to adjacent triangles and a reference to a previously defined, higher level, internal triangle (the vertices of which are the same as those of the boundary triangle). An external triangle simply consists of a reference to a previously defined, higher level, internal triangle with which it is identical. Each triangle within a level also references those triangles at the next more detailed lower level which intersect it. This assists hierarchical spatial search within a Delaunay pyramid to determine which triangle a given point lies inside.

The constrained Delaunay pyramid (cdp) modifies the original data structure by allowing insertion of chains of edges belonging to surface features. Retention of these edges within the triangulation as edges of triangles results in local violation of the criteria for Delaunay triangulation. In doing so, the surface more accurately models the real world surface by ensuring that it conforms to known structural features. The possible failure of Delaunay triangulation to model the real surface was one of the motivations for the multiresolution scheme of Scarlatos and Pavlidis (1991). The cdp differs from the latter however in ensuring that triangle edges conform to specified structural features and in using Delaunay criteria to model those parts of the surface for which no other structural information is available.

In the Delaunay pyramid, vertices are allocated to successive levels of

the hierarchy on the basis of their reduction in the elevation error at the location of the vertex, with respect to the original fully defined surface. Thus, starting at an approximation to the original surface, the vertically most distant vertices are progressively inserted in the triangulation at that level until a preset tolerance is reached, i.e. until no uninserted vertex is further from the current surface approximation than the tolerance distance for that level.

When constraining the surface with linear features, the vertical error criterion for vertex insertion does not provide adequate control over the degree of generalisation of linear features at a particular level, since it takes no account of lateral variation in form. In practice, the inserted linear feature vertices may have been derived from a two dimensional planar map and could not therefore contribute to a reduction of surface error, since their elevations were only obtained by interpolating them within the original terrain surface. To ensure that linear features are represented at an appropriate degree of generalisation at each level, it is necessary to introduce the concept of lateral tolerance, which is a measure of the 2D cartographic generalisation of these features. In the line generalisation tree (Jones 1984, Jones and Abraham 1986) and its spatially indexed variant, the multiscale line tree, linear feature vertices are classified according to shape significance by means of the Douglas and Peucker (1973) algorithm. This uses a tolerance value based on the laterally perpendicular distance of vertices from an approximating line passing through a subset of the original vertices.

In our modification of the cdp, the Multiresolution Topographic Surface Database (MTSD), each level of the hierarchy is defined by both a vertical distance tolerance and a lateral distance tolerance. Thus only those vertices of linear features which approximate the line to within the preset lateral tolerance are inserted as constrained edges within a particular level. The constrained edge insertion procedure is similar to that of De Floriani and Puppo, though it differs in allowing the insertion of intersecting edges, by creating a new vertex at the intersection.

The MTSD describes primitive surface features in terms of points, lines and polygons. Polygons are defined by lists of lines, while lines are defined by lists of vertices. Objects of a particular class are defined by the polygon, line and point features which constitute them. Each level of the MTSD is represented by tables which define the objects and the polygon, line and point features which are relevant to that level. In the case of the linear feature tables, each record consists of data comparable to that used in the line generalisation tree. Thus linear feature records contain a line feature identifier, the identity of the highest level at which the line appears and a list of vertex identifiers, each of which is accompanied by left and right

control values. The control values record the numbers of vertices to be inserted on either side of the referenced vertex at the next level down the hierarchy. Line reconstruction is performed by means of a recursive procedure described in Jones and Abraham (1986).

Spatial access to the MTSD is facilitated by the use of a pyramidal structure consisting of regular grids for each level of the database. The cell size of the grids differs between levels according to the density of objects and triangles. Each cell references a list of the objects or triangles which intersect it. The use of the pyramidal grid spatial indexing scheme is comparable to that employed in the multiscale line tree which was implemented both with quadtrees and with a regular pyramid index. This approach to spatial indexing diverges from the Delaunay pyramid design of De Floriani, which employs hierarchical links between pyramid levels to guide spatial search. The regular grids provide a simple method of object and triangle indexing. Their use in our experimental database is justified by simplicity rather than optimality.

The MTSD has been implemented with a relational database management system. The design does not conform fully to the relational model in that it makes use of variable length fields for storing lists of data items. Thus the linear feature tables store lists of vertex identifiers and control values, the polygon feature tables store lists of linear feature identifiers, the object tables store lists of point, line and polygon identifiers, and the object grid and the triangle grid tables store lists of objects and triangles respectively.

A major component of the storage space in the MTSD is that of the triangulation tables for each level of the database. Having constructed the triangulations and inserted the surface features at the various levels of resolution, it remains an option to store only the identifiers of the constituent vertices of each level, along with the point, line polygon and object features which map onto them. When a surface is required at a particular level of detail it may then be retrieved by obtaining the relevant vertices and executing a constrained Delaunay triangulation algorithm. Because reconstruction of the surface does not require the original tests for surface error, a more efficient algorithm may be used, such as that described by Chew (1987). Provided that the spatial window was expanded sufficiently to model the boundary of the surface, it could be reproduced in a form identical to when the database was constructed. This method is termed implicit triangulation and has been described by Kidner and Jones (1991).

Three Dimensional Object Modelling

Multiresolution triangulated surface data structures have potential for representing fully three dimensional structures of the sort required in geological modelling. The construction of triangulated surfaces bounding 3D polyhedra is, however, complicated by the fact that, unlike normal terrain, geological surfaces cannot be assumed to be single valued when projected onto a plane. In general, they may be of any orientation and may be overfolded, at least relative to the horizontal plane.

Procedures for Delaunay triangulation, in 2D and in 3D, of arbitrarily oriented polyhedral surfaces have been described by Boissonnat (1984). In the case of the 2D triangulation technique, vertices are projected onto local tangential planes, on which triangulation then takes place. The vertices which are triangulated belong to a neighbourhood which must be assumed to represent a surface which is single valued relative to the local tangential plane (which itself is found by a least squares fit). Working with subsurface geological data, this assumption would not always hold true, since data may be sparsely distributed and folding or reverse faulting could have occurred. However, if information were available from borehole or seismic data about the orientation of the boundary at each vertex, it would be possible to select vertices in order to improve the chances of working with a single valued surface. It might also be possible to introduce constraints relating to connectivity between vertices, based on geological interpretations of cross sections obtained from borehole and seismic data.

The 3D volumetric approach to Delaunay triangulation described by Boissonnat finds a surface represented by a set of vertices by determining the boundary of a 3D tessellation, based on tetrahedra. Initially the volumetric tessellation occupies the entire convex hull of the vertices, after which tetrahedra occupying what are assumed to be concavities of the surface are progressively eliminated. Provided the original vertices have been obtained by a relatively regular and dense sampling of the surface of an object, the resulting triangulated surface may be expected to correspond topologically to the real surface. As with the 2D triangulation scheme referred to above, the Delaunay criteria for triangulation cannot be guaranteed to reproduce the form of the real surface when using irregularly sampled vertices, particularly when combined with complex structures. It may be envisaged that Delaunay criteria should only be applied when no other constraining data are available.

Modification of the multiresolution database for topographic surfaces to incorporate 3D surfaces will depend not only upon finding an appropriate surface triangulation procedure but also upon finding a method of classifying vertices according to their contribution to the shape of the

surface. Faugeras et al (1984) have described a technique which does this by retriangulating an initial higher resolution surface to within a given error. The technique resembles the Douglas and Peucker algorithm for line simplification, in that it uses a recursive procedure to determine points which are most distant relative to a given approximation.

Whereas, in the Douglas and Peucker algorithm, distances are measured perpendicular to an approximating line (starting with the line joining the start and the end points), in the surface based method, distances are measured from the plane of an approximating triangle. An initial triangle segments the vertices into two halves, on the assumption that the vertices represent a closed volume. The initial triangle's component vertices are then retriangulated with the two vertices selected as most distant on both sides of the plane. Using the vertices in the respective initial partitions, the most distant vertices from each of these constituent triangles are then found, subject to a constraint which confines the search to the connected neighbourhood of each triangle. The procedure continues recursively until no vertex is further from its parent triangle than the tolerance distance. This procedure would preserve the edges of each triangle that was subdivided, resulting in the possibility of a poor representation of the surface. Faugeras et al alleviate this problem by breaking common edges at an intermediate vertex and retriangulating after each stage of the recursion process.

Triangulated Models and Cartographic Generalisation

The MTSD previously described provides a framework for accessing geographical information at multiple levels of generalisation. In describing the data in the context of constrained triangulated surfaces however, it provides a tessellated representation of space which may be useful in carrying out generalisation procedures used to derive representations at different scales.

An important aspect of generalisation is the competition for map space which results from the symbolisation of objects such that they occupy more space than their true scale representation. As scale decreases the symbolisation process dictates that object representations must be simplified, moved from their true scale location, or eliminated entirely from the map. Generalisation of individual objects within a map must be done with regard to the integrity of adjacent map objects. The triangulated model, or simplicial complex, is attractive in this respect, in that all space on the map is explicitly accounted for. Furthermore, the triangular topology can serve to record adjacencies between neighbouring objects. Thus if all space is triangulated with disjoint (i.e. non-overlapping) triangles, the edges emanating from the boundary of an object will be

connected to the nearest objects.

If generalisation operators are applied to individual objects within a triangulated space, any adverse effects of the operations can easily be monitored, by testing the integrity of the triangulation. Whenever an overlap occurs, characterised by the fact that a vertex of one object moves across the boundary of another object, the triangle which connects the originally adjacent objects will become inverted. This situation is distinguished by a reversal in the rotational order of the triangle's component vertices. Thus if initially ordered clockwise, they will become anticlockwise on inversion.

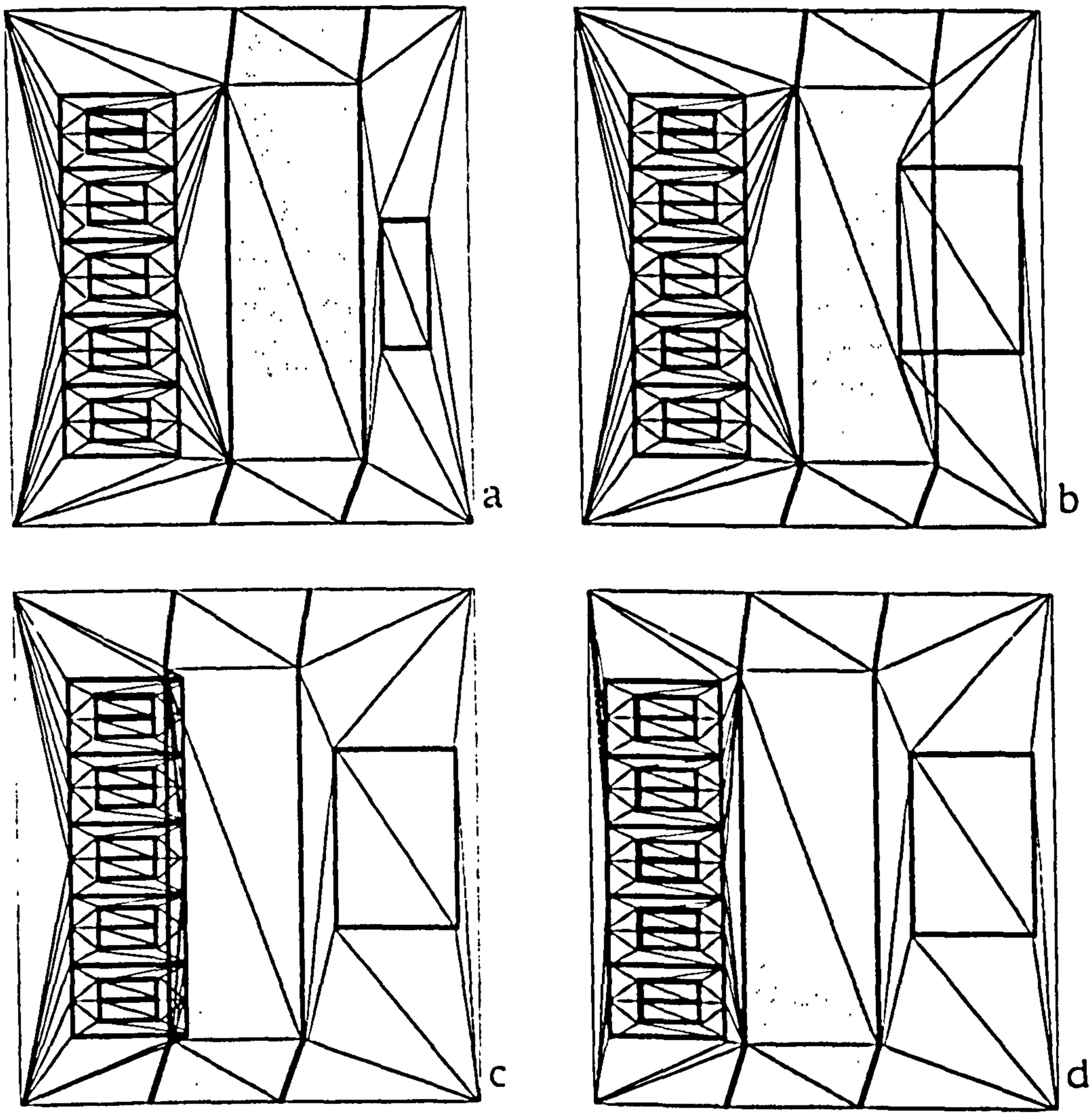


Figure 1. Enlargement and displacement can result in loss of integrity of a triangulation, indicated by inverted triangles (see text).

The presence of inverted triangles provides a computationally simple means of detecting potential conflict, actual overlaps and, in general, topological inconsistencies due to an object moving across another object. Identification of inversions can be used both to trigger operations such as displacement, shrinkage and object deletion, and to guide the way such operations are executed. Figure 1 illustrates a simple scenario in which an initially consistent triangulation (a) is violated by the expansion of the rectangular object on the right (b). An attempt to resolve the conflict by moving the transgressed object leftwards (c) results in a further overlap with the group of objects on the left. Resolution is achieved by moving this group leftwards (d). The way in which conflict is resolved depends upon the relative importance of the objects concerned. Research on the applicability of triangulations to generalisation is in progress and results will be published elsewhere. Control of the generalisation operators according to context and purpose of the map will be achieved by means of a frame-based knowledge representation and reasoning system.

Summary

A multiscale topographic surface database (MTSD), which integrates point, line and polygon features within a triangulated terrain model has been implemented. The hierarchical constrained triangulation scheme used for surface description is being extended to enable the representation of three dimensional geological objects bounded by multiresolution triangulated surface patches. The technique of constrained Delaunay triangulation is also being used for the purpose of generalising 2D maps. A data structure which records the topology of the triangulation facilitates the detection of spatial conflicts resulting from generalisation operators, and their subsequent resolution in a manner which maintains the topological integrity of the map. The MTSD design and the related generalisation techniques are being incorporated within a deductive database system, summarised in Jones (1991), for maintaining multiple representations of geographical information.

Acknowledgements

JMW and GLB are supported by a SERC studentships in collaboration with the British Geological Survey and the Ordnance Survey respectively.

References

Ballard, D.H. 1981. "Strip trees: a hierarchical representation for curves", Communications of the ACM, 24, pp. 310-321.

Boissonnat, J-D 1984. "Geometric structures for three-dimensional shape recognition", ACM Transactions on Graphics, 3(4), pp. 266-286.

Chew, L.P 1987. "Constrained Delaunay triangulations", Proceedings Third ACM Symposium on Computational Geometry, Waterloo, pp. 216-222.

Chithambaram, R., K. Beard and R. Barrera 1991. "Skeletonizing polygons for map generalization", Technical Papers ACSM-ASPRS Volume 2, pp. 44-55.

De Floriani, L. 1989. "A pyramidal data structure for triangle-based surface description", IEEE Computer Graphics and Applications, March 1989, pp. 67-78.

De Floriani, L. and E. Puppo 1988. "Constrained Delaunay triangulation for multiresolution surface description", Proceedings Ninth IEEE International Conference on Pattern Recognition, pp. 566-569.

Douglas, D.H. and T.K. Peucker 1973. "Algorithms for the reduction of the number points required to represent a digitized line or its caricature", Canadian Cartographer, 10(2), pp. 112-122.

Faugeras, O.D., M. Hebert, P. Mussi and J.D. Boissonnat 1984. "Polyhedral approximation of 3-D objects without holes", Computer Vision, Graphics and Image Processing, 25, pp. 169-183.

Frank, A.U. and W. Kuhn 1986. "Cell graphs: a provable correct method for the storage of geometry", Proceedings Second International Symposium on Spatial Data Handling, Seattle, pp. 411-436.

Gomez, D. and A. Guzman 1979. "Digital model for three dimensional surface representation", Geo-Processing, 1, 53-70.

Jones, C.B. 1984. "A tree data structure for cartographic line generalisation", Proceedings Eurocarto III, Research Center Joanneum, Institute for Image Processing and Computer Graphics, Graz.

Jones, C.B. 1989. "Data structures for three-dimensional spatial information systems in geology", International Journal of Geographical Information Systems, 3(1), pp. 15-31.

Jones, C.B. 1991. "Database architecture for multi-scale GIS", Proceedings Auto-Carto 10, ACSM-ASPRS, pp. 1-14.

Jones, C.B. and I.M. Abraham 1986. "Design considerations for a scale-independent cartographic database", Proceedings Second International Symposium on Spatial Data Handling, Seattle, pp. 384-398.

Jones, C.B. and I.M. Abraham 1987. "Line generalisation in a global cartographic database", Cartographica, 24(3), pp. 32-45.

Kidner, D.B. and C.B. Jones 1991. "Implicit triangulations for large terrain databases", Proceedings EGIS'91, pp.537-546.

Peucker, T.K., R.F. Fowler, J.J. Little, D.M. Mark 1978. "The triangulated irregular network", Proceedings Digital Terrain Models (DTM) Symposium, ASP-ACSM, St. Louis, pp. 516-540.

Scarlatos, L. and T. Pavlidis 1991. "Adaptive hierarchical triangulation", Proceedings Auto-Carto 10, ACSM-ASPRS, pp.234-246.

Weibel, R. 1991. "Amplified intelligence and rule-based systems", in Map Generalization, edited by B.P. Buttenfield and R.B. McMaster, Longman, pp. 172-186.

INT. J. GEOGRAPHICAL INFORMATION SYSTEMS, 1992, VOL. 6, NO. 6, 479–496

A multiresolution topographic surface database

J. MARK WARE and CHRISTOPHER B. JONES

Department of Computer Studies, The University of Glamorgan, Pontypridd, Mid Glamorgan, Wales, UK

Abstract. Multiresolution data structures provide a means of retrieving geographical features from a database at levels of detail which are adaptable to different scales of representation. A database design is presented which integrates multi-scale storage of point, linear and polygonal features, based on the line generalization tree, with a multi-scale surface model based on the Delaunay pyramid. The constituent vertices of topologically-structured geographical features are thus distributed between the triangulated levels of a Delaunay pyramid in which triangle edges are constrained to follow those features at differing degrees of generalization. Efficient locational access is achieved by imposing a spatial index on each level of the pyramid.

1. Introduction

Many existing geographical information systems (GIS) may be regarded as limited by the fact that the spatial models they employ represent the Earth's surface in only two dimensions and only at a fixed level of detail. When models of the terrain surface are included in GIS they are typically present as an independent representation consisting of an array of elevation values or a triangulated network of irregularly distributed elevations. For several types of spatial analysis, such as slope studies and visibility determination, this separate representation may be satisfactory and, particularly for raster-oriented analysis, very convenient. However, with the increasing use of GIS for urban and regional planning, and in the geosciences, it may be envisaged that the ground model could be more conveniently regarded as an integral part of a topographic database rather than as a separate layer. For many such applications it should be possible to visualize the model at different levels of detail, according to the areal extent, or scale, of the region of interest. Thus it is desirable to be able to treat urban infrastructure and natural features as embedded within a 2·5-D surface which can be retrieved at varying levels of detail. This would facilitate multi-scale 2·5-D visualization of existing and planned environments, and the interpretation and analysis of geoscientific data.

Since accurate representation of topographic features requires that they be locatable at arbitrary coordinates, there is considerable advantage in using a TIN surface model (Peucker *et al.* 1978), rather than a grid which would limit the locational resolution. A further advantage of triangulated surfaces is that they lend themselves both to a multi-resolution representation of the ground surface and to the incorporation of vector models of topographic features. A data structure which provides multi-resolution access to a triangulated surface is the Delaunay pyramid (De Floriani 1989). De Floriani and Puppo (1988) have shown how the construction of a multi-resolution trangulation can be constrained by linear features which become included as triangle boundaries.

This article presents the results of a continuing research project with the aim of designing a multi-scale database which integrates vector-defined geographical features

0269-3798/92 $3·00

with a digital elevation model. The data structures used in this database combine aspects of two-dimensional multi-scale database design (Jones 1984, Jones and Abraham 1986 and 1987) with a constrained Delaunay pyramid. In the following sections, the multi-scale storage of 2-D linear features is discussed briefly before summarizing the characteristics of De Floriani and Puppo's constrained Delaunay pyramid. Our modification of the latter is then described with reference to the issues of including generalized features and spatial indexing. An algorithm for building the data structures is also described, before presenting an implementation which uses a relational database management system. Finally, some implementation issues are discussed and future directions for research indicated.

2. Multi-scale access of linear features

Access to linear features at variable scales can be achieved either by storing multiple versions of the linear feature at predetermined scales; by storing a single large-scale version from which smaller scales are derived using generalization algorithms; or by means of a multiresolution data structure specifically adapted to retrieving variable degrees of detail. Storage of multiple versions results in significant storage overheads owing to duplication of the constituent vertices between different versions. Retrieval from a single version could incur major processing overheads when deriving a representation of much smaller scale than the original linear feature. Multiresolution data structures represent a compromise between the two approaches.

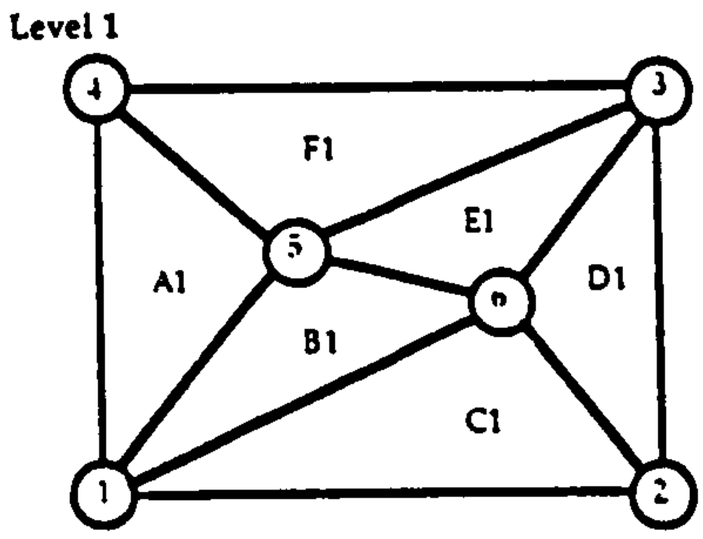
The line generalization tree (Jones 1984, Jones and Abraham 1987), which can be thought of as a relative of the strip tree (Ballard 1981), is an example of a multiresolution data structure in which vertex duplication is minimized, while providing selective access to those vertices required for a particular scale representation. It achieves this by firstly assigning to each point a level of scale significance using a line generalization algorithm and then storing that point at its corresponding level in the tree. Therefore, at each level, only those points which are intermediate to points at the previous level in the tree are stored. The order of points within a linear feature can be maintained by either associating with each point a left and right control value which records the number of adjacent intermediate points at the next lower level or by storing a sequence number for each point which records its position in the original line. Although this method introduces additional data in the form of either the control values or the sequence numbers, it significantly reduces the data overheads of mutliple line storage (Abraham 1988). Jones and Abraham (1987) also describe a strategy for spatially segmenting each scale-specific level in a data-adaptive manner using quadtree cells, the resulting structure being termed a multi-scale line tree.

A data structure with close similarities to the line generalization tree and the strip tree is the BLG-tree (van Oosterom and van den Bos 1989, van Oosterom 1990). This is a binary tree in which each node stores a line segment accompanied by the most distant intermediate point of the original curve, its distance, and pointers to the two line segments defined by the current start and end points and the intermediate point. The BLG-tree differs in particular from the line generalization tree in that the latter employs discrete levels of generalization. For the purpose of combining generalized linear features with the constrained Delaunay pyramid, the use of the line generalization tree appears the more appropriate since the pyramid structure also employs discrete levels of accuracy (or generalization). It is noted that, although originally designed for storing linear features, the line generalization tree is also capable of storing polygonal features.

3. The constrained Delaunay pyramid—a hierarchical terrain model

Hierarchical surface models, such as those described by Gomez and Guzman (1979), Barrera and Vazques (1984), Chen and Tobler (1986) and De Floriani *et al.* (1984), provide representation of a surface at different resolutions. These models are deficient in that they produce approximations which are either numerically inaccurate because of the elongated shape of their constituent triangles or are well-suited only to regularly sampled data (De Floriani 1989). The Delaunay pyramid (De Floriani 1989) is a hierarchical multiresolution surface model which overcomes these problems. It uses a data structure made up from a number of Delaunay triangulations, each approximating the true surface to a different level of accuracy, linked together in increasing order of accuracy in a tree-like manner (figure 1).

The pyramid is built from a set of points S by firstly constructing an initial constrained Delaunay triangulation, which would include those points of S which either define the domain boundary or are the most important surface-specific points (peak, pits and passes) and lines (ridges and valleys). Each triangle is defined by its three



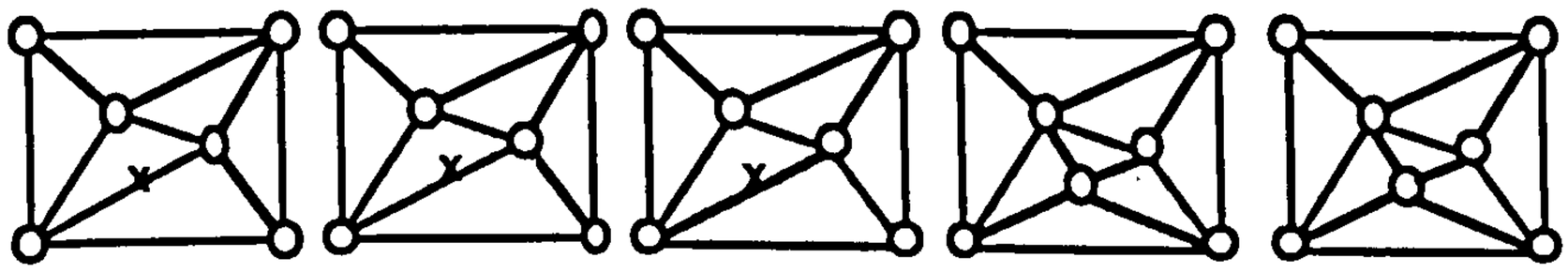
Id	Type	Verticies	Adjacent Triangles	Child Triangles
A1	0	1 5 4	B1 F1 -	A2
B1	0	1 6 5	C1 E1 A1	G2 H2 J2 K2
C1	0	1 2 6	- D1 B1	G2 H2
D1	0	6 2 3	C1 - E1	D2
E1	0	3 5 6	F1 B1 D1	E2
F1	0	3 4 5	- A1 E1	F2
A2	1		K2 F2 -	A3
D2	1		H2 - E2	D3
E2	1		F2 J2 D2	L3 M3
F2	2			L3 M3 N3 P3
G2	0	1 2 7	- H2 K2	G3
H2	0	2 6 7	D2 J2 G2	H3
J2	0	6 5 7	E2 K2 H2	J3
K2	0	7 5 1	J2 A2 G2	K3
A3	1		K3 P3 -	
D3	1		H3 - M3	
G3	2			
H3	2			
J3	1		L3 K3 H3	
K3	2			
L3	0	5 6 8	J3 M3 P3	
M3	0	6 3 8	D3 N3 L3	
N3	0	8 3 4	M3 - P3	
P3	0	5 8 4	L3 N3 A3	

Figure 1. An example of a 3 level Delaunay Pyramid. Triangles of type 0 correspond to internal triangles, triangles of type 1 to boundary triangles and those of type 2 to external triangles. Vertice and adjacency information of boundary and external triangles is found by examining the parent triangle.

vertices and its three adjacent triangles. Pyramid construction proceeds by taking an additional point from the set *S*, that point being the one furthest away from the approximated surface, and adding it to the surface, which is then re-triangulated (figure 2). It is important to note that, owing to the circle criterion in the Delaunay triangulation, only those triangles whose circumcircles contain the new point will have to be re-triangulated. The process of adding points and re-triangulating is repeated until a preset error tolerance is reached for that level. The next level, initially identical to the first level, is then created. Further points are added from *S* until the error tolerance for that level is reached. New levels continue to be added to the pyramid until the most detailed level has been created to the required accuracy.

It is likely that some of the triangles at a particular level will be completely retained in the next lower level or will differ only in regard to their adjacent triangles. The Delaunay pyramid overcomes data duplication by storing triangles as either internal, boundary or external. An internal triangle is defined by its vertices and its adjacent triangles. Boundary triangles will consist of a pointer to a parent triangle, from which its vertices will be obtained, and a reference to its three new adjacent triangles. An external triangle is completely described by a pointer to a higher level triangle. Each triangle in the Delaunay pyramid also maintains a pointer to those triangles contained in the next lower level which intersect it. This provides some direction to spatial search by allowing the pyramid to be traversed quickly once possible triangles have been identified at the top level.

Scarlatos and Pavlidis (1991) point out that the Delaunay pyramid, along with all terrain models based solely on Delaunay triangulation, tends to ignore the third dimension, and may therefore produce edges that contradict the topology of the actual surface. They attempt to overcome this problem by proposing a non-Delaunay triangulation scheme, termed adaptive hierarchical triangulation, which produces a multiresolution terrain model that adapts itself to surface characteristics. However, this method does not appear to be suited to the inclusion of topographic (non-elevation) feature data in the model. To counter the apparent inadequacy of the Delaunay pyramid, De Floriani and Puppo (1988) have proposed a dynamic, easy-to-code algorithm to produce a constrained Delaunay pyramid (figure 3). The ability to introduce constraints into a pyramid ensures that specific linear features, such as valleys and ridges, can be retained as connected edges within each level of the pyramid. In principle, this mechanism for constraining the triangulation facilitates the inclusion within it of any point, linear or polygonal feature, whether physical or cultural.



Triangulation before new point is inserted.	Identify triangle in which the new point lies.	Construct the polygon which bounds the triangles influenced by the new point.	Triangulate the polygon and delete the 'old' triangles.	Triangulation after new point is inserted.

Figure 2.

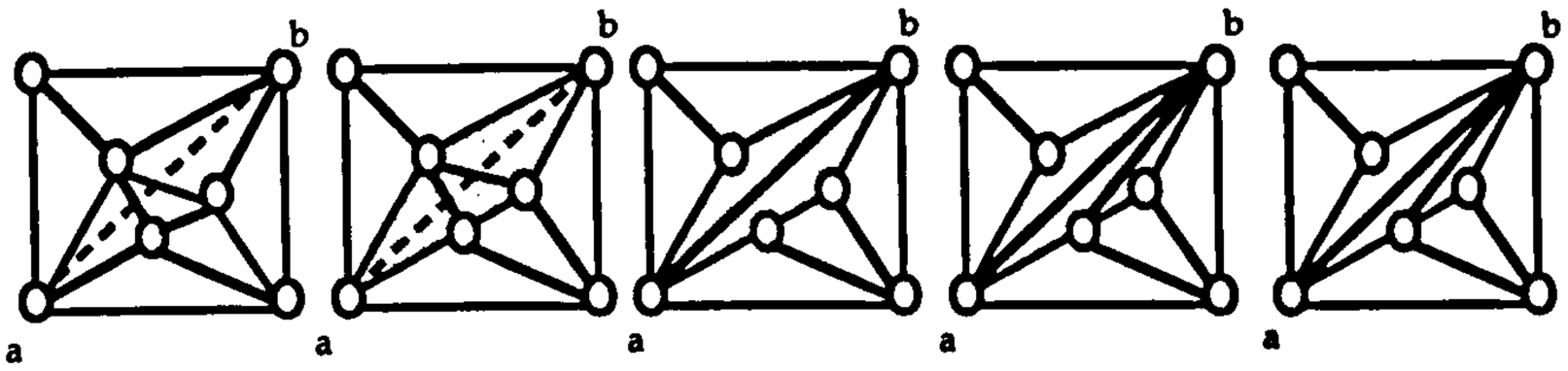


Figure 3.

4. A hierarchical model for both feature and terrain data

The model presented here, building upon the line generalization tree and constrained Delaunay pyramid, introduces a spatial data access scheme suited to the efficient storage and retrieval of terrain and topographical feature data at multiple scales. Vertices, or points, representing point, line and polygonal features, are merged with those points defining the surface to form a single data set. These points, each of which is allocated a unique identifier, are then used to construct a constrained Delaunay pyramid.

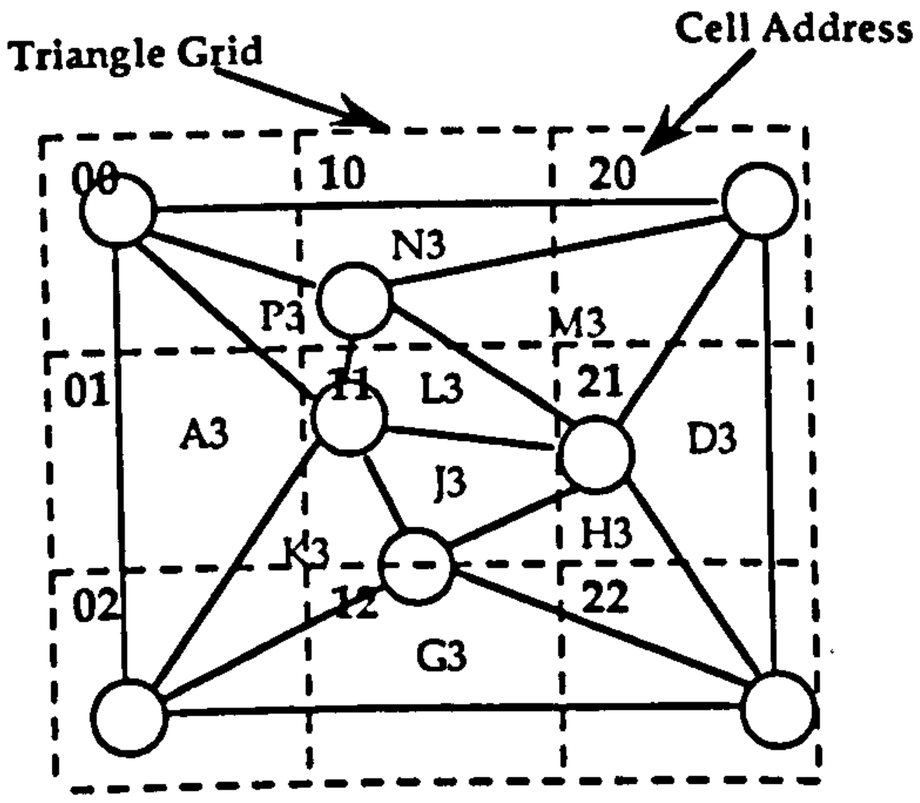
A unique aspect of the work is the ability to include topologically-structured features, such as pylons (point), railways (line) and county borders (polygonal) within the pyramid. In the case of line and polygonal features, these occur as chains of constrained edges within the pyramid. These are in addition to those surface features necessary to characterize the shape of the surface, such as ridges and valleys. Point, line and polygonal features, all of which are embedded within the pyramid, are arranged in a hierarchical manner. Each polygon is stored as a collection of one or more line features, which in turn references, via a line generalization tree, vertices within the pyramid. Point features are represented as direct references to these vertices. It is noted that Kraak and Gazdzicki (1991) present a triangle-based terrain model capable of representing both the terrain surface and spatial objects related to it. This model is applied to what they term Cartographic Terrain Modelling (CTM). The fundamental difference between CTM and the model presented in this paper is that CTM is limited to single-scale representation of data.

In certain cases, constrained edges within a triangulation may represent more than one feature. For example, a national boundary very often coincides with some physical boundary, such as a river. Furthermore, objects of interest may consist of sets of point, line and polygonal features. This can be illustrated by considering a factory as an object of interest, which is itself made up from point, line and polygonal features. To accommodate such occurrences, while at the same time minimizing data duplication, an additional entity, referred to here as an object, is introduced into the data structure hierarchy. Each object consists of an unique object identifier and a list of pointers to the appropriate point, line and polygonal features making up that object. For the first example, the physical boundary and the political boundary, each stored as a separate object, would refer to the same embedded feature or list of features. The factory, also stored as an object, would refer to each of its constituent point, line and polygonal features.

4.1. *Spatial access*

Efficient access to this hierarchical model is provided by introducing spatial indexing at each level in the pyramid. This indexing technique replaces the parent triangle to child triangles pointer method employed by De Floriani. The method used here employs a regular grid overlay scheme, where each grid cell maintains a list of references to all data which intersect it. For the model presented here, it is thought necessary to provide spatial indexing on objects and triangles. Also, since certain objects and triangles may be relevant to some levels in the pyramid and not to others, the spatial access structure has been separated into levels. Therefore, at each level in the pyramid there is an object grid, referencing all objects relevant to that level, and, similarly, a triangle grid, referencing all triangles relevant to that level (figure 4). In this particular model, an object or triangle is deemed to be related to a particular object cell or triangle cell, respectively, if any part of that object or triangle intersects the cell. It is clear that the number of cells in each spatial grid will determine the optimality of searching operations. A dense grid will, in general, be more efficient in terms of search time than a more refined grid. However, this benefit has to be weighed against the resulting increase in storage requirements. McCullagh and Ross (1980), when using a similar type of grid structure to assist in constructing the Delaunay triangulation of a set of points, suggest a grid which allows an average of four points per cell. In the model described here, the number of cells within each grid varies according to the total number of objects or triangles it references, following the approach described by Franklin (1983) for indexing lines for detecting intersections. A temporary grid index, which references points, is also used for the purpose of efficient pyramid construction.

Although this indexing scheme lends itself to referencing all occurrences of objects within a specified area, it does not take into account the spatial extent of individual objects. Thus locationally-specific retrievals involving areally extensive objects could lead to a large amount of unwanted data having to be read. This would be particularly true of high resolution data. This problem could be solved by ensuring that individual object components, that is, the point, line and polygonal features from which an object is made up, be limited in size. This can be achieved by segmenting any overly extensive line features and polygonal features (or to be more precise, the line features from which



Cell Address	Intersecting Triangles
00	A3 P3 N3
10	P3 N3 L3
20	N3 M3 D3
01	A3 P3 K3
11	P3 A3 K3 J3 L3 H3
21	M3 L3 J3 D3 H3
02	A3 K3 G3
12	K3 G3 H3
22	G3 H3 D3

Figure 4. Example of how triangles are spatially referenced using a regular grid overlay. Each cell references all triangles which intersect it. A similar structure is used for accessing objects.

the polygonal feature is made up) into a series of less spatially extensive line features. For example, an object, representing a river, might originally have had a single reference to one spatially extensive line feature. After segmentation has taken place, the same object will now have a list of references to a series of less extensive line features, these having replaced the original line feature. A further enhancement of this structure would be to replace the regular-sized grid with a data-adaptive indexing method such the bounding quadtree (Abel and Smith 1983). This would ensure that no single cell references more than a preset maximum amount of data.

The issue of spatially segmenting line data within a multiresolution model is quite a topical one. The multi-scale line tree (Abraham 1988, Jones and Abraham 1986 and 1987), an extension of the earlier line generalization tree (Jones 1984, Jones and Abraham 1987), provides efficient spatial access to linear data at various scales. It achieves this by classifying the internal points of digitized lines into hierarchies of scale-specific levels, which are themselves spatially segmented in a data-adaptive manner, using quadtree cells. A recent data structure, the Reactive-tree (van Oosterom 1991 and 1990), also provides efficient storage and retrieval of geometric objects at multiple levels of detail. By combining the R-tree (Guttman 1984), which provides efficient access to data objects by storing bounding rectangle information with each object, and the BLG-tree, the Reactive-tree allows both objects and the points making up the objects to be retrieved on the basis of position and scale. A more recently published paper by Becker *et al.* (1991) introduces the Priority Rectangle File (PR-file). Here, the points defining line and polygonal objects are assigned levels of scale significance using a line generalization algorithm. These points are then stored in a data structure which combines certain aspects of the line generalization tree and the R-file (Hutflesz and Six 1990). Here the possible retrieval of unwanted data is minimized by limiting the maximum number of points contained within a single bounding rectangle, thus limiting the spatial extent of individual object parts.

The type of spatial indexing method employed can be governed to a certain extent by the characteristics of the data that are being included in the model. If individual line features are likely to be spatially extensive it would be wise to consider a scheme which is able to segment spatially individual data items. However, it may be that relative to the total area being modelled, individual line features are not extensive. If this is the case it would appear sensible to remain with a simpler approach. The authors, while admitting that the simplicity of their spatial model is currently the sole motivating factor for its use, are as yet unsure whether a more elaborate scheme would significantly improve the spatial search facilities provided by their model. It may be noted that the recent work by van Oosterom (1991 and 1990) and Becker *et al.* (1991) provides examples of multi-scale storage schemes, which in the former case do not segment individual linear features, while in the latter case their component vertices are grouped into rectangular subdivisions. The relative efficiency of the two schemes is not known.

4.2. *Critical point selection*

A method of deciding at which level, and then subsequently at all lower levels, a particular point first appears in the pyramid has to be established. It will be governed by either the point's relevance to a particular object or its significance in describing the surface. The Delaunay pyramid selects points by means of a point insertion algorithm. Here, a point is included at a particular level if the vertical distance of that point from the approximated surface is greater than a given error tolerance for that level. Any point which does not form part of a topographic object will be dealt with in this way.

Those remaining points, all of which form part of an object, present a more difficult problem. The level at which a particular point of a linear feature is inserted can be generated by using a suitable line generalization algorithm (reviewed by McMaster 1987) to classify the internal points of the linear feature into a specified number of levels of scale-related significance. The method used here is that of Douglas and Peucker (1973), which has proved successful in retaining the shape information of a linear feature as the number of points describing it is reduced (McMaster 1983). Another of its properties, essential in allowing a linear feature to be stored hierarchically, is that points selected for small scales are a subset of those used in a larger-scale representation. This algorithm is also suitable to some extent for simple polygonal shapes. Generalization of the points of more complex polygonal features, such as buildings, into levels of scale significance cannot easily be achieved automatically. The level at which these points are inserted would, under the present version of our scheme, have to be determined manually. It should be noted that each point forming part of a feature will also have a height value associated with it which is used to evaluate the point's significance in describing the surface. This value may have formed part of the source data or will have been interpolated using a previously built Delaunay triangulation of all the surface points. It is therefore possible that points which form part of an object may be inserted at a higher level in the pyramid than the level originally indicated by the object generalization procedure.

5. An algorithm for building the hierarchical model

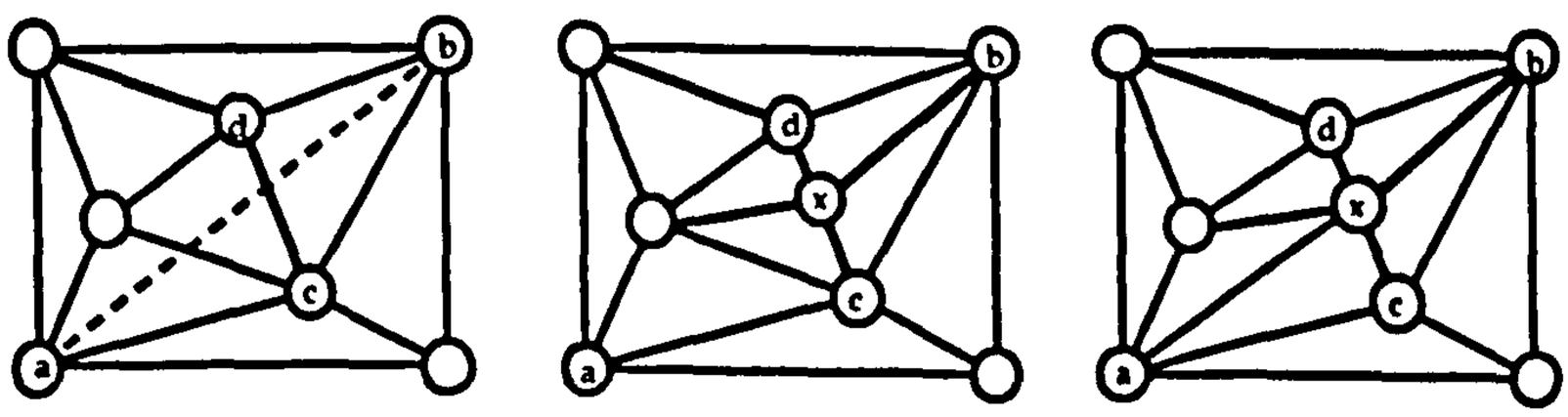
An algorithm, adapted from that of De Floriani and Puppo (1988) for building a constrained Delaunay pyramid (CDP), is given in figure 5. It constructs the model from a set of points *S* and a list of objects O. Each object is defined by a list of references to its constituent point, line and polygonal features. Polygonal features reference line features, which in turn reference points, via a line generalization tree. Each point which forms part of an object must be included in *S* and also have a level flag associated with it. This level flag ensures that it is inserted at the correct level of the pyramid, although it is possible that the point is inserted at some higher level according to its importance in approximating the surface.

Each object is inserted into a triangulation by sequentially inserting each of its line and polygonal feature components (point components will already have been included). Line and polygonal features are inserted as a series of straight-line segments. Algorithms for inserting points and straight-line segments into a Delaunay triangulation are given in the literature (De Floriani and Puppo 1988, Heller 1990). Brief descriptions of these methods have been shown in figure 2 and figure 3.

The CDP algorithm as presented by De Floriani and Puppo is restricted in that it caters only for non-intersecting straight-line segments. This creates a problem when introducing topographic features into the pyramid because their constituent straight-line segments can sometimes intersect each other. For example, this may occur when a road crosses over a county border. The adapted CDP algorithm makes provision for such occurrences by firstly introducing an additional point into the pyramid at the point of intersection of the two line segments (this point is given an interpolated elevation value), and then substituting the two original line segments with four replacement segments. This process is illustrated in figure 6.

```
adapted cdp algorithm
    set up object generalisation tolerances for each level
    assign levels of significance to each object point
    build line generalisation tree for line features
    set up surface error tolerances for each level in the pyramid
    create the initial triangulation
    set up spatial grids
    get surface error tolerance for first level
    set finished = false
    while ( more unused points in S ) and ( not finished )
        find next point to insert
        check accuracy of current level, ie. all unused points are within surface error tolerance
        if ( current level is accurate ) then
            insert any unused object points required at this level
            update spatial grids
            if ( current level is last level ) then
                set finished = true
            else
                get error tolerance for the next level
            endif
        else
            insert the point
            update spatial grids
            set point to used
        endif
    endwhile
    while ( more objects in O to insert )
        for each level in the pyramid
            for each object part
                insert object part into triangulation
            endfor
            update spatial grids
        endfor
    endwhile
end adapted cdp algorithm
```

Figure 5.



Segment a-b cannot be inserted since it intersects another line segment, c-d.

Therefore, delete segment c-d and insert point x where line segments intersected.

Finally, insert four substitute line segments, a-x, x-b, c-x and x-d.

Figure 6.

6. A relational database implementation

The modified constrained Delaunay pyramid algorithm has been implemented in C on a DEC Vax 8800 machine. Objects, features, points, the spatial index grids and the constrained Delaunay pyramid are stored as tables within a relational database management system (figure 7). All processes of pyramid construction and update operate directly on this database. A pyramid is built using the Building Program, which is also used for adding points and objects to any existing pyramid. The pyramid can also be accessed by application programs capable of tasks such as producing 2·5-D views, contouring, profiling and point location and height evaluation.

Each point belonging to S, having been assigned a unique identifier (point_id), is stored in the Point Table. Initially, the level_used flag for all surface points will be assigned a null value while the level_used flag of linear feature points will be set to the level of significance assigned to it by the line-generalization algorithm. The level_used flag of all points will be updated depending on which level in the pyramid the point is first used. It is noted that for large data sets, storing all the points in a single table could lead to slow data access. A more efficient method could be to store all the points associated with particular levels in separate tables.

The Triangle Table holds the details of each of the triangles in the pyramid. The table is constantly updated as the pyramid is built. Each triangle in the pyramid is given an identification number (tri_id) when it is created. It should be noted that if a triangle exists at more than one level (in the form of a boundary or external triangle in the lower level) it will have the same tri_id at each level. The level to which a triangle belongs is

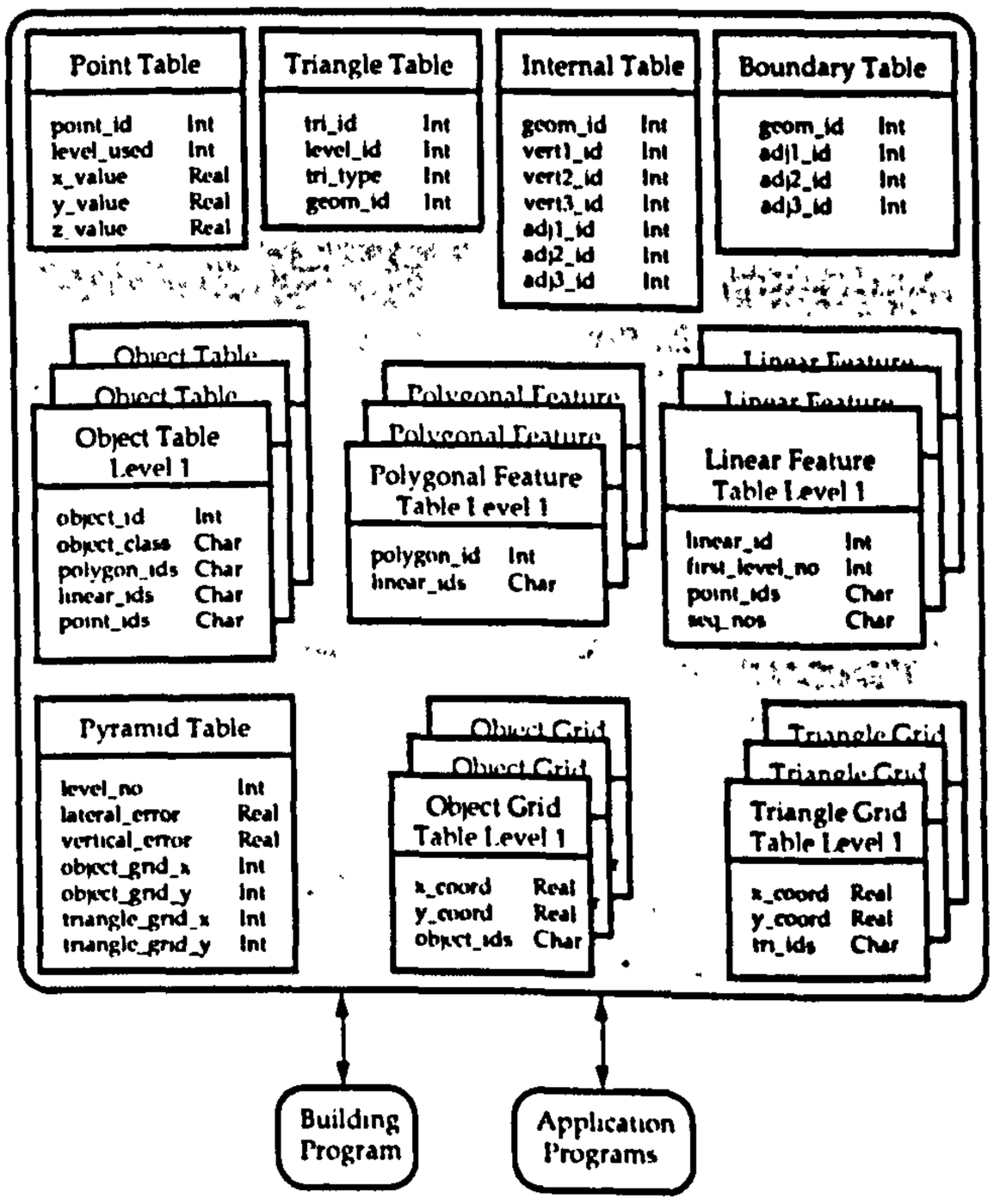


Figure 7. The relational Database Implementation, showing the column (field) descriptions for each table.

denoted by its level_id value. Therefore the tri_id coupled with the level_id forms a unique key for each triangle. As in the case of the Point Table, it may prove beneficial to store each triangulation, or level, in a separate table. The tri_type flag is set to 0, 1 or 2 depending on whether the triangle is internal, boundary or external, respectively. The geom_id field is used as a pointer to the appropriate record in either the Internal or the Boundary Table. The Internal Table holds the full geometry and adjacency information for internal triangles while the Boundary Table holds the adjacency information for boundary triangles. The vertices of Boundary triangles are found by obtaining details from a higher level triangle. No External Table is required since the geometry and adjacency information of an external triangle can be found by retrieving the details of the triangle with the same tri_id as it in the previous level of the pyramid.

There is a Linear Feature Table, a Polygonal Feature Table and an Object Table for each level in the pyramid. Each of the Linear Feature Tables can be thought of as a level in a line generalization tree. An individual linear feature (linear_id) is described by a list of points (point_ids), each of which has a final pyramid level of significance assigned to it. Each of these points has a sequence number (seq_nos) associated with it indicating its position in the original line. To assist in making the retrieval of data more efficient, the Linear Feature Tables also keep a record of the level at which each linear feature first appears in the pyramid (first_level_no). The polygonal features present at a particular level in the pyramid are stored in the Polygonal Feature Table assigned to that level. Each polygonal feature (polygonal_id) is described by a list of linear features (linear_ids) from which it is made up. Each Object Table contains a reference to all objects relevant to its particular level in the pyramid. Each object reference consists of an object identifier (object_id), a list of object class identifiers (class_ids) and lists of the point (point_ids), linear (linear_ids) and polygonal (polygon_ids) features which make up that object. The object class identifiers are used to assist in thematic retrievals of information.

The object Grid Tables and Triangle Grid Tables correspond to the spatial grids detailed in § 4.1. Each entry in a spatial table consists of the x, y coordinates (x_coord, y_coord), of the bottom left hand corner of the cell it represents and a list of objects or triangles (object_ids or tri_ids) which intersect that cell. The number of cells in the x and y direction of each object grid (object_grid_x, object_grid_y) and each triangle grid (triangle_grid_x, triangle_grid_y) is stored in the Pyramid Table. This table also stores the lateral error (related to object resolution) and vertical error (related to terrain resolution) associated with each level.

All lists stored in the relational database tables (point_ids, linear_ids, polygon_ids, seq_nos, object_ids and tri_ids) are, for implementation purposes, stored as character strings. These strings are converted into integer values by the Building Program (and any applications program) before being used.

7. Integration of data into the relational database system

The system described has been used to model data acquired from the British Geological Survey (BGS). Two data sets were involved—line data, representing geological outcrop boundaries, and terrain data in the form of irregular (x, y, z) point data. The data covers a 5 km by 5 km square in the Grantham area. Figure 8 is a plot of a 2·5 km by 2·5 km section of the terrain data and contains 894 points. Figure 9 is the geological line data for the same area. The lines are represented by a total of 727 points.

The data sets were used as input for the constrained Delaunay pyramid Building Program. The number of levels in the pyramid was restricted (arbitrarily) to two.

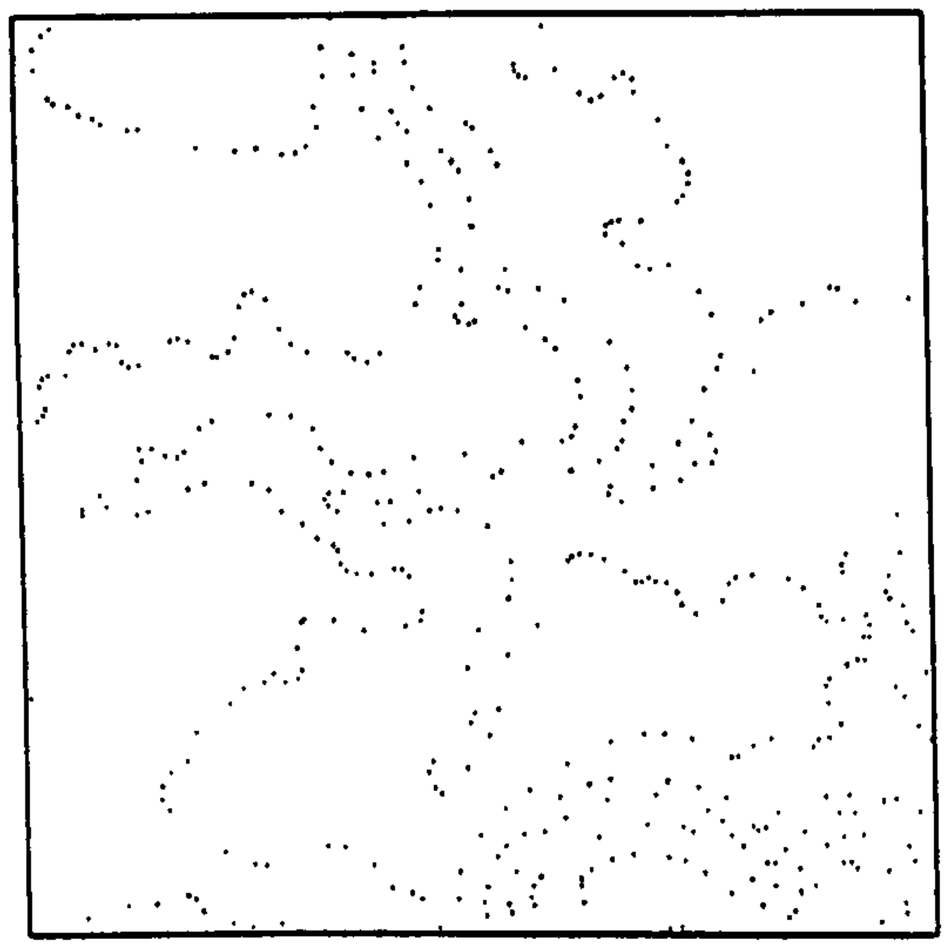


Figure 8. Number of points = 894.

Figure 9. Number of points = 727.

Additional, intervening levels could have been included, giving a more gradual change in detail between levels. It should be noted that the storage benefit gained by having boundary and external type triangles within the pyramid is fully realised only when changes in detail between successive levels are relatively small. This is because large changes in scale between successive levels will probably cause many of the triangles in the parent triangulation to be replaced by new, non space-saving internal triangles in the lower level triangulation. Each level has two-error tolerances associated with it: (i) vertical error which governs how accurately the level depicts the terrain, and (ii) lateral error which governs how accurately the line data are represented at that level. The error tolerances chosen in this case were

Level	Vertical error (m)	Lateral error (m)
1	10·0	50·0
2	5·0	10·0

These error tolerances were chosen simply to emphasize the change in detail encountered at each level of the pyramid. The authors are at present unaware of a relationship that exists between these error tolerances and true scale. The constrained Delaunay pyramid produced is shown in figures 10 to 13.

8. Performance

The worst-case time complexity of our adapted constrained Delaunay pyramid algorithm is $O(n^2)$, which represents a lower limit for any incremental Delaunay triangulation algorithm applied to a set of points S (see De Floriani and Puppo 1988). The relational database implementation has been tested on a number of large data sets. Performance for building the database was poor in terms of processing time, particularly when compared to an indexed sequential UNIX-based C version which

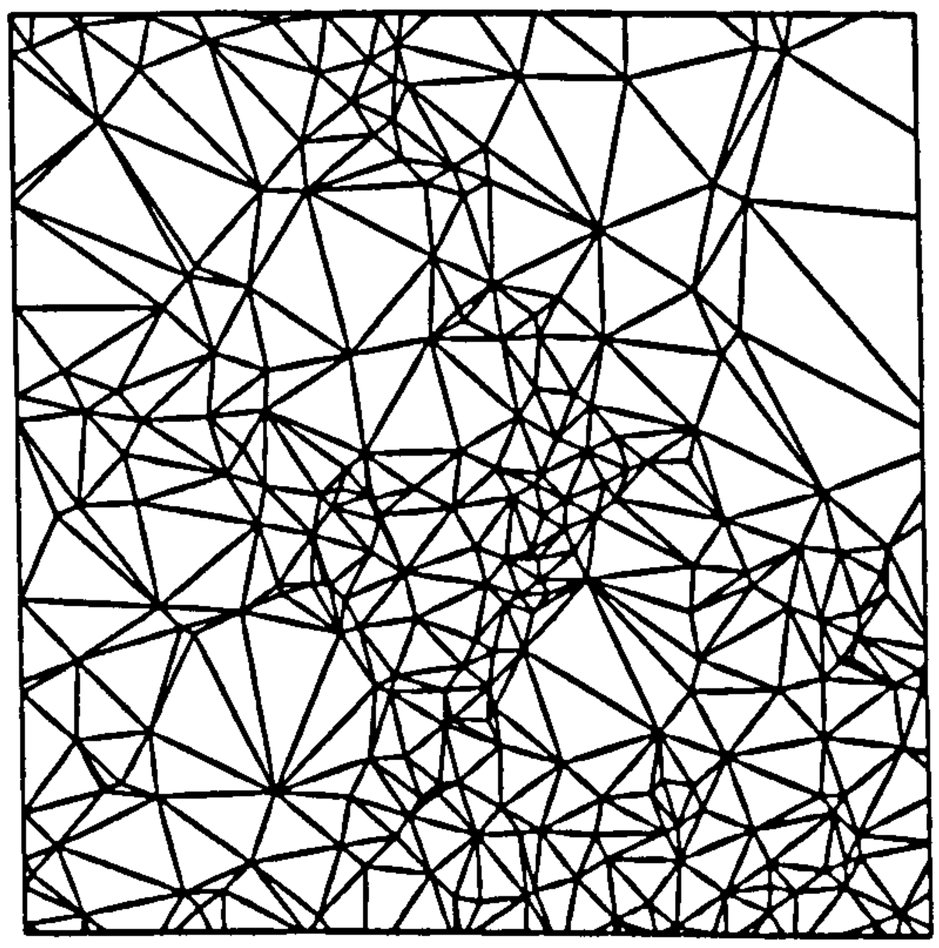


Figure 10. Level 1 unconstrained. 118 level points ($e = 10$ m), 177 line points ($e = 50$ m).

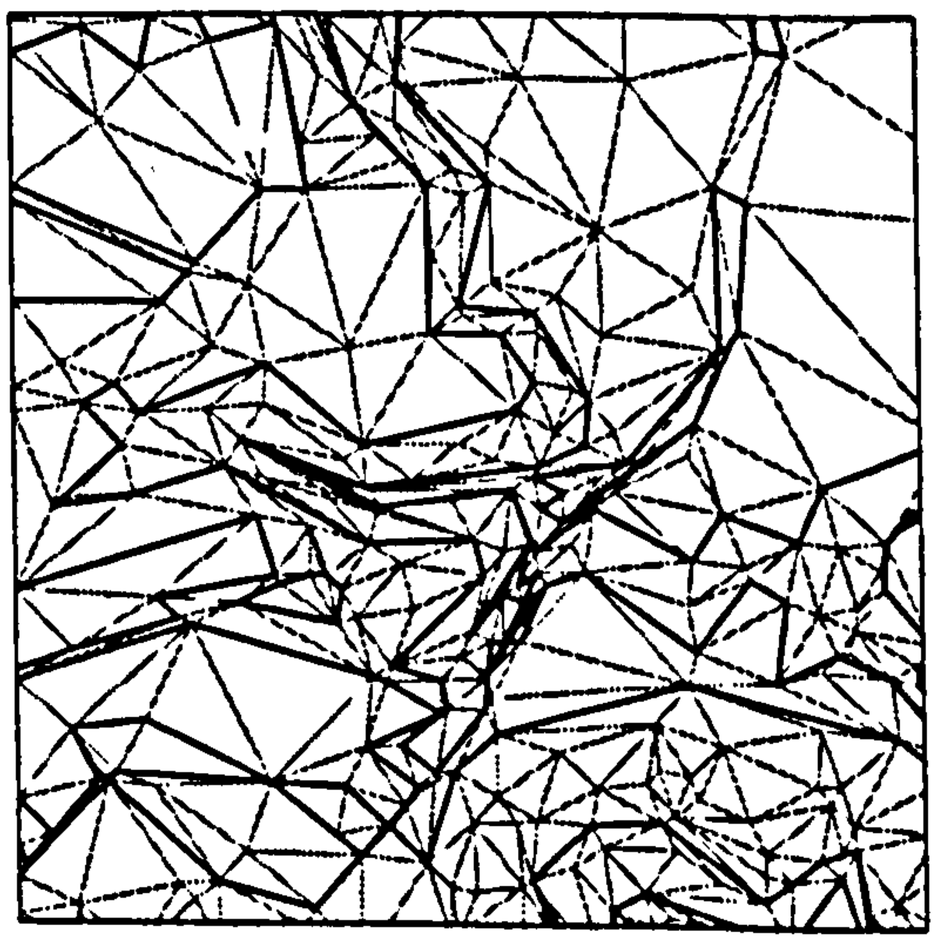


Figure 11. Level 1 constrained. 118 level points ($e = 10$ m), 177 line points ($e = 50$ m).

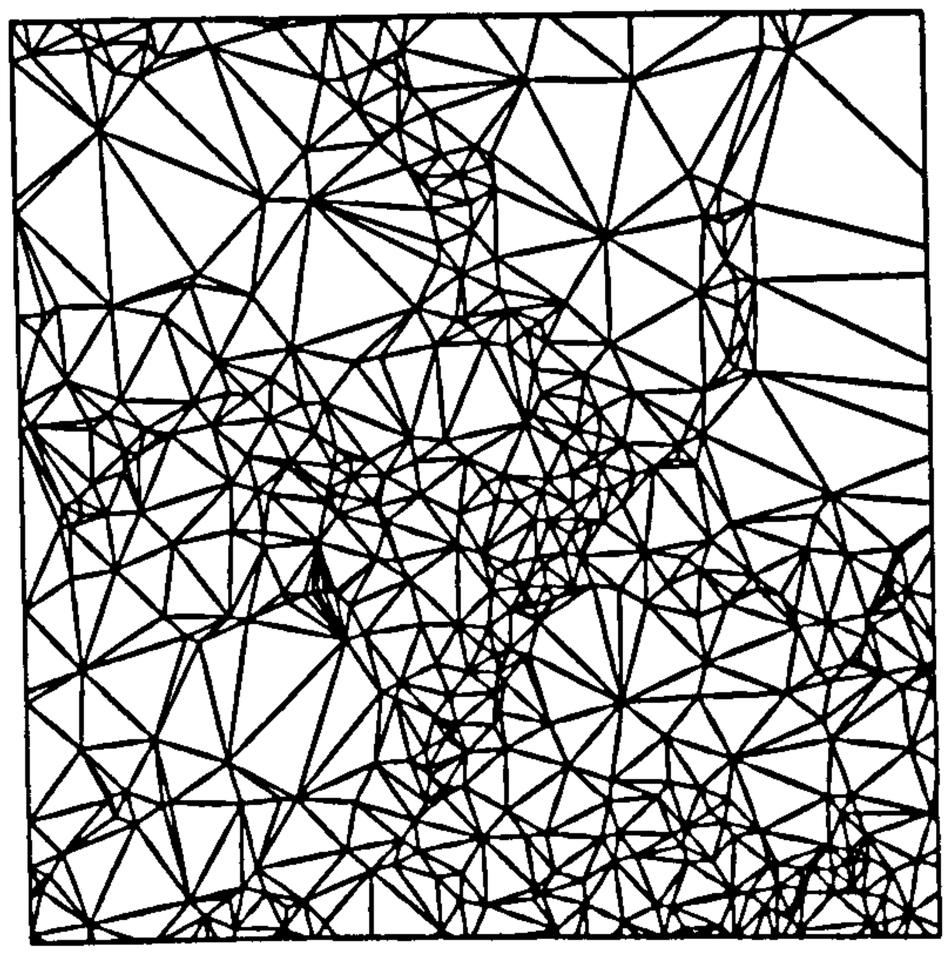


Figure 12. Level 2 unconstrained. 190 level points ($e = 5$ m), 338 line points ($e = 10$ m).

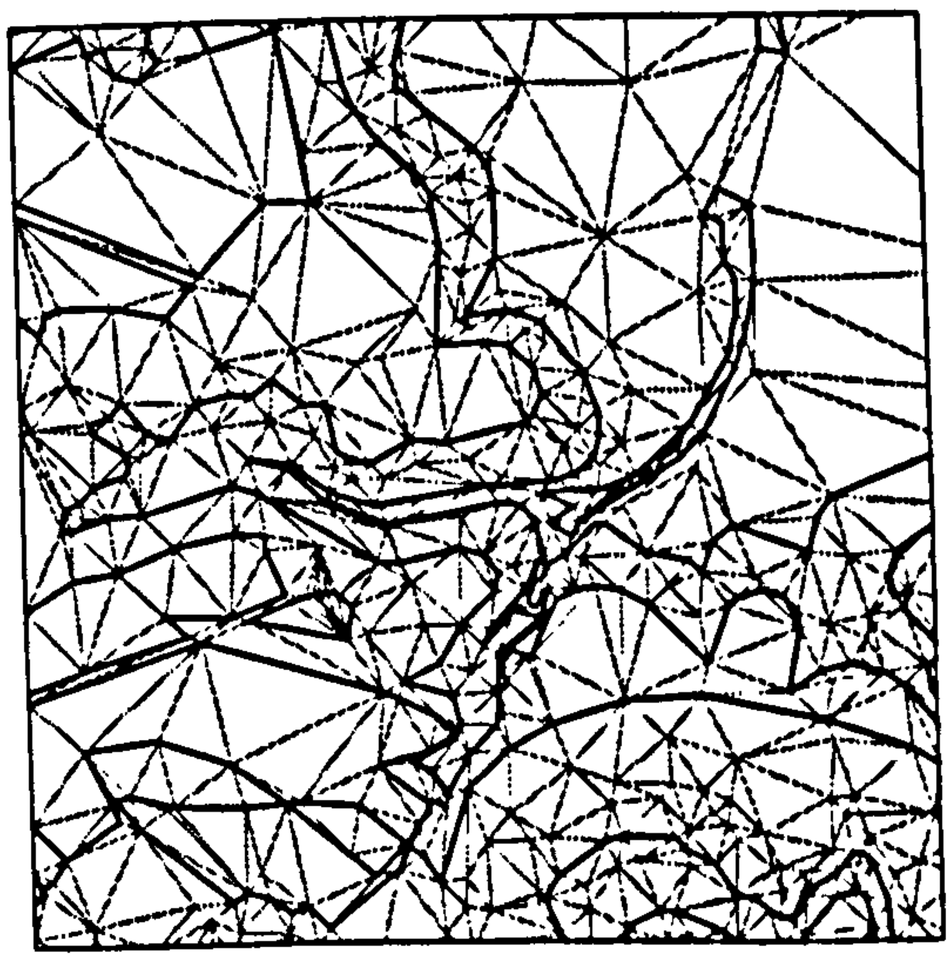


Figure 13. Level 2 constrained. 190 level points ($e = 5$ m), 338 line points ($e = 10$ m).

has also been implemented. For example, database construction for the BGS data can take up to 2 hours using the relational database version compared to a time of about a minute for the indexed sequential UNIX-based version. This is due for the most part to the large number of single record SELECT and INSERT operations delivered by the program to the database during the building process. It is believed that time efficiency will improve significantly by implementing within the program a type of paging mechanism. One possibility here would be for large blocks of tables, possibly whole tables, to be read into internal arrays where they would be processed before being returned to the database, thus simulating the main memory-based version. It should be noted, however, that the emphasis of the current research is on the logical design rather than an optimal implementation of the proposed database.

The amount of storage required for De Floriani's pyramid structure is the space used to store each triangulation plus that required to store the inter-level links (De Floriani 1988) and is given by

$$4n' + 5(m+1) + \sum_{i=0}^{m} (9t_i + 2s_i) \tag{1}$$

where n' is the total number of points used in the pyramid, $(m+1)$ gives the number of levels in the pyramid, t_i represents the number of triangles at level i and s_i gives the number of triangles of level $(i+1)$ intersected by triangles of level i. This equation indicates a storage requirement of $O(n)$, where n is the total number of points in S.

Using a similar notation, the storage requirements of the proposed scheme compare favourably with those of the constrained Delaunay pyramid and consist of the space required to store each triangulation plus the spatial index at each level. Using a separate table for each triangulation and considering the worst case, that is when all triangles are stored as internal and each triangle grid cell contains a list of all triangles contained in the triangulation (for the sake of a valid comparison with De Floriani, the storage required for spatially indexing objects is neglected), the amount of storage required is represented as

$$4n' + 2(m+1) + \sum_{i=0}^{m} (11 + 2x_i y_i) t_i \tag{2}$$

where x_i and y_i are the number of spatial cells in the horizontal and vertical directions at level i respectively. Since $m \ll n$, storage requirements will be independent of the number of levels in the pyramid and will be highly dependent on the number of spatial cells at each level.

Additional storage will be required for the Linear Feature Tables, Polygonal Feature Tables and Object Tables. This will be directly proportional to the number of features and objects encoded. It should be noted that object specification requires referencing only the identity of vertices, rather than the coordinates, which are stored only once in the Points Table.

9. Discussion

An alternative approach to implementing the spatial index for triangles would be to store only one triangle per cell, possibly the most central. All other triangles intersecting that cell could then be deduced by accessing recursively the surrounding adjacent triangles. Equation (2) would then become

$$4n' + 2(m+1) + \sum_{i=0}^{m} (11t_i + 3x_i y_i) \tag{3}$$

which is $O(n)$.

The incremental triangulation algorithm that we used relies on the fact that inserting a point or constraining edge results in only local modifications to the triangulation. It may be that the dual edge structure rather than the traditional triangle-oriented construct used in our model, would be more appropriate for such an algorithm (Heller 1990).

At present, points are selected for inclusion at a particular level depending on either their importance in describing the surface or their significance in describing a particular object. Thus surface-specific points are selected according to their vertical displacements relative to the surface, while object points may also be selected on the basis of

their lateral displacement. Work has recently begun to design data structures suitable for modelling 3-D geological data (see Jones 1989). With this in mind, investigations into developing a fully 3-D point selection algorithm are currently being undertaken.

An alternative approach to storing the pyramid, which would save a considerable amount of storage space, is that of using implicit triangulations (Kidner 1991, Kidner and Jones 1991). The pyramid would be built in the same way as previously described. However, once all required levels of accuracy were obtained and points allocated a level at which they are included, no permanent storage of the triangulations would be retained. Triangulations could then be reconstructed at run time using a suitable constrained Delaunay triangulation algorithm. This method could take advantage of the recent development of fast parallel algorithms for building the constrained Delaunay triangulation from a set of points. An example of such an algorithm, for normal Delaunay triangulations, has recently been implemented using a network of transputers (Ware and Kidner 1991). Also, for certain applications, it may not be appropriate to include all classes of object as constraints within the pyramid. Implicit triangulations would allow the user to select, at run time, which classes of object are to be included in the model. It is thought that such a system would benefit by storing those points forming part of an object and those which define the surface in separate tables.

10. Conclusion

A data storage scheme has been presented which succeeds in allowing terrain and topographic object data to be combined in a single database at multiple levels of detail. Points, lines and polygons are integrated with, and serve to constrain, an hierarchical triangulation which avoids data duplication. The database is accessed via a spatial index referencing topographic objects and the triangles that model the terrain surface. The scheme has been implemented with both a commercial relational database management system and with an indexed sequential UNIX-based file handling facility. For the purpose of building the database, the relational implementation gave a relatively poor time performance, which experiments with main memory data processing indicate could be greatly improved. Current research efforts are concerned with developing a fully three-dimensional multiscale model suitable for representing geological data.

Acknowledgments

The authors would like to express their appreciation to The British Geological Survey who have provided financial and technical support for parts of the research presented in this paper. JMW is currently supported by a SERC CASE studentship.

References

ABEL, D. J., and SMITH, J. L., 1983, A data structure and algorithm based on a linear key for rectangular retrieval. *Computer Visions, Graphics and Image Processing*, **2**, 1–13.

ABRAHAM, I. M., 1988, Automated cartographic line generalisation and scale-independent databases. Ph.D Thesis, Dept. Computer Studies, The Polytechnic of Wales.

BALLARD, D. H., 1981, Strip trees: a hierarchical representation for curves. *Communications, Association for Computing Machinery*, **24**, 310–398.

BARRERA, R., and VAZQUES, A. M., 1984, A hierarchical method for representing relief. *Proceedings Pecora IX Symposium on Spatial Information Technologies for Remote Sensing Today and Tomorrow, Sioux Falls, South Dakota* (South Dakota: IEEE), pp. 87–92.

BECKER, B., SIX, H.-W., and WIDMAYER, P., 1991, Spatial priority search: An access technique for scaleless maps. *Proceedings SIGMOD 1991* (Denver, Colorado: ACM), pp. 128–137.

CHEN, Z. T., and TOBLER, W. R., 1986, Quadtree representation of digital terrain. *Proceedings of Auto-Carto London* edited by M. Blakemore (London: Auto-Carto London Ltd), pp. 475–484.

DE FLORIANI, L., 1988, A data structure for encoding a Delaunay pyramid. Technical Report IMA n. 19/88 Genoa, Italy, February 1988.

DE FLORIANI, L., 1989, A pyramidial data structure for triangle-based surface description. *I.E.E.E. Computer Graphics and Applications*, **March 1989**, 67–78.

DE FLORIANI, L., and PUPPO, E., 1988, Constrained Delaunay triangulation for multiresolution surface description, *IEEE Computer Society Reprint* (Washington DC: Computer Society Press). (Reprinted from *Proceedings Ninth IEEE International Conference on Pattern Recognition, Rome, November 1988*).

DE FLORIANI, L., FALCIDIENO, B., NAGY, G., and PIENOVI, C., 1984, A hierarchical structure for surface approximation. *Computer and Graphics*, **8**, 183–193.

DOUGLAS, D. H., and PEUCKER, T. K., 1973, Algorithms for the reduction of the number of points requires to represent a digitised line or its caricature. *Canadian Cartographer*, **10**, 112–122.

FRANKLIN, W. R., 1983, Adaptive grids for geometric operations. *Proceedings of Auto-Carto 6*, edited by B. S. Wellar (Ottawa: Steering Committee of Auto-Carto 6), pp. 230–239.

GOMEZ, D., and GUZMAN, A., 1979, Digital model for three-dimensional surface representation. *Geo-Processing*, **1**, 53–70.

GUTTMAN, A., 1984, R-trees: A dynamic index structure for spatial searching. *Proceedings 1984 ACM-SIGMOD International Conference on Management of Data, June 1984* (Boston, MA: ACM), pp. 47–57.

HELLER, M., 1990, Triangulation algorithms for adaptive terrain modelling. *Proceedings Fourth International Symposium on Spatial Data Handling, Zurich*, edited by K. Brassel and H. Kishimoto (Zürich: International Geographical Union), pp. 163–174.

HUTFLESZ, A., SIX, H.-W., and WIDMAYER, P., 1990, The R-file: An efficient access structure for proximity queries. *Proceedings of the IEEE Sixth International Conference on Data Engineering* (Los Angeles: IEEE), pp. 372–379.

JONES, C. B., 1984, A tree data structure for cartographic line generalisation. *Proceedings Eurocarto III* (Graz: Research Center Joanneum, Institute for Image Processing and Computer Graphics).

JONES, C. B., 1989, Data structures for three-dimensional spatial information systems in geology. *International Journal of Geographical Information Systems*, **3**, 15–31.

JONES, C. B., and ABRAHAM, I. M., 1986, Design considerations for a scale-independent caruographic database. *Proceedings of the Second International Symposium on Spatial Data Handling, Seattle*, edited by D. F. Marble (Seattle: International Geographical Union), pp. 384–398.

JONES, C. B., and ABRAHAM, I. M., 1987, Line generalisation in a global cartographic database. *Cartographica*, **24**, 32–45.

KIDNER, D., 1991, Digital terrain models for radio path loss calculations. Ph.D. Thesis, Dept. Computer Studies, The Polytechnic of Wales. (available from Defence Research Information Centre, Kentigern House, Brown Street, Glasgow).

KIDNER, D., and JONES, C. B., 1991, Implicit triangulations for large terrain databases. *Proceedings of the Second European Conference on Geographical Information Systems, Brussels, April 1991*, edited by J. Harts, H. F. L. Ottens and H. J. Scholten (Utrecht: EGIS Foundation), pp. 537–546.

KRAAK, M. J., and GAZDZICKI, J., 1991, Triangulation based modelling of spatial objects in relation to the terrain surface. *Proceedings of the Second European Conference on Geographical Information Systems, Brussels, April 1991*, edited by J. Harts, H. F. L. Ottens and H. J. Scholten (Utrecht: EGIS foundation), pp. 564–572.

MCCULLAGH, M. J., and ROSS, C. G., 1980, Delaunay triangulation of a random data set for isarithmic mapping. *The Cartographic Journal*, **17**, 93–99.

MCMASTER, R. B., 1983, A mathematical evaluation of simplification algorithms. *Proceedings of Auto-Carto 6*, edited by B. S. Wellar (Ottawa: Steering Committee of Auto-Carto 6), **2**, pp. 267–276.

MCMASTER, R. B., 1987, Automated line generalisation. *Cartographica*, **24**, 74–111.

PEUCKER, T. K., FOWLER, R. J., LITTLE, J. J., and MARK, D. M., 1978, The triangulated irregular network. *Proceedings of the Digital Terrain Models Symposium, ASP/ACSM, May 1978* (Falls Church, Virginia: ASP and ACSM), pp. 516–540.

SCARLATOS, L., and PAVLIDIS, T., 1991, Adaptive hierarchical triangulation. *Auto-Carto 10, Baltimore, March 1991* co-Chaired by D. Mark and D. White (Baltimore: ACSM and ASPRS), pp. 234–246.

SCHEK, H.-J., and WATERFELD, W., 1986, A database kernel system for geoscientific applications. *Proceedings of the Second International Symposium on Spatial Data Handling, Seattle* edited by D. F. Marble (Seattle: International Geographical Union), pp. 273–288.

VAN OOSTEROM, P., 1991, The Reactive-tree—A storage structure for a seamless, scaleless geographic database. *Auto-Carto 10, Baltimore, March 1991*, co-Chaired by D. Mark and D. White (Baltimore: ACSM and ASPRS), pp. 393–407.

VAN OOSTEROM, P., 1990, Reactive data structures for geographic information systems. Ph.D. Thesis, Dept. Computer Science, Leiden University.

VAN OOSTEROM, P., and VAN DEN BOSS, J., 1989, An object-oriented approach to the design of geographic information systems. *Proceedings of the First Symposium on Large Spatial Databases SSD '89, July 1989*, (Santa Barbara: University of California), pp. 255–269.

WARE, J. A., and KIDNER, D., 1991, Parallel implementation of the Delaunay triangulation within a transputer environment. *Proceedings of the Second European Conference on Geographical Information Systems, Brussels, April 1991*, edited by J. Harts, H. F. L. Ottens and H. J. Scholten (Utrecht: EGIS foundation), pp. 1199–1208.

The Implicit Triangulated Irregular Network and Multiscale Spatial Databases

CHRISTOPHER B. JONES*, DAVID B. KIDNER† AND J. MARK WARE†

*Department of Geography, University of Cambridge, Downing Place, Cambridge CB2 3EN, UK
†Department of Computer Studies, University of Glamorgan, Pontypridd, Mid Glamorgan CF37 1DL, UK

The triangulated irregular network (TIN) provides a versatile and widely used approach to representing terrain models in a way that retains the original sample points, adapts to variation in data density and incorporates linear features corresponding to natural or man-made phenomena. Classification of the scale-related priority of the constituent points and linear features can be used to create hierarchical, multiresolution TIN representations. A large proportion of the data items included in conventional and hierarchical TIN data structures are concerned with recording the topology of the triangulation. Although TINs typically use many fewer points than the main alternative representation of regular rectangular grids, they do not usually occupy much less data storage, due to the topological data. This paper describes a novel multiresolution storage scheme which uses an approach termed the Implicit TIN, in which storage requirements are reduced significantly by storing only the vertices and constraining features. TIN topology is reconstructed by a procedure when required. The Implicit TIN storage scheme has been demonstrated in the context of an experimental multiscale database. Variable-scale access is provided to polygonal regions of a terrain model which includes polygon, line and point objects that constrain the constructed triangulated model.

Received April 2, 1993, revised July 30, 1993

1. INTRODUCTION

The rapid growth in the use of geographical information systems (GIS) has introduced the requirement for terrain models that combine digital elevation data with natural and man-made topographic features. Applications requiring such models include landscape architecture, civil engineering and radio communications network planning. A characteristic of many such GIS applications is the need to retrieve data at different levels of detail, or generalization, for purposes of scale-variable visualization and analysis.

A spatial model which can represent digital elevation data at its original locational precision and can conform exactly to known linear features, such as ridges, roads and valleys, is the triangulated irregular network (TIN) [22] A TIN defines a triangulation for a given set of sample points. In the absence of linear features, it is common practice to create a Delaunay triangulation. It is characterized by providing the set of most equiangular triangles [11, 23], a property which is desirable when interpolating within triangles. A Delaunay triangle is one in which a circumscribing circle passing through its vertices contains no other points (Figure 1). Sibson [23] states that for a finite set of distinct data sites, there is only one locally equiangular triangulation, known as the Delaunay triangulation, which is the dual of the Dirichlet, Voronoi or Thiessen tessellation. This is an important concept in geographical applications, since a Thiessen polygon can be used to define the region of influence of any point in an areal context [20]. In Figure 1, points 1-6 are known as the Thiessen neighbours of point P.

Since a Delaunay triangulation is dependent only upon the spatial distribution of vertices, it cannot be guaranteed to conform to known linear features that are defined by subsets of the vertices. However, the triangulation can be constrained, such that the linear features are correctly represented by a sequence of triangle edges [4].

The main alternative to the TIN is the regular rectangular grid. Although regular grids are convenient for storage and some spatial data processing operations, they are not able to preserve arbitrarily located point and linear data, except at the cost of significant data redundancy. Some commercial GIS use the TIN for the primary database and convert to a grid temporarily, to assist, for example, in visualization.

For the purposes of multiscale retrieval, hierarchical data structures based on the TIN have been developed. The Delaunay Pyramid [6] succeeds in retaining the properties of Delaunay triangulation at all levels of detail. It has been modified in the Constrained Delaunay Pyramid (CDP) [7] to incorporate linear features corresponding to known structural edges in the terrain. Differences in resolution or scale between the hierarchical levels of the CDP are determined solely on the basis of vertical error of a level relative to the highest level of detail available. If the surface model is constrained by linear features which represent objects, such as rivers and roads, the use of vertical error alone is inadequate, since it will not retain corresponding degrees of general-

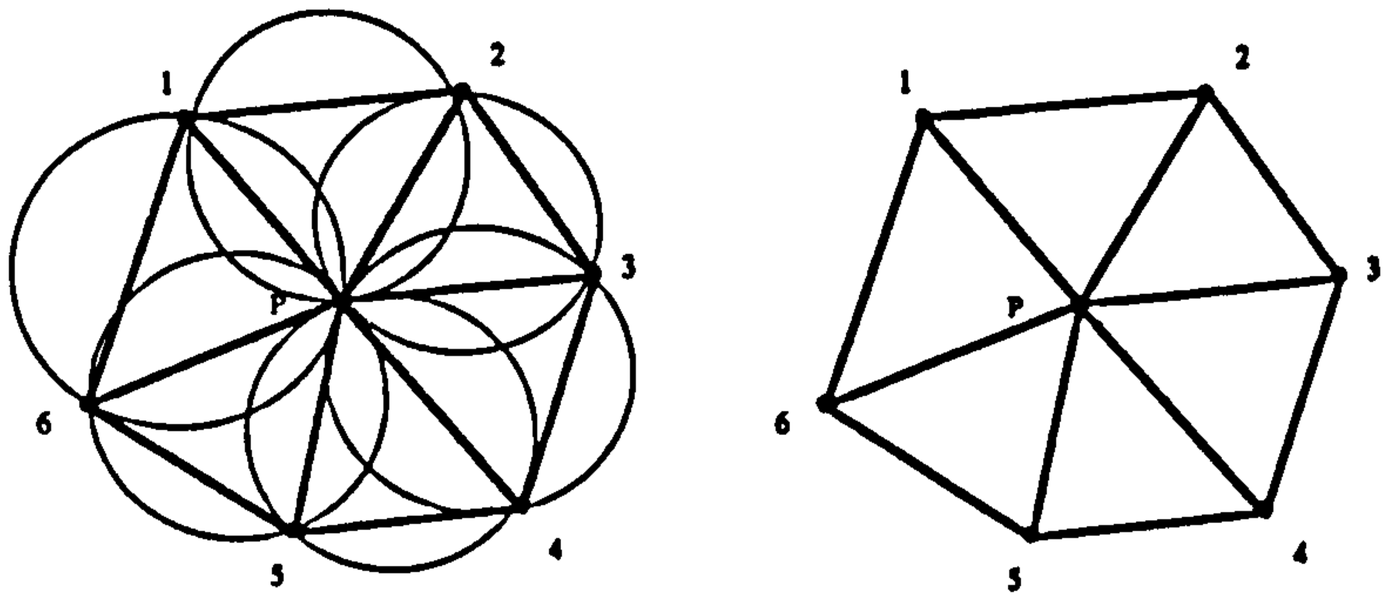
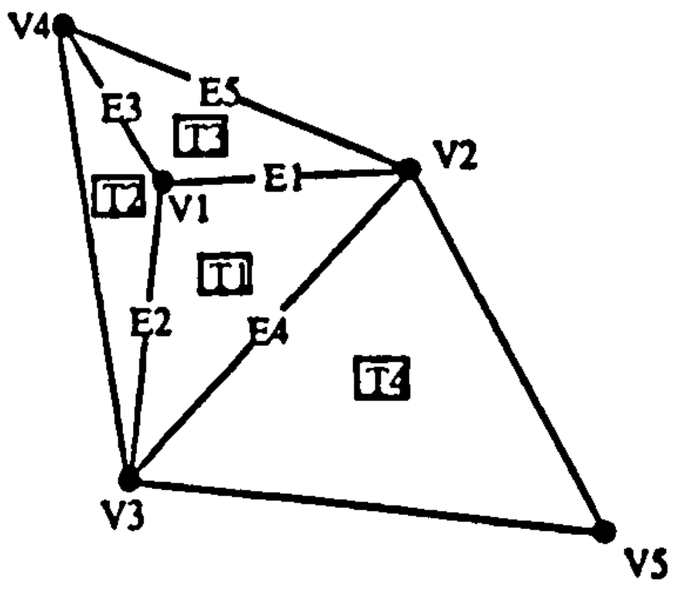


FIGURE 1. Local Delaunay triangulation about a point P.

ization in the lateral displacement of lines. This issue has been addressed in the Multiresolution Topographic Surface Database (MTSD) [27], which integrates the constrained Delaunay Pyramid with a multiresolution representation of linear features, the Multi-Scale Line Tree (MSLT) [15, 16].

In creating databases that represent TIN-based data structures, a large amount of storage is taken up by the pointers used to represent the topology, such as the connectivity of triangle vertices and edges. De Floriani [5] states that the topology of a triangular subdivision is completely and unambiguously represented by any suitably selected subset of nine adjacency relations between entities (vertices, edges, triangles). Referring to Figure 2, the assumption is that having stored the coordinates of each uniquely identified vertex *Vn*, some additional data must be stored to define the structure of the triangulation. Which of the nine schemes is most appropriate will depend upon a combination of type of operations to be carried out on the triangulation and the importance of saving storage space. For any triangulation of N nodes, B of which are on the boundary (convex hull), there are $2N-B-2$ triangles and a total of $3N-B-3$ edges, or $6N-2B-6$ directed pointers, if stored as links from each vertex. For a vertex-based TIN which references edges, each node's coordinates may be stored with a pointer to the list of connected vertices (referred to as a vertex–vertex relation, i.e. 1 in Figure 2). If coordinates and pointers require the same unit storage space, the total storage will approximate to $9N$ (i.e. x, y and z coordinates for each of the N nodes and $6N-2B-6$ links). The triangle-based TIN will require more storage (approximately $15N$) since for each of the $2N-B-2$ triangles, pointers to the three vertices and three neighbouring triangles are stored ($12N-6B-12$), together with the vertex coordinates ($3N$).

In contrast to the nine schemes referred to, it is possible to reduce the permanent storage requirements of the TIN very greatly by only storing the vertices and any linear constraints on the triangulation. When the triangulation, or a part of it, is required for a particular application, it can be reconstructed temporarily using a triangulation algorithm. Provided the algorithm operates on predetermined criteria such as the Delaunay triangulation and its constrained variant, it is possible to ensure that the topology of the original triangulation will be reconstituted. The approach is based on offsetting storage against computation and is called Implicit Triangulation. It was implemented originally for the application of retrieving profiles for radio path loss calculations from a large, single scale, terrain database [17]. In this paper we present, for the first time, the algorithms used to implement the Implicit TIN and show how they can be applied to create a multiscale



1 Vertex - Vertex : Given V1 Store V2, V3, V4
2 Vertex - Edge : Given V1 Store E1, E2, E3
3 Vertex - Triangle : Given V1 Store T1, T2, T3

4 Edge - Vertex : Given E1 Store V1, V2
5 Edge - Edge : Given E1 Store E4, E2, E5, E3
6 Edge - Triangle : Given E1 Store T1, T3

7 Triangle - Vertex : Given T1 Store V1, V2, V3
8 Triangle - Edge : Given T1 Store E1, E4, E2
9 Triangle - Triangle : Given T1 Store T2, T3, T4

FIGURE 2. Illustration of the 9 possible relations between pairs of entities in a TIN (where *Vn*, vertices; *En*, edges and *Tn*, triangles).

database which integrates ground elevation data with other topographic features.

By reconstructing a surface from its original vertices at the time it is retrieved, the Implicit TIN provides the basis for a versatile approach to building multiscale databases. In a multipurpose spatial database, both the type of features to be retrieved and the detail with which they should be represented may vary from one retrieval to another. If terrain elevation data are to be integrated in a flexible manner with ground surface features, it is desirable to defer the definition of constraints on the surface triangulation until the features of interest are defined. Thus the retrieved surface will consist only of the relevant features. Classification of vertices according to their importance in reducing error in vertical elevation and in laterally defined feature representation allows selective retrieval of those data items that are appropriate to the application requirements.

In the following sections we start by describing our current multiscale version of the Implicit TIN, with reference to the selection of vertices for different scales and the reconstruction of an explicit TIN within a spatially defined subregion of the database. This is followed in Section 3 by a review of multiscale representations of linear features and an explanation of how such features can be integrated with the Implicit TIN. Section 4 describes briefly how multiscale polygonal objects can be represented in terms of their consituent linear boundaries. Section 5 describes the design and performance characteristics of a multiscale topographic surface database which integrates terrain elevation data with points, lines and polygons, and composite objects defined in terms of these primitive spatial objects. The final section concludes with a summary of the use of the Implicit TIN and outlines future research directions related to multiscale geographical databases.

2. THE IMPLICIT TIN

The Implicit TIN provides a highly compact storage scheme for representing topographic surfaces originally encoded as triangulated irregular networks. The implementation reported in Kidner [17] and Kidner and Jones [18] stored the vertices of the original TIN in a regular rectangular cell spatial indexing data structure, in which vertex coordinates were represented by offsets from the origin of their containing cell. Reconstruction of the triangulated network in a query window involves retrieval of the relevant vertices and execution of a Delaunay triangulation algorithm. In a comparison of different methods for storing digital elevation data [17] it was found that the Implicit TIN was the most space efficient.

2.1. Vertex selection and initial TIN construction

Given a set of points representing vertical elevations there are several methods for selecting a subset of points which can be used to describe the sampled surface as a TIN with a specified vertical error [19]. The need for selecting vertices from an original set arises when that set is in the form of a grid which may have considerable redundancy, and when certain applications only require a given degree of accuracy which is less than that of the complete set of points. The approach used by De Floriani [6] and Kidner [17] is to triangulate initially a small subset of the original points. This could be known important points or it could be an artificial set of points defining a surrounding rectangle. Points are added to the initial triangulation by selecting the vertically most distant point from the approximating surface, retriangulating, and repeating the process until no untriangulated point is further than a specified tolerance from the triangulated surface. Lee [19] favours a more computationally expensive approach whereby, initially, all points are triangulated and points are removed selectively until no triangulated point can be removed without degrading the accuracy of the surface below a specified tolerance. Selective removal of points involves finding for a given triangulation that point which, after removal, is vertically nearest to the retriangulated surface. Thus it is an iterative process in which a single point is only selected for removal after all other points have been considered in the same way, involving repeated retriangulation.

2.2. Combining spatial access with vertex priority access

Methods such as the above for point insertion or removal enable all vertices of the original set to be ordered in terms of their significance in representing the surface. Although it would be possible to label all points with their priority, if storage requirements are an important criterion it may be preferable to place vertices into classes defined by an associated limiting error. Using this layered hierarchical approach, the vertices required to reconstruct a surface are those belonging to classes with an error less than or equal to that of the retrieval criterion. Vertices of each layer of the hierarchy may then be segmented spatially to facilitate the search process required to rebuild the triangulation. If the vertices belong to a spatially extensive database, spatial segmentation is also required to enable efficient retrieval of an areal subset of the data.

The ideal spatial access scheme, given priority-ordered vertices, would be one which combined spatial indexing with scale-related, or priority, indexing. Efficient spatial indexing depends upon being able to group together in storage those points which would also be grouped together in space. Having clustered data in spatial terms they cannot simultaneously be clustered with equal efficiency with respect to scale priority, since a cluster based on scale priority could not be expected to be clustered in space. In practice therefore it is necessary to compromise. The solution adopted here is to give preference to the spatial indexing, using a quadtree directory, but to introduce a hierarchy of spatially indexed levels where each level corresponds to a prespe-

cified vertical error associated with the surface. The highest level cells contain all vertices required to reconstruct the triangulation to the lowest level of resolution, i.e. largest error. The next level down the hierarchy contains the additional vertices which, when combined with those in the higher level, would reconstruct the surface to the second resolution level. Thus each lower level provides the additional vertices required to reduce the surface error to that corresponding to its level. This approach is comparable to that used by De Floriani in the Delaunay Pyramid, except that here we do not store a triangulation and we do, unlike the latter scheme, use a spatial index for each level. A closer comparison is with the methods used for the purposes of multiresolution storage of lines by Jones and Abraham [16] and by Becker *et al.* [3].

2.3. Structural and non-structural lines

Before considering algorithms for reconstructing a triangulation from an Implicit TIN it is necessary to consider the linear features that may act as constraints on the triangulation. Linear features that are combined with terrain models fall into two categories. There are those which are structural in the sense that their use as constraints in a terrain model will improve the accuracy of the model in describing the form of the terrain. These lines describe phenomena such as ridges, valleys and breaks of slope. We include in this category any lines that describe physical objects. Roads, rivers and the outlines of buildings are notable examples. Other, non-structural lines are those that may be used to constrain the triangulation to facilitate visual display of the surface. Such non-structural lines could, for example, represent administrative boundaries. In some circumstances these boundaries might coincide with structural lines.

The significance of this distinction is that when structural lines are inserted in a topographic surface database, their vertices can be added to the terrain model data and the lines designated as necessary constraints. Non-structural lines can be distinguished as such and they only act as constraints when a particular query requires their presence in the retrieved model.

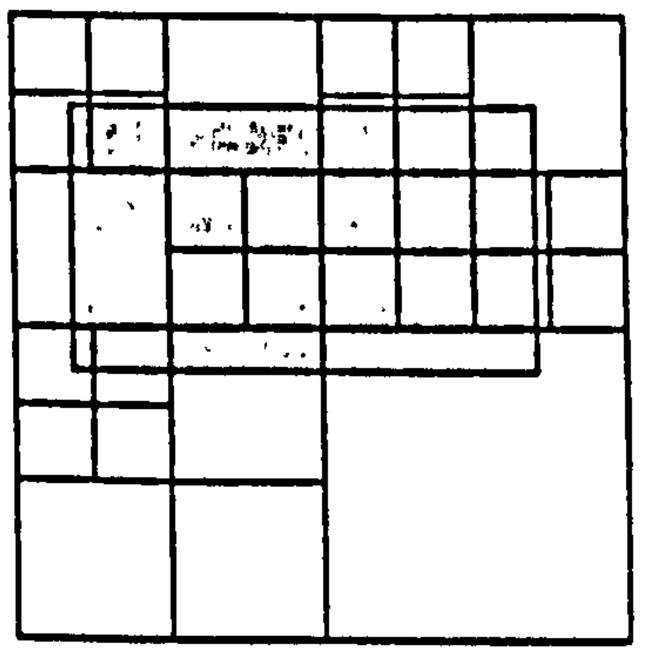
If linear constraints are to be imposed on a multiscale surface representation it is desirable to be able to control the degree of detail of the line descriptions. Ideally this should be comparable with that of the digital elevation model, but this is not always possible if there is a mismatch in the levels of detail of the original datasets. The MSLT provides a means of representing the vertices of linear features in a similar manner to that used in the multiscale Implicit TIN (see previous section). The MSLT is described in more detail in Section 3.

2.4. TIN reconstruction with linear constraints

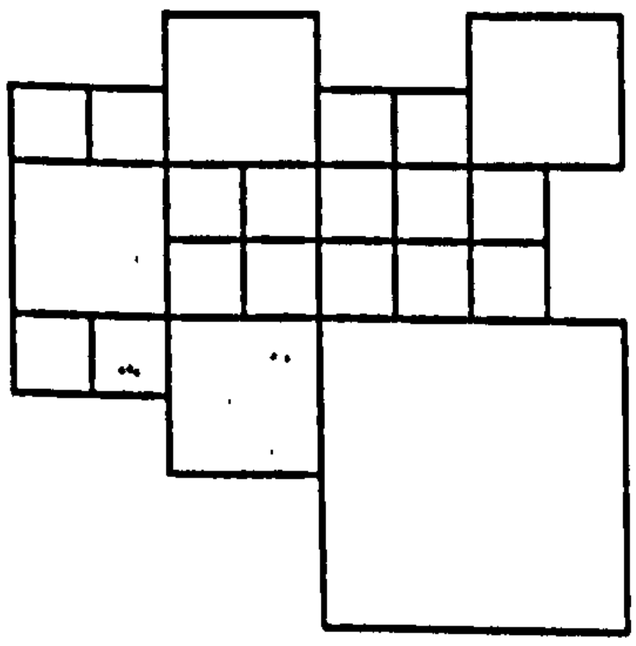
The effectiveness of the Implicit TIN depends upon the ability to reconstruct the original triangulation by means of an algorithm which operates on the relevant vertices and associated constraints. It is important to ensure that when only a part of the surface is being reconstructed (which will be the normal case for retrievals on an Implicit TIN), all of the relevant vertices and constraints are found. Note that if only a subsection of the original surface is required in a given spatial window, some of the relevant vertices of triangles crossing the border of the window will lie outside the window.

2.5. Extensive region triangulation

We now present an algorithm which will reconstruct a constrained TIN for a given query window. The algorithm starts by using the query window to generate a list of quadtree addresses (Figure 3). These are used to access the relevant elevation points and constraining objects. It should be noted that all geometric data defining objects referenced by quadtree cells are retrieved, not just geometric data that intersect the query window. The object data, that may consist of polygons, linear features or points, are reduced where appropriate to a list of edges, the constituent vertices of which are



(a)

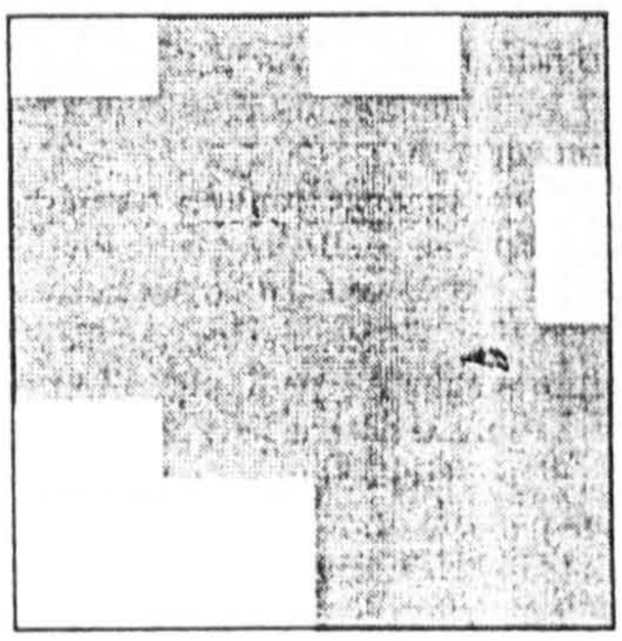


(b)

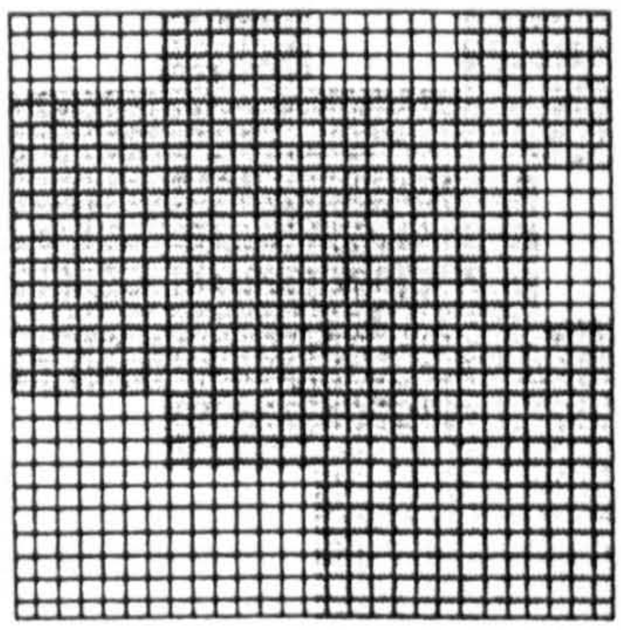
FIGURE 3. (a) The query region with respect to the database (b) The quadtree cells (and data) accessed in the database.

stored, along with the height vertices, in a main-memory based 'box-sort' data structure (Figure 4). This regular grid structure provides spatial indexing in the course of triangulation [20].

The correct constrained triangulation is obtained in two stages. First a Delaunay triangulation of all relevant vertices (from elevation data and constraining objects) is performed (see Procedure DELAUNAY_TRIANGULATE). Points that are interior to the query region will always belong to the final triangulation, so initially these points are put onto a stack of points to be triangulated. The Thiessen neighbours of each point CURRENT_POINT on the stack are then found in the following way. The nearest neighbour, NNB, of CURRENT_POINT is deemed to be the first Thiessen neighbour. The Thiessen neighbour K to the right of the edge (CURRENT_POINT, NNB) is then found and added to the list of Thiessen neighbours. The Thiessen neighbour to the right of the edge (CURRENT_POINT, K) is then found. The process is repeated until the latest neighbour is equal to the original neighbour. The search for Thiessen neighbours utilises the box-sort data structure, such that only local points within the neighbourhood of a Delaunay edge are tested. However, if the search for triangle vertices includes box-sort cells which are empty (lie outside the generated quadtree region) or extends beyond the box-sort coverage, the necessary quadtree cells in the database intersected by the local search region are accessed and the vertices are retrieved (Figure 5).



(a)

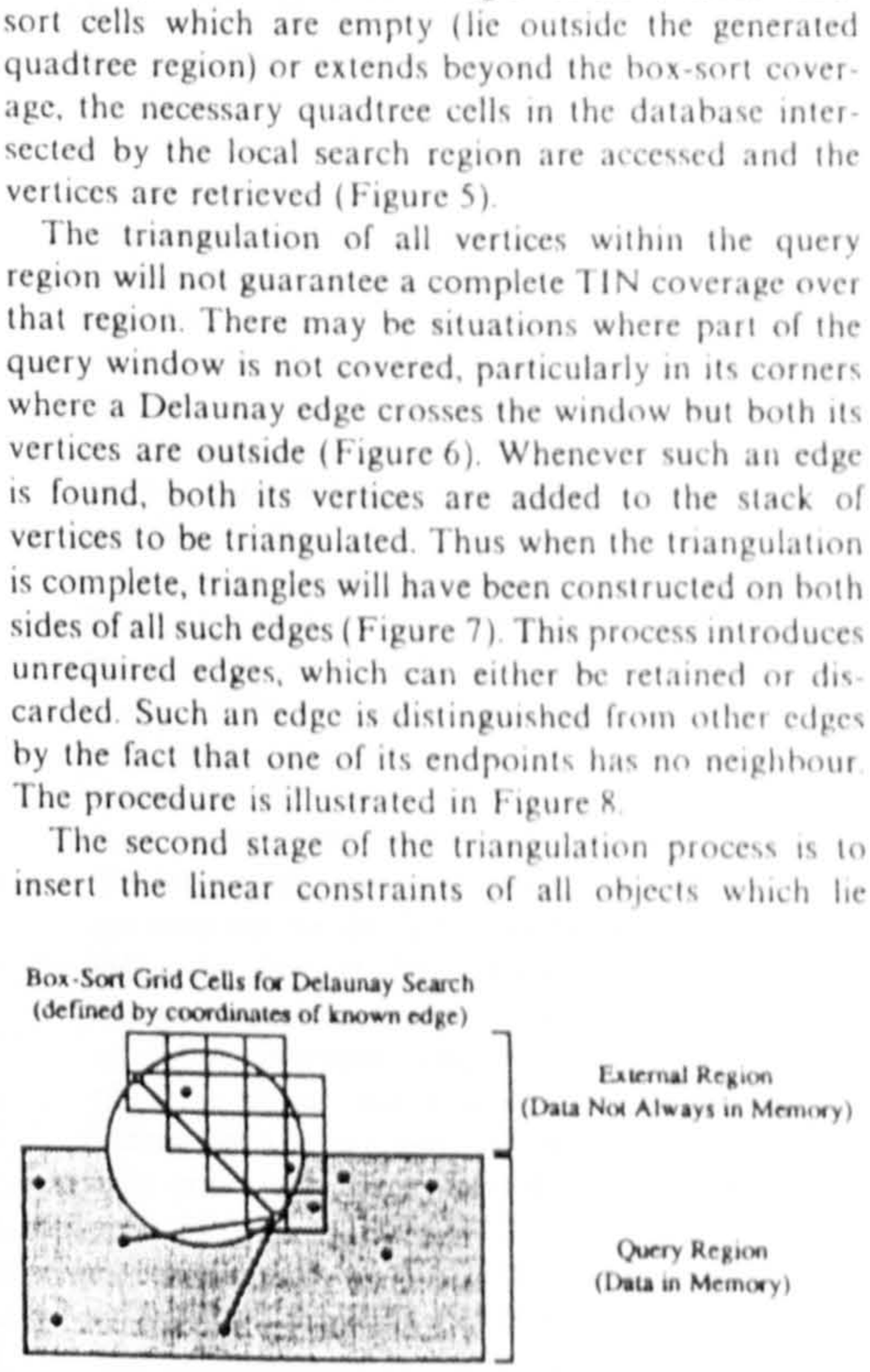


(b)

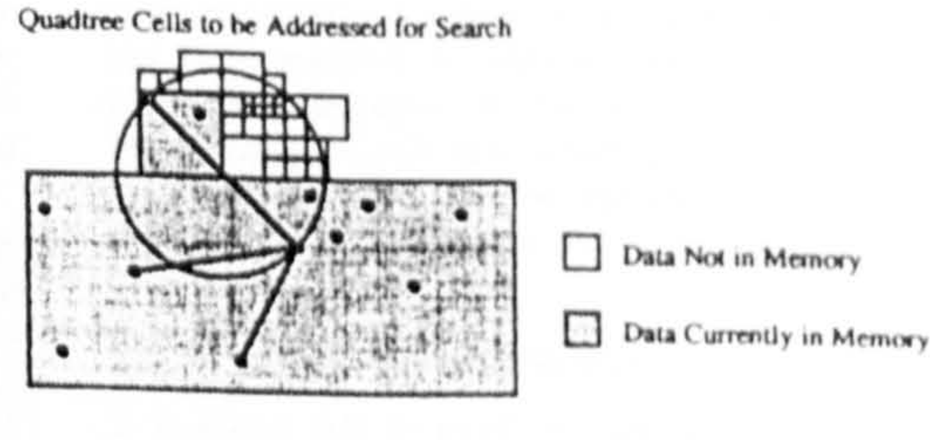
FIGURE 4. (a) A bounding rectangle placed around quadtree cells. (b) The initial box-sort data structure with referenced and empty cells.

The triangulation of all vertices within the query region will not guarantee a complete TIN coverage over that region. There may be situations where part of the query window is not covered, particularly in its corners where a Delaunay edge crosses the window but both its vertices are outside (Figure 6). Whenever such an edge is found, both its vertices are added to the stack of vertices to be triangulated. Thus when the triangulation is complete, triangles will have been constructed on both sides of all such edges (Figure 7). This process introduces unrequired edges, which can either be retained or discarded. Such an edge is distinguished from other edges by the fact that one of its endpoints has no neighbour. The procedure is illustrated in Figure 8.

The second stage of the triangulation process is to insert the linear constraints of all objects which lie



(a)



(b)

FIGURE 5. (a) The search for vertices extends beyond the query region. (b) The search region is mapped into quadtree addresses for retrieval

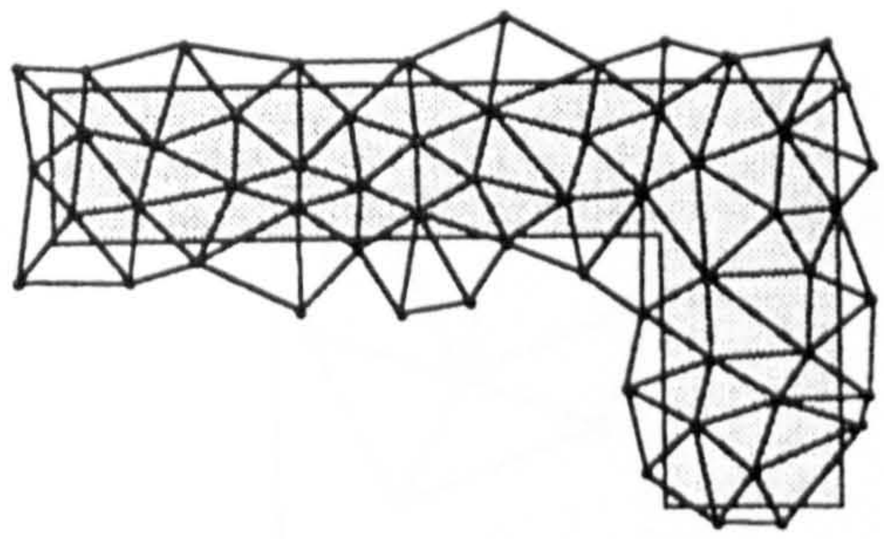


FIGURE 6. The triangulation of all vertices within the query region. (Note that coverage is not complete in the bottom two corners.)

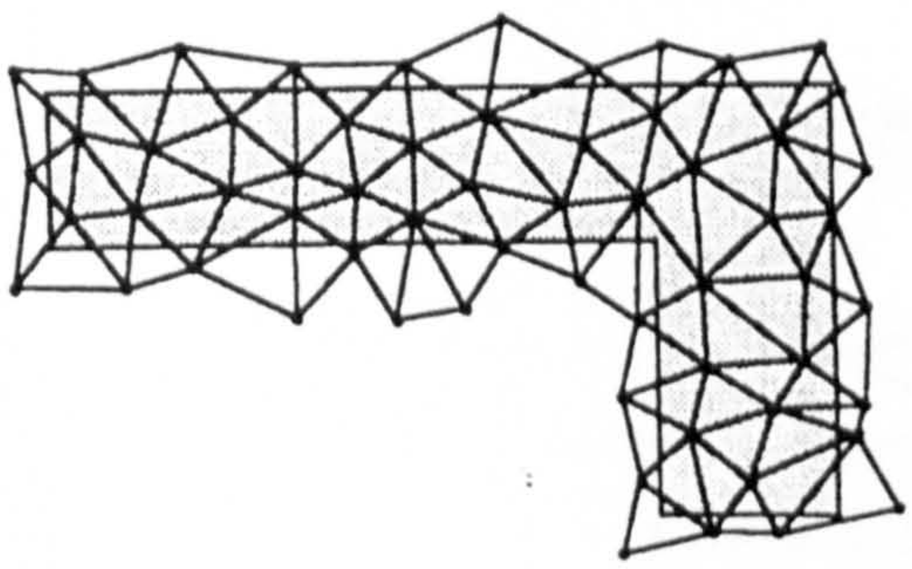


FIGURE 7. The final triangulation of query region.

within or intersect the current TIN (see Procedure CONSTRAIN_TRIANGULATION). Each constraining segment (A, B) can have one of five possibilities: (i) A and B are both vertices within the TIN and form a Delaunay edge, (ii) A and B are both vertices within the TIN with no connecting edge, (iii) either A or B is a TIN vertex whilst the other is external to the TIN, (iv) both A and B are external to the TIN, or (v) any of the cases (ii)–(iv) but where the constraining edge passes through a hole or concavity in the triangulation. The first two occurences are the most likely, but the probabilities of each will depend upon the sampling densities of the elevation and object data. In the first instance (i), the segment exists within the TIN and therefore no update is necessary. In the other cases, the segment does not exist and therefore the TIN must be constrained (Figure 9).

For case (ii), the procedure for inserting an edge constraint (A, B) into the TIN consists of determining the current edges which are intersected by the constraint (Figure 9a), eliminating these edges (Figure 9b), re-triangulating around the new edge (Figure 9c) and updating the TIN data structure. It may be noted that, initially for case (ii), there will always be one or more current edges that are intersected by the constraint (A, B) since this region will have been triangulated initially as it is within the original query window (otherwise A or B would be external). The problem of re-triangulating around the constraining edge is reduced to that of separately triangulating the two polygons formed either side of the edge. These polygons are sometimes referred to as the polygons of influence [7]. The triangulation of each polygon proceeds as follows. Consider the edge (A, B) to be the base edge of the polygon. The initial step is to find the vertex Q of the polygon, discounting the vertices A and B, which subtends the largest angle to the base edge. For the upper polygon in Figure 10(a) this is vertex 5. This vertex is added to the list of neighbours of both A and B, and similarly A and B become neighbours of vertex 5. Two sub-polygons have now been formed with base edge (A, 5) and (5, B) respectively (Figure 10b). The two sub-polygons, and any subsequent sub-polygons, are dealt with, recursively, in the same way as the original polygon (Figures 10c and d). The recursion continues until the latest new edge matches an edge in the original TIN.

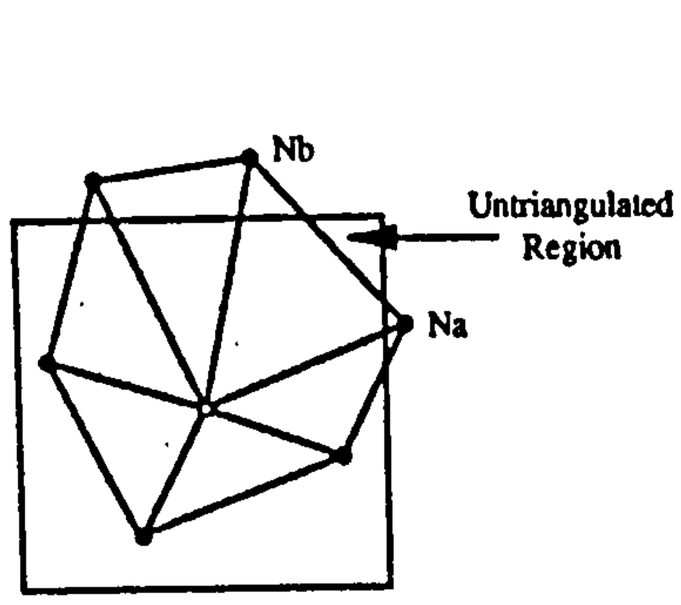
For a constraining segment with one vertex in the TIN and a second outside, the procedure is very similar, but triangle edges may extend to vertices outside the original TIN. For example, consider the insertion of the highlighted segment (A, B) in Figure 11(a). Here the search for the vertex with the largest subtended angle must include the vertices making up the polygon of influence (discounting A and B) plus all vertices which are external to the TIN but lie within the polygon of influence (Figure 11b). Searches involving any subsequent sub-polygon must include the vertices making up the sub-polygon (discounting the base edge vertices) plus any vertex which is external to the TIN but lies within the sub-polygon. The recursive procedure in this case continues until either the latest edge matches an edge in the original TIN or the latest edge fails to intersect the original TIN (Figure 11c).

The fourth possible situation is where both vertices of the constraining edge lie outside the original TIN. The procedure for constrained edge insertion follows that of the insertion of a constraining edge with one external vertex (Figure 12).

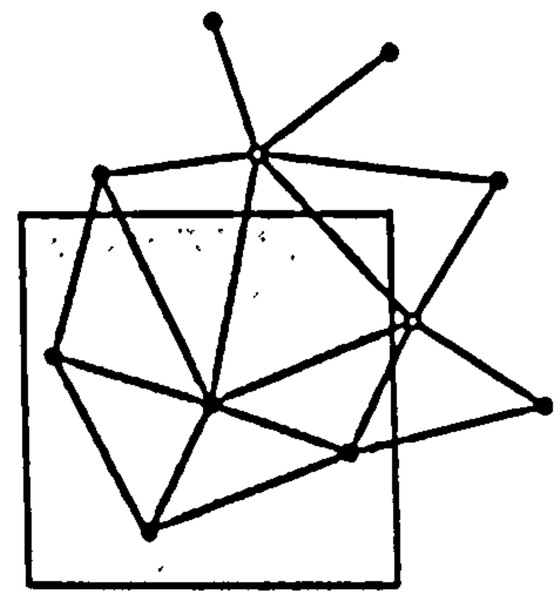
The Implicit TIN algorithm will sometimes produce triangulations containing holes due to triangles crossing concave regions of a query window. Thus the algorithm has been designed to handle query windows that are themselves concave in shape or include a hole. Introducing a constraint which passes through such a hole or concavity can be catered for by using the methods for cases (iii) and (iv), as shown in Figure 13.

2.6. Restricted region triangulation

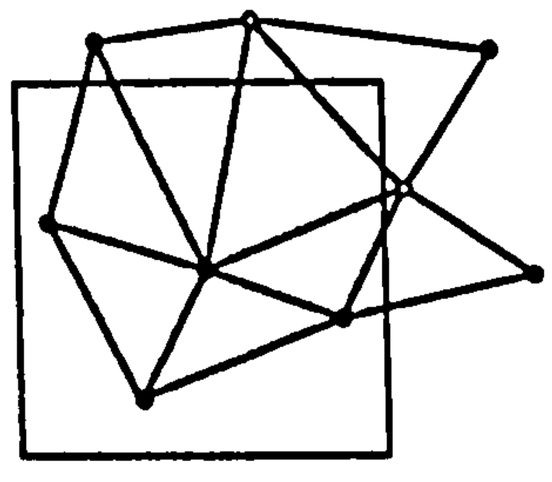
It is noted that to construct the Implicit TIN for any query, the algorithm requires an initial vertex to start the triangulation process. In most cases an arbitrary vertex from within the query region is chosen. However, in certain circumstances, no vertices lie within the initial



(a) External Edge Intersection.



(b) Triangulation of External Vertices.



(c) Removal of unrequired edges.

FIGURE 8. Test for complete coverage and resolution of completeness by triangulation of external vertices

query window. This situation may arise if the query window is narrow or has no width as in the case of a profile. In such a case the initial vertex can be found in a number of ways. One method is to find the nearest neighbour of the centre of gravity of the vertices defining the query region. Another method is to search for straight line segments (constraints) belonging to linear features that cross the query window and to select one of their bounding vertices. This would have to act as a preliminary method in that it will of course only work if there is an intersecting constraint. Having found an initial vertex, the algorithm proceeds by finding its Thiessen neighbours and testing whether the connecting edge to each neighbour intersects the query window. If it does, then the neighbour is placed on a 'to triangulate' stack. If no connecting edge crosses the window, another vertex in the vicinity must be selected and the procedure is repeated. Once a vertex of an intersecting triangle edge has been found, its neighbours across the window can be processed in the same way. All such opposite neighbour vertices are then processed. The remainder of the algorithm finds any unprocessed border triangles in the same way as in Procedure DELAUNAY_TRIANGULATE. The case where no intersecting edge exists, when the query region is completely contained within a Delaunay triangle, is also catered for. This is achieved by simply finding the Thiessen neighbours of the vertex closest to the query region. One of the triangles thus formed will contain the query region.

3. MULTISCALE STORAGE OF LINEAR FEATURES

There are several published descriptions of multiresolution schemes for representing lines. Examples of these are the strip tree [2], the MSLT [15, 16], the Reactive Tree [25] and the Priority Rectangle (PR) File [3]. The original strip tree includes no facility for efficient spatial access and is therefore not satisfactory for use in a large database.

The MSLT does provide spatial indexing and it is intended for large databases. Like all of these schemes it uses a line generalization algorithm (that of Douglas and Peucker [8]) to simplify linear features. The algorithm is used to classify vertices of a line according to their scale significance or contribution to the shape of the line. A hierarchy is then constructed in which, at the top level, all vertices required to represent the line in its most simplified form are stored. At the next level are

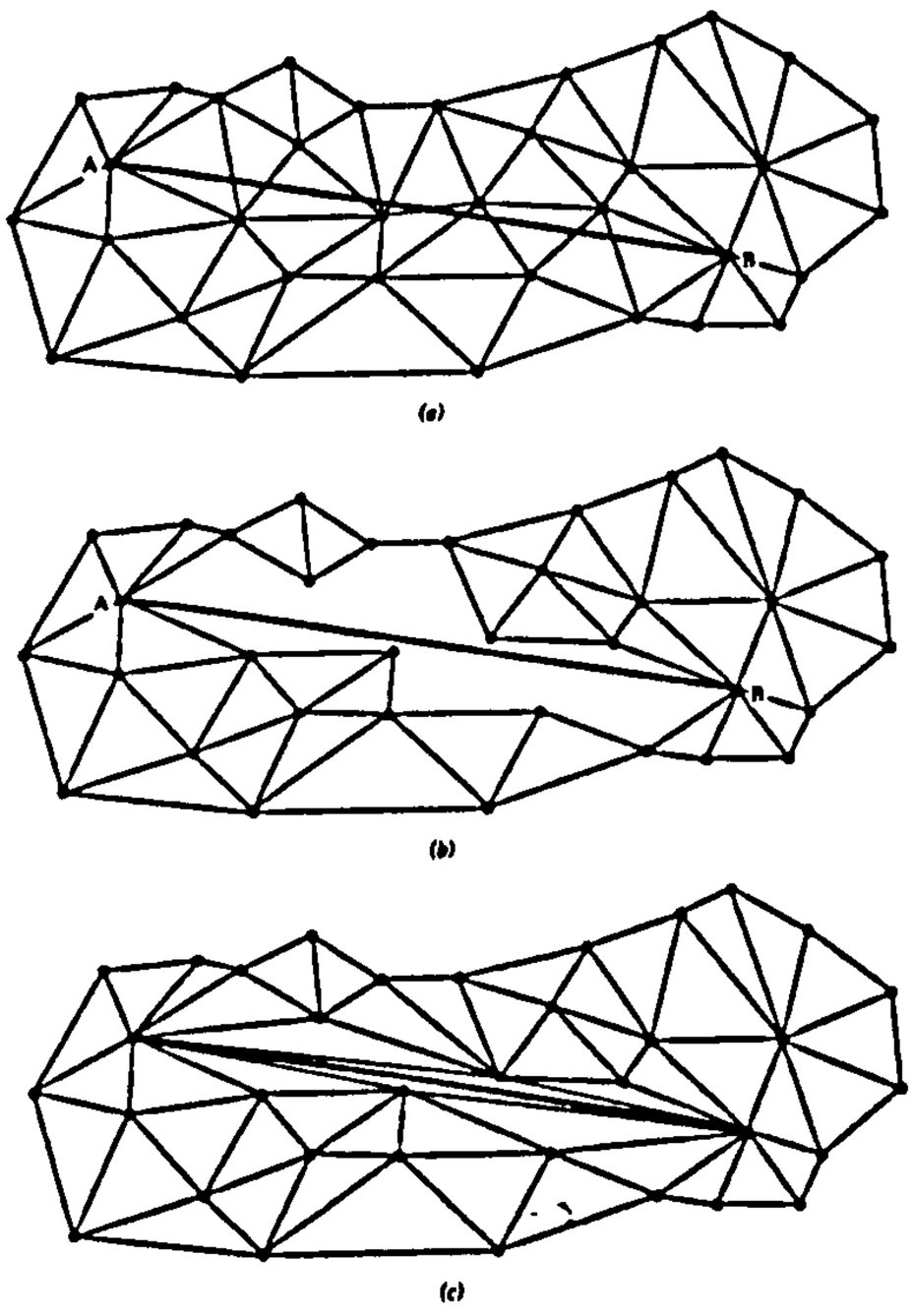


FIGURE 9. Constrained edge insertion within a TIN. Insertion of a constraining edge AB involves finding existing edges that intersect it (a), deletion of these edges (b) and re-triangulation to include AB as an edge in the triangulation (c).

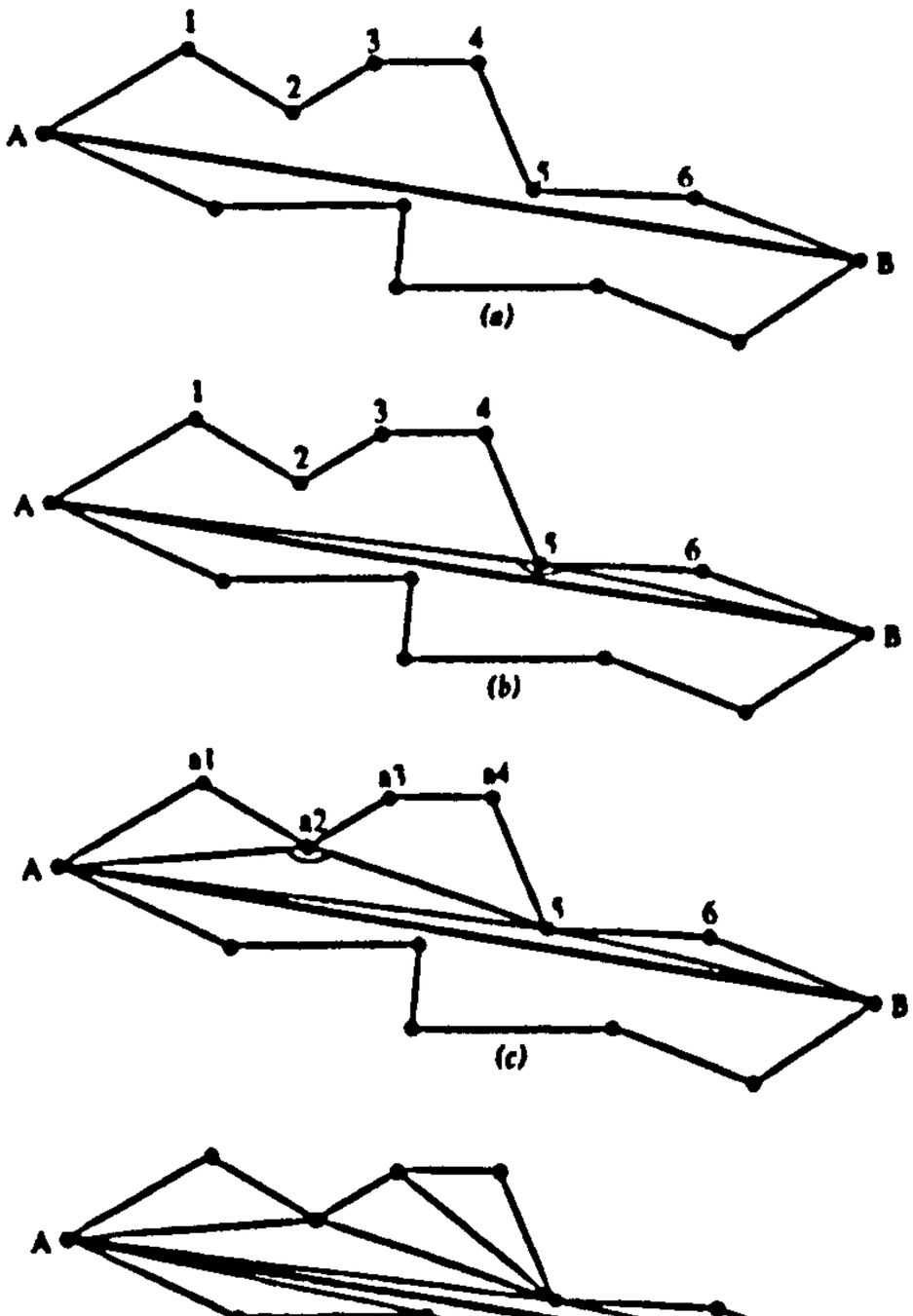


FIGURE 10. Triangulation within a polygon around a constrained edge. The inserted edge forms the base of two polygons to be triangulated (a). Triangulation of each polygon proceeds by selecting the vertex which subtends the largest angle with the base edge (vertex 5 in b). Each new edge is treated recursively as a base edge of a new polygon (c). Triangulation is completed for each polygon when the edges from the selected vertex to the base edge belong to the original triangulation (d).

stored intermediate vertices which when added to those at the higher level would represent the line to a prespecified lateral error tolerance. Subsequent lower levels provide further degrees of detail. Abraham [1] implemented several spatial indexing methods whereby each level of the hierarchy could be accessed on the basis of a specified spatial window. For a given level of detail all spatially relevant parts of the levels down to and including that level need to be accessed. The retrieved vertices are then reassembled to constitute the linear feature at the required resolution. The MSLT spatial indexing method subdivided vertices into rectangular cells, based on a quadtree. The scheme carried a storage overhead due to the fact that each cell included boundary vertices, which were spatially located outside the cell's (spatial) extent.

The Reactive Tree of van Oosterom [25] uses an R-Tree spatial index [12] to refer to the occurrence of linear features which are stored separately in a hierarchical (but not spatially segmented) data structure, the BLG-tree, which provides access to the scale-classified vertices of the linear features. The BLG-tree is then traversed to the level of detail required. Efficiency of this approach is dependent upon required linear features not being greatly spatially extensive beyond the region of the spatial query window, since the BLG-tree requires accessing the entire line which may subsequently be clipped to the area of interest.

The PR File [3] is more closely related to the MSLT in that the vertices of linear features are separated into spatial units which are present at different levels of detail (or 'priority'). It differs however in that the spatial units are minimum bounding rectangles of arbitrary subdivisions of the stored line and are indexed using an R-Tree related scheme. This reduces problems of boundary vertices, though the scheme appears complex to implement.

In our application of the Implicit TIN we have adopted an approach which takes aspects of the MSLT and of the Reactive Tree in order to reference line features. Like the MSLT (and Becker *et al.*'s PR File)

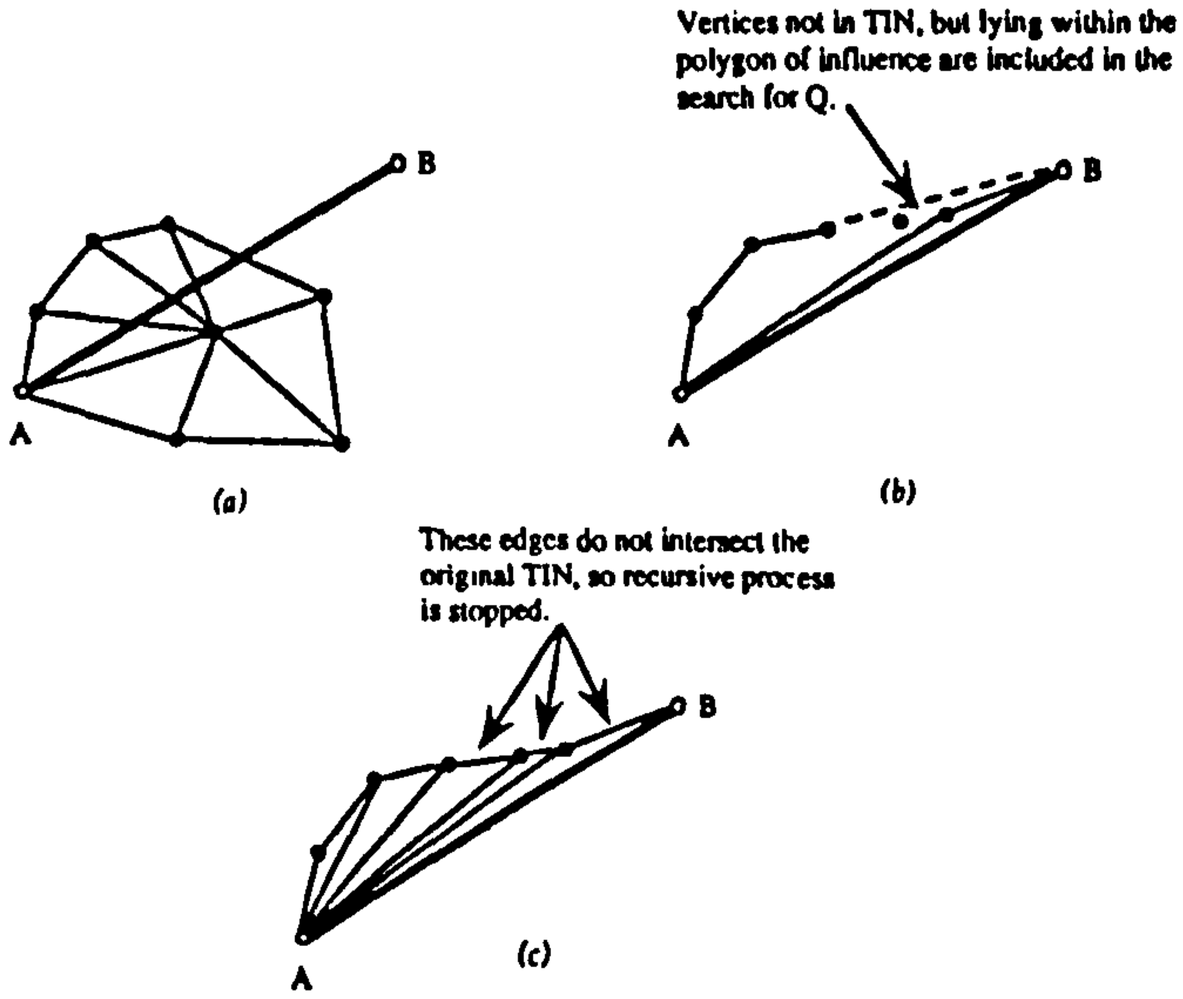


FIGURE 11. Insertion of an edge with one external vertex (see text for explanation).

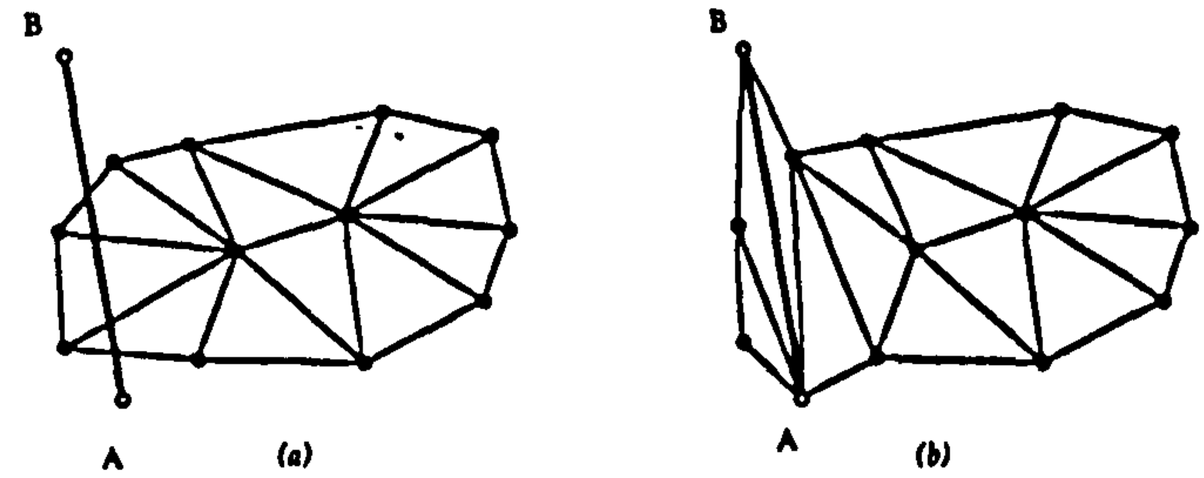


FIGURE 12. Insertion of an edge with two external vertices A and B (a) follows the procedure for insertion of an edge with one internal vertex, to result in the constrained triangulation (b).

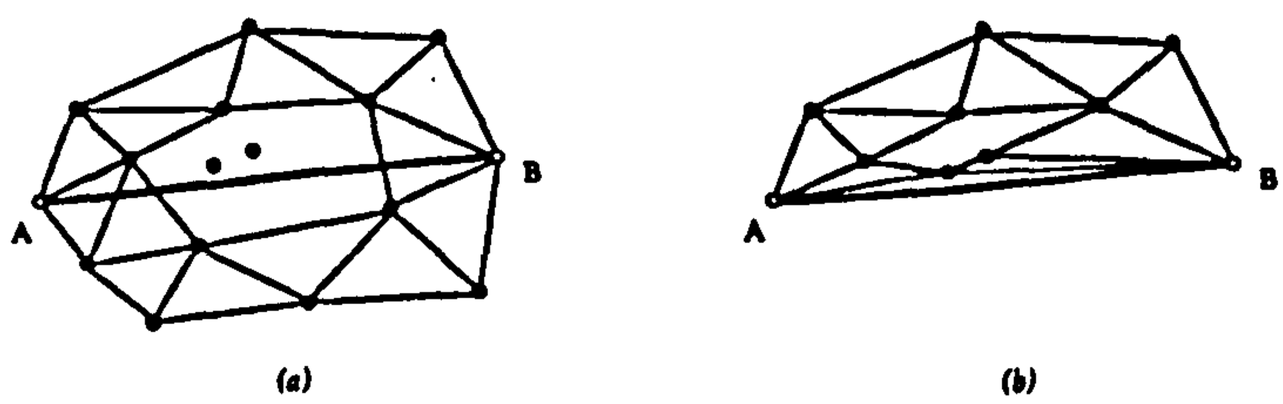


FIGURE 13. Insertion of an edge through a hole in the triangulation of a concave query window. Triangulation of the constraining edge AB (a) continues within the polygons of influence until new base edges either belong to the original triangulation or do not intersect the query window (b shows the triangulation of one side of AB).

vertices are classified into hierarchical levels. Since these vertices may be a necessary part of the terrain model, each level stores lists of the sequentially numbered line vertex identifiers. The corresponding coordinates are stored separately. The lists are not themselves spatially segmented, though their presence is referenced by the quadtree spatial index.

4. POLYGONS AND MULTISCALE LINEAR FEATURES

Storage of polygonal objects is achieved by subdividing each polygon into linear components representing their boundary. In doing so, it is possible to avoid unnecessary data duplication, since for a map entirely covered by polygonal regions, all boundaries interior to the map

will belong to the two adjacent polygons which share the linear boundary. Provided that polygon representations are reduced to lists of linear boundaries, the linear feature multiresolution storage scheme described in the previous section can be used. Multiscale representation of polygons can then be achieved by a combination of a polygonal object description which refers, for a particular scale of representation, to the relevant linear components, and the multiresolution representation of the linear components themselves. Retrieval of a particular representation is accompanied by a check for topological integrity of the polygon, which may have been violated by the line generalisation procedure. This is then corrected by inserting additional higher resolution vertices in the linear boundary representations (Figure 14).

5. A MULTISCALE DATABASE

In this section we describe the components of a multiscale database which applies the Implicit TIN concept to the storage of points, lines, polygons, complex polygons and composite objects constructed from these spatial objects. All of these spatial objects are regarded as part of a terrain model, which we refer to as the topographic surface.

The objective in designing the database was to enable subsets of spatial objects that are part of a topographic surface to be retrieved at variable levels of detail determined by the scale of the required output. The assumption is that for large scale (detailed) retrieval the geometry will be required at higher resolution than for small scale retrieval. It is also assumed that the actual objects retrieved will be required at larger scales and, furthermore, different types of object will be required according to the purpose of the retrieval. In a geological context, finer subdivisions of geological formations might be represented at larger scales along with classes of geological unit that were relevant to particular types of mineral exploitation. In the context of local government planning the boundaries of individual land parcels or planning regulation zones might be required along with, for example, the proposed path of new roads.

All geometric objects in the database are defined in terms of component points. Individual points are identified uniquely and may be regarded as belonging to both the terrain surface and any point or linear or polygonal features on the surface. When a point is inserted into the database it is allocated a level which characterizes its priority or scale significance. This priority is determined by a combination of two factors. One is the importance in defining the geometry of the terrain surface and the other is in defining the geometry of any objects mapped onto the surface. Methods for determining the priorities were described Section 2.1 and Section 3.

Spatial objects representing phenomena mapped onto the terrain surface are viewed hierarchically. Thus point objects are defined geometrically simply by reference to a single vertex at a specified level; linear objects are defined by ordered lists of vertices which may occupy a specified range of levels; simple polygons are defined by lists of bounding linear objects; complex polygons are defined by lists of component simple polygons (a primary external bounding polygon and the internal bounding polygon, i.e. holes). Higher level objects defining various real world phenomena are then defined in terms of the constituent points, lines or polygons.

In the implemented database, all spatial objects and all component vertices are indexed by spatial location, using a quadtree directory, which is itself organised in levels corresponding to levels of storage of the geometric coordinate data. Each cell of the quadtree directory references the objects that intersect it. The number of objects per quadtree cell has been chosen somewhat arbitrarily as 5. The purpose of the experimental database is to provide a framework for demonstrating the multiscale Implicit TIN and no attempt has been made to optimise spatial indexing. It may be remarked however that the quadtree indexing scheme is similar in principle to the PMR quadtree [21], which has performance characteristics that are competitive with other major alternatives [13].

Queries to the database are answered by accessing the level or levels appropriate to the specified 'scale' or resolution. At any given level, all objects from the coursest scale down to that level are recorded in the spatial index. Thus having entered the database at a

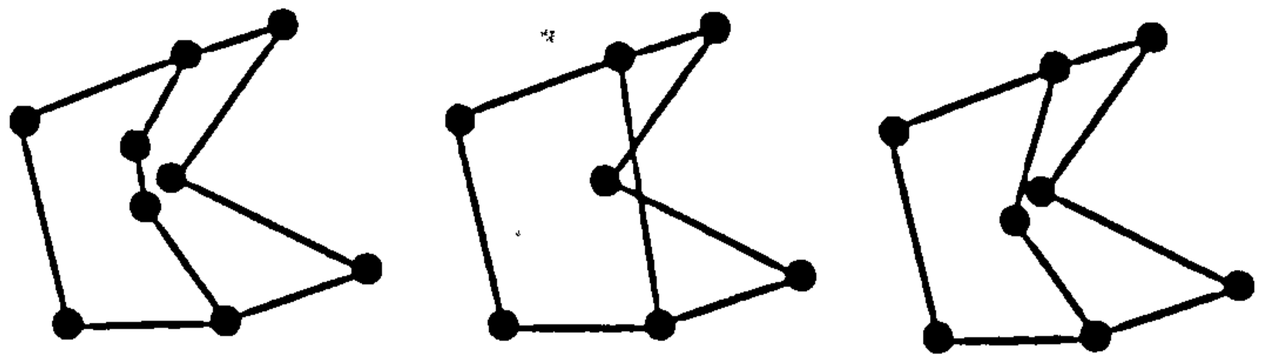


FIGURE 14. Example of error which may occur during line simplification and how the error is corrected

particular level, retrieval of the geometry for selected objects is achieved by accessing the range of levels from the highest recorded for the object down to the current level. Since adjacent quadtree cells may reference the same (non-point) objects if they lie in both cells, it is necessary to keep a temporary index of already accessed objects for any particular query, in order to prevent duplicate retrieval of the associated geometry. The TIN construction algorithm is applied to the combination of vertices and constraints retrieved for the specified spatial window.

The experimental database consists of a set of indexed tables with variable length records. An overview of the database tables is given in Figure 15, which we will now explain. The database represents the multiresolution topographic surface by a sequence of levels where the top level is the coarsest resolution and the bottom is the finest resolution. The *Levels Table* has an entry for each such level and records the level number, the maximum vertical terrain height error associated with the level and the maximum lateral (ground location) error associated with linear features which may be embedded within the level and hence constrain the triangulation.

There are two quadtree tables at each level of the database. The *Object Quadtree Table* records, for each quadtree cell, the list of the spatial objects (or features) that lie inside or intersect the cell. The *Point Quadtree Table* stores the point identifiers of the points that lie inside its respective quadtree cells. The reason for maintaining separate 'object' and 'point' quadtrees is due to a distinction between real world objects with name and class attributes and the lower level point geometry used to describe the objects. Many points will only be used for describing the ground surface and will not be part of the boundary of objects mapped onto that surface.

The *Point Tables*, of which there is one for each level of the database, store, for each vertex, its point identifier and x, y and z coordinates. Vertices that define linear features that are not regarded as essential to defining the form of the terrain model are assigned a null z value. When combined with terrain data their z coordinates are inferred from the terrain elevation data.

The *Object Tables*, of which there is one for each level of the database, have an entry for each object at the corresponding level referred to in the Object Quadtree Tables. Objects may be composed of polygons, linear features and point features. The data items for each object are its 'real world' classification and lists of references to its component polygon, line and point features. Note that Point objects refer directly to the Point Table where the coordinates are stored. Clearly up to two of these lists could be empty if it consisted of only one type of spatial geometry.

The *Polygon Feature Tables*, with one table per database level, store the polygon identifier and a list of the identifiers of the linear features which compose the polygon.

The *Linear Feature Tables*, again with one per level of the database, contain the linear feature identifier, the highest level of the database in which it is referenced and a list of the point identifiers and their sequence numbers within the line. Each such Table only stores the identifiers of the vertices which are introduced at that level. Thus to construct a linear feature at a given level, it is necessary to access all Linear Feature Tables

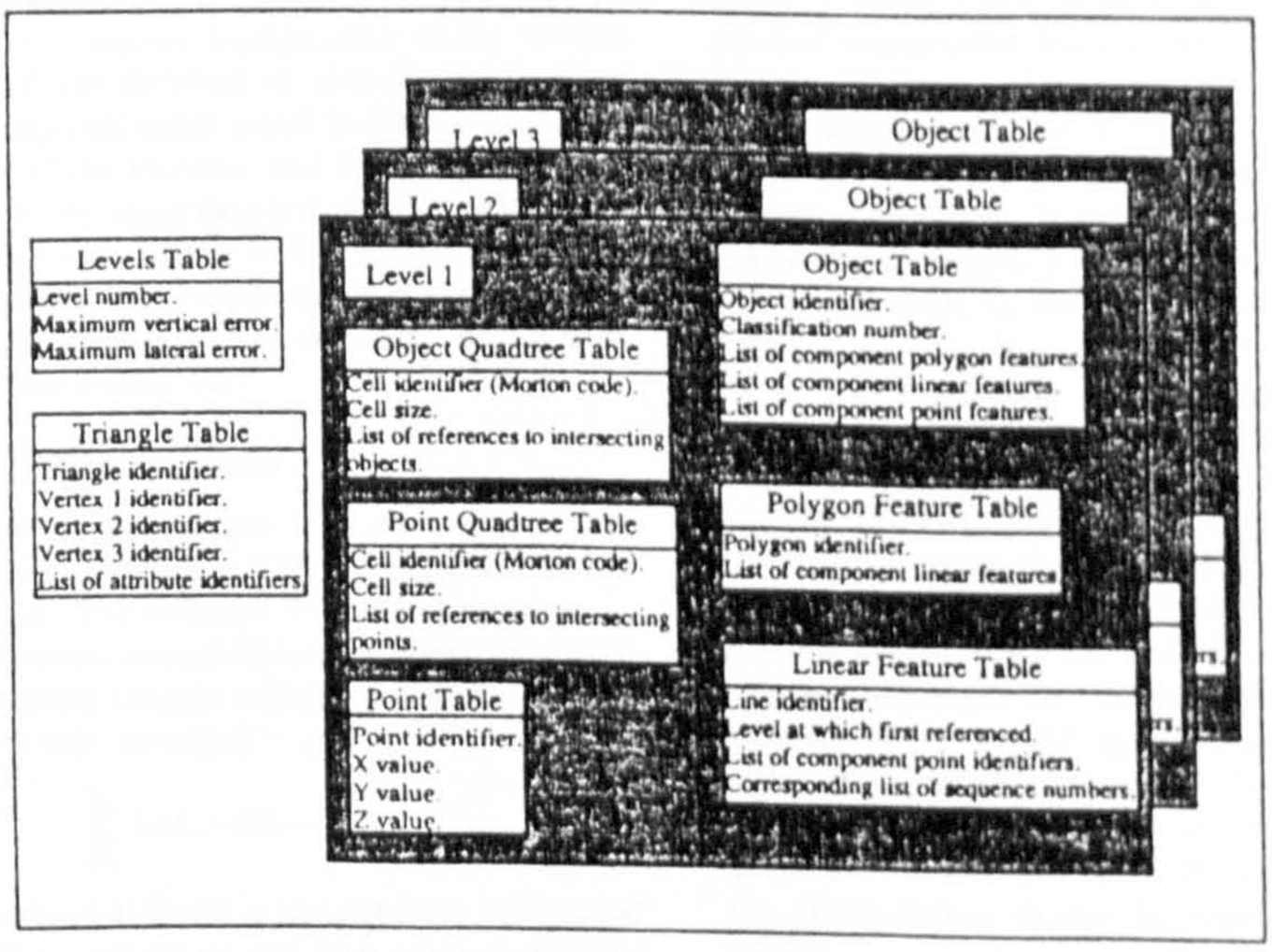


FIGURE 15. Overview of the Multiscale Database (in this instance with three levels of detail)

from the highest level of occurrence down to the current level.

The *Triangle Table* is used to record the form of explicit triangulations stored temporarily as a result of triangulating the Implicit TIN. It records for each triangle, the triangle identifier, the Point identifiers of its three vertices, and attribute identifiers that may be associated with the triangle. Such attribute identifiers are assumed to be obtained by the triangle taking on the classification-related properties of any polygons of which it is a member.

5.1. An application to geological data

Figure 16 shows the Implicit TIN output produced when the algorithm described in Section 2 is applied to a small implementation of the database design described in Section 5, using an L-shaped query region. The test database consists of 20 objects comprised of 13 geological outcrop regions and seven geological faults. The outcrop regions are defined by 20 polygons, while there are a total of 143 linear features used to define these polygons and the fault lines. The terrain surface is defined by 612 points, while the constraining features are defined by 967 points. The quadtree cells have a maximum of five objects and five points per cell, respectively, for the two types of quadtree. Two levels of detail are shown to illustrate the differences in the amount of data relevant to each level. The first level (Figure 16a and b) was created with vertical and horizontal tolerances of 10 m, while the second, more detailed level was created with vertical and horizontal error tolerances of 5 m. There are 45 retrieved points (14 for the terrain elevation and 31 for the linear constraints) for the query window at the first level and 77 points (21 terrain and 56 linear constraints) at the second level. For each of the two levels, a complete triangulation of the corresponding part of the database is also shown (created using the conventional constrained Delaunay triangulation algorithm of De Floriani and Puppo [7]). There are 891 points in the more detailed representation (part of which is shown in Figure 16d) and 587 in the less detailed one (Figure 16b). Inspection of Figure 16 shows that the Implicit TIN produces the same triangles as those in the conventional TIN.

5.2. Database performance issues

One of the major advantages that an Implicit TIN system holds over an explicit TIN system is the saving in storage space. The Implicit TIN database scheme, described here, when compared to an equivalent explicit TIN database using triangle adjacency pointers, has an approximate storage saving of

$$\sum_{i=0}^{m} 12N_i - 6B_i - 12$$

where there are $(m+1)$ levels in the database and a total of N_i points (from elevation and linear features) in the reconstructed TIN at level i, B_i of which are boundary points. The assumption is that the explicit TIN database stores multiple versions of the triangulations, i.e. one for each of the $m+1$ levels. Storage costs for the explicit scheme include an additional element proportional to $3N$ to represent coordinates of the points or $4N$ if we assume that a unique point identifier is also stored for each point. Further storage is required for the definition of objects in terms of their geometry. If the object definitions are stored as lists of point identifiers and their sequence numbers, an aproximate upper limit on the storage required would be $1N$. Thus $5N$ is an estimate of the storage required in addition to triangulation topology pointers. For the Implicit TIN there is no triangulation topology, other than constraints, thus $5N$ is a measure of the storage costs for this scheme. Regarding the size of B_i, it is determined here by the number of points on the convex hull and remains a constant for all levels of representation. It is usually small compared with N. Thus for a single level of storage the triangulation topology approximates to $12N$ and the relative size of the Implicit and Explicit TIN scheme is in the ratio 5/17. As the number of levels increases, the overheads for the explicit scheme increase significantly. Taking the example of five levels of storage each of which involved a reduction in the number of points by two-thirds, the overhead would amount to about 50% of that of the of most detailed level, i.e. in proportion to $6N$. In this case the ratio of storage between Implicit and explicit schemes would be 5/23.

It is important to note that storage saving is not the only justification for using the Implicit TIN. The approach provides flexibility in integrating selected topographic features with a terrain model at user-specified levels of detail. Thus the constraints introduced by the selected topographic features are not predetermined, as they would be in a stored constrained explicit TIN.

The usefulness of the Implicit TIN will depend, for many applications, on the ability to reconstruct the correct constrained Delaunay triangulation for a given query region within a satisfactory time, the length of which will relate to the specific needs of the particular application. The major time penalty introduced by the Implicit TIN system is that of having to reconstruct the constrained Delaunay triangulation from the main memory data. The reconstruction algorithm currently used in the system has a worst case time complexity of $O(N \log N)$, where N is the number of points (elevation and linear feature data) to be triangulated. This represents an upper bound on time for any serial Delaunay triangulation algorithm (constrained or unconstrained), although some parallel algorithms improve on this, with $O(\log N)$ reported by ElGindy [9]. Early experimental results indicate that a satisfactory reconstruction time is achieved. For example, the CPU time taken to produce the triangulation shown in Figure 16(c) is less than 250 ms.

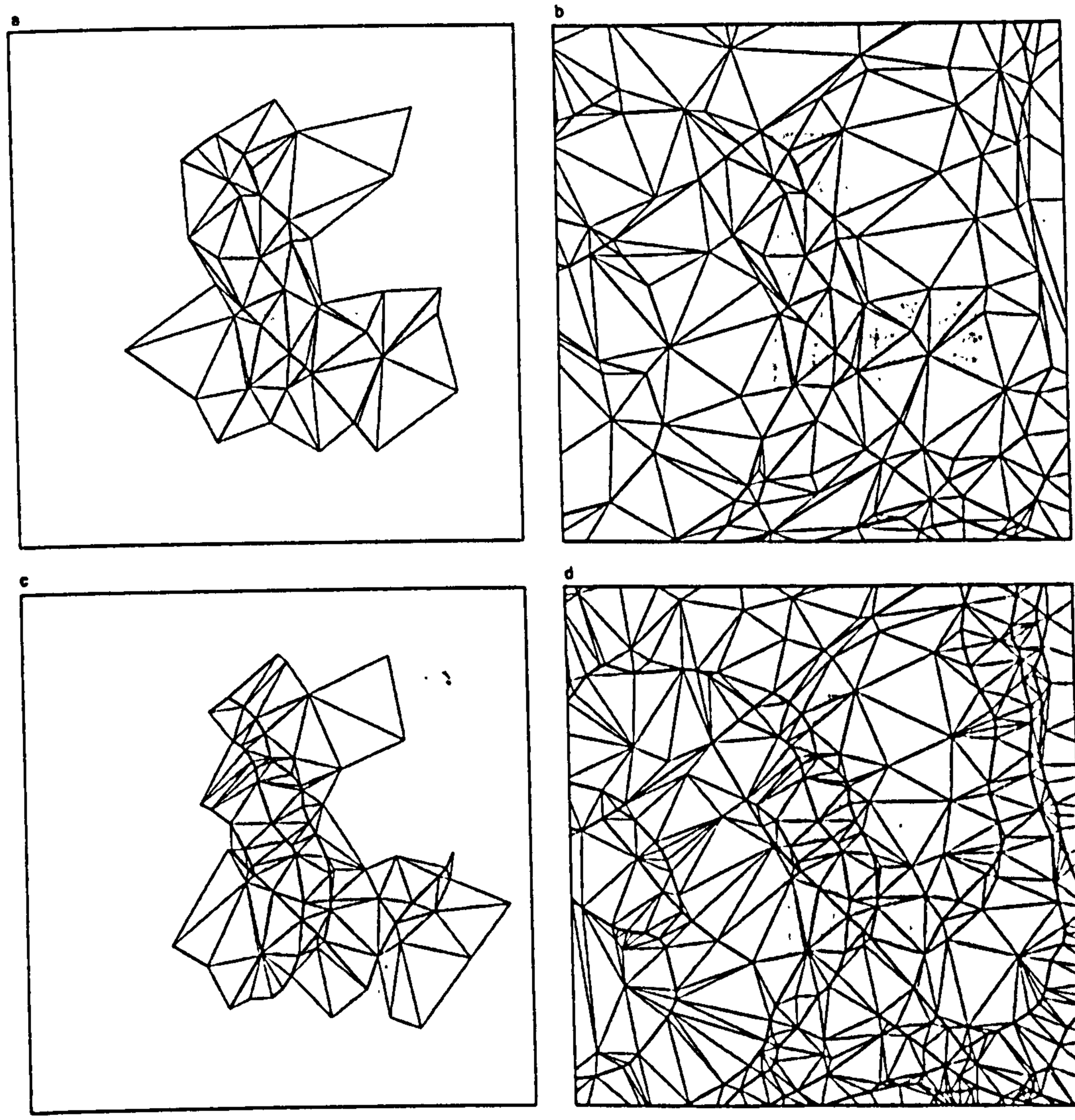


FIGURE 16. (a) and (c) show the triangulations produced by the Implicit TIN algorithm when applied to two levels in a multiscale geological database; (b) and (d) are the triangulations for the same area produced using a conventional triangulation algorithm.

6. CONCLUSIONS AND DISCUSSION

The Implicit TIN is a space-efficient triangulated data storage scheme that has considerable potential for representing topographic surfaces in a multiscale database. For the purposes of representing only elevation data it can provide a highly compact storage scheme. By classifying vertices according to their scale-related priority, it can be combined with a multiscale representation of geographical objects defined by polygons, lines and points, which are embedded in, and act as constraints on, the triangulation of the elevation model. The vertices defining these objects are themselves classified according to their scale-related priority, allowing them to be combined selectively with the elevation data when a particular scale of representation is required. A scale of representation can be defined at least partially by the vertical error tolerance associated with the elevation model vertices and the lateral error tolerance of any additional features.

Retrieval of an explicit triangulation requires applying a constrained triangulation algorithm to the terrain and associated objects relevant to the spatial query window.

Execution of the algorithm inevitably introduces a time overhead in retrieval from the database. Whether this is acceptable will depend on the application. Having built a model from the database components it is envisaged that it will be retained at least temporarily for purposes of analysis and visualization. How long it is worth retaining a triangulated model will depend upon the time taken to create it. Clearly actual retrieval times will depend on the quantity of data and the speed of execution. The introduction of parallel processing methods to triangulation can be expected to improve performance in the future [9, 26].

The major benefits of the approach are the storage efficiency and the flexibility it gives in integrating relevant topographic features with a digital elevation model at user specified scales.

Multiscale databases for geographical information systems raise many challenging issues relating to the integration of data of different quality from different spatial models and to the automated generalization of the retrieved data.

When several spatial objects are retrieved and integrated in a model which is at a smaller scale than that of the original data, major problems can arise in visualising the model such that its components are clearly represented and distinguishable. This requires symbolizing the original geometry in ways that may involve simplification, exaggeration and change in location, all of which are aspects of cartographic generalization. Such generalization operations may be applied to data following retrieval from a multiscale database.

Update of a multiscale database is potentially a complex process in that, for any given spatial region, new data may be at different levels of detail from that already stored [14]. If the source scale of new data is the same as existing data the new data may replace the old, unless a temporal record is required. If it is more detailed it might replace existing data, though if the new, more detailed data were highly localized, relative to existing data, it might be appropriate to maintain it additionally, without replacement. Likewise, less detailed data might also be maintained additionally if it provided a potentially useful generalization, perhaps over an extensive region. Such multiple representation is particularly relevant if there are no satisfactory automatic means for generalizing the data.

The multiscale database described in this paper is applicable to the storage of spatial objects that are defined in terms of their original surveyed geometry, or simple subsets of it. This is currently applicable, but full exploitation of the multiscale database will depend on the implemention of more advanced update and generalization procedures that enable data from multiple source scales to be integrated and to undergo major generalization transformations on retrieval.

ACKNOWLEDGEMENTS

This research was partially supported by SERC grant GR/F92688 and by the Ordnance Survey. J.M.W. was supported by an SERC CASE studentship in collaboration with the British Geological Survey.

APPENDIX

```
Procedure CONSTRAIN_TRIANGULATION

For each constraining edge A to B
    If edge not already in the TIN, then
        Identify all points which have 1 or more links that intersect the edge (A, B), keeping a
         record of the distance from A of the point of intersection (for points with
         more than 1 link intersecting, it is only necessary to record 1 distance).
        Delete each point's intersecting links
        Split the points into 2 sets, S1 and S2, each containing points lying either side of (A, B).
        Sort S1 with respect to intersection distance from A.
        Sort S2 with respect to intersection distance from B.
        Add A to the list of neighbours of B
        Add B to the list of neighbours of A.
        TRIANGULATE_POLYGON(A, B, S1).
        TRIANGULATE_POLYGON(B, A, S2).
    EndIf.
EndFor.


Procedure TRIANGULATE_POLYGON(P1, P2, S).

Find all points not in current TIN but which lie within current polygon and store in V.
Search S and V for the point Q  with largest subtended angle to edge (P1, P2).
If Q not already in TIN, then
    Add Q to TIN.
EndIf
Add P1 to the list of neighbours of Q
Add P2 to the list of neighbours of Q
Add Q to the list of neighbours of P1.
Add Q to the list of neighbours of P2.
If Q is in S, then
    If edge (P1, Q) is not an edge in original TIN, then
        Create set S3, containing points in S lying between P1 and Q.
        TRIANGULATE_POLYGON(P1, Q, S3).
    EndIf.
    If edge (P2, Q) is not an edge in original TIN, then
        Create set S4, containing points in S lying between Q and P2.
        TRIANGULATE_POLYGON(Q, P2, S4).
    EndIf.
Else
    If edge (P1, Q) intersects original TIN, then
        TRIANGULATE_POLYGON(P1, Q, S)
    EndIf.
    If edge (P2, Q) intersects original TIN, then
        TRIANGULATE_POLYGON(Q, P2, S).
    EndIf.
EndIf.


Procedure FIND_THIESSEN_NEIGHBOURS(CURRENT_POINT, TNBS, NTN)

/* TNBS - an array used to hold the list of Thiessen neighbours of a point */
/* NTN - the number of Thiessen neighbours a point has */
/* NNB - the nearest point (neighbour) to the CURRENT_POINT */

Initialise SEARCH_AREA (in terms of box-sort cells).
FOUND = FALSE
Do While NOT FOUND
    Check SEARCH_AREA for nearest point (NNB) to CURRENT_POINT
    If NNB was found, then
        FOUND = TRUE.
    Else
        Expand SEARCH_AREA
        If SEARCH_AREA now extends outside the buffer limits, then
            Generate quadtree addresses for external region and read data into appropriate
            external cell
        EndIf.
    EndIf
EndDo.
NTN = 1
TNBS(NTN) = NNB
I = NNB
FINISHED = FALSE.
Do While NOT FINISHED
    Initialise SEARCH_AREA.
    FOUND = FLASE
    Do While NOT FOUND
        Check SEARCH_AREA for Thiessen neighbour (K) of edge (CURRENT_POINT, I).
        If K was found, then
            FOUND = TRUE
        Else
            Expand SEARCH_AREA.
            If SEARCH_AREA now extends outside the buffer limits, then
                Generate quadtree addresses for external region and read data into
                appropriate external cell
            EndIf
        EndIf.
    EndDo.
    If K = NNB, then
        NTN = NTN + 1.
        TNBS(NTN) = K.
        I = K.
    Else
        FINISHED = TRUE
    EndIf
EndDo.
```

```
Procedure DELAUNAY_TRIANGULATE

/* TNBS - an array used to hold the list of Thiessen neighbours of a point */
/* NTN - the number of Thiessen neighbours a point has */

Define query region.
Use region definition to generate the required quadtree addresses for both object data quadtree
and height dataquadtree
Read required data. Store objects as a sequential list of edges, each referencing two vertices.
Store object vertices and height vertices in the box-sort data structure.
Put all vertices VERTS (NUMBER_OF_VERTICES) on the TRIANGULATION_STACK.
Initialise the stack pointer (STACK_POINTER = 1).
Do While STACK_POINTER ≠ NUMBER_OF_VERTICES
    Let CURRENT_POINT = top point of TRIANGULATION_STACK(STACK_POINTER).
    FIND_THIESSEN_NEIGHBOURS(CURRENT_POINT, TNBS, NTN)
    For each pair of Thiessen neighbours (Na,Nb)
        If both neighbours are outside of the query region, then
            If external edge (Na,Nb) intersects the query region, then
                If Na has not already been triangulated, then
                    NUMBER_OF_VERTICES = NUMBER_OF_VERTICES + 1
                    TRIANGULATION_STACK(NUMBER_OF_VERTICES) = Na
                EndIf
                If Nb has not already been triangulated, then
                    NUMBER_OF_VERTICES = NUMBER_OF_VERTICES + 1.
                    TRIANGULATION_STACK(NUMBER_OF_VERTICES) = Nb.
                EndIf.
            EndIf
        EndIf
    EndFor
    Add CURRENT_POINT and Thiessen neighbours (TNBS) to the TIN.
    STACK_POINTER = STACK_POINTER + 1.
EndDo.
```

REFERENCES

[1] Abraham, I. M. (1988) Automated cartographic line generalisation and scale-Independent databases. Unpublished PhD Thesis, Department of Mathematics & Computer Studies, The Polytechnic of Wales, UK.

[2] Ballard, D. H. (1981) Strip trees: a hierarchical representation for curves. Commun. ACM, 24, 310–321.

[3] Becker, B., Six, H-W. and Widmayer, P. (1991) Spatial priority search: an access technique for scaleless maps. In Clifford, J. and King, R. (eds) *Proceedings of the 1991 ACM SIGMOD International Conference on the Management of Data*, Denver, Colorado, May 29–31. *ACM SIGMOD Record*, 20, 128–137.

[4] Chew, L. P. (1987) Constrained Delaunay triangulations. In *Proceedings of the Third ACM Symposium on Computational Geometry*, Waterloo, June, 216–222.

[5] De Floriani, L. (1987) Surface representations based on triangular grids. *The Visual Computer*, 3, pp. 27–50.

[6] De Floriani, L. (1989) A pyramidal data structure for triangle-based surface description. *IEEE Computer Graphics and Applications*, 67–78.

[7] De Floriani, L. and Puppo, E. (1988) Constrained Delaunay triangulation for multiresolution surface description. In *Proceedings of the 9th International Conference on Pattern Recognition*, Rome, November, 566–569. (Reprinted by IEEE Computer Society Press.)

[8] Douglas, D. H. and Peucker, T. K. (1973) Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10, 112–122.

[9] ElGindy, H. (1990) Optimal parallel algorithms for updating planar triangulations. In *Proceedings of the 4th International Symposium on Spatial Data Handling*, Vol. 1, Zurich, July, Publisher? 200–208.

[10] Gargantini, I. (1982) An effective way to represent quadtrees. *Commun. ACM*, 25, 905–910.

[11] Green, P. J. and Sibson, R. (1978) Computing Dirichlet tessellations in the plane. *Computer J.* 21, pp. 168–173.

[12] Guttman, A. (1984) R-trees: a dynamic index structure for spatial searching. In *Proceedings 1984 ACM SIGMOD International Conference on Management of Data*, Boston, June, Location?, 47–57.

[13] Hoel, E. G. and Samet, H. (1992) A qualitative comparison study of data structures for large line segment databases. In *Proceedings 1992 ACM SIGMOD*, San diego. *ACM SIGMOD Record*, 21, 205–214.

[14] Jones, C. B. (1991) Database architecture for multi-scale GIS. In *Proceedings Auto-Carto 10*, Baltimore, ACSM-ASPRS, Full details?, 1–14.

[15] Jones, C. B. and Abraham, I. M. (1986) Design considerations for a scale-independent cartographic database. In *Proceedings of the 2nd International Symposium on Spatial Data Handling*, Seattle, Washington, July, International Geographical Union, 384–398.

[16] Jones, C. B. and Abraham, I. M. (1987) Line generalisation in a global cartographic database. *Cartographica*, 24, 32–45.

[17] Kidner, D. B. (1991) Digital terrain models for radio path loss calculations. Unpublished PhD Thesis, Department of Mathematics & Computer Studies, The Polytechnic of Wales, UK. (Available from Defence Research Information Centre, Kentigern House, Brown Street, Glasgow, UK.)

[18] Kidner, D. B. and Jones, C. B. (1991) Implicit triangulations for large terrain databases. In *Proceedings of the 2nd European Conference on GIS*, Vol. 1, Brussels, Belgium, April, Publisher?, 537–546.

[19] Lee, J. (1991) Comparison of existing methods for building triangular irregular network models of terrain from grid digital elevation models. *Int. J. GIS*, 5, 267–285.

[20] McCullagh, M. J. and Ross, C. G. (1980) Delaunay triangulation of a random data set for isarithmic mapping. *Cartographic J.* 17, 93–99.

[21] Nelson, R. C. and Samet, H. (1986) A consistent hierarchical representation for vector data. In *Proceedings of ACM SIGGRAPH*, Dallas, August. *SIGGRAPH*, 20, 197–206.

[22] Peucker, T. K., Fowler, R. J. Little, J. J. and Mark, D. M. (1978) The triangulated irregular network. In *Proceedings of the Digital Terrain Models (DTM) Symposium*, Location?, ASP/ACSM, May, 516–540.

[23] Sibson, R. (1978) Locally equiangular triangulations. *Computer J.*, 21, 243–245.

[24] van Oosterom, P. (1990) Reactive data structures for geographic information systems. PhD thesis, Department of Computer Science, Leiden University, The Netherlands.

[25] van Oosterom, P. (1991) The Reactive-Tree: a storage structure for a seamless, scaleless geographic database. In *Proceedings Auto-Carto 10*, ACSM/ASPRS, Vol. 6, Baltimore, March, 393–407.

[26] Ware, J. A. and Kidner, D. B. (1991) Parallel implementation of the Delaunay triangulation within a transputer environment. In *Proceedings of the 2nd European Conference on GIS*, Vol. 2, Brussels, Belgium, April, Publisher?, 1199–1208.

[27] Ware, J. M. and Jones, C. B. (1992) A multiresolution topographic surface database. *Int. J. GIS*, 6, 479–496.