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-

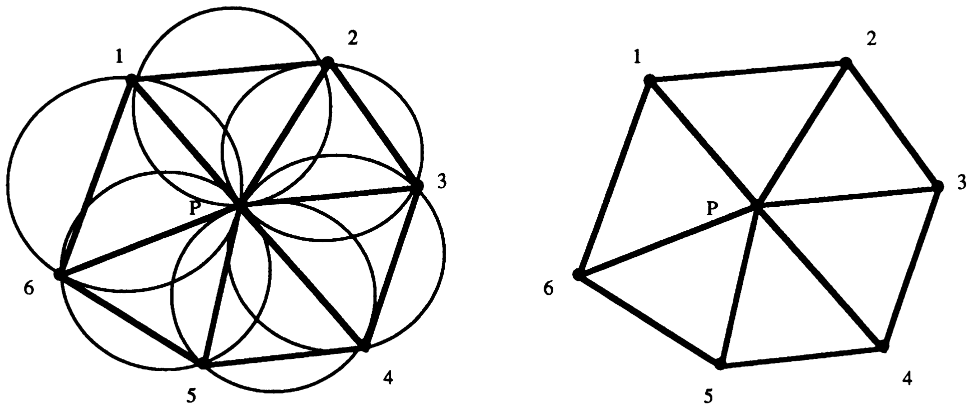
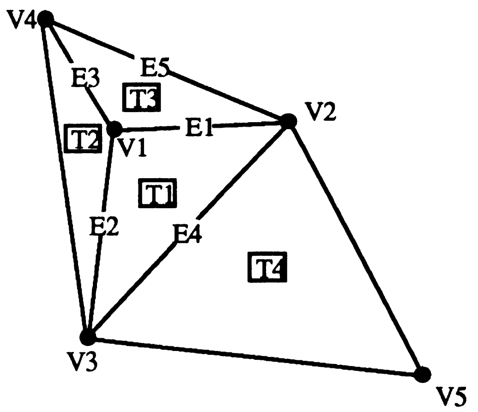


FIGURE 1. Local Delaunay triangulation about a point P.

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 ιopographic 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 prespecified 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

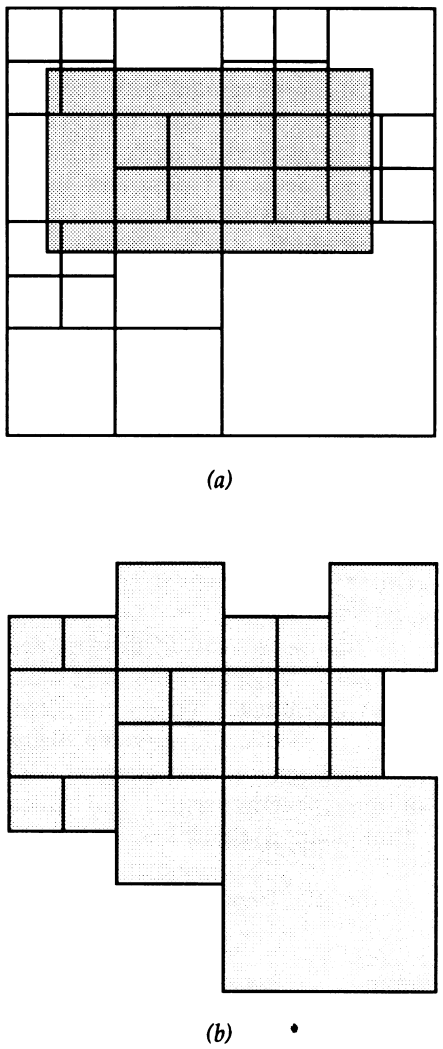


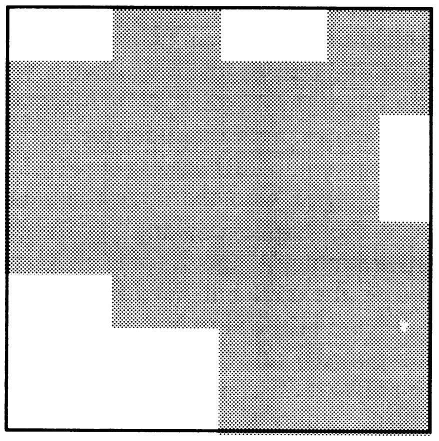
FIGURE 3. (a) The query region with respect to the database. (b) The quadtree cells (and data) accessed in the database.

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].

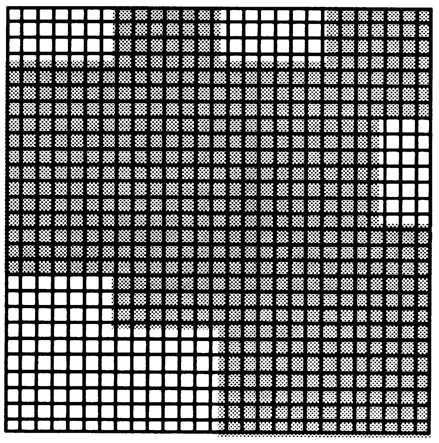
The correct constrained triangulation is obtained in two stages. First a Delaunay triangulation of all relevant vertices (from elevation data and constraining objects) is performed (see Procedure DELAUNAY_TRIANGULATE). Points that are interior to the query region will always belong to the final triangulation, 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).

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

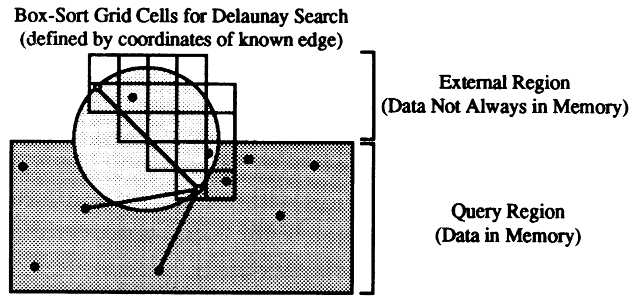


(a)

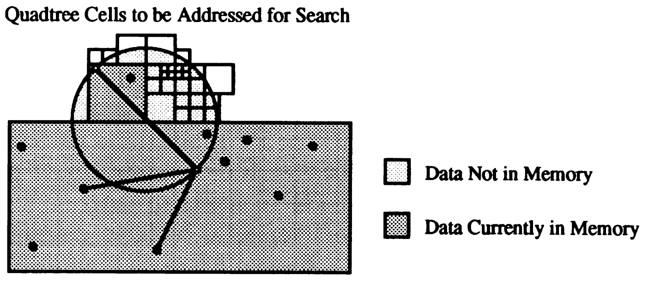


(b)

FIGURE 4. (a) A bounding rectangle placed around quadtree cells. (b) The initial box-sort data structure with referenced and empty cells.


Box-Sort Grid Cells for Delaunay Search
(defined by coordinates of known edge)
External Region
(Data Not Always in Memory)
Query Region
(Data in Memory)

(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.

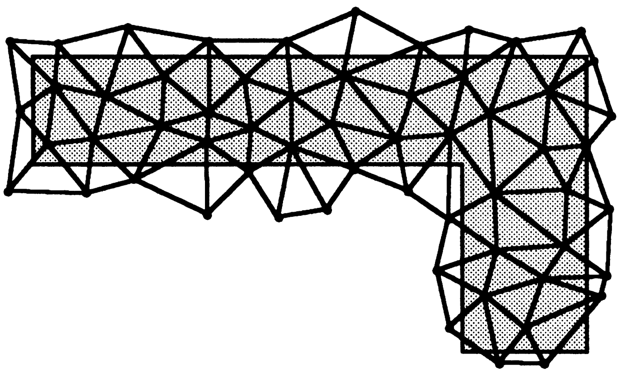


FIGURE 6. The triangulation of all vertices within the query region. (Note that coverage is not complete in the bottom two corners.)

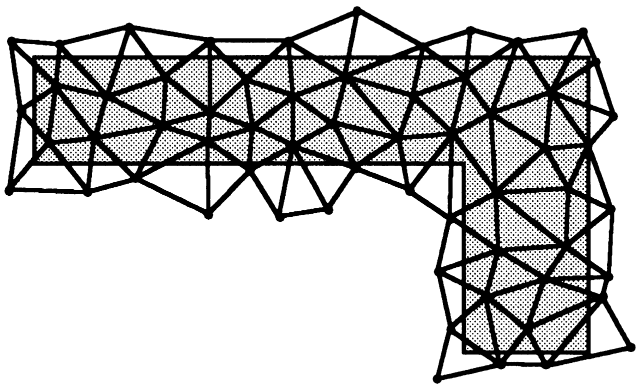


FIGURE 7. The final triangulation of query region.

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.

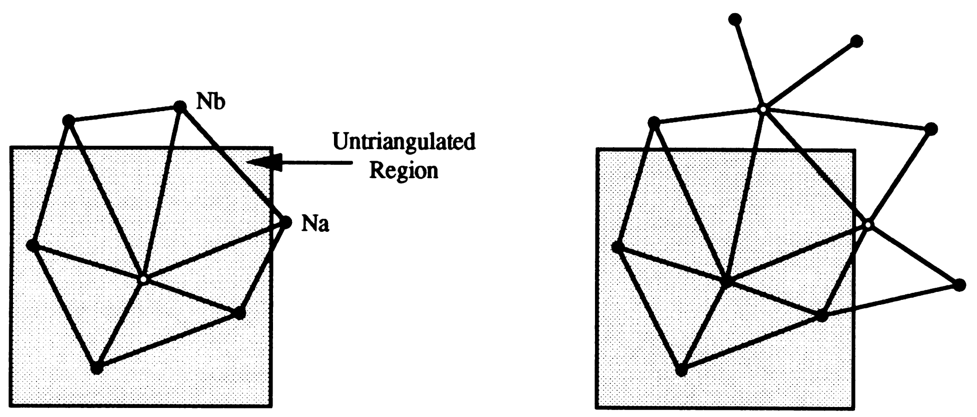
For a constraining segment with one vertex in the TIN and a second outside, the procedure is very similar, but triangle edges may extend to vertices outside the original TIN. For example, consider the insertion of the highlighted segment (A, B) in Figure 11(a). Here the search for the vertex with the 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.

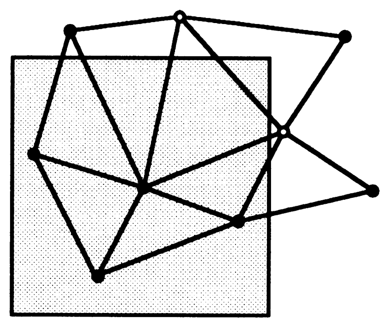
2.6. Restricted region triangulation

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

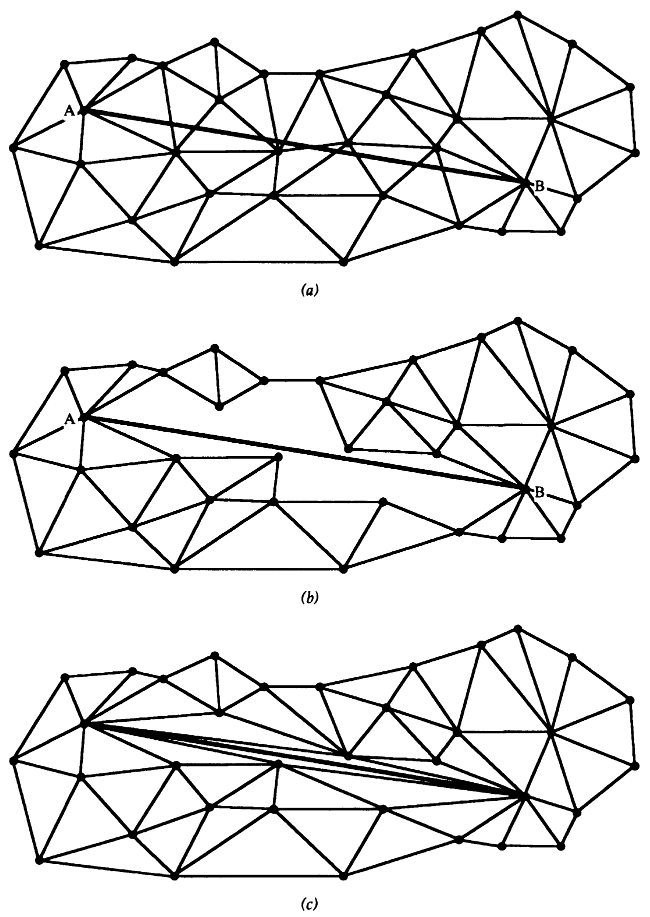


FIGURE 9. Constrained edge insertion within a TIN. Insertion of a constraining edge AB involves finding existing edges that intersect it (a), deletion of these edges (b) and re-triangulation to include AB as an edge in the triangulation (c).

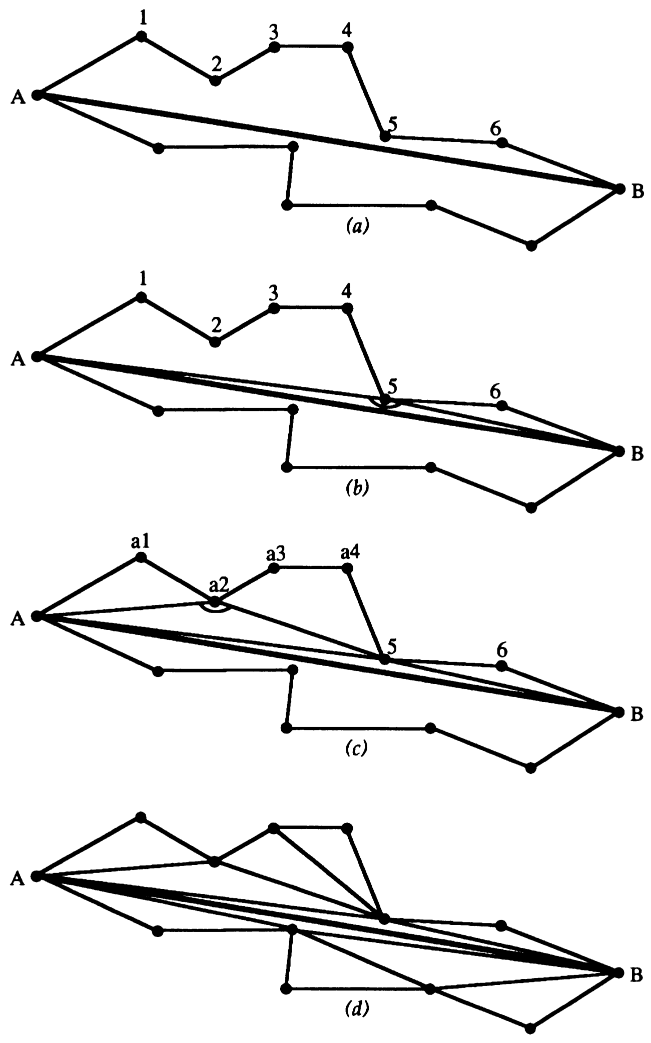


FIGURE 10. Triangulation within a polygon around a constrained edge. The inserted edge forms the base of two polygons to be triangulated (a). Triangulation of each polygon proceeds by selecting the vertex which subtends the largest angle with the base edge (vertex 5 in b). Each new edge is treated recursively 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)

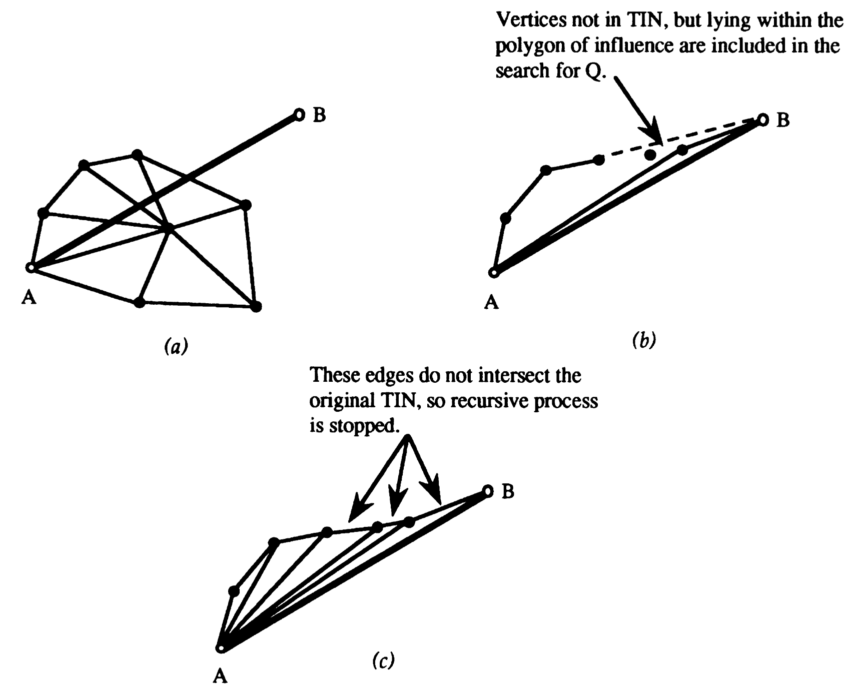

Vertices not in TIN, but lying within the polygon of influence are included in the search for Q.
B
A
(a)
B
A
(b)
These edges do not intersect the original TIN, so recursive process is stopped.
B
A
(c)

FIGURE 11. Insertion of an edge with one external vertex (see text for explanation).

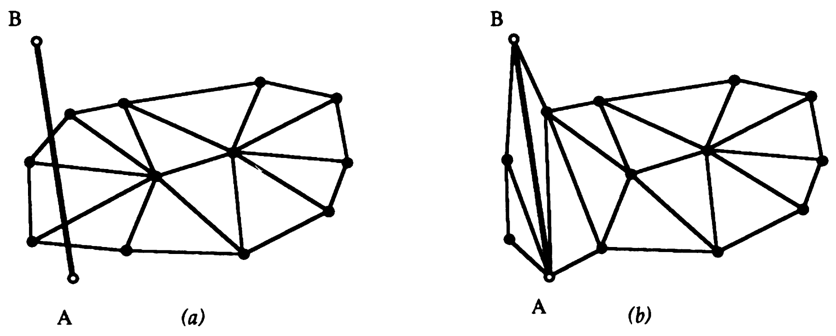


FIGURE 12. Insertion of an edge with two external vertices A and B (a) follows the procedure for insertion of an edge with one internal vertex, to result in the constrained triangulation (b).

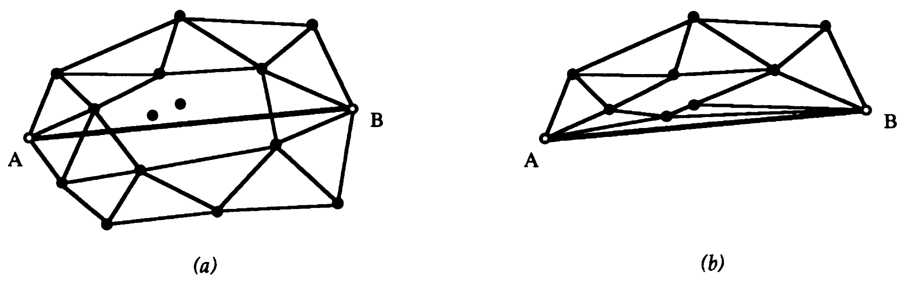


FIGURE 13. Insertion of an edge through a hole in the triangulation of a concave query window. Triangulation of the constraining edge AB (a) continues within the polygons of influence until new base edges either belong to the original triangulation or do not intersect the query window (b shows the triangulation 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

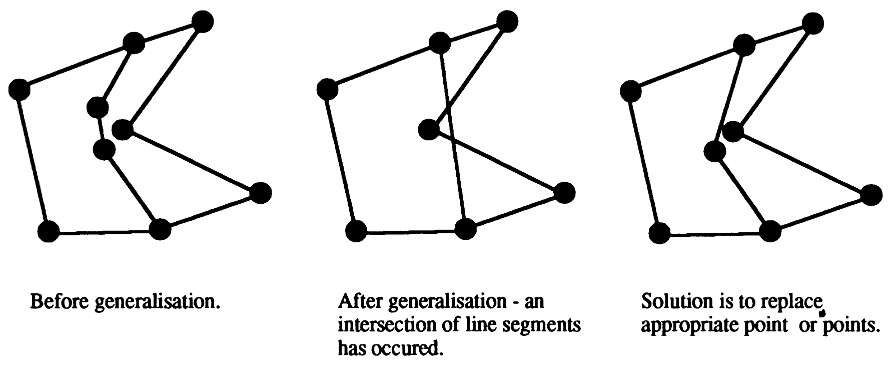

Before generalisation.
After generalisation - an intersection of line segments has occured.
Solution is to replace appropriate point or points.

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

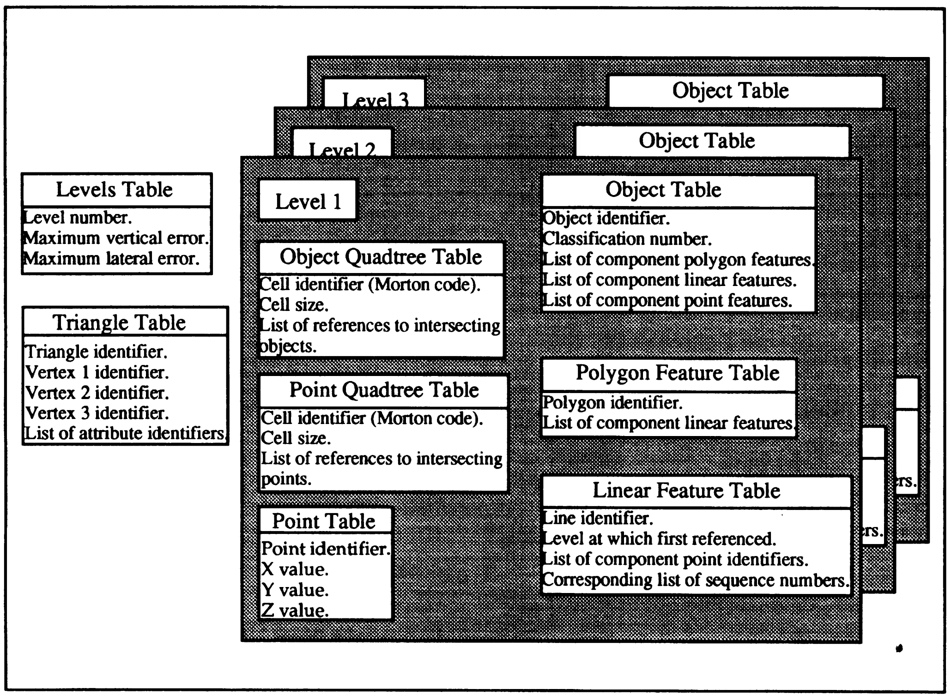


FIGURE 15. Overview of the Multiscale Database (in this instance with three levels of detail).

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.

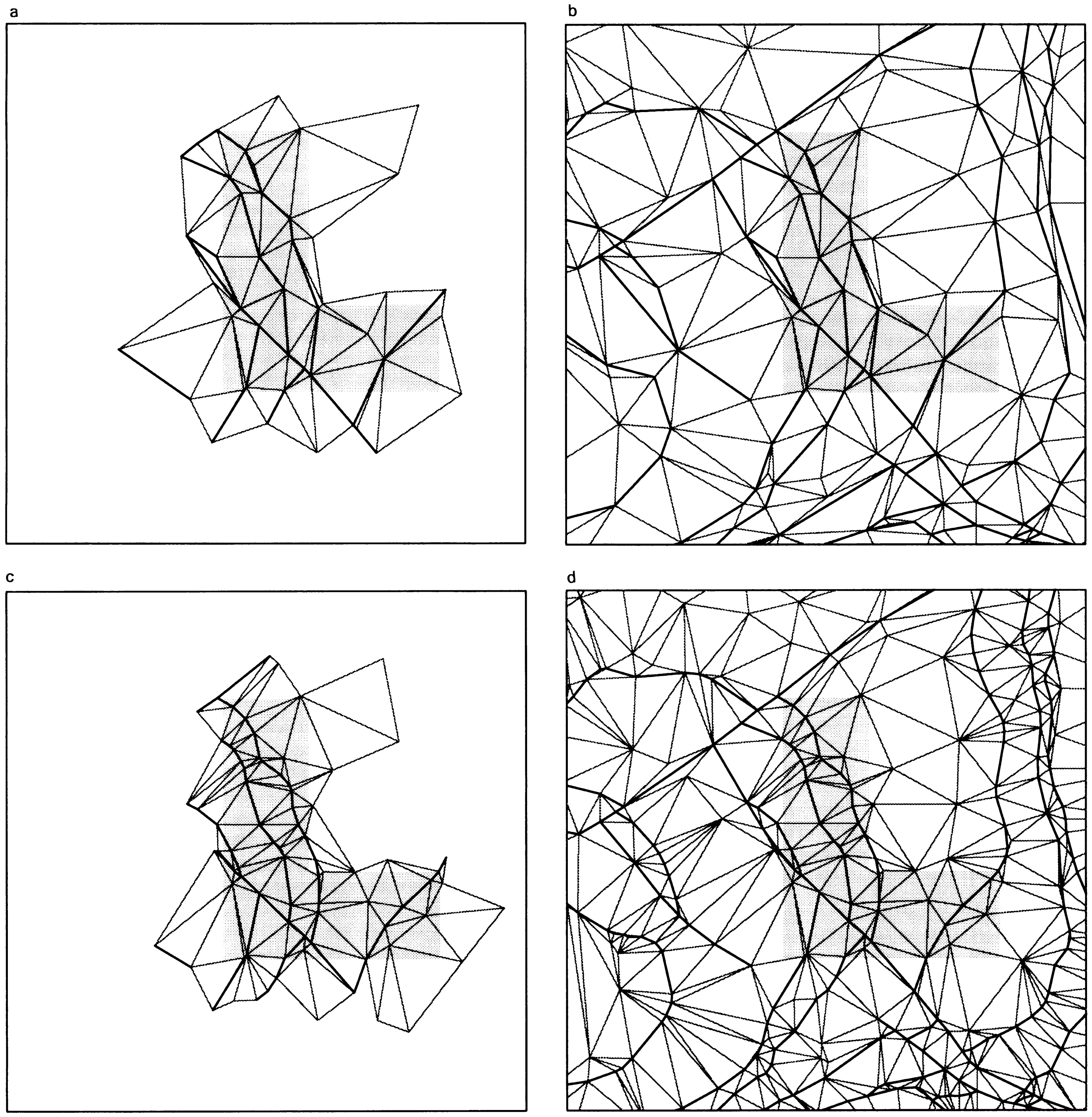


FIGURE 16. (a) and (c) show the triangulations produced by the Implicit TIN algorithm when applied to two levels in a multiscale geological database; (b) and (d) are the triangulations for the same area produced using a conventional triangulation algorithm.

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.
J = 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, J).
        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.
        J = 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.