Multi-Scale Spatial Database and Map Generalisation

Sheng Zhou & Christopher B. Jones
Department of Computer Science
Cardiff University
Cardiff, CF24 3XF
United Kingdom
{s.zhou, c.b.jones}@cs.cf.ac.uk
Fax: (+44) 29-20874598
Corresponding author: S. Zhou

Abstract

The constitution of multi-scale spatial databases is closely linked to map generalisation. Elements of a multi-scale spatial database include multi-scale data models, multi-scale data storage schemes, methods to generate multi-scale datasets and methods to access these datasets and present results. In this paper we proposed a formal multi-scale spatial data model with an information-rich structure which supports representation of multiple objects at the same location and the same scale. In using existing generalisation procedures to generate multi-scale spatial datasets, an indicator which conforms to the retrieval precedence of vertices and a mapping mechanism to link scales and indicator values are required. The binary tolerance tree (BVT) is presented with the objective of meeting these two requirements. When data are retrieved from a multi-scale database, conflict resolution procedures can help to present the results in legible form. We present a shape-preserving method for on-line linear feature displacement using local coordinate transformation and analysis on curve characteristics. Finally, a brief introduction on several multi-scale geometry storage schemes as well as some conclusions drawn from experiments is given.

Keywords Multi-scale spatial database, line displacement, binary tolerance tree (BVT)

1. Introduction

Geographical space is an extremely information-rich environment. Human perception on geographical phenomena via observations at different time and from different points of view (which may be regarded as *scales*) generates mental images of these phenomena. Subsequently, geographical objects are recognised from these mental images and concepts of map feature classes are abstracted. Since an object may have different spatial forms at various scales with the same semantically unique identification, scale may be treated as a fifth dimension, alongside the three spatial dimensions and the temporal dimension in a 5-D space.

A map is a model of a part of geographical space while map features represent geographical phenomena. A conventional map reflects our knowledge of a part of geo-space with the limitation that only a small subset of all phenomena which present at a certain time and a fixed scale can be included. The advance of GIS and spatial database [1] technologies has not changed the way we access spatial data fundamentally as a conventional GIS or spatial database may still best be described as a collection of maps at various fixed scales without support for scale and temporal changes in geographical phenomena.

A truly *integrated spatial database* should be seamless over spatial extensions and temporal changes as well as scale changes. While scale issue is our main concern in this paper, such an integrated database is a *multi-scale spatial database*. The main issues relevant to the design, implementation and application of a multi-scale spatial database include:

- Multi-scale spatial data model
- Multi-scale database schema and geometric data storage schemes for DBMS to support efficient data retrieval based on spatial extent, scale and other properties.
- Generating multi-scale spatial datasets from various data sources to populate a multi-scale spatial database
- Producing legible presentation of data retrieved from a multi-scale spatial database

From a database-centred point of view, generation of multi-scale spatial datasets may be regarded as a pre-process while the presentation of results is a post-process. Therefore we believe the first principle of design and implementation of a multi-scale spatial database is to strike a balance between the pre-process and the post-process [2]. We also believe that map generalisation are essential to these two processes.

There have been several studies on designing multi-scale spatial models (e.g. [3]). We will discuss some limitations of existing models and present a conceptual model for multi-scale spatial data in section 2. In section 3 and 4 we will discuss issues in applying map generalisation procedures to generate multi-scale spatial datasets and present query results from a multi-scale spatial database. In section 5 we present several storage schemes, some of which are built upon ideas presented in [4-6]. In section 6 a brief introduction to proposed future works is given.

2. Spatial Object with a Scale Dimension and Multi-scale databases

2.1 Scale, Scale Interval and Scale Range

As already mentioned, scale may be broadly regarded as the reflection of viewpoint of observation. For clarity we will follow the standard cartographic terminology in this paper. Therefore, for two *scales* or *scale values* $s_1, s_2 \in (0, \infty)$ and $s_1 < s_2$, we regard s_2 as a larger scale corresponding to a more detailed map and s_1 as a smaller scale for a map with less details.

A *scale interval* is defined by two scales: $SI = [s_L, s_H]$, $s_L \le s_H$ (or alternative forms: (s_L, s_H), $[s_L, s_H)$ and $(s_L, s_H]$). We regard s_L as the *lower scale bound* and s_H as the *higher scale bound* of the scale interval.

A *scale range* is a set of scale intervals: $SR = \{SI_i \mid i = 1, n\}$ with $MIN(s_{Li})$ and $MAX(s_{Hi})$ as its lower and higher scale bounds.

2.2 Multi-Scale Spatial Object (*MSO*)

A spatial object is an identifiable instance of a geographical concept (or *spatial object type*). Therefore, a spatial object has its unique *object identifier*. As a geographical phenomenon occurs across a range of scales, a spatial object is associated with a scale range (*sr*) property. For reasons stated later, in some cases this scale range may be divided into two scale ranges: *dominant scale range* (*dsr*) and *recessive scale range* (*rsr*).

For a scale value $s \in sr$, there is a *geometry* which represents the geometric form of the object at the scale. All geometric representations form the *geometry set* (*gs*) of a spatial object. Note that which one of these geometries will be retrieved to represent the spatial object may not be solely decided by scale. However, normally it is convenient to index these geometries by scale values.

A spatial object also contains a *presentation compatibility set* (*pcs*) to indicate its compatibility with other objects (the term "*compatibility*" generally refers to phenomena that can be graphically overlapped on the presentation media). This set may contain individual values for each instance of a spatial object type or it may be defined as a class property (similar to static members in a C++ or Java class).

Geographical phenomena often show a hierarchical structure over scale transition and a spatial object is often derived from a group of objects at larger scales while the object itself may be a source of another spatial object at smaller scales. To track these relations, a spatial object may contain two reference sets: *derived-from* and *source-of*.

Finally, a spatial object normally has other application-dependent properties.

2.3 Multi-Scale Geometry (*MGEO*)

It is impractical to store a geometry for any scale value in the scale range of a spatial object due to the virtually infinite scale values in the range (assuming $s_L < s_H$). Methods have to be found to share geometric data among various geometries in the geometry set of a spatial object. For this reason we

define the concepts of multi-scale geometry on the basis of single-scale geometric data models (e.g. [7-9]).

2.3.1 Single-scale geometric data structures and transformation

From a vector point of view, we regard any geometric object as a set of points constructed by different rules. There are three simple geometry types and an indefinite number of complex geometry types built on simple types as well as nesting complex types. Note that for these geometry types only those properties closely related to our later discussions are defined.

2.3.1.1 Simple geometries (*SG*)

A simple geometry is defined at a fixed scale *s*:

- Point: $p^s(x, y, z)$ or $P(s) = \{ p(x, y, z)\}$ in a set form.
- Simple Polyline: $PL(s)=\{p^s_i | i = 1, n; n \geq 2\}$ which may be self-intersected but not closed; *i* is regarded as the *sequence number* of *vertex* p_i; p_1 and p_n are the starting and ending points of PL.
- Simple Polygon: $PG(s)=\{p^s_i | i = 1, n; n \geq 3\}$which may be self-intersected; *i* is regarded as the *sequence number* of *vertex* p_i;

Supported simple geometry transformations include:

- $T_{p\_p}(P_1(s) \rightarrow P_2(s))$: $P_1=\{p_1(x_1,y_1,z_1)\}$ and $P_2 = \{p_2(x_2, y_2, z_2) | x_2 \neq x_1 \vee y_2 \neq y_1 \vee z_2 \neq z_1 \}$. This transformation represents a point displacement operation.
- $T_{PL\_PL}(PL_1(s) \rightarrow PL_2(s))$: $PL_1 = \{p_i | i = 1, n; n \geq 2\}$ and $PL_2 = \{p_j | j = 1, m; n \geq m \geq 2; p_j \in PL_1 \vee p_j = T_{P\_P}(p_k) \in PL_2, p_k \in PL_1 \}$. This transformation represents a line simplification operation from a generalisation point of view.
- $T_{PL\_P}(PL(s) \rightarrow P(s))$: $P = \{ p(x, y, z)\} \subset PL$. By applying this transformation, a polyline collapses to a point.
- $T_{PG\_PG}(PG_1(s) \rightarrow PG_2(s))$: $PG_1 = \{p_i | i = 1, n; n \geq 3\}$ and $PG_2 = \{p_j | j = 1, m; n \geq m \geq 3; p_j \in PL_1 \vee p_j = T_{P\_P}(p_k) \in PL_2, p_k \in PL_1 \}$. This transformation simplifies the boundary of a simple polygon.
- $T_{PG\_P}(PG(s) \rightarrow P(s))$: $P =\{ p(x, y, z)\} \subset PG$. As T_{PL_P}, a simple polygon collapses to a point.

We regard T_{PL_PL} and T_{PG_PG} as non-destructive transformations because the type of the geometry involved is not changed. On the other hand, T_{PL_P} and T_{PG_P} are destructive transformations.

2.3.1.2 Complex geometries (*CG*)

A complex geometry is a collection of simple geometries as well as other complex geometries. Different rules for collection define different complex geometry types.

Examples of complex geometries include:

- Multi-Segment Polyline: $MPL(s) = \{(MPL_i/PL_i(s), dir_i \in \{-1, 1\}) | i = 1, n\}$ which is a sequence of end-connecting simple or multi-segment polylines. $MPL_i/PL_i(s)$ means that the members in the set can be either simple polylines or other multi-segment polylines. dir_i represents the direction relating to the original polylines.
- Multi-Segment Polygon: $MPG(s) = \{(MPL_i/PL_i(s), dir_i \in \{-1, 1\}) | i = 1, n\}$ whose boundary is defined by a closed sequence of end-connecting simple or multi-segment polylines.
- Network: a collection of simple or multi-segment polylines connected at network nodes
  - NetworkNode: $NN(s) = (MPL_i/PL_i(s), dir_i \in \{-1, 1\})$
  - Network: $N(s) = \{NN_i(s) | i = 1, n\}$
- Complex Polygon (CPG): $CPG(s) = \{ MPG/PG, \{MPG_i/PG_i | i = 0, n\} \}$ which has one simple or multi-segment polygon as boundary and may contains holes.

Some other potential complex geometries defined by simple set construction rules include:

- Point Collection(PC): a point set
- Polyline Collection (PLC): a set of simple/multi-segment polylines
- Polygon Collection (PGC): a set of simple/multi-segment/complex polygons
- Mixed Geometry Collection (MGC): a set of geometries of various types

There are many transformations which may be defined for these types. We are particularly interested in those transformations which

- do not change the type of the complex geometry;
- applying only non-destructive transformations to embedded simple geometries;
- for complex polygon and various collection types, a component which is simple or complex is eliminated entirely during the transformation.

These transformations form the foundation for defining and constructing simple and complex multi-scale geometries. There are other destructive transformations of complex geometries as well as transformations between simple and complex geometry types which may be regarded as corresponding to various generalisation operations.

2.3.2 Simple and complex multi-scale geometry

A multi-scale geometry has a scale range which contains at least one scale interval of non-zero length. For each point in a multi-scale geometry, a scale range which may have zero-length interval is associated.

2.3.2.1 Simple multi-scale geometry

Under the multi-scale context, a multi-scale point (MS point) has a scale range property *sr*:

- MS Point: $p^{ms} = (p^s(x, y, z), sr_P)$

The definitions of simple MS polyline and MS polygon are similar to the single-scale ones except that the vertices in a MS polyline/polygon are MS points:

- Simple MS Polyline: MSPL = $(\{p^{ms}_i \mid i = 1, n; n \geq 2\}, sr_{PL}), p^{ms}_i.sr_P \cap sr_{PL} \neq \varnothing$
- Simple MS Polygon: MSPG = $(\{p^{ms}_i \mid i = 1, n; n \geq 3\}, sr_{PG}), p^{ms}_i.sr_P \cap sr_{PG} \neq \varnothing$

Using the polyline type as an example, the process of constructing a multi-scale geometry from various single-scale geometries of a spatial object can be formally described as follows:

Given two single-scale simple polylines $PL_1(s_1)$ and $PL_2(s_2)$, $s_1 > s_2$ as representations of the same spatial object, if $PL_2 = T_{PL\_PL}(PL_1)$, a MS polyline may be defined as:

$$MSPL_1 = (\{p^{ms}_i(p^s_i, sr_p) \mid p^s_i \in PL_1 \vee p^s_i = T_{PP}(p^s_j) \in PL_2, p^s_j \in PL_1\}, sr_{PL\_1})$$

At this stage, $sr_{PL\_1} = \{[s_2, s_2], [s_1, s_1]\}$. For vertices contained by PL_1 and by PL_2 (directly or by a T_{PP} transformation), the scale range is $\{[s_2, s_2], [s_1, s_1]\}$; for other vertices not presented in PL_2, the scale range is $\{[s_1, s_1]\}$.

If there is not any representation $PL_i(s_i)$ with a scale $s_1 > s_i > s_2$, which is a reasonable assumption as in practice only a finite set of single-scale representations will be available (e.g. the representations of the same object in maps of different scales) for a spatial object, the representation required has to be interpolated from existing representations (i.e. PL_1 and PL_2). We may use PL_1 as an approximation at scale $s_i \in (s_2, s_1)$. Alternatively, we may decide on an intermediate scale value $s_m \in (s_2, s_1)$ and use PL_1 at $[s_m, s_1)$ and PL_2 at (s_2, s_m). Consequently the scale range of vertices which are in both PL_1 and PL_2 is extended to become $\{[s_2, s_1]\}$ while for other vertices it is $\{[s_1, s_1]\}$(the first case) or $\{[s_m, s_1]\}$. In either case, the scale range of the multi-scale geometry $MSPL_1$ is extended to be: $sr_{PL\_1} = \{[s_2, s_1]\}$. Indeed, in practice we use single-scale maps in the same manner, assuming they are good approximations at scales relatively close to the maps' designated scales.

For multi-scale spatial objects with multiple representations, this process may be repeated to merge other single-scale geometries into the multi-scale geometry. In case there are single-scale geometries which can not be merged by the operations defined above as some or all of their vertices can not meet the transformation criteria, one or more new multi-scale geometries will be required to represent the multi-scale spatial object.

When a multi-scale geometry with a scale range $\{[s_n, s_1]\}$ is generated, we regard those vertices with scale ranges in the form of $\{[s_i, s_1]\}$ while $s_1 \geq s_i \geq s_n$ as *subsetting* vertices of the geometry. Other *non-subsetting* vertices normally correspond to operations like vertex displacement during the process.

2.3.2.2 Complex multi-scale geometry

Complex multi-scale geometries contain simple multi-scale geometries or other complex multi-scale geometries as their components, which are constructed under the constraints of the rules for constructing simple multi-scale geometries as well as the non-destructive transformation rules described in 2.3.1.2.

It is worth noting by allowing a point in one geometry to be referred by other geometries and a geometry to be a component of more than one complex geometry, the model described above can be more flexible and storage-efficient. However, from a practical point of view, with the exceptions of points or geometries stored as standalone entities in an implementation, these types of references will be very inefficient in a DBMS environment. Therefore, in the following discussions we are not going to explore these possibilities beyond the above exceptions.

2.4 Scale-Transition, Dominant and Recessive Spatial Objects

2.4.1 Location and scale transition

Spatial objects represent information of geographical phenomena at various locations on the earth's surface. The minimum physical size of a *location* depends on scale. At the largest scale of interest, there may be only one or a few mutually-compatible phenomena occurring at a single location. At a smaller scale, a location is the aggregation of a group of locations at larger scales. As a geographical phenomenon may occur over a range of scales, several incompatible phenomena which are separated into different locations at larger scales may occur in the same location at a smaller scale. In the process of conventional map-making, we select the one or a few compatible phenomena to present according to our requirement.

In a spatial database a spatial dataset is stored inside the *database space* while queries on the database retrieve data and information and present them on some sort of media in a *presentation space* which can then be further interpreted by the users.

To a great extent, a paper map may also be regarded as a primary spatial database. One fundamental characteristic of a conventional map is that its database space and the presentation space are the same with a physical linkage between the storage and the presentation of spatial information. Consequently, the limited amount of presentation area in the presentation space restricts the number of spatial objects which can be stored and presented, i.e. only one (or a few compatible) spatial object may occupy a certain location. On the other hand, a computer-based spatial database physically separates the database space from the presentation space and spatial objects are linked logically to their presentations in the presentation space. In theory, it is possible in a multi-scale spatial database to store all available knowledge and information of any location at any supported scales, which will then provide multiple choices of retrieval according users varying preferences.

Indeed, if a multi-scale spatial database is constructed for a single purpose only, e.g. providing legible topographical maps at any scales, the demand of retrieval is then predictable and there is no need to store data of multiple objects at the same location. In a general-purpose spatial database, queries based on different, often mutually conflicting, requirements have to be supported and hence it is highly desirable to have a dataset with richer information stored.

2.4.2 Dominant and recessive spatial objects

For a location at a certain scale within the scale ranges of several incompatible objects, all these objects may be easily stored in a computer-based spatial database. However, which of these objects will be retrieved has to be decided at run-time on information specified by users, which is, however, unfortunately unpredictable at the time when the database is constructed. Nevertheless, when a conflict at a location occurs at the construction stage, a default decision may be made to select one or a few compatible objects as *dominant* objects (i.e. to appear in the final presentation) at this scale, where others at the location become *recessive* at this scale. Consequently, the scale range (*sr*) of an object may be divided into two sub-ranges: a dominant range (*dsr*) and a recessive range (*rsr*). In practice, it might be that the *sr* is used along with the *dsr*. When a default query is submitted, the *dsr* is used to retrieve the dominant object(s) at the query scale; otherwise, the *sr* is used to retrieve all objects at the location and further processes are then carried out possibly by the application programs to decide the non-default dominant object(s).

In summary, we believe it is the concepts of location aggregation and multiple competing objects at a single location that give our model a stronger power of expression and differentiate it from many existing ones.

3. Map Generalisation and Generation of Multi-Scale Spatial Dataset

Using processes described above, individual representations of a multi-scale spatial object can be merged to one or several multi-scale geometries which form the geometry set of the spatial object. These multi-scale geometries covering various scale ranges may have different types (simple, complex, etc.). The union of scale ranges of all multi-scale geometries in the geometry set forms the scale range of the spatial object.

As existing single-scale paper maps (and map series) and their digitised equivalents remain the main source of our information and knowledge on geographical space, a multi-scale spatial database will have to be built on the basis of these datasets, i.e. to generate objects' representations at scales smaller than the designated scale of the source map. Since map generalisation is regarded as the process of generating relatively small scale maps from more detailed source maps, it is natural to make use of map generalisation procedures to facilitate the generation of integrated spatial datasets.

At present, we restrict our discussion to processing a simple geometry at a large scale to create multi-scale geometries. The main issue in this process is to define a vertex precedence with an objective indicator that may be mapped to scale values.

3.1 Decomposing the Scale Space

There are two ways to classify vertices in a geometric object to decide their scale ranges. We may pre-define a scale range with adjacent scale intervals like $\{[s_n, s_{n-1}), ...[s_{i+1}, s_i), [s_i, s_{i-1}), ...[s_2, s_1]\}$ and for each of these intervals we check whether a vertex should be present. The union of all intervals a vertex presents forms the scale range of this vertex. We regard this method as the *key-space decomposition* [10] approach as it is undertaken on the basis of a pre-defined decomposition strategy of the key-space (i.e. the scale space), which is also used in variouswith regard to space. Alternatively, following the example of R-Tree construction, we may want to find some method to calculate the scale range directly for each vertex on the basis of its characteristics (e.g. its relations with other vertices in the geometry). This is the *object-space decomposition* [10] approach as the scale space is decomposed according to the concrete objects (vertices).

3.2 Methods for Vertex Scale Range Classification

3.2.1 Key-Space decomposition methods

Most line simplification algorithms may be regarded as key-space decomposition methods when adopted for vertex classification. In the Douglas-Peucker algorithm, for example, a tolerance value is set before a subset of vertices are selected by the displacement criterion. The remaining problem is how to link the parameter value(s) (in this case, the tolerance length) to scale.

By using very fine pre-defined scale intervals, key-space decomposition methods can classify vertices and produce vertex scale ranges to any required precision although the whole process may become extremely tedious. On the other hand, it is understandable there is still some chances of under-sampling or over-sampling when a particular dataset is processed, which will not satisfy perfectionists.

3.2.2 Object-Space decomposition methods

Object-space decomposition methods have the potential of generating the exact scale ranges for vertices and hence provide better support to "continuous scale changes" [6]. Some of methods falling into this category include the BLG-tree [6] which calculates tolerance values of vertices in a top-down manner to construct a binary vertex tree with maximum tolerance value stored for each node, and the band-width hierarchical tree [11] which is a top-down or bottom-up constructed binary tree structure.

The difficulty in adopting the above methods to create multi-scale geometries which will be stored in a DBMS is that either the tolerance values or the bandwidth values are not decreasing monotonically from the top of the binary tree down to the bottom. In other words, these values do not conform to the order in which the vertices are traversed. Therefore, for a given tolerance or bandwidth value, all nodes in the tree have to be checked in order to select the qualified vertices, which will cause severe performance problem in a DBMS environment.

3.3 BVT-Tree

In this sub-section we introduce the Binary Vertex Tolerance Tree (BVT-tree), an object-space decomposition method which solves the problem associated with the BLG tree and the hierarchical bandwidth tree. The vertex tolerance used in the BVT-tree is calculated in the same way as in the BLG tree.

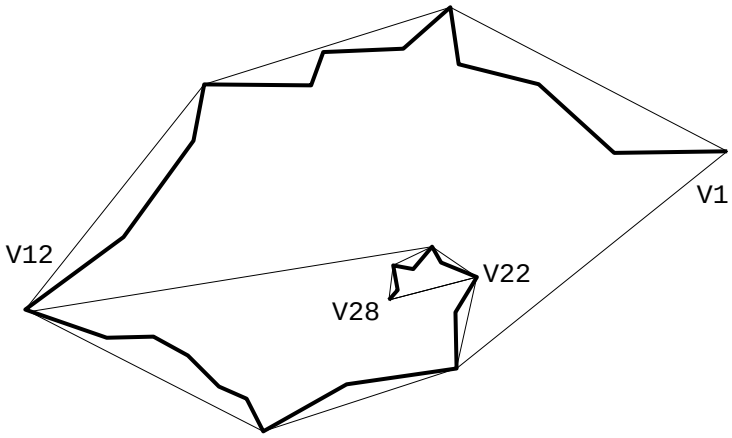

Fig. 1 Geometry sectioning using convex hull. The object (thick line) is divided to three sections: $v_1\_v_{12}$, $v_{12}\_v_{22}$ and $v_{22}\_v_{28}$)

3.3.1 Convex hull and geometry sectioning

An issue in applying some hierarchical process methods to geometric objects is to decide the right starting point(s), especially for linear objects which extend beyond endpoints [12] (e.g. in Fig. 1, the polyline extends beyond the two endpoints v_1 and v_{28}). Before calculating tolerance values for a geometry, we calculate the convex hull of the geometry to divide the geometry into sections at the two vertices (*section vertices*) which are the endpoints of the longest diagonal line in the convex hull. For a polyline, if a section vertex is not an original endpoint, the convex hull sectioning process will be repeated on the section between this section vertex and the neighbouring endpoint until there is no remaining "extending beyond endpoints" section; for a polygon, two sections are generated.

3.3.2 Vertex tolerance promotion

When a geometric object is sectioned, a BLG-like process will be carried out for each section in the geometry. The tolerance value for a section vertex is the distance between the two section vertices. When a section vertex is shared by two sections, the larger value will be used.

When all vertices are processed, a check will be carried out to find out any node in the binary tree with a tolerance value larger than its parent node and to "promote" its parent nodes' tolerance value to this larger value. This process will be repeated until the tolerance values of all nodes are monotonically decreasing from the top down to the bottom, which reflects the precedence of vertices in the tree hierarchy. We name this process as *tolerance promotion*.

The rationale behind this method is that vertices in a geometry are inter-dependent in the process of tolerance calculation. The presence of the larger tolerance value of a child node depends on the presence of its parent nodes which precede it in the tree hierarchy. For example, in Fig. 2, vertex v_3 has an initial tolerance of 4.03 relating to the two section vertices v_1 and v_4 while v_2 has a tolerance of 4.97 which is larger than that of v_3. However, this larger value depends on the presence of v_3, otherwise, the tolerance of v_2 in relating to $v_1\_v_4$ is only 2.84. On the hand, tolerance of v_3 relating to v_2 and v_4 is even larger (5.47, indeed, it can be proven geometrically this will always happen under these circumstances)

but it also depends on v_2. For a query tolerance value between 4.03 and 4.97, v_3 will not be retrieved if the initial tolerance of v_3 (4.03) is used, but, v_2 will be retrieved while its larger tolerance value can not be derived because of the absence of v_3. On the other hand, if we change the tolerance of v_3 to 5.47 which depends on v_2, for query tolerance between 4.97 and 5.47, v_2 will not be retrieved and v_3's larger tolerance will not be derived either. Therefore, we promote v_3's tolerance from 4.03 to 4.97, which will guarantee a proper retrieval for all query tolerance values.

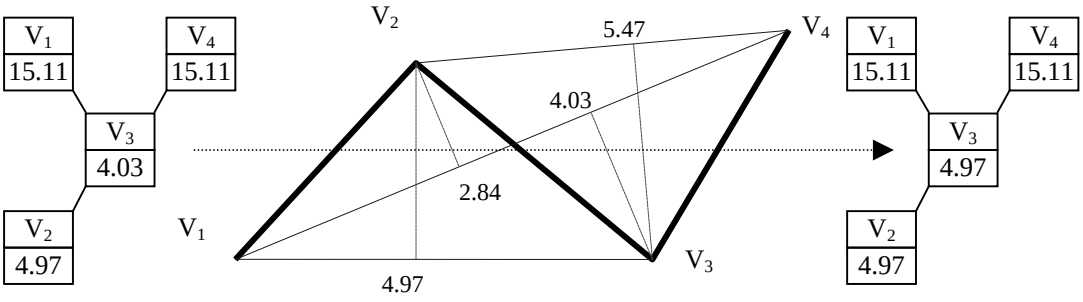

Fig. 2 Vertex tolerance promotion

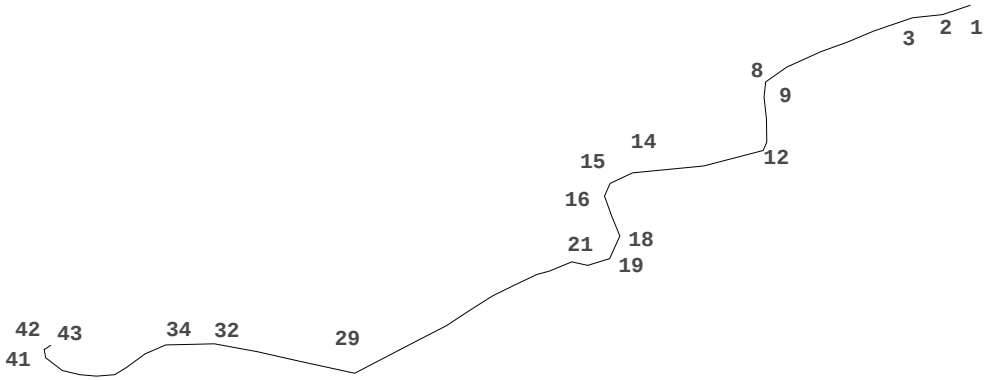

Fig. 3 A simple polyline with 43 vertices and two sections(v_1_v_{41} & v_{41}_v_{43})

3.3.3 BVT-Tree reconstruction

When the tolerance calculation is completed for an object (Fig. 3), its BVT-tree has a forest shape with all section vertices at the top (Fig. 4). The process of tolerance promotion will then be carried out and the forest will be readjusted to form a true binary tree with one root node (one of those with largest tolerance value). During this process, a pair of parent-child nodes may be swapped in order to reduce the overall height of the tree (in this example, the height is reduced by two levels). The reconstructed BVT-tree (Fig. 5) is then a binary tree in which each node has a tolerance value no smaller than the tolerance(s) of its child node(s) and whose left child node has a smaller sequence number (i.e. nearer to the starting vertex of the geometry) and whose right node has a larger one.

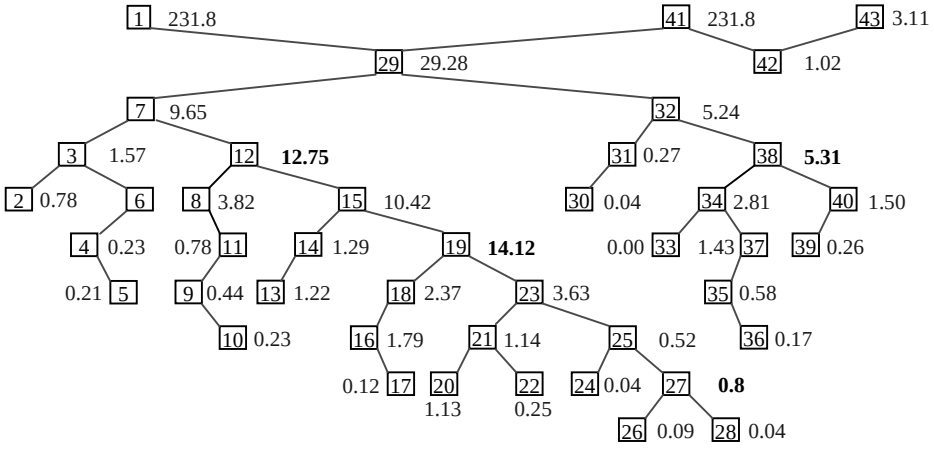

Fig. 4 The initial forest-shape BVT-tree

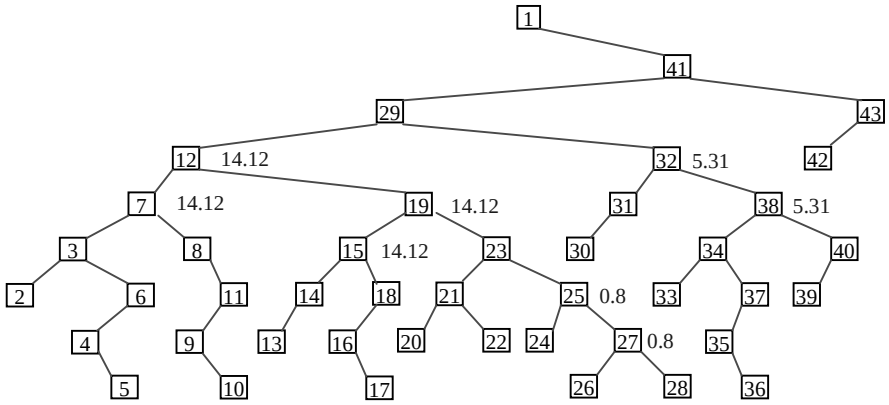

Fig. 5 BVT-tree after tolerance promotion and re-shaping

3.3.4 Linking vertex tolerance to query scale

In our case, the requirement to link generalisation parameters with actual query scales is to map a query scale value to the tolerance value of vertices in BVT-tree in order to retrieve data at the required level of detail from a multi-scale database.

First, we define a default screen resolution R^b_{scr} which represents the finest resolution on a presentation medium (VDU, paper, etc.). This value is decided with reference to the process of rasterising vector objects, i.e. the threshold when a new point inserted into a line segment will not generate new pixels. This threshold is an approximate or average one due to the impact of the orientation of line segments although we may calibrate the tolerance values of vertices in some way with orientation of segments taken into consideration. Indeed, the process of rasterisation may serve as a metaphor to illustrate the geometric meaning of the vertex tolerance used in the BVT-tree.

Given the physical size of the query window *DE* and the corresponding field extent *ME*, the query scale s_q is *DE/ME* and we can map it to a tolerance value $R_m = R^b_{scr} / s_q$. Obviously, by setting a different screen resolution value, we can retrieve data at different detail levels for the same s_q.

Since cartographical scale values may be represented reciprocally by resolution values (Scale*Resolution = *Constant*) which change linearly and have the same metric as the BVT tolerance's and at the same time the tolerance value has a relatively clear geometric and cartographical meaning, it is not very difficult to establish a simple and direct mapping between scale values and tolerance values used in the BVT-tree. It may not be the case for other methods, particularly when multiple parameters are used.

3.4 Other Issues Relevant to Calculating Scale Ranges for Vertices

The BVT-tree method presented above is in its basic form, which does not consider the distance between a vertex and other vertices or line segments in the same geometry or other neighbouring geometries. This distance may be smaller than the tolerance value assigned to it. This problem may be solved by reducing the vertex's tolerance value (an equivalent to deletion) or by displacing it. Such conflicts may also be a sign of needing to change the type of the object (e.g. a simple polyline becomes a network) which will result in creating a new multi-scale geometry. For conflicts between different geometries, deletion or displacement may also provide solutions. Alternatively, a selection operation may be needed to choose some of the objects while others become recessive at this scale and beyond. Generally speaking, there may not be a "best" solution to these conflicts but rather several equal alternatives that may be chosen according to different generalisation philosophies and for various purposes.

The BVT-tree is presented here only as a simple example to demonstrate the principal requirements of methods for generating object-space decomposed multi-scale spatial datasets in a DBMS environment: 1) to select an indicator which conforms to the scale-based retrieval precedence of vertices in geometries and 2) to define a one-to-one mapping between the indicator space and scale space. From a

practical point of view we do not even suggest it is a good method in its current form. We believe a practical method for generating multi-scale spatial datasets should first meet the above requirements and also address the relevant issues discussed in this sub-section, which might be the combination of several generalisation operations.

4. Map Generalisation and Presentation of Results of Querying Multi-Scale Database

4.1 Conflicts in the presentation of query results from a multi-scale spatial database

In the foreseeable future, users who make queries on a multi-scale spatial database will still want the information they required to be presented on a VDU or as hard copies which are in a form comparable to traditional paper maps with limited presentation space. Therefore, besides normal graphic conflicts caused by varying types and sizes of map symbols specified by users, a multi-scale spatial database with the information-rich architecture proposed in this paper could also generate conflicts when objects which are originally recessive are retrieved, as the inherently incompatible nature of these objects makes it impossible to solve these conflicts at the pre-process stage. There is, however, no space in this paper to bring this issue further.

For methods to solve either of the two types of conflict, the first priority is performance as these processes are carried out on-line during interactive query sessions. Some previous methods for conflict resolution (e.g. [13, 14]) have severe performance overheads for on-line use but significant progress in this regard has been made recently [15, 16] to meet the requirements of on-line application.

4.2 A Shape-Conserving Line Displacement Method

In this sub-section we present a method for on-line linear feature displacement which preserves the shape characteristics of the object.

Fig. 6(a) shows the original polyline, displacement vector (v_0_v_{0N}) and the displaced polyline. We use the notation $v_i(x, y)$ and $v_{iN}(x, y)$ to refer to a vertex and its displaced position in the original coordinate system and $v'_i(x', y')$ and $v'_{iN}(x', y')$ for their transformed positions in a new coordinate system (Fig. 6(b)). Assuming the displacement vector v_0_v_{0N} is applied at vertex v_0, scan lines will be derived from v_{0N} to find the nearest (in sequence, not in distance) points of tangency v_L and v_R. Then vertices between v_L and v_R (inclusive) will be transformed to a local coordinate system with v'_0 as the origin and v'_0_v'_{0N} as the positive Y' direction. In this local coordinate system, the displacement of v'_0 is $d'_0 = v'_{0N}.y' - v'_0.y'$. This displacement will be evenly distributed through the section of the polyline v'_0_v'_L and v'_0_v'_R. Two parameters are used to control the distribution to make sure the curvature of the sections will not be changed too much. One parameter is the compression rate $RC \le (v'_{LN}.y' - v'_{0N}.y') / (v'_L.y' - v'_0.y') < 1$ (if $v'_L.y' > v'_0.y'$, as the section will be "compressed"); the other is the stretch rate $RS \ge (v'_{LN}.y' - v'_{0N}.y') / (v'_L.y' - v'_0.y') > 1$ (if $v'_L.y' < v'_0.y'$, as the section will be "stretched").

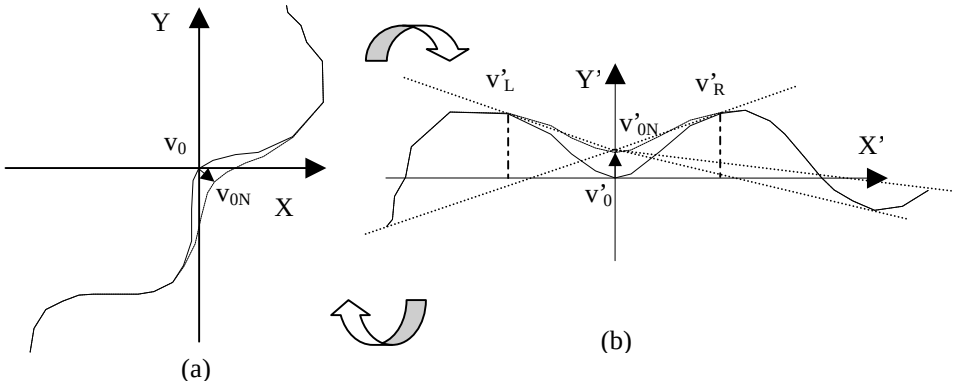

Fig. 6 Shape preserving line displacement - I (sectioning at points of tangency)

Using v'_L as an example, if $(v'_L.y' - v'_{0N}.y') / (v'_L.y' - v'_0.y') \ge RC$, d'_L(displacement of v'_L) is 0 and v'_L will not be displaced; otherwise:

$$d'_L = v'_{LN}.y' - v'_L.y' = (v'_L.y' - v'_0.y') * RC + (v'_{0N}.y' - v'_L.y')$$

This value then becomes the original displacement value for the next section with v'_L as the right section point. In general, to meet the *RC* criterion, d'_i (the displacement on Y' for a vertex v'_i between v'_L and v'_0), is:

$$d'_i = v'_{iN}.y' - v'_i.y' = (d'_0-d'_L)*(v'_i.x' - v'_L.x')/(v'_0.x' - v'_L.x') + d'_L$$

which will be d'_L when $v'_i.x' = v'_L.x'$ and d'_0 when $v'_i.x' = v'_0.x'$.

The formulas for stretch cases are similar. d'_L is 0 if $(v'_L.y' - v'_{0N}.y') / (v'_L.y' - v'_0.y') \leq SR$ and otherwise $d'_L = v'_{LN}.y' - v'_L.y' = (v'_L.y' - v'_0.y') * RS + (v'_{0N}.y' - v'_L.y')$ and for an arbitrary vertex v'_i, we also have:

$$d'_i = v'_{iN}.y' - v'_i.y' = (d'_0-d'_L)*(v'_i.x' - v'_L.x')/(v'_0.x' - v'_L.x') + d'_L$$

A more computation-efficient alternative to sectioning with points of tangency is to use local maximum/minimum points in the new coordinate system for sectioning object (Fig. 7).

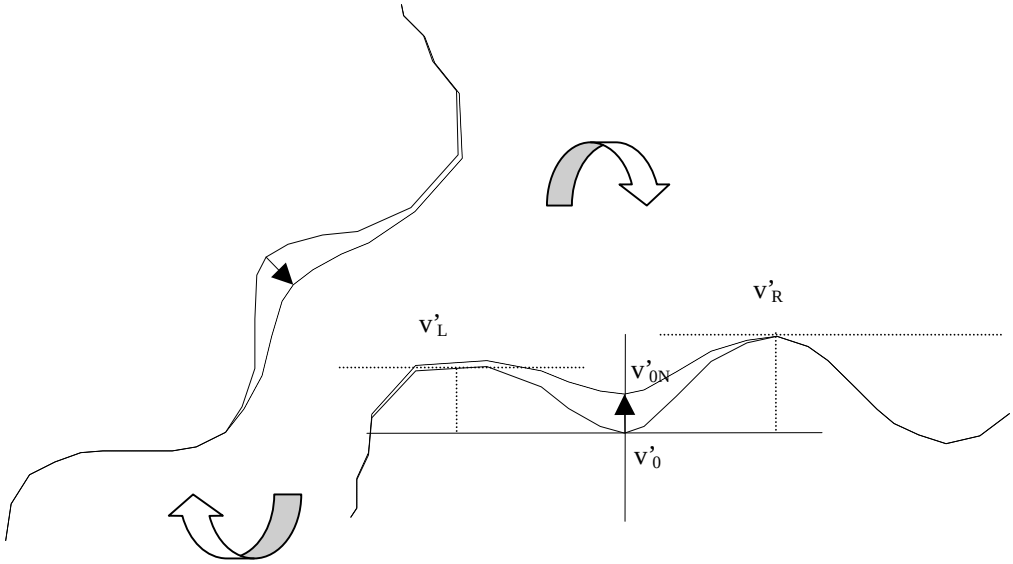

Fig. 7 Shape preserving line displacement - II (sectioning at local points of extreme)

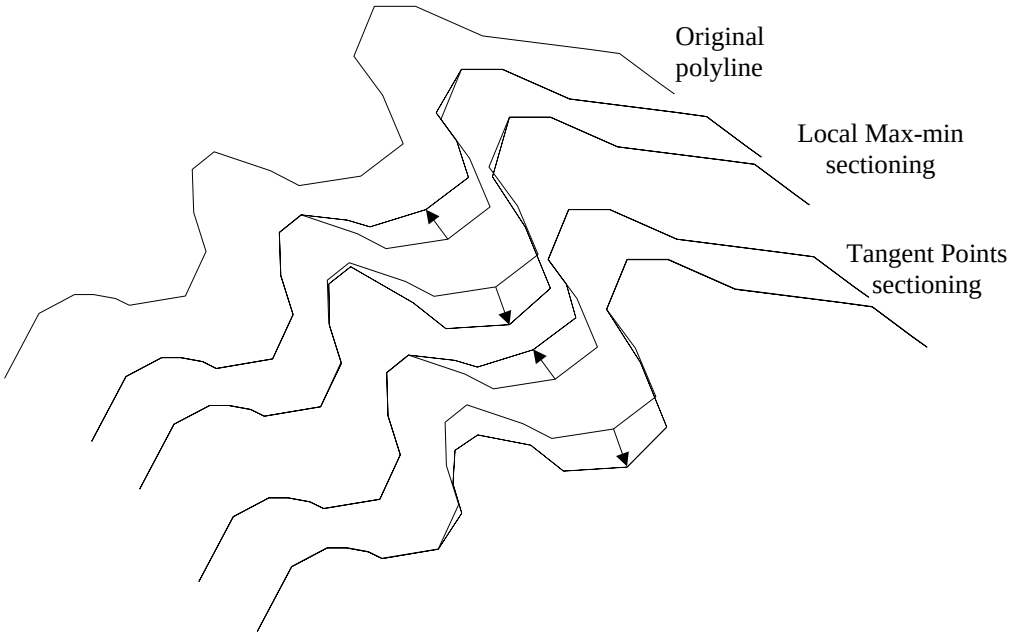

Fig. 8 Comparison of tangent point sectioning and max-min point sectioning

After a displacement is distributed to one or more sections on each side of the point of impact (v'_0), the vertices whose coordinate values have been modified accordingly will be transformed back to the

original coordinate system. Fig. 8 shows some more examples of the methods (currently implemented with AutoLISP in AutoCAD).

The main advantage of the methods presented here is that all basic curves in an object are preserved. In addition, the use of the two control parameters *RC* and *RS* prevents large changes of curvature of basic curves. Impacts may be restricted inside a relative local area. As only affected vertices will be processed, these methods (especially the one using local max-min vertices for sectioning) are potentially very efficient and capable of working in an on-line environment.

From our experiment (Fig. 8), we find that results generated by these methods are quite sensitive to the characteristics of objects. It seems that the combination of the tangent point based and local max-min point based methods may generate better results in many situations.

At present, we use fixed values for the two control parameters in our experiment. For real applications, these values may be decided adaptively, e.g. taking the X' offset of a section (i.e. $v'_0.x' - v'_L.x'$) into consideration. In addition, the issue of handling propagated conflicts has not been addressed here as the prime objectives of designing these methods are shape preservation and running efficiency. Indeed, we believe it is not difficult to link these methods to other existing methods for conflict detection to form a complete solution for real applications.

5. Multi-scale Geometry Object Storage Schemes in Brief

Based on the multi-scale data model presented in section 2, we have designed several storage schemes for simple multi-scale geometry objects and implemented them in an extendible object-relational DBMS *Informix* using C++ and *Informix*'s Object Interface for C++ v.2.6 as DBMS API.

5.1 Storage Schemes

Four storage schemes are presented here. For each scheme (except the multi-version scheme), there are two potential implementations: 1) a multi-scale geometry may be stored in a BLOB object as a single entity or alternatively 2) a large geometry may be divided into several segments represented by a user defined "opaque" type in *Informix* and each segment may be stored directly in a database row (record).

5.1.1 Multi-Version scheme (MV)

This scheme stores a complete representation for each scale interval si_i in a scale range $sr = \{si_i | i = 1, n\}$. Basically this scheme is for datasets generated on the basis of key-space decompositions as too many versions would have to be stored for a dataset with object-space decomposition. For example, the polyline in Fig. 3 has 31 different tolerance values and therefore under the MV scheme 31 versions have to be stored in order to support all these values.

5.1.2 Sequence number based schemes(SN)

Under this scheme, an explicit sequence number is assigned to each vertex to label its position in the original vertex sequence in the geometry. In a BLOB object, all vertices in a geometric object are stored according to their tolerance/scale value in a descending (in case of tolerance) order. There may be an index which includes some or all the tolerance/scale values stored in front of the vertex data section. For the segment based method, the above sorted vertex sequence is divided into several blocks according to a maximum size for blocks (number of vertices, etc.) and labelled with the largest tolerance value of vertices in the block.

When vertices are retrieved, they are sorted by their sequence number to restore the correct vertex order in the original sequence. This scheme may be suitable to both the key-space decomposed and object-space decomposed datasets.

5.1.3 MS-Tree based schemes(MS)

This scheme is primarily for key-space decomposed datasets. All vertices from the same scale/tolerance interval and between two vertices at a larger tolerance interval form a vertex group. The order of

vertices is maintained by references between a vertex and the two vertex groups in front of and behind it and at smaller tolerance intervals. For example, given a tolerance range {[0, 1), [1, 5), [5. 10), [10, 20),[20, ∞)}, the object in Fig. 3 will form the following MS-Tree (Fig. 9):

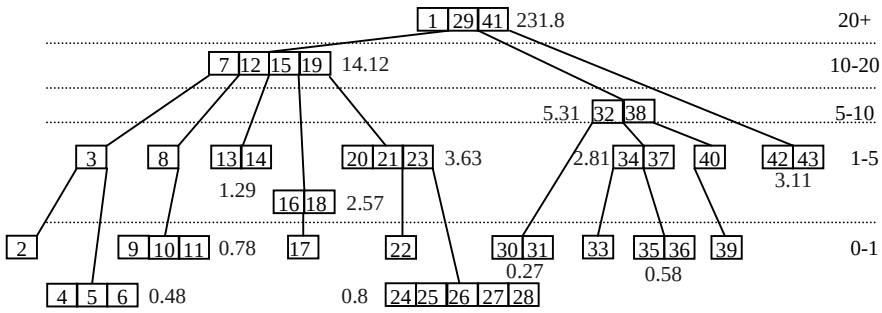

Fig. 9: The MS-Tree of the object in Fig. 3

In a BLOB object, these vertex groups will be stored in a descending order of the maximum group tolerance value. References to child groups are physical offset values of the group in the BLOB object. For an segment based implementation, several groups at the same level may be stored into a single vertex block. Therefore, references will also include information on a group's position inside the block (in our implementation, we stored references in child groups so the position of the parent vertex in its group should also be stored).

If the MS-Tree is used to store object-space decomposed datasets, the structure of the tree will normally be very similar to a persistent BVT-tree introduced below.

5.1.4 Persistent BVT-tree (BVT)

Under this scheme, vertices of a geometry are sorted in descending order according to their tolerance value and then stored in a BLOB object. References to child nodes are physical offset values of the child nodes in the BLOB object. For the segment based implementation, a BVT-tree may be divided into several sub-trees of a specified maximum height (e.g. for a height of 2, a sub-tree has a maximum 3 nodes) and each sub-tree is stored as a vertex block. The regular nature of a binary tree means it is easier to map the position of a node at the bottom level of a sub-tree to its child sub-trees.

5.2 Results

Due to the limited space, only some observations made on our experiment results are shown here. Some detailed results on MV and SN schemes can be found in [2].

The MV scheme, understandably, provides a slightly better performance in data retrieval and object reconstruction. However, we observed severe storage overhead when handling real datasets (Fig. 10).

BLOB based MS-Tree and BVT-Tree schemes have a slightly better performance in comparison to then SN scheme when object size is relatively large (which means it takes longer to sort vertices on their sequence numbers under the SN schemes). For opaque-type implementation, the SN scheme outperforms other alternatives. In addition, the opaque-type segment implementation seems to be faster than BLOB based implementation as it is an expensive operation to open and access BLOB objects. In summary, we believe an implementation which combines all available schemes when appropriate may achieve best overall performance.

6. Future work

An important issue not thoroughly discussed in this paper is intra-object and inter-object conflict detection and resolution when a multi-scale dataset is generated or a query result is presented. We proposed an information-rich architecture for a multi-scale spatial database which allows multiple objects occupying the same location at a given scale but we have not discussed the issue of designing priority systems for multiple properties which depend on specific map specifications or geo-referenced

data classification systems. At present, our plan is to use multi-agent based methods along with triangulation methods to tackle these problems.

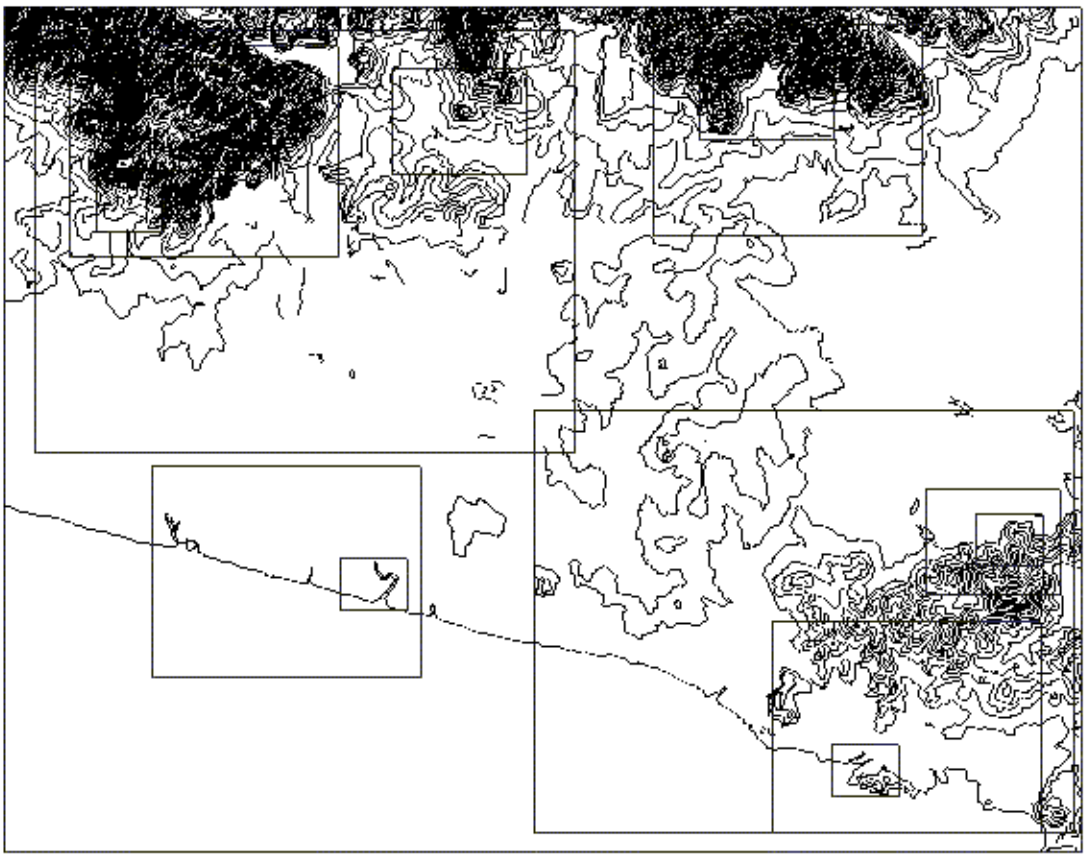

Fig. 10 A real dataset (1:5000) used in experiments (smaller rectangles are query windows used)

Biographical Sketch

S. Zhou has mainly worked on developing methods to generate multi-scale dataset in a DBMS-centred environment.

C.B. Jones has worked on several aspects of map generalisation with a focus on a) triangulations to support topologically consistent implementation of operators such as amalgamation, collapse, line simplification and displacement; b) graphical conflict resolution applying iterative improvement procedures such as simulated annealing and c)multi-agent systems for interactive maps.

Bibliography

1. Guting, R.H., *An introduction to spatial database systems.* VLDB Journal, 1994, **3**(4), p. 357-399.
2. Zhou, S. and C.B. Jones, *Design and implementation of Multi-Scale Databases*. in *SSTD'01*, to appear, Los Angles.
3. Puppo, E. and G. Dettori, *Towards a formal model for multiresolution spatial maps*, in *Advances in Spatial Databases*, M.J. Egenhofer and J.R. Herring, Editors. 1995, Springer-Verlag, p. 152-169.
4. Becker, B., H.-W. Six, and P. Widmayer, *Spatial priority search: an access technique for scaleless maps.* ACM SIGMOD Record, 1991, **20**(2), p. 128-137.
5. Jones, C.B. and I.M. Abraham, *Design considerations for a scale-independent database*. in *Second International Symposium on Spatial Data Handling*. 1986, Seattle, International Geographical Union, 384-398.
6. van Oosterom, P., *Reactive Data Structures for Geographic Information Systems*. 1993, Oxford: Oxford University Press.

7. Worboys, M.F., *A generic model for planar geographical objects.* International Journal of Geographical Information Systems, 1992, **6**(5), p. 353-372.
8. Egenhofer, M.J., E. Clementini, and P.Di Felice, *Topological relations between regions with holes.* International Journal of Geographical Information Systems, 1994, **8**(2), p. 129-142.
9. Open GIS Consortium, *OpenGIS Simple Features Specification*, 1998, Open GIS Consortium (OGC).
10. Shaffer, C.A., *A Practical Introduction to Data Structures and Algorithm Analysis*. 1997: Prentice Hall.
11. Cromley, R.G., *Hierarchical Methods of Line Simplificatoin.* Cartography and Geographic Information Systems, 1991, **18**(2), p. 125-131.
12. Gunther, O., *Efficient Structures for Geometric Data Management*. Lecture Notes in Computer Science. Vol. 337. 1988: Springer-Verlag.
13. Ware, J.M. and C.B. Jones, *Conflict Reduction in Map Generalisation Using Iterative Improvement.* Geoinformatica, 1998, **2**(4), p. 383-407.
14. Harrie, L.E., *The Constraint Method for Solving Spatial Conflicts in Cartographic Generalization.* Cartography and Geographic Information Science, 1999, **26**(1), p. 55-69.
15. Ware, J.M., C.B. Jones, and N. Thomas, *Map Generalisation, Object Displacement and Simulated Annealing: Two Techniques for Executiion Time Improvement*. in *Proceedings of the GIS Research UK 9th Conference GISRUK 2001*. 2001. Wales, UK: University of Glamorgan.
16. Harrie, L. and T. Sarjakoski, *Simultaneous Graphic Generalisation of Vector Data Sets.* GeoInformatica, Submitted.