Workshop of the ICA Commission on Map Generalisation and Multiple Representation – June 25th 2006

Vario-scale data server in a web service context

Peter van Oosterom, Marian de Vries, and Martijn Meijers

Delft University of Technology, OTB, Section GIS-technology,
Jaffalaan 9, 2628 BX Delft, The Netherlands.
Phone: +31 15-2786950, Fax +31 15 2782745.
Email: P.J.M.vanOosterom@tudelft.nl, M.d.Vries@otb.tudelft.nl, and martijn@zw9.nl

KEYWORDS: map generalization, topological structure, planar partition, client/server, progressive data transfer

1. Introduction

This paper presents the first implementation of a topological vario-scale data structure: the topological GAP structure (Generalized Area Partitioning), or tGAP structure for short. Purpose of this structure is to store the data only once, with no redundancy of the geometry, and derive different representations of this same data on-the-fly according to the level of detail needed. Last year at the AutoCarto conference and published in CaGIS (van Oosterom, 2005), the theory behind this structure was presented. In this paper the lessons learnt from the implementation are described, including some modifications to the earlier described theory and some new ideas (which still need to be investigated further). The focus is on the implementation of the topological GAP-structure at the server side in an Oracle Spatial DBMS environment. The second main contribution of this paper is the exploration of how this vario-scale data server can be put to effective use in a Web service/client environment. This will be based on the OGC and ISO standards (WFS and GML) and some extensions to these protocols are proposed. The focus is on two aspects: how the tGAP-structure can support progressive transfer of vector data from Web service to client, and how adaptive zooming can be realized, preferably in small steps (‘smooth’ zooming).

1.1 Vario-Scale

Data structures supporting variable scale data sets are still very rare. There are a number of data structures available for multi-scale databases based on multiple representations (MRDB’s), that is, data to be used for a fixed number of scale (or resolution) intervals. These multiple representations data structures try to explicitly relate the objects at the different scale levels, in order to offer consistency during the use of the data. Drawbacks of the multiple representations data structures are that they do store redundant data (same coordinates, originating from the same source) and that they support only a limited number of scale intervals. By far most data structures are intended to be used during the pan and zoom (in and out) operations of a user, and in that sense multi-scale data structures are already a serious improvement for interactive use as they do speed-up interaction and give reasonable representations for a given level of detail (scale).

1.2 Progressive transfer

Another drawback of the multiple representation data structures is that they are not suitable for progressive data transfer as each scale interval requires its own (independent) graphic representation to be transferred. Nice examples of progressive data transfer are raster images, which are first presented relatively fast in a coarse manner and then refined when the user waits a little longer. These raster structures could be based on simple (raster data pyramid) or more advanced (wavelet compression) principles. For example, JPEG2000 (wavelet based) allows both compression and progressive data transfer from the server to the end-user. Also, some of the propriety formats such as ECW from ER Mapper and MrSID from LizardTech are very efficient raster compression formats based on wavelets and offering multi-resolution suitable for progressive data transfer. Similar effects

are more difficult to obtain with vector data and require more advanced data structures, though recently a number of attempts are made (Bertolotto and Egenhofer 2001, Buttenfield 2002, Jones et al., 2000, Zhou et al., 2004).

1.3 Paper overview

In section 2 the most important concepts of the Generalized Area Partitioning theory are shortly introduced. A first implementation of a vario-scale data server based on these concepts form theory is given in section 3 together with some adjustments and suggestions for improvements. Section 4 then puts the vario-scale data server in the context of a web-service/client setting and discusses the communication protocols needed to support vario-scale and progressive transfer. Finally, the main conclusions, some additional ideas and future work can be found in Section 5.

2. Background Generalized Area Partitioning

It will not be attempted to describe the tGAP (topological Generalized Area Partitioning) structure in sufficient detail to completely understand it. For this, the reader is referred to the publication in CaGIS (van Oosterom, 2005). In this section the most important components of the tGAP structure will be shortly introduced: the GAP-face tree, the GAP-edge forest, the BLG-tree, and the Reactive-tree (or the 3D R-tree as Pseudo Reactive-tree). The first proposal of a tree data structure for a generalized area partitioning (GAP-tree) was by van Oosterom (1993). The idea was based on first drawing the larger and more important polygons (area objects), which then corresponds to a generalized representation. However, one can continue by refining the scene through the additional drawing of the smaller and less important polygons on top of the existing polygons (based on the Painters algorithm). This drawing order fits in the concept of progressive transfer: a first rough image is created, which is then refined further and further. If one keeps track of which polygon refines which other polygon, then the result is a tree structure as a refining polygon completely falls within one parent polygon and there is one single root polygon (covering the whole domain). The tree structure is built by thinking the other way around: starting with the most detailed representation, find the least important object (child) and assign this to the most compatible neighbour (parent). This process is then repeated until only one single polygon remains, the root; see Figure 1 (top).

Drawbacks of this original GAP-tree are: 1. redundancy as boundaries between neighbour polygons are stored twice at a given scale (and redundancy of the boundaries shared between scales) and 2. boundaries are always drawn at the highest detail level (all points are used, even if it would not be meaningful for a given scale). Therefore, the tGAP structure was developed: a topological structure based on faces and edges but using the same principle of the original GAP-tree to cope with the vario-scale requirements: rough area descriptions are refined by more detailed ones. The parent-child relationships between the faces again form a tree structure with a single root: the GAP-face tree. Also the parent-child relationships between edges at the different scale (detail) level form tree structures, but now with multiple roots; therefore this is called the GAP-edge forest. In order to take care of the line simplification, the edges are not stored as simple polylines, but as Binary Line Generalization trees; BLG-trees (van Oosterom, 1990). When two edges are combined to form one edge at a smaller scale (more important), then their BLG-trees are linked (but no redundant geometry is stored).

Finally, the entries of the GAP-face tree and GAP-edge forest are stored as records (tuples) in DBMS tables. In order to efficiently select the requested faces and edges for a given region (rectangle) and scale (importance) a specific index structure is proposed: the Reactive-tree (van Oosterom, 1992). As this type of index is not available in most systems, an alternative is to use a 3D R-tree to index the data: the first two dimensions are used for the spatial domain and the third dimension is used for the scale (or more precisely for the importance level as this is called in the context of the GAP structure). Together, the GAP-face tree, the GAP-edge forest, the BLG-trees and the Reactive-tree, are called the tGAP structure. The tGAP structure can be used in two different ways (see Figure 1): 1. to produce a representation at an arbitrary scale (a single map) or 2. to produce a range of representations from rough to detailed representation. Both ways of using the tGAP structure are useful, but it will be clear that in the context of progressive transfer (and smooth zooming) the second way must be applied.

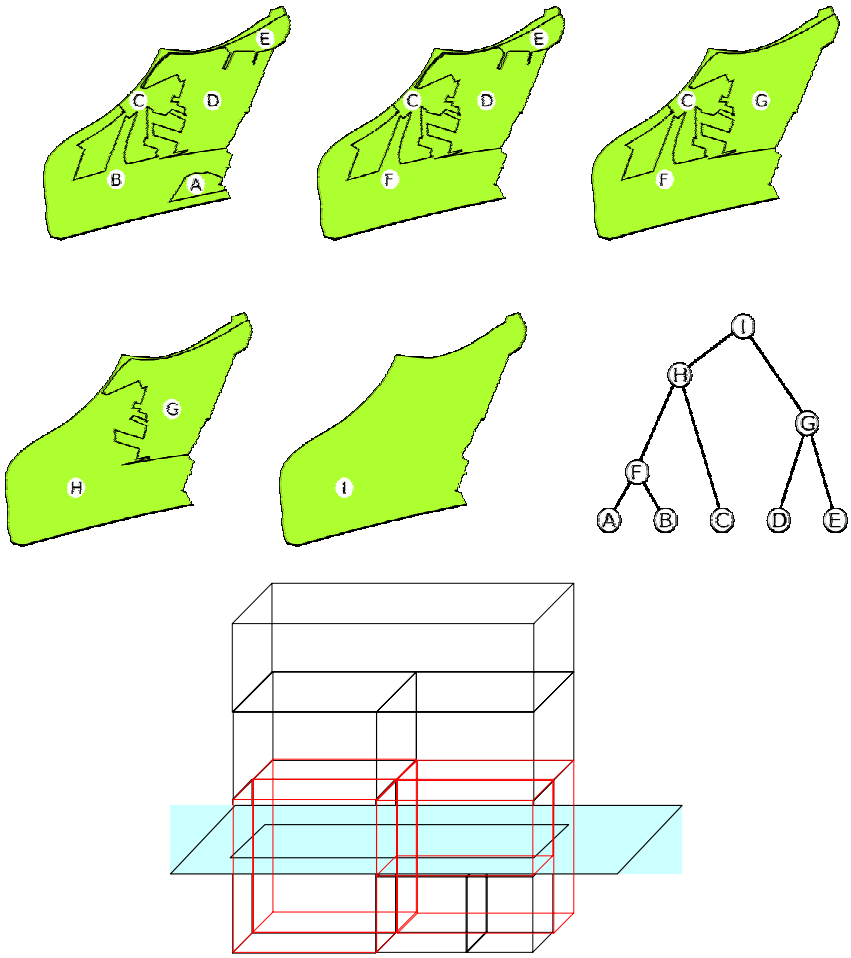

Figure 1. Top: set of area features (A, B, C, D, E) and their series of merges, which finally results in one root area feature (I) and the whole process is reflected in the GAP-face tree. Bottom: Importance levels schematically represented by the third dimension (at the most detailed level, bottom, there are several objects while at the most coarse level, top, there is only one object); The hatched plane represents a requested level of detail and the intersection with the symbolic '3D volumes' then gives the faces. Taken from (Vermeij et al., 2003).

3. First implementation

The server side of the implementation platform consists of the Oracle spatial DBMS and uses facilities such as the extensibility to add new data types (BLG-tree as new type), PL/SQL to develop own functions within the DBMS (e.g. return_polygon of face), and the spatial data types and indices. Figure 2 shows the UML class diagram of the implemented tGAP structure. Remember that the BLG table can contain both lowest level edges (with true BLG-tree with points), but also merged edges (with references to two sub BLG-trees and a associated tolerance value for the new root). The main changes in implementation (Meijers, 2006) compared to the theory of the tGAP structure (van Oosterom, 2005) are:

1. Besides a face and edge table, also a node table was introduced, so there is no redundancy in end points of edges; this also fits well to the BLG-tree structure (which stores the intermediate points in the tree).
2. Only the left/right edge-face references are stored and not the winged-edge (edge-edge) references, because this takes less storage space (less references per edge to store, but also less versions of a single edge), is easier to build and does not slow down performance during use.
3. As it is convenient to know the direction of the parts in a merged edge (join of BLG-tree) the references to the BLG sub-trees are signed ('-' when direction has to be reversed as member of the merged edge).

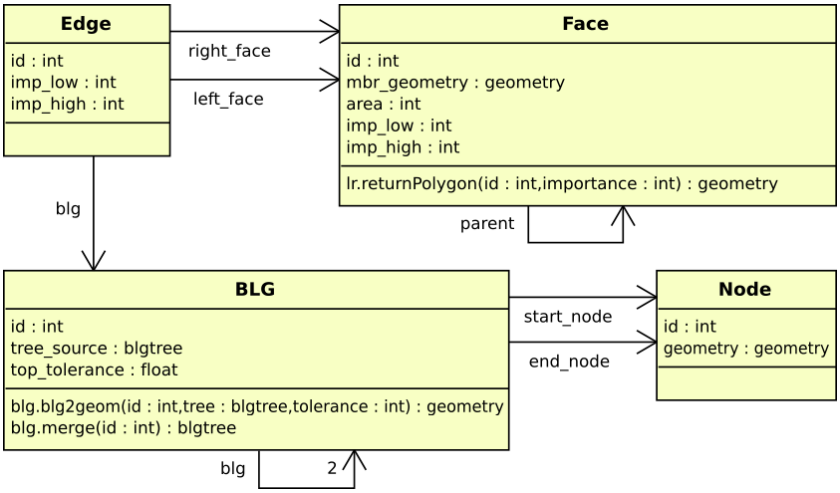

Figure 2. UML class diagram showing the GAP face tree and GAP edge forest

Appendix B1 contains the SQL table descriptions of the tGAP structure tables. Loading some real data sets and building the tGAP structure, it was possible to investigate the storage requirements (for test data sets from a few hundred to a few hundred thousand faces). These turned out to be about 5 times more than a single-scale topological structure (table descriptions in Appendix B2). Note that in itself our topology structure is mean and lean: just a node, edge and face table. Our structure is different from the standard topology structure in Oracle, which requires more storage space; as is the case for LaserScan Radius topology. This is due to the fact that there is no separation between geometry, topology and feature tables, which cause quite a bit of additional storage in Oracle topology and LaserScan Radius topology. For example, the Oracle topology table structure requires about 3 times more storages space compared to just storing plain polygons (Penninga, 2004). Our basic topology structure requires 107 Mb, while plain polygon storage (just one table with id and one sdo_geometry for the polygons) requires 82 Mb. So, just a factor 1,3 (instead of a factor 3).

structure	*#face/Mb*	*#edge/Mb*	*#blg/Mb*	*#node/Mb*	*Total Mb*
Basic topology	170.368/2	418.530/94	-/0	281.216/11	107
tGAP structure	340.735/56	7.113.680/291	658.219/133	281.216/11	491

Table 1. Number of rows and size in Mb in tGAP strucure (compared to basic topology)

Nevertheless, 5 times more storage (and not yet considering indices) than the single-scale basic topology tables is a lot, and therefore further analysis is needed; see Table 1. The number of faces in the tGAP structure is exactly 2 times the number of faces in the basic topology minus 1 (due to the property of the binary GAP-face tree). The number of BLG records is about 1.5 times the number of edges (at basic topology), which is reasonable if one considers that the BLG records represent all levels of details. The number of nodes in both structures is equal (as can be expected). Below a number of suggestion to the reduction of storage space:

1. the tgap_face table could have less attributes: area and mbr may be computed when needed (and parent_id may not really be necessary);
2. the tgap_edge table has an explosion in the number of rows (17 times more than edge table in basic topology structure), the majority of these additional rows are due to edge versions with only different left or right faces. Further, the imp_low of one version is equal to the imp_high of its predecessor. Instead of storing separate rows for every version it is proposed to store one row for every every and use varray's to store the varying attributes (left, right, and imp_high);
3. (a) the tgap_blg table size may be reduced by a more lean implementation of the BLG-tree data type (as used in the attribute tree_source in the BLG table). Currently, every node has explicit left/right references, but this could be replaced by an implicit tree structure (which is more

compact). (b) More storage space can be saved by not having a BLG-tree at the bottom level: just store polylines and compute generalization when needed. At the higher level the BLG-tree concept is still present (after merging two edges);

4. the tgap_blg table does contain both leaf BLG records (with attributes blg_id and tree_source filled and other attributes empty) and non-leaf BLG records (with attributes blg_id, child1_id, child2_id and top_tolerance filled and other attributes empty). All these empty attributes do waist some storage space and therefore an alternative with two types of BLG tables and a view (tgap_blg_orininal for leafs, tgap_blg_joined for non-leafs, and tgap_blg view for their union) implementation is more efficient;
5. the tgap_blg table does not have to store start_node_id and end_node_id as these can be obtained from the corresponding edge table records (less distinct id's 614.707 edges versus 658.219 blgs)'and top_node is added to tgap_blg to refer to new top of joined BLG-trees.

Appendix B.3 shows the proposed new SQL tables in the tGAP structure. It is expected that these improvements will result in a tGAP structure size only 2 or 3 times the size of the basic topology structure. This seams very reasonable if one considers that the tGAP structure does support all possible scales (importance levels). Future work related to this implementation might be to compare the storage requirements of the tGAP structure in some realistic MRDB scenarios. A Pseudo Reactive tree was used (3D R-tree). As no new physical attributes (such as mbr) should be added to the tables, a functional index in Oracle was used. For this purpose it was needed to compute the 3D bbox (with a function) to merge the 2D bbox and the 1D importance range as the functional index expects one parameter (3D bbox in this case).

Though the main focus was on the server side development, it is also nice to see some resulting maps. Therefore at the client side Google Earth was used as platform, because of its user-friendly interface. The middleware to connect the server and client consists of the Apache web server, the Python programming language and the Geospatial Data Abstraction Library (GDAL with bindings for Python). Apache, the web server software, takes care of processing the requests from Google Earth (made over the HyperText Transfer Protocol, HTTP) and the answers are sent back in KML (short for Keyhole Markup Language, an XML language). Figure 3 shows the tGAP data visualized in Google Earth, at 4 different zoom levels. Appendix A shows 6 different levels zoom levels, but now also with the result of BLG-tree on the right side. Depending on the amount of zooming, a specific (but arbitrary) scale (importance value) is requested from the server and all polygons are sent to client. The server returns complete polygons from the tGAP structure. The function 'return_polygon' needs an additional parameter, besides face_id, i.e. the BLG-tree tolerance in order to produce the right level of detail. Note that there is no topology structure and no progressive transfer in these Google Earth tests. For that purpose we need another type of Web client/service set-up (see section 4).

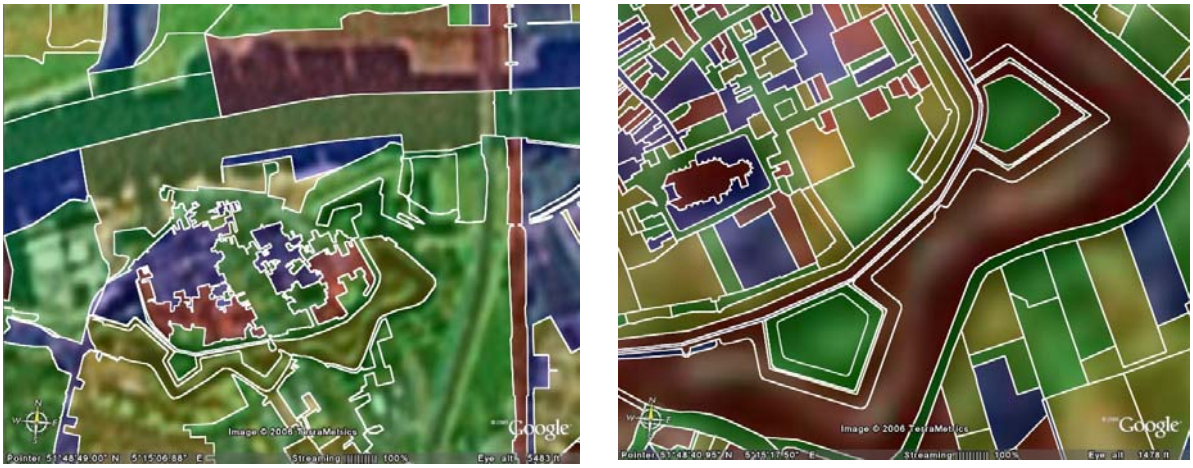

Figure 3. The tGAP data structure in action: more detail when zooming in (Google Earth client) (white lines indicate boundaries of the area features from the data server).

4. Server-client set-up and progressive refinement

Progressive refinement of data being received by a client could be implemented in the following way. The server starts by sending the most important nodes in tGAP-structure (including top levels of associated edge BLG-trees) in a certain search rectangle. The client builds a partial copy of tGAP-structure, which can then be used to display the coarse impression of the data. Every (x) second(s) this structure is displayed and the polygons are shown at the then available resolution on the screen. The server keeps on sending more data and the tGAP-structure at the client side is growing (and the next time it is displayed with more detail). Several stop criteria can be imagined: a. 1000 objects (meaningful information density), b. required importance level is reached (with associated error tolerance value), or c. the user interrupts the client.

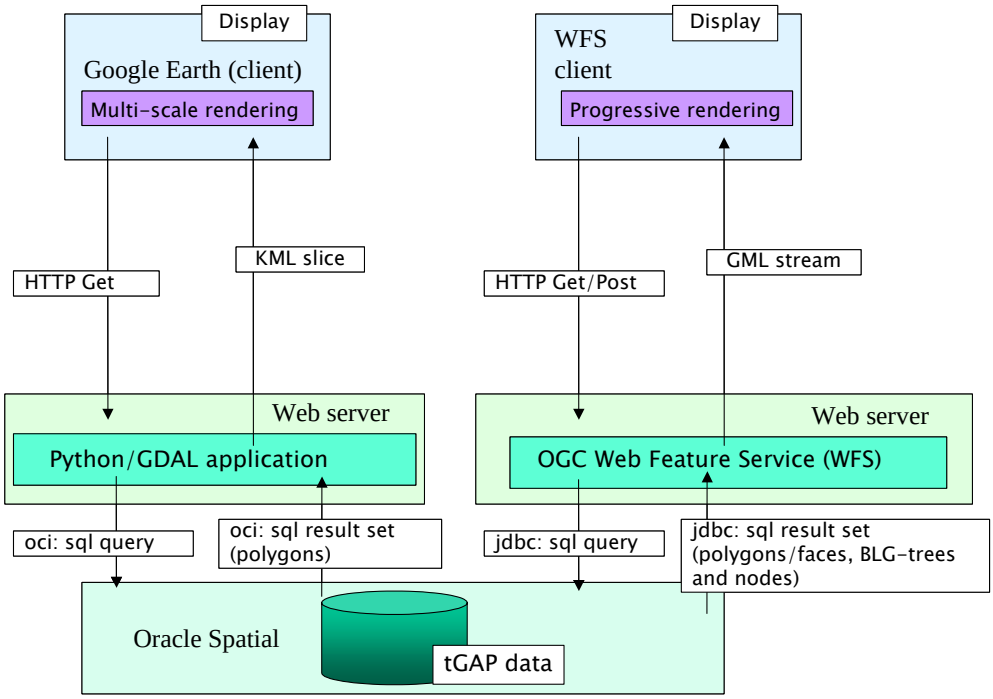

Figure 4. The two Web service/client implementations

In this section we will look in some detail how the tGAP-structure can be used in a Web service/client setting for progressive transfer and for refinement during zooming. In addition to this, we wanted to use more standardized interfaces, such as Web Feature Service (WFS) and Geography Markup Language (GML) instead of the Google Earth specific solutions. As shown in Figure 4 (right side),

the WFS service acts as middle layer between Web client and geo-database. So we have to establish two things: 1. what queries are necessary from WFS to database to retrieve the vario-scale data, enabling progressive transfer and smooth zooming later on (section 4.1) and 2. what does this mean for the requests and responses between Web client and WFS service (section 4.2). We will start with these two questions. In the last paragraph of this section (4.3) we will shortly discuss the requirements of progressive transfer and refinement for the software used for geo-database, WFS server and WFS client.

4.1 Queries to the database

In this section again, the transfer of complete polygons from client to server is discussed (that is, no topology at the client side). From a functional perspective there are two situations in the Web service/client interaction: 1. the initial request to the WFS to get features for the first time during the session; and 2. additional requests when zooming in. When the Web client requests the data at one specific importance level (e.g. 101) the query to the database would be something like this:

```
select face_id as id, '101' as impLevel, RETURN_POLYGON(face_id, 101) as geom
from tgap_face
where imp_low <= 101 and 101 < imp_high;
```

This query will retrieve polygons from the database (constructed dynamically by the RETURN_POLYGON function). The result is one ‘slice’ of the data set at a specific importance level. The disadvantage is that all objects have to be visualized at once (to get a complete map without holes), progressive transfer is not possible and the GAP-tree structure is not ‘rebuilt’ at the client for smooth zooming later on. The next query will also retrieve polygons from the database, but now not one slice, but a range of objects from the most important level until a certain importance level (e.g. 90), sorted in order of importance (descending from high to low):

```
select face_id as id, imp_high-1 as impLevel,
  imp_low, imp_high, RETURN_POLYGON(face_id,imp_high-1) as geom
from tgap_face
where imp_high > 90
order by imp_high desc;
```

Note no upper boundary of the required importance is specified, only a lower boundary of ‘90’. This means that everything starting above 90 and up to the root importance will be selected. When the WFS service receives the results from the data layer the data is already in the right importance order (from high importance to low importance) and can be passed to the client (as GML) in that same order. When the client receives the GML there are three possibilities that differ with respect to the moment that the data is visualized as map:

a. rendering in small steps. The client software already starts visualizing parts of the incoming GML data before the whole data stream is received. Here we have an example of ‘true’ progressive transfer. The objects are received and visualized in sets of two objects which replace one parent at a time (see section 3, binary GAP-face tree). The map is progressively rendered in small steps. The visualization is very smooth: the user first sees the contours and then slowly the details are filled in.

b. visualize the incoming GML in two or more larger steps, refreshing the complete spatial extent that is displayed in the map after a certain time interval (of 2 seconds for example). In this case the client will at given times visualize the level of the latest objects it has received. Here also the data is in the right importance order so that progressive refinement/ generalization in the later stages is possible, only the refresh is done for the whole area.

c. no progressive transfer and rendering (the common situation with current WFS client software): the client waits until the complete data stream is received and then visualizes the data at the appropriate level all at once. The data is still vario-scale and in order of importance, so the basis for smooth zooming during user interaction is also there.

Not using the topology (at client side) in the above scenarios is a limiting factor. When the client receives server-side constructed polygons and there is no line simplification, the Painters algorithm

works fine: it takes care of hiding the coarser objects when the more detailed objects are received. However, in the case of line generalization the Painters algorithm does not suffice because with line simplification the shape of the derived polygons will change, so the more detailed lines will not always 'hide' the coarser lines. This means that in case of line simplification the approach will only work when the topology-to-polygon reconstruction is carried out in the Web client and not in the database. The GAP-edges (and blg's and nodes) with their geometry will now be streamed to the client; the polylines can already be visualized during retrieval, so there is progressive transfer, but instead of the Painters algorithm other methods are needed to hide the previously received, coarser lines.

In case of a WFS service (providing vector data) zooming and panning by the user can often be handled in the client itself, without having to send new GetFeature requests to the server. But when only a part of the objects for that spatial extent is already in the client, new requests might be necessary: for example when zooming in, more detailed data could be needed. With panning beyond the original spatial extent the situation is a bit more complicated because it depends on the characteristics of the new spatial extent (the density of objects there) what needs to happen.

When an additional GetFeature request is not necessary, zooming in will mean hiding coarser objects and making visible more detailed ones. In the case of zooming out this process is the opposite: now the more detailed objects have to be 'switched' off, and the coarser objects will be made visible. Important is that all of this is handled in the client. Depending on the type of client this hiding/displaying could be based on event handling. When the GML data is visualized using SVG (Scalable Vector Graphics) for example, the 'onzoom' event of the SVG DOM can be used.

An important issue for further research is, how to calculate the right importance range for the 'new' set of objects to be displayed. The algorithm will have to be a function of the size of the map window (in pixels), the zooming factor for that particular zoom action, and some kind of optimal number of objects for that map size (Töpfer and Pillewizer 1966). When a new GetFeature request is necessary, the scenario is partly the same as in the previous case: first the right importance range for the new objects has to be calculated. The second step is then to request the new objects (only the ones that are not yet already in the client). Finding out whether or not a new request is necessary implies that the client software should not only keep track of the spatial extent of the already received data, but also of the importance range(s) of these objects.

4.2 Extensions to OGC/ISO standards?

Are extensions of the existing OGC WFS protocol necessary for the progressive transfer and refinement scenarios described in the previous paragraph? The first option is to use the existing GetFeature request and specify the importance range (imp_low, imp_high) as selection criteria in the Filter part of the request. And using the ogc:SortBy clause that is available since WFS version 1.1 the client can instruct the WFS service to return the objects in order of importance.

Still this is not an ideal solution. Somehow the Web service has to communicate to the client that it supports progressive transfer and refinement. The self-describing nature of the OGC WFS GetCapabilities response is an important part of creating interoperable service/client solutions. For the WFS protocol this means that somewhere in the GetCapabilities document it must be stated that this particular WFS server can send the objects sorted in order of importance. Another addition to the WFS Capabilities content is reporting the available importance range (imp_low – imp_high) of each feature type should be given, comparable to the way the maximum spatial extent (in lat/long) of each feature type is given in the GetCapabilities response. For this reason (to supply solid service metadata that clearly state the capabilities of the service) it is better to add a new request type to the WFS protocol, for example with the name GetFeatureByImportance. Just like there is besides the Basic WFS also a WFS-T (Transactional WFS) with extra requests for editing via a Web service, we would then have a WFS-R (Progressive Refinement WFS) with an extra GetFeatureByImportance request and two parameters minImp and maxImp to specify the importance range of the features to be selected. When no value is specified for these parameters, all features are requested, but still in order of importance from high to low. And when minImp = maxImp exactly that importance level is

requested, as a 'slice' of the data set (first query in section 4.1). The HTTP Post request would then look like this (the 'D' in the ogc:SortBy is for 'descending'):

```
<wfs:GetFeatureByImportance service="WFS" version="1.0.0" outputFormat="GML2" ...>
<wfs:Query typeName='gdmc:tgap_face' minImp='50' maxImp='150'>
  <ogc:Filter>
      <ogc:BBOX>
        <ogc:PropertyName>geom</ogc:PropertyName>
        <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#28992">
           <gml:coordinates>136931,416574 139382,418904</gml:coordinates>
        </gml:Box>
      </ogc:BBOX>
  </ogc:Filter>
  <ogc:SortBy>gdmc:imp_high D</ogc:SortBy
</wfs:Query>
</wfs:GetFeatureByImportance>
```

4.3 Implications for server and client software

In order to accomplish the progressive refinement when the geo-data is retrieved from the WFS-R service an 'order by' expression (or functional equivalent) has to be included in the WFS query to the data source. Necessary requirement for the data storage system is therefore that the WFS service can retrieve the geo-objects from the data source in a sorted way. A 'Progressive Refinement' WFS-R service, that serves data in order of importance and in a certain importance range, does not need large extensions to WFS software. What is extra in the WFS layer is: 1. adding a 'order by' to the queries sent to the data source and 2. adding a importance selection to the Filter conditions (either spatial or non-spatial) that the user already has specified in the request to the WFS.

5. Conclusion

Main results: This is the first time that the implementation of a non-redundant (with respect to geometry) variable scale data structure has been presented. The previous versions of the GAP-tree all did have some geometry redundancy (between the polygons at a given scale and/or between the scales). The key to this solution was applying a full topological structure. Though this is (far) more complicated than topological structures designed for the traditional single-scale data sets, this implementation proved that the structure is realizable and does perform well in practice. The tGAP structure is very well suited for a Web environment: the client requirements are relatively low (almost no geometric processing of the data at the client side) and progressive refinement of polygon data is supported (allowing quick feedback to the user and smooth zooming). This paper did also illustrate the functioning of a vario-scale structure in a Web service context. The support of progressive transfer is very important (and quite unique for vector data). Existing OGC standards are proposed to be extended in order to support retrieval and visualization of vario-scale geo-data.

***Future work*:** Crucial for the quality of the GAP-tree generalization remains the importance value of the involved feature classes (and importance function) and the compatibility values between two different feature classes (and compatibility function). For sure more research is needed in this area to automatically obtain good generalization results for real world data. Related to this aspect is finding the relationship between the importance value and the actual scale.

One could argue that generalization should be application driven and does depend on the task of the user and a given tGAP structure does only represent one type of use. However, with modest adjustment it should be possible to build more than one tGAP structure on top of a single data set. So different types of generalization are supported without redundant geometry between the scales or different applications. One could compare this to a single table in the database with multiple indices defined: e.g. one b-tree index on the id and on r-tree index on the spatial attribute.

In the current implementation the different edges of narrow features may cross after line generalization (see 4th and 5th figure on the right side in Appendix A). During the construction of the tGAP strucure more attention has to be paid to avoid these situations.

Instead of receiving polygons at the client side, also tests should be done with clients receiving a streamed tGAP structure (mix of ordered face, edge, blg and node records).

The current tGAP structure has only been applied to static data (both in theory and practice). In reality the data is dynamic and the tGAP structure should adjust itself when the source data is changing (comparable to the manner an index changes when the source data is changed in the table). A difficulty is the 'defining condition' that the least important object must be removed (and merged with the most compatible neighbor). Such a global criterion could in theory influence the whole tGAP structure when a small edit is made; e.g. an even less important object (than the current minimum) is created. This global criterion of the least important object is not useful, and an alternative criterion should be defined (allowing local updates of the tGAP structure).

Acknowledgements
This publication is the result of the research program 'Sustainable Urban Areas' (SUA) carried out by Delft University of Technology. This specific research is part of the Dutch Space for Geo-information project RGI-233 'Usable and well scaled mobile maps'. Special thanks to Wilko Quak for discussing the various tGAP structure aspects (both theory and practice) and for being the supervisor of Martijn Meijers during his MSc thesis project.

References

Bertolotto, M. and Egenhofer, M.J. (2001), Progressive Transmission of Vector Map Data over the World Wide Web. *GeoInformatica* 5 (4): 345-373.

Buttenfield, B.P. (2002), Transmitting Vector Geospatial Data across the Internet. In *Proceedings GIScience* 2002, Egenhofer, M.J. and Mark, D.M (eds.). Berlin: Springer Verlag, Lecture Notes in Computer Science 2478: pages 51-64.

Jones, C. B., Abdelmoty, A.I., Lonergan, M.E., Van Der Poorten, P.M. and Zhou, S. (2000), Multi-Scale Spatial Database Design for Online Generalisation. In *Proceedings, 9th International Symposium on Spatial Data Handling*, sec. 7b, 34-44.

Meijers, B.M. (2006), *Implementation and testing of variable scale topological data structures - Experiences with the GAP-face tree and GAP-edge forest.* TU Delft, MSc Geomatics thesis, June 2006

van Oosterom, P. (1990), *Reactive Data Structures for Geographic Information Systems*. PhD-thesis Department of Computer Science, Leiden University.

van Oosterom, P. (1992), A Storage Structure for a Multi-Scale Database: The Reactive-tree. *International Journal, Computers, Environment and Urban Systems*, 16(3): 239-247.

van Oosterom, P. (1993), The GAP-tree, an approach to "On-the- Fly" Map Generalization of an Area Partitioning. GISDATA Specialist Meeting on Generalization, Compienge, France, 15-19 December 1993. Chapter 9 in: *GIS and Generalization, Methodology and Practice*. Editors J.C. Müller, J.P. Lagrange and R. Weibel. Taylor & Francis, London, pages 120-132.

van Oosterom, P. (2005), Variable-scale Topological Data Structures Suitable for Progressive Data Transfer: The GAP-face Tree and GAP-edge Forest, *Cartography and Geographic Information Science*, 32 (4): 331-346.

Penninga, F. (2004), Oracle 10g Topology; Testing Oracle 10g Topology using cadastral data, GISt Report No. 26, Delft, 2004, 48 p.

Töpfer, F. and Pillewizer, W. (1966) The Principles of Selection. *Cartographic Journal*, 3: 10-16.

Vermeij, M., van Oosterom, P., Quak, W., and Tijssen, T. (2003), Storing and using scale-less topological data efficiently in a client-server DBMS environment, In the *proceedings of the 7th International Conference on GeoComputation*, University of Southampton, Southampton, UK 8-10 September 2003.

Zhou, X., Prasher, S., Sun, S. and Xu, K. (2004), Multiresolution Spatial Databases: Making Web-based Spatial Applications Faster. In *proceedings The Sixth Asia Pacific Web Conference* (APWeb'04), 14-17 April, 2004, Hangzhou, China, Lecture Notes in Computer Science 3007, Editors: Jeffrey Xu Yu, Xuemin Lin, Hongjun Lu, et al., pp. 36-47.

Appendix A: tGAP structure

(left: without BLG, right with exaggerated BLG)


Zaltbommel
Google


Zaltbommel
Google


Google


Google


Google


Google

Appendix B: Table definitions

B.1 tGAP structure

```
sql> desc tgap_face;
 name                                      null?    type
 ----------------------------------------- -------- ---------------------
 face_id                                            number
 mbr_geometry                                       mdsys.sdo_geometry
 area                                               number
 imp_low                                            number
 imp_high                                           number
 parent_id                                          number

sql>  desc tgap_edge;
 name                                      null?    type
 ----------------------------------------- -------- ---------------------
 edge_id                                            number
 left_face_id                                       number
 right_face_id                                      number
 imp_low                                            number
 imp_high                                           number
 blg_id                                             number

sql> desc tgap_blg;
 name                                      null?    type
 ----------------------------------------- -------- ---------------------
 blg_id                                    not null number(11)
 start_node_id                                      number(11)
 end_node_id                                        number(11)
 child1_id                                          number(11)
 child2_id                                          number(11)
 tree_source                                        blgtree
 top_tolerance                                      float(126)

sql> desc tgap_node
 name                                      null?    type
 ----------------------------------------- -------- ---------------------
 node_id                                            number
 geometry                                           mdsys.sdo_geometry
```

B.2 topology structure

```
sql> desc face;
 name                                      null?    type
 ----------------------------------------- -------- ---------------------
 face_id                                   not null number

sql> desc edge;
 name                                      null?    type
 ----------------------------------------- -------- ---------------------
 edge_id                                   not null number
 left_face_id                                       number
 right_face_id                                      number
 start_node_id                                      number
 end_node_id                                        number
 geometry                                           mdsys.sdo_geometry

sql> desc node;
 name                                      null?    type
 ----------------------------------------- -------- ---------------------
 node_id                                   not null number
 geometry                                           mdsys.sdo_geometry
```

B.3 Improved tGAP structure proposal

```
sql> desc tgap_face;
 name                                      null?    type
 ----------------------------------------- -------- ---------------------
 face_id                                            number
 imp_low                                            number
 imp_high                                           number

sql>  desc tgap_edge; /* note: all versions of edge in single record */
 name                                      null?    type
 ----------------------------------------- -------- ---------------------
 edge_id                                            number
 imp_low                                            number
 imp_highs                                          varray(number)
 start_node_id                                      number
 end_node_id                                        number
 left_face_ids                                      varray(number)
 right_face_ids                                     varray(number)
 blg_id                                             number

sql> desc tgap_blg_original;
 name                                      null?    type
 ----------------------------------------- -------- ---------------------
 blg_id                                    not null number(11)
 tree_source                                        blgtree

sql> desc tgap_blg_joided;
 name                                      null?    type
 ----------------------------------------- -------- ---------------------
 blg_id                                    not null number(11)
 child1_id                                          number(11)
 child2_id                                          number(11)
 top_node                                           number(11)
 top_tolerance                                      float(126)

sql> desc tgap_node;
 name                                      null?    type
 ----------------------------------------- --------
 node_id                                            number
 geometry                                           mdsys.sdo_geometry

sql> create view tgap_blg as (select
            blg_id,
            tree_source,
            null as child1_id,
            null as child2_id,
            null as top_node,
            null as top_tolerance
        from tgap_blg_original)
    union all (select
            blg_id,
            null as tree_source,
            child1_id,
            child2_id,
            top_node
            top_tolerance
        from tgap_blg_joined);
```