Real-Time Data Generalisation and Integration using JAVA

Lars Harrie and Mikael Johansson, National Land Survey of Sweden

To present real-time maps on a computer or on a mobile device requires data integration and generalisation in real-time. This paper describes a Java program that performs some simple generalisation methods. The program is based on open source products that conform to the Open GIS Consortium (OGC) standards, and contains robust implementations of the most fundamental geometrical algorithms. In the paper a minor case study is presented.

Introduction

Maps are important tools for visualising geographic locations. Traditionally, printed paper maps have been used, but technological developments have made possible the production of digital maps. Furthermore, the introduction of the Internet and mobile technology has made it possible for a computer (or a mobile device) to communicate with remote databases. This way cartographic data from a remote database, distributed in real-time, can be displayed locally. In this paper we denote this kind of map as a *real-time map*.

A real-time map has both advantages and disadvantages compared to a printed paper map. One obvious disadvantage of the real-time map is the small size of the display, which makes it difficult to get a good overview of a larger area. The main advantages of a real-time map are the possibility to always use the most updated data source and the possibility to integrate the map into a service. Examples of services where maps are valuable are navigational services, Yellow Pages and emergency services.

Today, most of the cartographic data that are distributed via the Internet and to mobile devices are raster data, but the emerging XML standards will make it easier to distribute vector data. The use of vector data has two basic advantages. Firstly, a map stored in vector format generally requires less storage space than a map stored as raster data. This also means that transmission times are shorter for vector data. Secondly, it is easier to integrate and generalise vector data than raster data. These integration and generalisation operations are important when creating the map components of user-friendly services.

This paper describes methods for manipulating cartographic vector data for creating real-time maps. It focuses particularly on a technical environment for integrating and generalising vector data. The paper starts with a description of the system architecture for distributing cartographic data from a database to an end user. This is followed by a description of a Java program for manipulating the vector data and a case study of this program. The paper concludes with a discussion.

System architecture

The system architecture described here has been developed in the EEC-project GiMoDig (GiMoDig, 2003). The objective of the GiMoDig project is to develop and test methods for delivering geospatial data to a mobile user by means of real-time data-integration and generalisation. The project aims at the creation of a seamless data service providing access, through a common interface, to the primary topographic geo-databases maintained by the National Mapping Agencies (NMAs). A special emphasis will be put on using cartographic visualization appropriate for mobile terminal with limited display capabilities. GiMoDig is not primarily a project for creating services, the aim is rather to build an infrastructure which other partiescan use for building services.

The GiMoDig system architecture is only briefly described in this paper (see GiMoDig, 2003 and references within for further details). Furthermore, only the data flow from the database to the end user is presented (an end user is, in this context, a person that is using a service built on the GiMoDig infrastructure).

The end user request in the GiMoDig project, follows the standards *Web Map Service* (WMS) and *Web Feature Service* (WFS) from Open GIS Consortium (OGC, 2003).

The distribution of cartographic data is performed using two XML standards: *Geographic Markup Language* (GML; OGC, 2003) and *Scaleable Vector Graphics* (SVG; W3C, 2003). These two standards are complementary. GML is used for storing and distributing geographical data. The GML standard supports storage of object attribute information about geodetic reference systems etc., but it does not support storage of symbolisation. SVG is a general standard for presenting vector data on the Internet (and there are special dialects for use in mobile devices). The format supports symbolisation but not typical geographical data, such as attributes and information about geodetic reference systems.

In this study, the cartographic data are distributed from the database to the end user as shown in Figure 1. A request from the end user is sent to the database and then the cartographic data is distributed from the database as GML files. The generalisation and integration of the cartographic data is performed in a Java program and a new GML file is generated. This GML file is then translated into an SVG-file in an XSLT-transformation. Finally, the end user can browse through the SVG file in his computer or mobile device.

There are several variants of this workflow. One possibility would be to create an SVG file directly in the Java-program. This means that an XSLT transformation is not necessary. A second possibility would be to manipulate the cartographic data in the XSLT transformation (with some Java extensions); which means that there would not be a need for the Java program. The latter approach has successfully been used by Lehto and Kilpeläinen (2000, 2001a, 2001b). However, this approach has certain limitations. Since XSLT transformations only treat one object at a time, it is not possible to implement methods which involve interactions between objects. This type of interaction modelling is often required when creating a real-time map for a service. Some examples where modelling interactions of objects are required:

- Solving spatial conflicts between objects (or more correctly, the symbols that represent these objects). These spatial conflicts do often occur, for example, when building symbols are exaggerated (to be readable) and therefore infringe on neighbouring road symbols.
- Integrating "service data" and cartographic data. For example navigational data in the form of arrows are added to the map in a navigation service. It is here important that the arrows do not hide important cartographic data.
- Aggregating objects. For example, aggregating buildings into built-up areas (cf. Figure 3).

As the main theme of this paper is the Java program for manipulating the cartographic data, the XSLT transformation step is not further described here. Details of how this step is used in GiMoDig system architecture can be found in Lehto and Kilpeläinen (2001a, 2001b).

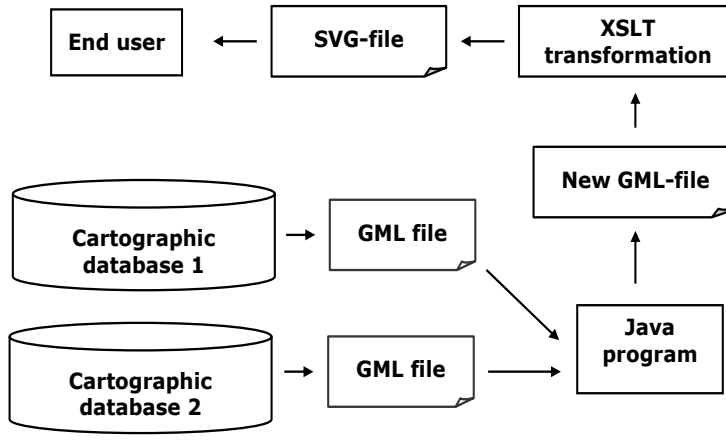

Fig. 1. A schematic view of the distribution of cartographic vector data from a database to a user.

A Java program for generalisation and integration of vector data

This section is devoted to the Java program for generalising and integrating cartographic data (cf. the architecture in Figure 1). The program consists of six packages, of which some are freeware and others have been written within the GiMoDig project. Furthermore, some packages are general, whilst others are data dependent (such as object type classes). The following six packages are included:

Java Topology Suite

Java Topology Suite (JTS; Vivid Solutions, 2003) is a geometry and topology class library. JTS was chosen as environment for generalisation and integration mainly for the following three reasons. Like GML, JTS conforms to the *Simple Features Specification for SQL* (OGC, 2003). This means that the basic geometrical entities are the same in the Java environment as in GML and that there is no need for geometrical transformations during the import of the data into the Java environment. The second reason is that JTS contains robust implementations of the most fundamental geometrical algorithms (in 2D). Thirdly, JTS is open source and free to use and modify in research.

Abstract feature classes according to OGC-standards

In the OGC standards, the general structure of cartographic objects is stated (in the standard the objects are denoted *features*). This package contains abstract Java classes that implement this structure. The geometries of these abstract feature classes are stored in JTS classes and linked using associations.

Object type classes

This package contains one Java object class for each object type. The object classes inherit the general structure from the abstract feature classes. Clearly, this package is data dependent; the classes are dependent on the conceptual model of the cartographic data. This differs from the two packages above, which are data generic.

Generalisation and integration classes

The classes in this package govern the generalisation and integration processes. They contain both the conceptual framework of the process (the framework for triggering the methods) and the actual implementations of the generalisation and integration methods. Since these methods are partly dependent on the cartographic data, this package has to be modified for the data that it is applied to.

GML reader classes

This package contains translation classes for data stored in a GMLfile; the classes are based on a free parser (Xerces) from Apache (2003). In principal, our parser is a modified version of the SAX-parser described in McLaughlin (2000).

Viewer

A viewer (see Figure 2) based on standard Java API Swing (Sun, 2003). The viewer is only used for development work.

The case study

The case study described below deals with presenting a real-time map on a small-display for personal navigation. Since the display is small, it puts high demands on the selection of the cartographic data that is to be shown. This becomes problematic when the user requires a considerable amount of cartographic information. In personal navigation, users often need both a detailed map of the area surrounding the user's current position as well as an overview map. In cartographic terms, this means that the user requires both large-scale and small-scale cartographic data. One possible way to solve this problem is to use a variable-scale map.

The variable-scale map used in this case study is based on Harrie et al. (2002). This type of variable-scale map has a circular cap where the scale is homogeneous and beyond which the radial scale constantly decreases to a threshold value (Figures 2 and 3). The mapping function is conformal in the centre of the map (which, normally, should be the user's location) but not in all parts of the map.

As seen in Figure 3, the central part of the map is shown with a larger scale than the parts

Fig. 2. The original data from the City of Malmö, Sweden, shown in the Java viewer. © The local municipalities in Skåne and the National Land Survey of Sweden, 2002.

Fig. 3: A variable-scale map of the same area as in Figure 2. Some of the building objects in the western part of the map were aggregated to built-up areas.
© The local municipalities in Skåne and the National Land Survey of Sweden, 2002.

closer to the map borders. The selection and level of detail of the cartographic data is appropriate in the central part but, otherwise, the map is too detailed. In other words, cartographic generalisation is required for outer parts of the map.

The building objects in the areas towards the edges of Figure 3 are not discernible. To improve the readability of the map, aggregation of the building objects into built-up area objects was performed in the western part of the map. The aggregation method was based on convex hull (which is a method in JTS). Unfortunately, we are currently lacking tools for performing this generalisation fully automatically. The reason is that no topological relationships are stored in the GML file. This implies that it is cumbersome to write a program that creates the neighbourhood partitions. In this case study we have created the neighbourhoods manually in advance by digitising. However, the next version of GML (3) supports storage of topological relationships and then it will be fairly easy to create a fully automatic building aggregation function.

As seen in the southern part of the map in Figure 3 the variable-scale mapping function introduced topological errors between the road objects and the building objects (the symbols overlap). These types of topological errors can occur even though the variable-scale mapping is continuous. The reason is that a line segment is only described by its end points in the mapping. To circumvent this problem additional points should be added on long line segments.

Discussion

Currently, we have only implemented few generalisation and integration methods within the Java program. To make the program useful, more methods need to be implemented. The structure of the program is built so that the program easily should be extendable. We would like to include methods for:

- simplifying and aggregating building objects (e.g. methods by Regnauld, 1996),
- solving spatial conflicts and simplifying objects by using least squares methods (Harrie and Sarjakoski, 2002; Sester, 2000), and
- treating data stored in a multiple representation database (a database which consists of different data sets connected by links between objects representing the same physical entities; see for example. Buttenfield, 1993; Kilpeläinen, 1997; Harrie and Hellström, 1999).

Several of these methods require a spatial data structure based on constrained Delaunay triangulation. Such a data structure is implemented in the Java program using code from Shewchuk (1996), but the triangulation has not yet been utilised.

Real-time maps are critical for time response. In this paper we have not dealt with computational or storage complexity as the focus of the paper is on the presentation of cartographic data on a small display. In this case, the cartographic objects that are distributed are relatively few and, therefore, the fact that GML and SVG generate large files is not too problematic nor is the computational complexity of the parsing and generalisation.

Acknowledgements
The research described in the paper is part of the GiMoDig project, IST-2000-30090, which is funded from the European Union via the Information Society Technologies (IST) programme (GiMoDig, 2003). We would like to thank Lassi Lehto (Finnish Geodetic Institute) for the idea of using JTS, Lars-Håkan Bengtsson (National Land Survey of Sweden) for data transformations, Ian Brook for correcting our English, the editors for constructive comments, and Tiina Sarjakoski (Finnish Geodetic Institute) and Monica Sester (University of Hanover) for co-operation. Test data for Malmö, Sweden, were kindly provided by the local municipalities in Skåne and the National Land Survey of Sweden.

References

Apache, (2003). *Web site of Apache XML Project*, xml.apache.org (accessed 21 January 2003).

Buttenfield, B. P., (1993). *Research Initiative 3: Multiple Representations*, Closing report, National Center for Geographic Information and Analysis, NCGIA, Buffalo.

GiMoDig, (2003). *Geospatial info-mobility service by real-time data-integration and generalisation*, http://gimodig.fgi.fi/ (accessed 21 January 2003).

Harrie, L., and A.-K. Hellström, (1999). A Prototype System for Propagating Updates between Cartographic Data Sets. *The Cartographic Journal*, Vol. 36, No 2. pp. 133-140.

Harrie, L., and T. Sarjakoski, (2002). Simultaneous Graphic Generalization of Vector Data Sets. *GeoInformatica,* Vol. 6, No. 3, pp. 233-261.

Harrie, L., Sarjakoski, L. T. and L. Lehto, (2002). A Mapping Function for Variable-Scale Maps in Small-Display Cartography. *Journal of Geospatial Engineering*, Vol. 2, No. 3, pp. 111-123.

Kilpeläinen, T., (1997). Multiple Representation and Generalization of Geo-Databases for Topographic Maps. *Publications of the Finnish Geodetic Institute,* No. 124, Doctoral dissertation.

Lehto, L., and T. Kilpeläinen, (2000). Real-Time Generalization of Geodata in the Web. *International Archives of Photogrammetry and Remote Sensing*, Vol. XXXIII, Part B4, Amsterdam, pp. 559-566.

Lehto, L. and T. Kilpeläinen, (2001a). Real-time Generalisation of XML-encoded Spatial Data on the WEB. Kidner, D. B. and G. Higgs, eds., (2001), GIS Research in the UK, *Proceedings of the GIS Research UK, 9th Annual Conference GISRUK 2001,* April 18th – 20th, University of Glamorgan, Wales, pp. 182–184.

Lehto, L. and T. Kilpeläinen, (2001b). Generalizing XML-encoded Spatial Data on the Web. *Proceedings of 20th In-*

ternational Cartographic Conference, August 6–10, 2001, Beijing, China, Volume 4, pp. 2390–2396.

McLaughlin, B., (2000). *Java and XML*, O'Reilly, Cambridge.

OGC, (2003). *OpenGIS®*, http://www.opengis.org (accessed 21 January 2003).

Regnauld, N., (1996). Recognition of Building Clusters for Generalization. *Proceedings of the 7th Spatial Data Handling Symposium*, Delft, the Netherlands, pp. 185-198.

Sester, M., (2000). Generalization Based on Least Squares Adjustment. *International Archives of Photogrammetry and Remote Sensing*, Vol. XXXIII, Part B4, Amsterdam, pp. 931-938.

Sun, (2003). *Java Foundation Classes*, http://java.sun.com/products/jfc/ (accessed 21 January 2003).

Shewchuk, J. R., (1996). Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *First Workshop on Applied Computational Geometry,* Philadelphia, Pennsylvania, pp. 124-133.

Vivid Solutions, (2003). *Java Topology Suite*, http://www.vividsolutions.com/jts/jtshome.htm (accessed 21 January 2003).

W3C, (2003). *Scalable Vector Graphics (SVG)*, http://www.w3.org/Graphics/SVG/ (accessed 21 January 2003).

Om forfatterne

Lars Harrie and Mikael Johansson, National Land Survey of Sweden, SE-801 82 Gävle
lars.harrie@lantm.lth.se, micke.j@goteborg.utfors.se