Variable-scale Geo-information

Martijn Meijers

ISBN 978 90 77029 28 2

Typeset by: Martijn Meijers, with the LaTeX 2ε document preparation system.
Printed by: Ridderprint · Ridderkerk, The Netherlands.
Cover design: Martijn Meijers. Location of cover photo: 37°49'46.17"N 25°45'16.42"W.

This PhD thesis is published under the same title in the series:

Publications on Geodesy 77
ISBN 978 90 6132 335 8
ISSN 0165 1706
NCG, Nederlandse Commissie voor Geodesie, Netherlands Geodetic Commission
P. O. Box 5030, 2600 GA Delft, The Netherlands
E-mail: info@ncg.knaw.nl
Website: www.ncg.knaw.nl

The NCG is part of the Royal Netherlands Academy of Arts and Sciences (KNAW).

VARIABLE-SCALE GEO-INFORMATION

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof. ir. K. Ch. A. M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op
vrijdag 16 december 2011 om 12.30 uur
door

Bernard Marinus Meijers

Master of Science in Geomatics
geboren te Delft

Dit proefschrift is goedgekeurd door de promotoren:

Prof. dr. ir. P. J. M. van Oosterom
Prof. dr. M. J. Kraak

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof. dr. ir. P. J. M. van Oosterom	Technische Universiteit Delft, promotor
Prof. dr. M. J. Kraak	Universiteit Twente, promotor
Prof. dr. ir. F. W. Jansen	Technische Universiteit Delft
Prof. dr. M. Menenti	Technische Universiteit Delft
Prof. dr. ir. I. S. Sarıyıldız	Technische Universiteit Delft
Prof. Dr.-Ing. habil. M. Sester	Leibniz Universität Hannover
Prof. Dr. R. Weibel	Universität Zürich

ACKNOWLEDGEMENTS

And then suddenly it has gotten this far: you are writing the acknowledgements section for your PhD thesis in the year in which you become thirty, dad and, apparently, doctor… Some words of thanks are appropriate and although doing a PhD sometimes feels a bit lonely it is definitely not something you can perform alone. For sure I could not have come this far without the help and support of numerous people.

First of all, I like to thank Peter van Oosterom, who gave me the chance to carry out this research at the GIS-technology group. Because of the open and honest atmosphere, our long, interesting and sometimes diverging discussions on different topics, and above all his enthusiasm, I look back upon a wonderful time of 4 years. Years that really have flown by.

Secondly, I like to thank Menno-Jan Kraak, although his role in the project has been less prominent than Peters, we have had some nice discussions with input coming from a different, geo-visualisation angle.

Thirdly, I like to thank all committee members, Erik Jansen, Massimo Menenti, Sevil Sarıyıldız, Monika Sester and Robert Weibel for taking on the task of reviewing my manuscript and giving constructive remarks that helped to improve the manuscript.

Another word of gratitude goes to Rod Thompson, who read the whole thesis and gave numerous comments on how to improve the English of the manuscript. Also thanks to Rien Elling. Thanks to his course on structuring the writing process, this went relatively smooth and he was open for giving some last minute advice.

Furthermore, I feel privileged that my PhD project started as a research being part of the RGI-233 project 'Usable (and well scaled) mobile maps for

consumers', in which TU Delft, ANWB, municipality of Amsterdam, TNO, ITC, Esri, Leibniz Universität Hannover and 1Spatial collaborated. I have fond memories of a great week of collaboration at 1Spatials office, strolling through the old city of Cambridge and staying in the 'Fawlty Towers' hotel. The crowning glory of the project has been the Geo-Innovation Award 2009, category Science, that was awarded to the project.

In this respect, also thanks to Jan-Henrik Haunert and Arta Dilo for the collaboration in an early stage of the research and to Sandro Savino for staying in Delft and being always cheerful, somewhat late and chatty and full of ideas on how to solve specific generalisation problems.

Thanks also to all colleagues of the GIS group. Theo Tijssen helped me starting off my project and always provided a helping hand either by having an open door for discussions or installing the newest software on casagrande. Hugo Ledoux introduced me to the wonderful world of triangulations, which I have not completely unravelled yet. Edward Verbree shared an office with me, which we both seem to prefer without annoying noises. Also thanks to the rest of the group members: João Paulo Hespanha, Tjeu Lemmens, Wilko Quak, Jantien Stoter, Wiebke Tegtmeier, Marian de Vries and Sisi Zlatanova. Elfriede Fendel has been of help organising the printing of this book. Liu Liu and Ken Arroyo Ohori as 'new' PhDs within our group I wish you all the best with your own projects. I hope you experience the same 'wow-factor', when it is time that you have finished your projects.

Thanks to STW and NWO I have the chance to investigate some of the remaining and open problems and hopefully bring the vario-scale concepts to the next level.

Another word of thanks is appropriate for all of my friends, who happened to ask so now and then what the status of my 'boekje' was and who were really interested in the topic. I hope I did not bore you too much with my stories about 'points', 'lines' and 'areas'…

Lastly, I could not have done this project without the support of my family: Jaap and Koos, my parents, it is you that have stimulated me to get the best out of education and have given me all the freedom to become an independent person. Fieke, the final words are for you. No, this paragraph is not 3 pages long, but from voicing over a promo-video or encouraging me to get back to writing when I was on a strict deadline (again), you were always there. *Thank you.*

Martijn Meijers
11th November 2011

TABLE OF CONTENTS

1 SETTING THE SCENE

~

This chapter gives an introduction to the thesis by showing the importance of the problem, the scientific gap and it hints at advantages of the proposed solution (§ 1.1). It puts forward the objective of the research and the research questions, which follow from the objective, in § 1.2. Furthermore it highlights the research scope (§ 1.3), the research methodology, the data sets and tools used for testing and prototype development § 1.4, and gives an overview of how to read the thesis in § 1.5.

1.1 Motivation for maps at variable scale

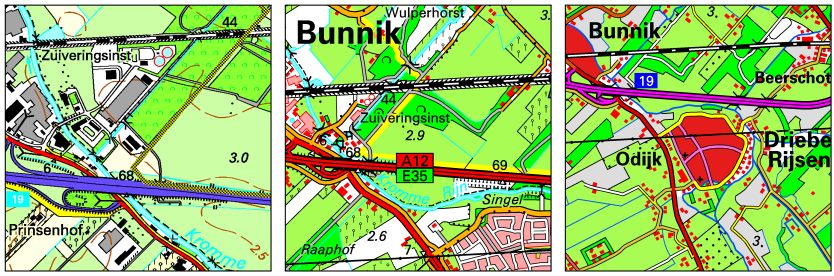
Geographic information is more and more applied in main-stream digital consumer products. These devices, such as personal navigation devices, mobile phones and tablet pc's are getting larger and larger screens – recent tablets, like the Samsung Galaxy Tab, have a screen upto 10.1" providing high resolution graphics – and more and more mobile processing power is available – the recently introduced Apple iPad 2 has a 1GHz dual-core processor. This is an important technology driver for being able to publish interactive maps on those devices. Advanced map user interfaces, making it is easy to navigate seamless 2D maps and aerial imagery, have seen a large uptake since the introduction of the Google maps product in 2005 (*cf.* Figure 1.1). This type of user interface has become a de facto standard for publishing geo-information online and is the basis for applications, such as searching for a new house to buy or rent, sharing your location with friends (using social network services), exploring your newly planned

Figure 1.1: Geo-information is used more and more in a network centric environment, *e. g.* via smartphones accessing the Internet.

holiday location and playing location-based serious games for secondary school education.

These examples also illustrate that geo-information is used more and more in a network centric environment (*i. e.* via the Internet). Mobile phones and tablet pc's can use the Web via WiFi or 3G connections and these possibilities are increasingly used (Jongeneel, 2011). Furthermore, plans to speed up the spread of wireless broadband networks are on the agenda of the European Parliament (European Parliament, 2011). Dissemination of geographic information over such networks offers several advantages over other dissemination techniques (such as distribution by DVD), *e. g.* it is easier for a user to retrieve up-to-date information (by means of incremental updates) and fair-pricing mechanisms can be created, *e. g.* based on pay-per-view (*cf.* van Oosterom, 2001).

The advances in mobile hardware also have changed the way people can interact with the geographic information at hand, compared to 'old-fashioned' paper maps. Users can zoom in or out to the desired level of detail their tasks require. Due to these advanced interaction possibilities, in a digital environment there is not really a need for a *fixed* map scale – the scale of a map is defined as the ratio of a distance on the map to the corresponding distance on the ground – as is the case with paper maps, where the printed, and therefore fixed, portrayal is prepared by professional cartographers and the physical size of the paper limits



(a) 1:25 000 (b) 1:50 000 (c) 1:100 000

Figure 1.2: Dutch topographic map series. Shown are 3 map fragments at different map scales around the city of Bunnik, The Netherlands.

the amount of information to be displayed. Even though a fixed scale is not necessary in a digital environment, it is convenient that a user still has a notion of the map scale, so that he can make a proper translation of distances between and sizes of objects on the map to the real world.

However, current state-of-the-art solutions for storing, maintaining and disseminating digital maps still mimic the analogue map-series concept in the sense that for every map scale in the serie (*e. g.* 1:25K, 1:50K, 1:100K, such as shown in Figure 1.2) a different digital copy with independent data is kept and maintained at the producers site, like national mapping agencies (NMAs) and commercial parties. Because several of these copies are independently stored and maintained, this can cause inconsistencies. With more advanced data management solutions, these problems could be alleviated and improved solutions offered, also to end users.

A step in the right direction for these inconsistency problems are multi-representation databases (MRDBs). According to Hampe et al. (2004) there are 'two main features that characterise an MRDB: 1. different levels of detail (LoD) are stored in one database; 2. the objects in the different levels are linked'. Individual objects are explicitly linked with each other and each object knows its corresponding objects in other representations which helps when updates have to be carried out. Although objects are linked, this is still not the most ideal situation, because data redundancy exists and the fixed levels are still the same as analogue map series: Firstly, map content can not be adjusted to scale accordingly, if the MRDB lacks the appropriate level. Better adjustment of contents could take place if more granular levels were present, but this would lead to more

redundant levels. Secondly, this has effects on using the database for interactive viewing: while zooming in on a geographic region discontinuities ('shocks') will be visible (discrete jumps from one representation intended for a specific map scale to another). Thirdly, graphic representations stored for objects are different, requiring separate geometric descriptions to be transferred, and therefore it is not easy (or even not possible) to send data in a progressive manner (which entails sending a coarse representation first and incrementally updating already sent data with additional details).

An alternative solution is the use of variable-scale data structures. In the context of the net-centric usage scenario an important requirement is that the digital map data is structured in such a way that redundancy for data storage is prevented as much as possible (*i. e.* no duplicate elements are stored), as this has severe impact on the amount of information to be stored and transmitted over the network. An example of a variable-scale data structure is the tGAP data structure (as proposed by van Oosterom, 2005). This data structure consists of two structures: 1. the face tree that captures the merging of areas as the result of generalisation via a parent-child relation in a binary tree and 2. the edge forest that holds the boundary edges of these faces needed for any level of generalisation. Instead of explicitly duplicating the boundaries of areas at a lower level of detail, references are stored to the original boundaries and for these a data structure is provided so that the level of detail can be dynamically adjusted. Data consistency between different map scales is guaranteed. In addition to the geometry and references, an *importance* value for every object is stored and based on this importance value different representations can be derived on-the-fly from the structure according to the needed level of detail. Those structures are the key for representations at arbitrary map-scale and making progressive data transfer possible. It was shown that those structures have great potential for storing geographic information with little redundancy and that it is possible to implement them in a main stream geo-enabled database management system (by the initial implementation of Meijers, 2006). Those structures are still in their infancy (*e. g.* van Oosterom et al. (2006) showed that quite some storage space may be needed), and as they will be used as the starting point for this research, they need to be improved.

Altogether, the challenge of this work is to get to a representation of the real world with continuous level of detail, instead of representations with discretised levels of detail (organised in multiple layers, each layer representing only one resolution level). Advantages of such a model with continuous levels of detail can be threefold:

1. from a geo-data-producers perspective: there is only one integrated model from which maps with lower level of detail can be derived on-the-fly (within relevant and reasonable limits) with fewer inconsistencies than is the case for separately stored resolution levels,

2. for software producers those structures can form the basis for a new generation of technology for publishing 2D vector map data on the Internet (improving the already existing smooth map interfaces) and providing multi- or vario-scale based analysis and algorithms and

3. from an end user point of view it would be possible to benefit from new possibilities, like true smooth zoom and progressive transfer.

Thus, it is crucial to investigate the new technological possibilities that variable-scale data structures can bring to make better digital 2D map solutions possible.

1.2 Objective and Research questions

At this moment it is not exactly known how to create a digital environment that can accommodate geographic information at variable-scale having minimal redundancy. The overall aim of this research is therefore to investigate variable-scale (or vario-scale for short) geo-information, by improving the initial tGAP structures (as described by van Oosterom, 2005). From this objective follows that the main question that we[1] try to answer in this thesis is:

How can we realise improved vario-scale geo-information having minimal redundancy?

To reach the main objective the research project, we refine the main question in eight more specific questions (in parentheses the chapter in which the question will be dealt with). To be able to define starting points for a vario-scale environment, the first question addresses:

1. What is the state-of-the-art in: 1. multi-scale data management and 2. generalisation of vector data? (Chapter 2)

[1] In this thesis the pronoun 'we' is used, when reference is made to the author, to acknowledge that co-authors, colleagues and members of the scientific community all have influenced this work.

Then, to lay a theoretical foundation for implementation of vario-scale data structures and to make it possible to define what is valid vario-scale data, we ask ourselves:

2. How can we formally describe what is variable-scale geo-information? (Chapter 3)

Once we have defined what is valid data, we need to be able to construct input data that fits the definition, therefore we will look at validation and creating input data:

3. How can we create valid 2D input data as much automated as possible? (Chapter 3)

One of the initial design goals of the tGAP structures was to have minimally redundant data storage. This is important for use of the structures in a networked environment, because when less data is stored, it is likely that also less data needs to be transferred. Often this requires a balance between storage and computation. This therefore is the topic of the fourth question:

4. How does minimal geometric redundancy influence the design of the data structures? (Chapter 4)

On a related note, we will investigate possibilities for carrying out line simplification, the consequences for the amount of data that needs to be stored and how this can be performed such that the resulting boundaries are valid (*e. g.* do not have unwanted crossings):

5. How can we simultaneously simplify edges so that the result is topologically consistent? (Chapter 4)

Initially, the tGAP structures are created by making use of a merge operation (in which two neighbouring area objects are merged and then form one new object, *cf.* § 2.4). However, merging might be not appropriate for all feature classes (*e. g.* long and linear features, such as roads). Therefore we investigate an alternative operation, splitting. How to perform this operation is the question that we answer next:

6. How can we split linear features over their neighbours, instead of merging to one of their neighbours? (Chapter 4)

Viewing of geographic data is a very important operation for any GIS and this operation should be supported efficiently. Furthermore, the tGAP structures should be able to support progressive data streaming (for which first a coarse map is retrieved and then refined with additional details). Therefore the last two questions that we will answer are:

7. How can we query the data structures to retrieve a 2D map from the structures? (Chapters 3 and 5)

8. How should progressive data streaming in a client-server setup look with respect to increments, communication and data structures (both at the client- and at the server-side)? (Chapter 5)

1.3 Research scope

This section shows what is considered to be in and out of scope, as well as what are the starting points for this research.

1.3.1 *Starting points*

The focus that was adopted at the beginning of the research to limit the possible solution space is as follows:

VECTOR DATA. Designed data structures will be suitable for storing vector data only, as vector data brings several advantages over raster data:

- Vector data allows descriptions with arbitrary topology (instead of a number of fixed direct neighbouring cells);
- It is easier to move, scale and rotate graphical primitives than raster based data sets (*i. e.* it is easier to combine data when different map projection systems are used);
- Such data brings (possibly offline) interactivity at client-side, *e. g.* for querying attributes and interactive editing of features;
- Allows for custom styling in an interactive rendering environment (this can be compared with later binding of presentation style to web documents, with techniques such as CSS and HTML, *e. g.* described in Lie, 2005);

- Rendering is less dependent on resolution, giving a crisper and more aesthetically pleasing result. Vector descriptions are key for printing technology, where the fonts are stored in vector format; *e. g.* making it possible to scale a print to banner size, while retaining high quality output.

GEO-DATABASE MANAGEMENT SYSTEM. Prototypes will be implemented using an extensible, object-oriented database management system (DBMS). A geo-DBMS provides 'a single undivided storage system' (van Oosterom, 1990), giving integrated access to geometry and thematic attributes (thus a query planner and optimizer can take advantage of spatial characteristics of geographical data). This approach solves the problem of keeping separate files in sync, as is for example the case with the de facto shapefile format, which physically splits thematic and geometric attributes in separate files (ESRI, 1998). Furthermore, such a system provides access for multiple users at the same time, while supplying built-in security and authentication models.

DESIGN BY CONTRACT. For engineering prototypes, the paradigm of 'Design by Contract' (Meyer, 1992, 1997) is preferred. Software modules are assumed to only accept and output data, that fulfills a certain contract by means of contracted assertions. For geometrical data this includes that input data is valid, so one can make assumptions, while processing the data (*e. g.* that a description of a polygon follows a certain specification). In contrast, another approach that we could have taken is to follow the 'Defensive Programming' paradigm. In this paradigm incorrect behaviour of software modules is prevented by anticipating wrong input for every module and dealing with this input, when encountered. However, this requires detecting errors in the input, which can take a lot of computation time, or ignoring the input at that moment, which can lead to unexpected behaviour. Handling degenerate cases for geometric algorithms is already hard when all input data is valid – not the least because of floating point arithmetic, of which cases can be found in, amongst others, Douglas (1974); Hoffmann (1989); Schirra (1997); Kettner et al. (2008). Therefore, we prefer a system architecture where said validation (and eventually repair) is performed explicitly and only once, *i. e.* when 'foreign' and unvalidated data enters the database.

1.3.2 *Research scope*

The following topics are explicitly included within the research scope:

- The research is performed from a GIS technological perspective, *i. e.* focusing on low-level data management issues (with 'technical glasses' on);
- As the newly proposed approach already brings enough challenges, input data will be 2D only. 3D data is not considered, although 3D data certainly has potential and advantages over 2D for certain applications (*e. g.* noise modelling);
- Progressive data streaming (in a networked environment, based on a client-server architecture);
- Building prototypes (to demonstrate and evaluate solutions);
- Giving a precise description (formalisation) of what variable-scale geographic information is.

1.3.3 *Out of scope*

The following topics are explicitly excluded from this work:

- Raster data approaches (or hybrid approaches, where vector data is rasterised at the server-side before sending to the client), as such approaches loose some of the advantages of 'raw' vector data, such as custom styling and interactivity;
- Implementing a client with very smooth display and morphing capabilities;
- Data compression techniques to further improve performance;
- Performing Human Computer Interaction and Usability Engineering research (*i. e.* testing and evaluating the proposed solutions by means of letting end users perform certain tasks, or involving potential users into the development cycle);
- Concentrating on thematic attributes and object classification for generalisation (thematic semantics and ontology engineering, as in van Smaalen (2003); Lüscher et al. (2009));
- Studying cartographers at work to improve their work processes (*e. g.* Stoter et al., 2009b);

- Cartography ready for high-quality printing in a form that is hardly changeable: volatile and easy to change visualisations are the starting-point (*i. e.* certain graphical problems can be alleviated by simply zooming in or out in a digital environment – a printed map does not offer these possibilities);
- Dealing with updates and update propagation (*i. e.* propagating changes from the real world into the data structures);
- Creating production-ready software.

1.4 Methodology

This section shows which methodology we followed to get to an answer for the main question. It also shows which tools and test datasets were used in the process of building the prototypes.

1.4.1 *Design science and Experiments*

As exemplified by the above sketched research questions, this research is about the design of variable scale data structures. In this research therefore we have adopted the paradigm of design science. In this paradigm 'knowledge and understanding of a problem domain and its solution are achieved in the building and application of the designed artifact' (Hevner et al., 2004, p. 75).

In a widely cited paper, March and Smith (1995) propose 4 outcomes of design research: *constructs*, the language in which problems and solutions are defined and communicated, *i. e.* they 'form the specialized language and shared knowledge of a discipline or sub-discipline', *models*, which are 'a set of propositions or statements expressing relationships among constructs', *methods*, that define processes and provide guidance on how to solve problems and *instantiations*, that 'demonstrate feasibility, enabling concrete assessment of an artifact suitability to its intended purpose'.

As 'the realm of Information Systems research is at the confluence of people, organizations, and technology' (Hevner et al., 2004, p. 77), systems can be studied within their environment in which they are put to work (*i. e.* studying their application within a certain business process) or as one separate entity in a laboratory setting. Both type of studies contribute to the knowledge base of information management. However, this research does not attempt to evaluate the application of the designed artifact within business processes, nor does it evaluate if and how end users will benefit from it (performing Human Computer Interaction

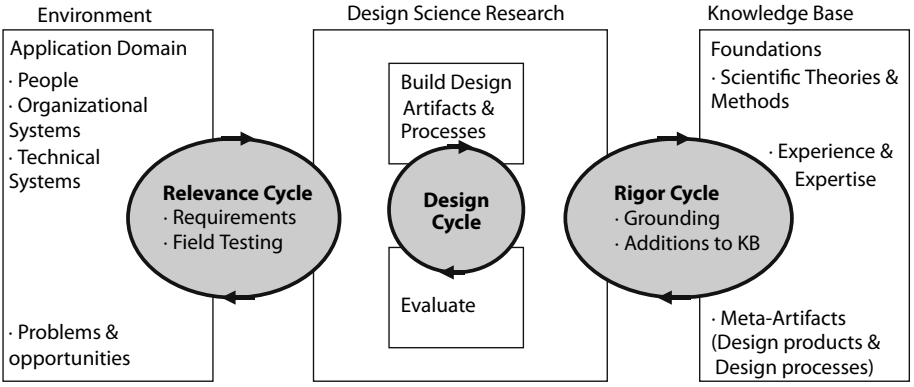


Figure 1.3: The paradigm of Design Science applied to Information Systems research (taken from Hevner and Chatterjee, 2010, p. 16).

research, or usability engineering), but is focused on the technical assessment of the sole system. The research is performed, in an iterative fashion: develop theory, make a software prototype, test prototype with real world data, generalize results by testing with different datasets, check developed theory, develop new or improve theory (based on insights from software prototype), etcetera. As became clear in § 1.1, it is necessary to define theoretical underpinnings of variable-scale geo-information (contributing 'constructs' and 'models' to the knowledge base, *cf.* Figure 1.3). Furthermore, in this research we will develop prototypes ('instantiations') by means of which we test our new theories and illustrate our proposed 'method' of data management. As final remark, the experiments permit us to show absence of missing elements in the developed theories (in his seminal paper on experimentation, Tichy (1997) illustrates the importance of this type of practice-oriented research for computer science).

1.4.2 *Tools and Test datasets*

Prototypes were built with the following tools and using these test datasets:

PYTHON / CYTHON Python is an open source, dynamically typed and interpreted programming language, offering different programming paradigms, *e. g.* object-oriented programming is possible. Rapid prototyping is possible due to the interpreted nature (which is comfortable in an environment of rapidly changing ideas). Furthermore, it has an extensive library of modules available also for spatial programming (Westra, 2010), plus a large

user community (it has been marketed as Swiss army knife of scientific computing: functionality is not only available for spatial development, also for making graphs, automating experiments, building user interfaces, network communication, etcetera).

Furthermore, it is possible to add highly optimised libraries to the dynamic scripting environment by using Cython. Cython provides wrapping of native C and C++ libraries (Seljebotn, 2009). Furthermore it permits implementation of functionality in a more strongly typed and compiled language (*i. e.* Cython is a compiler that compiles an extended subset of the Python programming language into C source code, which then has to be compiled with a C compiler into native binary code for the platform on which the code runs).

POSTGRESQL, EXTENDED WITH POSTGIS PostgreSQL is a database management system, with sub-system for dealing with geographical data. PostGIS is a plugin for the database system and PostGIS follows the OpenGIS 'Simple Features Specification for SQL' (Herring, 2001, 2006), under an Open Source license. PostgreSQL provides good insights to which extent the SQL standards are followed and permits by its open nature diverse applications to connect to it.

QGIS AND OPENJUMP Quantum GIS is an open source Geographic Information Systems (GIS), in this project mostly used as tool in conjunction with the database for selecting and visualising stored data (visualising data is very helpful while debugging geometric algorithms).

CGAL Computational Geometry Algorithms Library that has as goal to provide easy access to efficient and reliable geometric algorithms and data structures. CGAL offers a variety of data structures, like triangulations and Voronoi diagrams in 2D and 3D, and algorithms, such as computing a convex hull of a set of n-D points.

TEST DATA SETS Throughout the project a variety of test data sets have been used. The diversity of data sets guarantees different characteristics with respect to geometry.

Approximately 20 small data sets were created 'by hand', to be able to see whether functionality in prototypes worked as expected. These data sets acted as unit tests, testing correct working of parts of the implemented algorithms. A complementary data set for this purpose that was used was

a set with cadastral parcels from the Netherlands' Cadastre – this data set was directly available as it was used in previous research (Penninga, 2004). Characteristics of the cadastral data: small number of coordinates per line string (2 intermediate points on average), polygons with holes are common and the data theme does not really allow one to perform sensible generalisation in a vario-scale manner. However, because the data was well-checked with respect to topology (*i. e.* valid data), this data set was a suitable candidate for testing the workings of prototypes as no topological errors are present in the data set.

To validate how the proposed solutions will perform in practice also real world data sets were used. For example, different parts of the CORINE2000 dataset were used. To monitor the land cover changes in the European Union, the European Environment Agency (EEA) collects the Coordinated Information on the European Environment (CORINE) Land Cover (CLC) dataset (see Wiggins et al., 1987, for an early description of the programme). The European member states deliver this data every few years. This dataset spans the whole of Europe's member states, and it is freely available[2]. Characteristics: polygons with a relatively high number of vertices per polygon, lots of polygons with holes, intended map scale: 1:100K. In addition, 4 different topographic dataset fragments were used (containing both rural as well as urban data, intended to be used at map scales between 1:3K and 1:50K). As topographic maps often are used as backdrop map, this type of data can be considered as a common denominator for a large range of applications, where multiple themes are integrated (houses, roads, land use, etcetera). The used topographic data sets are:

1. a Dutch topographic data set, obtained via municipality of Amsterdam, 1:10K (dense urban city center of Amsterdam with lots of canals and roads, no individual houses);
2. TOP10NL data, clip of land use polygons forming a planar partition around Delft, intended for 1:10K, with roads, no individual houses;
3. German ATKIS data sets, intended for 1:50K, rural area (land use data without roads and houses), near Hamburg (Buchholz in der Nordheide);

[2]More information can be found on http://www.eea.europa.eu/themes/landuse/clc-download

4. British data, from Ordnance Survey, rural area, with roads and individual houses near Colchester, intended for 1:5K.

Other relevant data characteristics will be described, when the data sets are used.

1.5 Thesis outline

This section gives an outline of which chapters are based on earlier publications and how to read the thesis.

1.5.1 *Publications on which the chapters are based*

Table 1.1 shows an overview of the publications on which the chapters in this thesis are based.

1.5.2 *Guide to the reader*

This thesis is organised in the following manner, of which Figure 1.4 depicts a graphical outline.

CHAPTER 1 (this chapter) gives an introduction to the thesis by showing the importance of the problem and the scientific gap. Furthermore it highlights the research methodology, the data sets and tools used for testing and prototype development, suggests advantages of the proposed solution, gives links to publications on which this work is based and gives an overview of how to read the thesis.

CHAPTER 2 reviews related work so the reader can put the topic of the thesis in context. The chapter starts by showing different view points on the modelling process of geographic space. It gives an overview of the research field of map generalisation and issues related to the concept of map scale. A myriad of data structures have been proposed for storing the result of generalisation operations (to provide multi-scale data access) of which some relevant proposals are reviewed. Progressive streaming of data is described, as this type of networked data retrieval is necessary for a more efficient, net-centric way of data transmission and permits smooth zooming. Finally the chapter concludes with starting points for minimally redundant, variable scale maps. Chapter 2 is mainly based on literature review.

Table 1.1: Publications and their relation with the chapters of this thesis.

No.	Publication (sequenced by publication date)	Ch.
1.	Stoter, J., Morales, J., Lemmens, R., Meijers, M., van Oosterom, P., Quak, W., and Uitermark, H. (2007). Considerations for the design of a semantic data model for a multi-representation topographical database. In Kremers, H., editor, *Proceedings of the 2nd ISGI 2007: International CODATA symposium on generalization of information, Geneva, Switzerland, 1-3 October 2007*, Lecture notes in Information Sciences, pages 53–71, Berlin. CODATA.	2
2.	Meijers, M. (2008). Retrieving tGAP data with a stateless client for visualization. RGI Project Report 233-03, Delft University of Technology, Delft.	5
3.	Stoter, J., Morales, J., Lemmens, R., Meijers, M., van Oosterom, P., Quak, W., Uitermark, H., and van den Brink, L. (2008). A data model for multi-scale topographical data. In Ruas, A. and Gold, C., editors, *Headway in Spatial Data Handling: Proceedings of the 13th international symposium on Spatial Data Handling, SDH 2008*, Lecture Notes in Geoinformaton and Cartography, pages 233–254, Berlin. Springer.	2
4.	Meijers, M., van Oosterom, P., and Quak, W. (2009). A storage and transfer efficient data structure for variable scale vector data. In Sester, M., Bernard, L., and Paelke, V., editors, *Advances in GIScience*, Lecture Notes in Geoinformation and Cartography, pages 345–367. Springer Berlin Heidelberg.	2, 4
5.	Meijers, M. and van Oosterom, P. (2009). Applying DLM and DCM concepts in a multi-scale environment. In Mustière, S., Sester, M., van Harmelen, F., and van Oosterom, P., editors, *Dagstuhl Seminar Proceedings 'Generalization of Spatial Information (09161)'*.	2
6.	Stoter, J., Meijers, M., van Oosterom, P., Grünreich, D., and Kraak, M.-J. (2010). Applying DLM and DCM concepts in a multi-scale data environment. In Buttenfield, B., Brewer, C., Clarke, K., Finn, M., and Usery, L., editors, *Proceedings of GDI 2010: Symposium on Generalization and Data Integration*, pages 1–7, Boulder, USA. University of Colorado.	2
7.	Ledoux, H. and Meijers, M. (2010). Validation of planar partitions using constrained triangulations. In *Proceedings of the 14th Joint International Conference on Theory, Data Handling and Modelling in Geospatial Information Science*, pages 51–56, Hong Kong.	3
8.	van Oosterom, P. and Meijers, M. (2011a). Method and system for generating maps in an n-dimensional space. Dutch patent application 2006630, filed April 19, 2011, expected to be published October 2012.	6
9.	van Oosterom, P. and Meijers, M. (2011b). Towards a true vario-scale structure supporting smooth-zoom. In *Proceedings of 14th ICA/ISPRS Workshop on Generalisation and Multiple Representation*, pages 1–19, Paris.	6
10.	Meijers, M. (2011a). Cache-friendly progressive data streaming with variable-scale data structures. In *Proceedings of 14th ICA/ISPRS Workshop on Generalisation and Multiple Representation*, pages 1–19.	5
11.	Meijers, M. (2011b). Simultaneous & topologically-safe line simplification for a variable-scale planar partition. In Geertman, S., Reinhardt, W., and Toppen, F., editors, *Advancing Geoinformation Science for a Changing World*, Lecture Notes in Geoinformation and Cartography, pages 337–358. Springer Berlin Heidelberg.	4
12.	Meijers, M., Savino, S., and van Oosterom, P. (2011). SplitArea: An algorithm for splitting faces in the context of a hierarchical data structure. Manuscript submitted for review to an academic journal.	4
13.	Meijers, M. and van Oosterom, P. (2011). The space-scale cube: An integrated model for 2D polygonal areas and scale. In *28th Urban Data Management Symposium*, volume 38 of *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 95–102.	3

CHAPTER 3 first stresses that minimal redundancy also has to do with fundamental choices: it discusses whether only one or multiple models of reality have to be maintained. Then we formalise variable-scale data from a mathematical point of view and describe a conceptual model – the space-scale cube (SSC) – for vario-scale data storage. To realise the conceptual model in practice, there is a need to obtain a valid, single-scale input data set that will be used as starting point to create content for the variable-scale data structures. Therefore, this chapter also proposes an approach to validate (and automatically repair) this 2D input data.

CHAPTER 4 shows that specific generalisation algorithms to create vario-scale data are needed and that these algorithms bring specific requirements for the storage of this data into the tGAP data structures. Firstly, it shows that some design changes are necessary to obtain leaner structures compared to earlier published versions of the data structures in terms of needed storage space. Secondly, this chapter proposes an algorithm to simultaneously simplify a set of polylines to obtain a topologically consistent result. Thirdly, this chapter investigates possibilities to use a triangulation for splitting polygons. Furthermore, the implications of the proposed algorithms for the data structures are analysed.

CHAPTER 5 investigates (and improves) the data structures by using them for streaming transmission of the stored vector data over a network. A description is given of how a 2D map at a specific scale point can be derived. Further, it is investigated whether the structures make more dynamic map solutions possible by using the structures for retrieval of data by incrementally adding additional details to an already sent map – *i. e.* progressive transfer. Next, the chapter proposes an additional data structure, so that the variable-scale approach can become more cache-friendly.

CHAPTER 6 brings together the insights from chapters 3, 4 and 5. It proposes a final design of the data structures that are minimally redundant and are suitable for progressive transfer. Due to the progressive transfer demonstrator and the proposed line simplification we realised that the data structures are not pushed to their limits in terms of continuous generalisation. Therefore this chapter also proposes a new way of obtaining data, leading to continuously generalised vario-scale data; key to smooth zooming.

CHAPTER 7 highlights the main achievements of this work as well as the insights gained into variable-scale geo-information. It also provides a myriad of suggestions for future research.

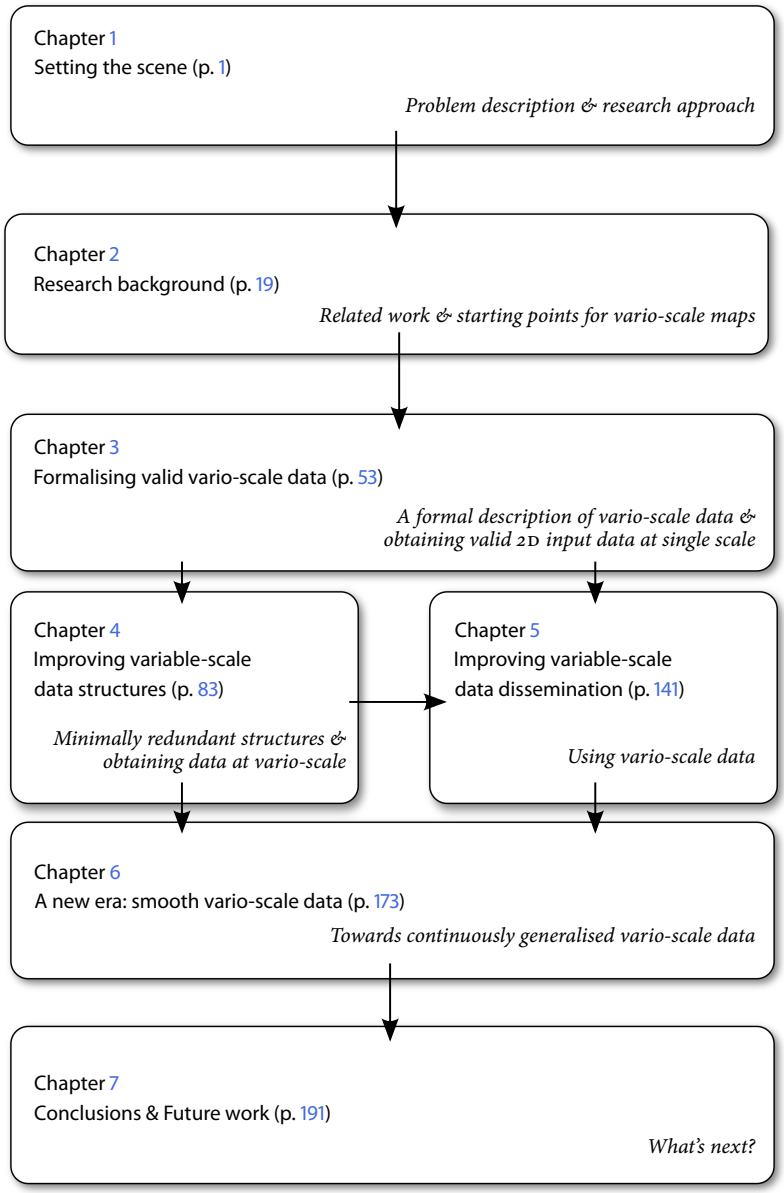


Figure 1.4: Schematic outline of the structure of the thesis.

2 RESEARCH BACKGROUND

~

This chapter reviews related work so the reader can put the topic of the thesis in context. The chapter starts by showing different view points on the modelling process of geographic space (space-first vs. object-first, § 2.1). It gives a high level overview of the field of map generalisation and how dealing with map scale is a central issue in this field (§ 2.2). As generalisation is computationally intensive, a multitude of data structures have been proposed for multi-scale data access, of which some relevant proposals are briefly discussed in § 2.3. Vario-scale structures, a different approach for organising multi-scale data is discussed in § 2.4. Then progressive transmission is described in § 2.5, as this type of networked data retrieval is necessary for a more efficient, net-centric way of working and permits smooth zooming. Finally the chapter concludes with starting points for minimally redundant, variable scale maps (§ 2.6).

Own publications

This chapter is partly based on the following own publications:

- Stoter, J., Morales, J., Lemmens, R., Meijers, M., van Oosterom, P., Quak, W., and Uitermark, H. (2007). Considerations for the design of a semantic data model for a multi-representation topographical database. In Kremers, H., editor, *Proceedings of the 2nd ISGI 2007: International CODATA symposium on generalization of information, Geneva, Switzerland, 1-3 October 2007*, Lecture notes in Information Sciences, pages 53–71, Berlin. CODATA.

- Stoter, J., Morales, J., Lemmens, R., Meijers, M., van Oosterom, P., Quak, W., Uitermark, H., and van den Brink, L. (2008). A data model for multi-scale topographical data. In Ruas, A. and Gold, C., editors, *Headway in Spatial Data Handling: Proceedings of the 13th international symposium on Spatial Data Handling, SDH 2008*, Lecture Notes in Geoinformaton and Cartography, pages 233–254, Berlin. Springer.

- Meijers, M., van Oosterom, P., and Quak, W. (2009). A storage and transfer efficient data structure for variable scale vector data. In Sester, M., Bernard, L., and Paelke, V., editors, *Advances in GIScience*, Lecture Notes in Geoinformation and Cartography, pages 345–367. Springer Berlin Heidelberg.

- Meijers, M. and van Oosterom, P. (2009). Applying DLM and DCM concepts in a multi-scale environment. In Mustière, S., Sester, M., van Harmelen, F., and van Oosterom, P., editors, *Dagstuhl Seminar Proceedings 'Generalization of Spatial Information (09161)'*.

- Stoter, J., Meijers, M., van Oosterom, P., Grünreich, D., and Kraak, M.-J. (2010). Applying DLM and DCM concepts in a multi-scale data environment. In Buttenfield, B., Brewer, C., Clarke, K., Finn, M., and Usery, L., editors, *Proceedings of GDI 2010: Symposium on Generalization and Data Integration*, pages 1–7, Boulder, USA. University of Colorado.

2.1 Modelling digital geographic space

This section shows different view points on the modelling process of geographic space (space-first vs. object-first) and puts forward why we regard the space-first approach the best approach for modelling variable-scale geo-information.

2.1.1 *Space-first versus object-first modelling*

When looking at modelling of geographic features, data model designers view the data to be modelled at different levels. These levels progress from reality into data stored in a machine. Peuquet (1984) distinguishes four different levels:

REALITY the phenomenon as it actually exists, including all aspects which may or may not be perceived by individuals;

DATA MODEL an abstraction of the real world which incorporates only those properties thought to be relevant to the application or applications at hand, usually a human conceptualization of reality;

DATA STRUCTURE a representation of the data model often expressed in terms of diagrams, lists and arrays designed to reflect the recording of the data in computer code;

FILE STRUCTURE the representation of the data in storage hardware.

At the level of creating a data model, three main approaches can be distinguished: 1. space-first approach, 2. object-first approach, and 3. a hybrid approach (see Figure 2.1). Because the related models have quite a different starting point, there is sometimes confusion between modellers.

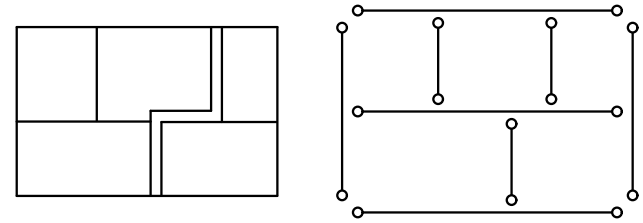
In the space-first approach, the models start from the perspective that the geometries of objects interact in and subdivide the embedded space (by focusing on the geometrical and topological relationships). Attributes are added to these geometries in order to classify the objects. The result is typically a set of tables in a database such as point or symbol table, text or label table, line table and area table. Within a table all objects (records) will have the same set of attributes. For example in the area table there may be houses and roads, but they all have the same set of attributes. In this approach, it is also possible to explicitly model the topological structure (*e. g.* linear network, or partition of space) with well-known advantages (explicit connectivity, avoiding redundancy, better guarantees for quality under updates). The Dutch cadastral map in LKI (Landmeetkundig Kartografisch Informatiesysteem) is a typical example of this space-first approach (van Oosterom and Lemmen, 2001). In this solution objects may share, via topology, their geometry with other objects. It could be argued that map representations (on paper or screen) themselves, *i. e.* the visualization of the spatial data, is also a space-first type of model as all objects are considered together in a geometric model.

The second approach, the object-first approach, models first the object classes with added geometry attributes (*i. e.* there is less focus on the division of space as a whole, as in the space-first approach). Every object class can have its own set of thematic attributes, which may vary for the different object classes. Also every object has its own geometric description independent of any other object. The TOP10NL model is an example of this approach (Bakker, 2005). Typically the result is a set of tables in the database such as houses, roads, waterways, which have among others their own simple object geometry type attribute. Sometimes

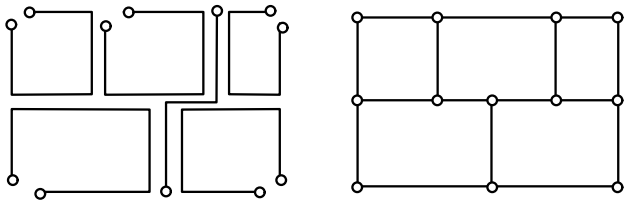
additional rules (constraints) are added in order to avoid unwanted situations (often topology based); *e. g.* a polygon representing a house should not overlap with a road polygon at the same layer. The drawback is that all these constraints have to be explicitly stated (and checked when updates are performed) and are not embedded in the main structure of the model. Also the model does not explicitly contain the topological relationships, which may support various types of analysis (*e. g.* quality control of updates). It must be noted that topological relationships are very important for map generalisation; *e. g.* what are the neighbours of this object (candidates for aggregation), is the network connectivity damaged when this road segment is removed, etcetera.

The third approach is the hybrid approach which treats the geometries and object classes equally. It combines the strengths of both approaches: the (thematic) attributes are specifically designed for every object class, but the model also enables shared geometry and use of embedded structure. The spatial domain is partitioned and the result is described using tables for nodes, edges, and faces (and solids in 3D). The objects are modelled in the same way as in the object-first approach with the exception that the objects do not have their own independent geometry-attributes, but refer to primitives in the geometry/topology part of the model (node, edge, face, …). This is the approach as described in the Formal Data Structure (FDS) theory of Molenaar (1989, 1998) and is more recently implemented in products such as 1Spatial Radius Topology (Baars et al., 2004) and Oracle Spatial Topology (Kothuri et al., 2007).

Peuquet (1984) and de Hoop et al. (1993) discuss the different modelling approaches and the consequences for realisation and use. It cannot be claimed that one model is 'better' than another model. This depends on the application context and use. If one specifies a number of important characteristic of the application domain and typical use, then it is possible to state which approach is preferred. Considerations could be: 1. allow exceptional overlapping of objects in certain cases (*e. g.* bridge over water), 2. allow modelling of systematically overlapping sets of object classes (*e. g.* topographic objects at one hand and administrative units at the other hand), 3. enable multiple geometry representations of single objects (*e. g.* road area polygon and road centre line, or building footprint polygon, building rooftop polygon, and building centroid), 4. support consistent updating/maintenance, 5. support efficient querying, analysis and viewing of data, 6. avoid storage space consuming representations (which might also be expensive for data transfer), 7. support easy delivery for customers (simple objects might be easier to receive in another system than a topology structure), etcetera.



(a) Terrain situation (b) 'Spaghetti' lines



(c) Object-first approach (d) Space-first approach

Figure 2.1: Several approaches for representing the geometry of the terrain situation that is shown in Figure 2.1a. Figure 2.1b does not contain explicit area objects and it is not possible to link the objects unambiguously to their geometric elements (this in fact is what Peuquet (1984) and Theobald (2001) term the spaghetti data structure, see § 2.1.2). In Figure 2.1c objects have their own geometric representation from which it is possible to derive information about position, shape and size of the individual objects, but it is rather difficult to analyse topologic relationships for neighbouring objects. The approach shown in Figure 2.1d allows to link objects to geometric elements and also allows to derive topologic relationships between the objects via their geometric elements (taken from Molenaar, 1998, p. 35).

All three approaches can be extended in one way or another to add multi- or vario-scale aspects to the model. We regard the space-first approach the best approach for variable-scale geo-information because of the advantages this approach brings (reduced data storage – and reduced data transfer in a net-centric environment – and explicitly modelled topological relations, which are very important to carry out automated generalisation) and, for the time being, the complexity of the hybrid approach is not needed.

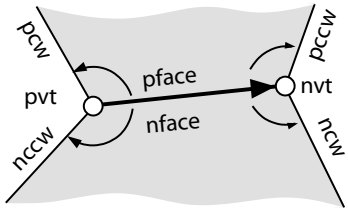
2.1.2 *Single-scale data structures*

As data modelling clearly deals with trade offs, it is impossible to design a general-purpose data structure that is equally useful for all situations. Some data structures are *e. g.* efficient for producing graphics, but will be very inefficient for

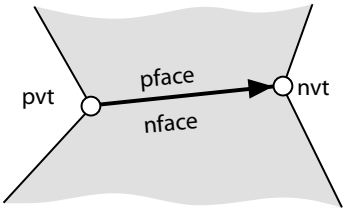
analytical purposes. Therefore, it is no wonder that to date a variety of data structures have been proposed.

With the first advent of digital technology, the most apparent choice for digitizing paper maps was to represent geographic features with a bunch of individual points and polylines. A polyline is a sequence of n line segments with two end points and $n - 1$ intermediate points. With the analogy of a plate of tangled pasta, this 'unstructured' data structure is often called the *spaghetti data structure* (Peuquet, 1984; Theobald, 2001). In the intertwined 'mess' of polylines, boundaries of geographic objects do not necessarily correspond to the polylines that are stored. This changes for the *cartographic data structure* (CDS), in which geographic objects correspond to a meaningful counterpart in the data structure (Theobald, 2001): geographic objects are abstracted to points, polylines and polygons. To capture the semantics of these modelling primitives, for an unambiguous interpretation that enables interchange of geographic data, the Open Geospatial Consortium (OGC) and International Organization for Standardization (ISO) have established standards for structuring the spatial data (Herring, 2001; ISO/TC 211, 2003). These standardisation efforts have also led to a specification (Simple Features for SQL) that defines how to support 0D, 1D and 2D spatial objects in object-relational DBMS environments (Herring, 2006). Database vendors have implemented these specifications, which has resulted in a shift towards geographic data being embedded in mainstream information technology environments.

The Simple Features specification is a clear example of using the geometry-first principle for modelling geographic objects. As topological relationships between objects with this approach are not explicitly stored, it is necessary to compute these relations 'on-the-fly'. For the 'on-the-fly' approach to work 'it is critical that there exists a high performance topology engine that … instantiates the topological primitives for the given collection of features within the topology' (Hoel et al., 2003). In contrast, Bertolotto (1998) points out that a space-first approach will result in a set of structured geographic entities, for which the topological relations are persisted and explicitly managed ('persistent topology') and that this is more efficient for her application (map generalisation). Her argument for this is that 'map overlay is a complicated operation and it is expensive from the computational point of view. Thus it seems more convenient to compute it only once, instead of having to recompute it each time such a query is being processed.' Other advantages that are known of such a persistent topology approach is reducing duplicate storage (boundaries between objects have to be stored only once and this also applies to shared attributes for boundaries, *e. g.* date of survey),

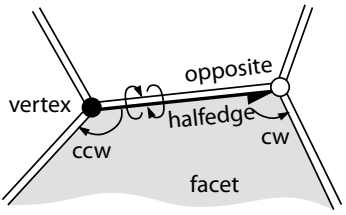


(a) Winged Edge data structure. Per edge 2 vertices (pvt, nvt), 2 face references (pface, nface) and 4 incident edge references, the so-called *wings* (pcw, pccw, ncw and nccw), are stored.

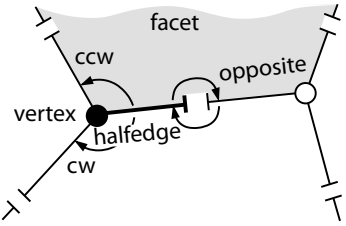


(b) Node Edge Face (NEF) data structure. For each directed edge the source and target node references (pvt, nvt) and the left and right face references (pface, nface) are stored.

Figure 2.2: Data structures (Figure 2.2a taken from Kettner, 1999).



(a) Halfedge (FE structure).



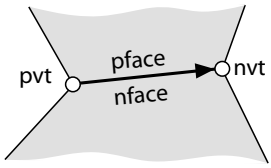
(b) Halfedge (VE structure).

Figure 2.3: Halfedge data structures (taken from Kettner, 1999). Kettner explains that "there are two ways of splitting the edge: Either the edge is split along the faces, such that the oriented halfedges belong to the two facets incident to this edge, see Figure 2.3a, or it is split into two halfedges belonging to the two vertices incident to this edge, see Figure 2.3b. […] In both splitting variants a halfedge contains a pointer to an incident vertex, an incident facet and the opposite halfedge. It is a matter of convention whether the source or target vertex is the one chosen to be stored in a halfedge or whether the facet to the left or the right is stored." (Kettner, 1999, p. 70)

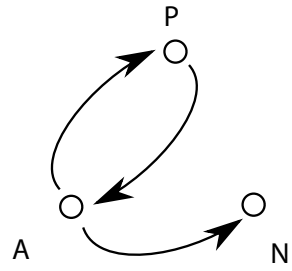
explicit neighbourhood relationships can be derived (speeding up certain types of queries), possibilities of automated error checking and consistency under edit operations (see for example Theobald, 2001; Penninga, 2004; Matijević et al., 2008).

For storing persistent topology, diverse structures have been proposed, not only in the field of GIS, but also in associated fields, such as computer graphics. Baumgart (1975) proposed the Winged-Edge structure to represent polygonal models for computer graphics (see Figure 2.2). It stores information on nodes, edges and faces and for each edge four edge pointers are stored. These pointers make it possible to easily navigate in the structure from edge to edge, which makes the structure suitable for editing. This structure is the basis of what has been standardised in ISO/TC 211 (2003) (topology packages). A somewhat simpler structure targetted at GIS is the Node-Arc-Area (NAA), also known as the Node-Edge-Face (NEF) structure, and described in any good GIS textbook (such as Worboys and Duckham, 2004). It is nearly equivalent to the Winged-Edge structure, except that the edge to edge pointers are not stored, therefore more geometric operations might be required when performing edits or updates on the structure. Another family of structures that also resembles the Winged-Edge structure is the Halfedge data structure. For this type of data structure, each edge is decomposed into two directed halfedges with opposite orientations, *cf.* Figure 2.3. Both Kettner (1999) and Brönnimann (2001) give an excellent introduction to deep implementation issues of such structures. The concept of directed halfedge is also used for the Doubly-Connected-Edge-List (DCEL, Muller and Preparata, 1978; de Berg et al., 2000), which is at the basis of the design and implementation of planar maps in CGAL (Flato et al., 1999).

A useful, educational tool for analysing the data structures and their stored relations is the PAN-graph, as proposed by Gold (1988). Figure 2.4 shows that a PAN graph has three vertices, denoted P (for polygons), A (for arcs) and N (for nodes). Each vertex in the PAN graph represents an object class rather than any specific object of that class. Pointers between object classes in the data structure become edges in the PAN graph, enabling visual comparison of the diverse data structures. This presents insights into the diverse relationships between object classes of the data structures that are modelled. However, this tool and the data structures discussed so far are only suitable for storage of geographic entities at one level of detail (mono- or single-scale).



(a) Node Edge Face (NEF) data structure.



(b) An illustration of the PAN graph for the NEF data structure (shown in (a)). Note that the PAN graph also shows a link from Polygons (face) to Arcs (edge), such that polygons can be formed out of the data structure. It depends on the implementation whether for *every* edge such a pointer is stored or, for example, only one pointer per ring.

Figure 2.4: The PAN-graph: an educational tool for analysing data structures (Gold, 1988).

2.2 From single-scale to multi-scale maps

This section shows that map scale expressed as representative fraction is always varying over the mapped domain. Therefore, it clarifies terminology and defines what is meant with *variable-scale* (or for short *vario-scale*) data. Then it reviews the field of map generalisation, that deals with spatial and thematic resolution of geographic objects under change of map scale.

2.2.1 *Map scale*

As Goodchild (2001) correctly argues that the term *scale* is 'a heavily overloaded term in English', with diverse meanings, some clarification of terminology is necessary.

Lam and Quattrochi (1992) discuss that in geography three meanings of scale exist, along the three dimensions spatial, temporal or spatio-temporal, see Figure 2.5. As the temporal dimension is out of scope of this thesis, we will not discuss the latter two here further. For spatial scale Lam and Quattrochi discriminate three meanings:

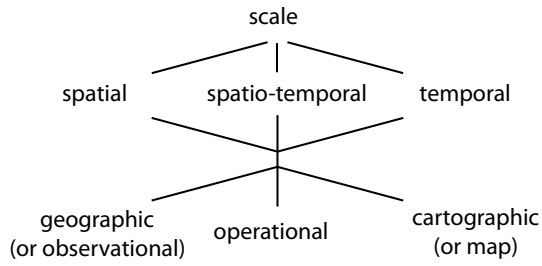


Figure 2.5: Meaning of scale (taken from Lam and Quattrochi, 1992)

GEOGRAPHIC OR OBSERVATIONAL SCALE denotes the spatial extent of study, where large-scale means that the study area is large in extent and small-scale means only a small area is studied;

OPERATIONAL SCALE is the spatial extent at which a particular phenomenon functions;

CARTOGRAPHIC SCALE OR MAP SCALE is defined as the ratio of a distance on a map printed on paper and the corresponding distance on the earth, where generally a large-scale map covers a small area with more detail and a small-scale map covers large area, with less detail.

On paper map sheets the map scale is usually expressed with what is called the *representative fraction* (RF), *e. g.* a RF of 1:25 000 indicates that 1 centimeter on the map relates to 250 meter on the earth (Robinson, 1953). Note that the RF is dimensionless, assuming that the unit of distance on both sides of the fraction is the same. From the RF also the meaning of large and small scale maps is derived. When the number represented by the fraction is 'large', a large scale map is implied, for example 1:25 000 $\gg$ 1:5 000 000 *viz.* $4 \times 10^{-5} \gg 2 \times 10^{-7}$. As different meanings for large and small scale exist – depending on the context in which the word scale is used – this terminology often leads to confusion. Next to giving the RF also graphical depictions for map scale exist, the so-called scale bars (Figure 2.6). Advantage of such a graphical depiction is that automatic adjustment takes place, when a paper map changes size (*e. g.* due to photo copying), contrary to a textual representation that for example explicitly states that '1 cm on the map represents 250 m in the terrain'.

Tightly related to map scale is resolution. Resolution concerns the ability to distinguish closely placed objects. The RF has an impact on the resolution that a map provides (Töpfer, 1974), affecting the minimal object size and the positional

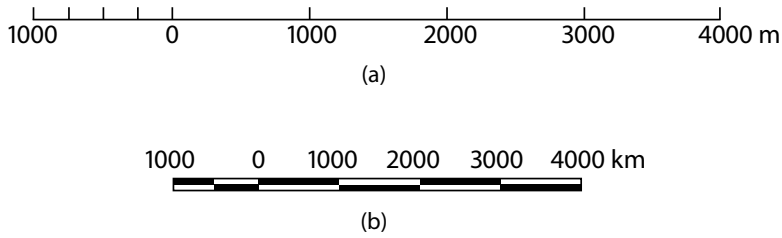


Figure 2.6: Two examples of scale bars. After De Maeyer et al. (2004), p. 98.

accuracy of displayed objects (*cf.* Figure 2.7). Goodchild (2001) discusses that it is typical on a paper map to show features not less than 0.5 millimeter apart[1] and that together with the RF this for example means for a 1:25 000 map, that the provided resolution will be 12.5 meters (in the terrain). In this respect, it is worth to mention the work of Tobler (1987), that posed the same rule: 'divide the denominator of the map scale by 1 000 to get the detectable size in meters. The resolution is one half of this amount' (*i. e.* 12.5 meters for the 1:25 000 map).

Furthermore, the RF is an approximate measure, as it is not possible to maintain scale correctly for an entire map, due to that the earth is a 3D sphere, a map is a 2D planar surface and the spherical surface will rip, while flattening it. A *geographic coordinate system* defines point locations on the earth (Robinson, 1953; Kennedy and Kopp, 2001). These locations are expressed in a coordinate system with latitude and longitude (*i. e.* λ and ϕ) which are angles from the center of the system to its surface. The shape and size of the surface is defined by a sphere or ellipsoid, that approximates the earth its surface. Together with a set of reference points (that are called the *geodetic datum*[2]) the ellipsoid composes the geographic coordinate system. A *projected coordinate system* represents coordinates (*i. e.* x and y) on a flat, 2D surface. When working with geographic data, 'a projected coordinate system is always based on a geographic coordinate system' (Kennedy and Kopp, 2001, p. 16). Therefore, it is necessary to carry out a map projection. A map projection relates the spherical coordinates on the sphere or ellipsoid to the flat, planar coordinates by using mathematical formulas. In symbolic form this is expressed as: $(\lambda, \phi) \rightarrow (x, y)$. Figure 2.8 shows an example of the distortion

[1] The visual acuity of the human eye is approximately 0.2 mm at a reading distance of 30 cm (SSC, 2005, p. 26).

[2] The term datum is often used interchangably with geographic coordinate system. In the strict sense, it only refers to the chosen elements of such a system that defines the origin of the coordinates (Ordnance Survey, 2010).

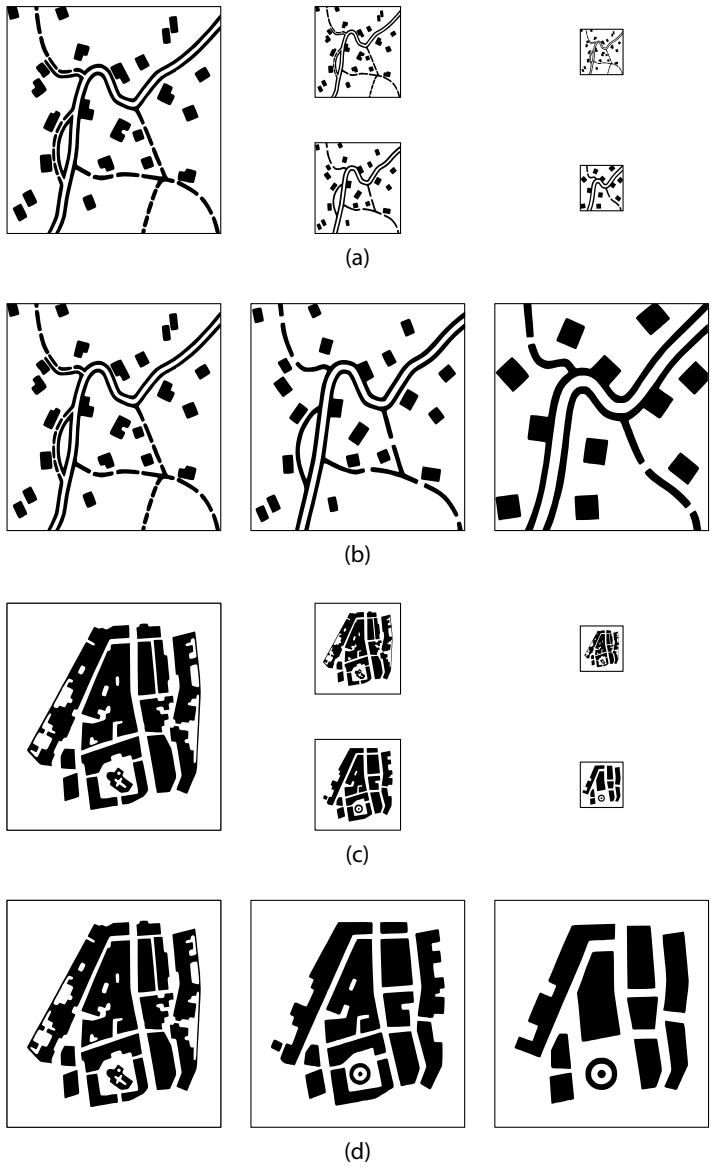


Figure 2.7: The need for map generalisation. The maps show loosely scattered farmsteads and a dense urban core. Both Figure 2.7a and 2.7c show the maps at their correct map scale, i.e. 1:10 000 (left), 25 000 (middle) and 50 000 (right). The maps at the top row are reduced (*i. e.* resized) versions of the 1:10 000 map. These versions are clearly unreadable, while in the bottom row map generalisation has taken place to improve legibility. Furthermore Figure 2.7b and 2.7d show the generalised 1:25 000 and 1:50 000 maps enlarged to the 1:10 000 map scale to make the changes that have taken place in the generalisation process more visible. Figures 2.7b and 2.7d taken from Imhof (1972), p. 222.

that a map projection causes and that this distortion can be visualised with the so-called Tissot's indicatrix (*cf.* Robinson, 1953, appendix D). In most cases a few lines on a 2D map will have scale maintained correctly. Some projections preserve true scale between one or two points and every other point on the map and sometimes a local scale factor is used to compensate for the variability in scale.

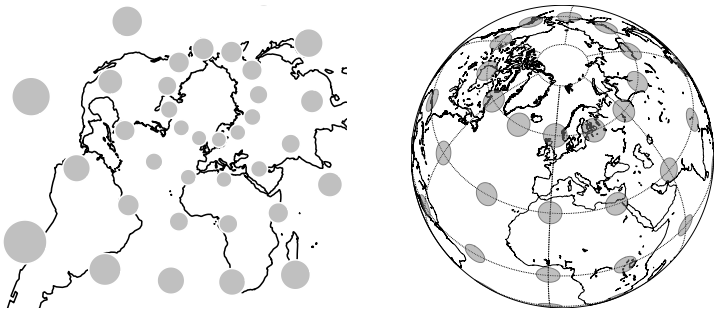


Figure 2.8: Tissot's indicatrix with 2 projections where The Netherlands is placed in the centre. Tissot's indicatrix, also known as the ellipse of distortion, illustrates the type of distortions a map projection introduces. The size and shape of the ellipse shows how much the scale is changed and in what direction. If a projection locally preserves angles (conformal) the ellipse would remain circular. If a map projection preserves area (equal-area) the projected ellipses will have the same size. If a projection is neither equal-area nor conformal at a point, both the shape and area of the ellipse vary. It is clear that for both projections, area distortions are present and that map scale is not constant.

Although we have shown that the RF is only approximately uniform for a mapsheet (and thus varies over the map), this is not the intended meaning of variable-scale (or vario-scale for short) maps in this thesis. The intended meaning is that when the RF of a map is changed a small amount, the content of the map is also changed by a small amount (*i. e.* it must be possible to zoom in or out on the digital map in a continuous, step less way). Another known meaning of a 'variable-scale map' has been defined by Harrie et al. (2002), that proposed a method to show a map with large-scale data for a circle positioned in the centre of the map and small-scale data outside this circle. Note that this type of maps is also known by the name of Fish Eye maps (Misue et al., 1995) and that we term this type of maps 'mixed-scale maps' (see § 6.2 and Figure 6.7).

2.2.2 *Generalisation*

Traditionally, cartographers have made topographic paper maps for different map scales. This process in which larger scale maps are reduced to smaller scale maps is termed cartographic generalisation. Because of resolution aspects (§ 2.2.1, notably Figure 2.7), it is not possible to just reduce the content of the map only graphically. Therefore, Weibel (1991) defines that the aim of this process is to 'produce maps at coarser levels of detail, while retaining essential characteristics of the underlying geographic information'.

Figure 2.9 illustrates that since the digital era in the context of GIS, generalisation can be understood as being composed of three separate processes: *object generalisation* deals with product specifications and data capture through the universe of discourse (*i. e.* the view that defines everything of interest in the real world for a data set and by means of which data quality can be expressed, *cf.* Jakobsson, 2002). This object generalisation leads to a highly detailed Digital Landscape Model (DLM). From this landscape model versions with lower levels of detail can be produced, mainly targeting data sets with less data for more efficient computation (or even making computation possible, when datasets are huge). This process is termed *model generalisation*. When it is time to make a visual representation of the digital data, there is a need to carry out a process, termed *cartographic generalisation*, leading to specialised models, targeting production of maps (*i. e.* geometric representation of objects is adapted to the styling of the map), the so-called Digital Cartographic Models (DCMs). Note that for this process either the high quality DLM can be used, or one of the DLMs with reduced accuracy.

Although the separation between Digital Landscape Model (DLM) and Digital Cartographic Model (DCM) is theoretically considered as the optimal way of maintaining data sets at multiple scales, in practice data producers, like national mapping agencies (NMA), wrestle with the question what to store explicitly in order to efficiently maintain their geographic databases and maps (Stoter et al., 2011) and consequently what operations to apply to obtain DLMs and DCMS at lower accuracies.

In this respect, Stoter (2005) has shown that different production workflows and conceptual architectures have been implemented among diverse NMAs over Europe. A central question seems to be whether to follow a ladder or star approach for creating smaller scale data sets. For the ladder approach, in a sequence of steps a smaller scale dataset is derived from the (previously produced) larger

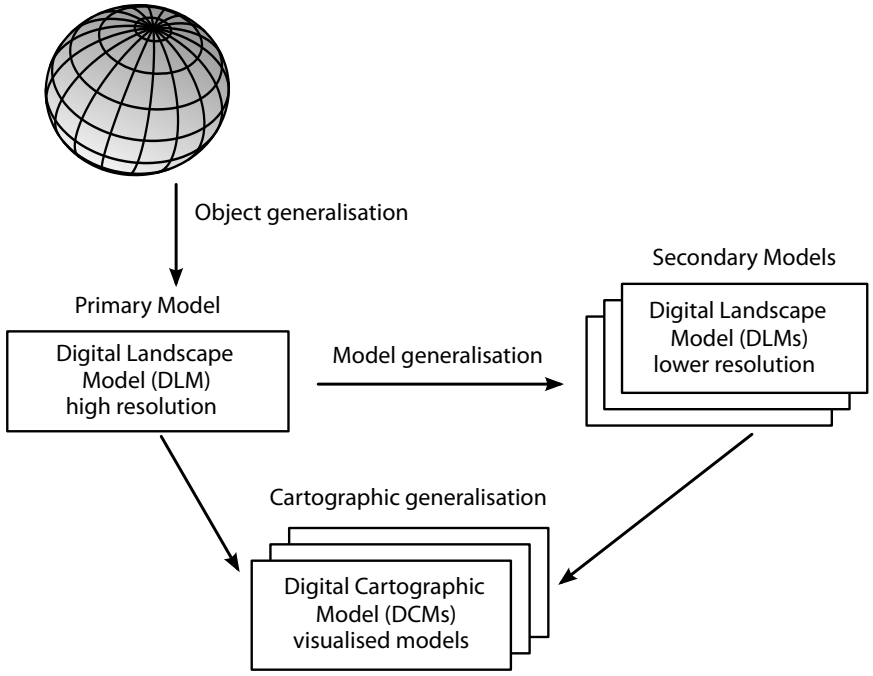


Figure 2.9: Generalisation can be understood as being composed of three separate processes. This terminology has originally been developed for the German ATKIS project (Grünreich, 1985, 1992).

scale dataset. The alternative is the star approach, where every small scale dataset is generalised from the same (large scale) base dataset (Foerster et al., 2010).

For either approach, to derive datasets with lower levels of detail, so-called generalisation operators are needed. These operators express how modelled geographic objects have to change with respect to the map scale (and thus the resolution) of the newly derived dataset. Foerster et al. (2007) provide a formal classification of operators (shown in Table 2.1). Some operators mainly change the geometric description of the objects (*e. g.* simplification and displacement), while others focus more on the thematic attributes (such as class selection, re-classification) and some on both (*e. g.* typification and amalgamation).

Algorithms to implement the generalisation operators have been widely investigated. Without any pretension of being complete, we now will briefly mention a few of these algorithms. In the early days of automated cartography several attempts have been made to provide methods for line generalisation

Table 2.1: Classification of generalisation operators (Foerster et al., 2007)

Model generalisation	Cartographic generalisation
Class selection	Enhancement
Reclassification	Displacement
Collapse	Elimination
Combine	Typification
Simplification	Enlargement
Amalgamation	Amalgamation

(Douglas and Peucker, 1973; Dettori and Falcidieno, 1982; McMaster, 1987; Jenks, 1989; Cromley, 1991) of which the Douglas and Peucker algorithm is the most well-known. Visvalingam and Whyatt (1993) proposed an alternative method for line simplification, on which context-aware simplification methods have been built (Zhou and Jones, 2004; Kulik et al., 2005) and which we will employ in § 4.2.

In general, for the implementations of generalisation algorithms diverse spatial data structures will be used. Jones et al. (1995) show that it is possible to implement diverse generalisation operators using a constrained triangulation. Also Dyken et al. (2009) use such a triangulation for simplifying a set of curves simultaneously, *cf.* Figure 2.10. Haunert and Sester (2008) use the straight skeleton for obtaining centre lines for a road network and show that the straight skeleton also can be used as a collapse operator. McAllister and Snoeyink (2000) obtain centre lines of a river network. The river in their approach is represented by separate polylines that are tagged as left and right banks describing the polygonal river area. Using a robust implementation of the Voronoi diagram and a topological data structure (the quad-edge data structure) they obtain an approximation of the medial axis of the river network. More examples of data structures for generalisation are given in § 2.3.2, p. 38.

As also illustrated by the above mentioned algorithms, Weibel (1997) points out that it is not sufficient to only have 'simplistic' and 'local' algorithms (*e. g.* not considering objects in the spatial neighbourhood of the objects being generalised) and that for generalisation it is of importance that algorithms:

1. observe the spatial and semantic constraints imposed by map context. As the resolution aspect is an important geometric constraint for generalisation for producing paper maps, it is not surprisingly that displacement of features has been studied in detail. Mackaness (1994) shows how spatial

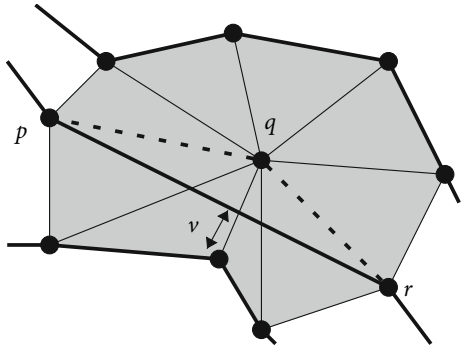


Figure 2.10: Line simplification using a constrained triangulation. The original part of the line, formed by the vertices p, q and r is simplified into $\overline{pr}$, if the separation distance v between the new line and the other lines is large enough. The triangulation is used to be able to define and check this distance v. Taken from Dyken et al. (2009).

conflicts for displacement can be identified and how cluster analysis is of help. Bader (2001) provides three algorithms, using snakes, elastic beams and ductile truss for displacement of roads and buildings. Sester (2000) and Sester (2005) model the displacement of buildings as an optimisation procedure for which a global optimal solution is found by employing conventional least squares adjustment. In this respect it is important that optimisation goals are formalised, so that they can be evaluated (*cf.* Schmid, 2008; Stoter et al., 2009a; Burghardt and Schmid, 2010).

2. make implicit 'knowledge' explicit, as it is needed for the generalisation process. An example is that of Mackaness and Mackechnie (1999) who present a method for the detection and simplification of road junctions. Moreover, Chaudhry and Mackaness (2007) present an approach for aggregation of geographic objects into composite objects at higher level of abstraction based on partonomic relationships. Another example of such a method is found in Haunert and Wolff (2010), who first presented an optimisation approach to simplify sets of polygonal building footprints and where Haunert (2011) extended this work to allow preservation of symmetries in the footprints in the simplification process (*cf.* Figure 2.11).

3. be able to draw from rich data models (using a combination of different data representations and auxiliary data structures). When these are not available, it is necessary to 'enrich' the original data, performing so-called data enrichment. For example, Neun (2007) investigates this process of

provision of auxiliary data by making implicitly contained high level knowledge explicit and also Lüscher et al. (2009) employs data enrichment, by using an ontology-driven approach and supervised Bayesian inference for inferring complex spatial concepts.

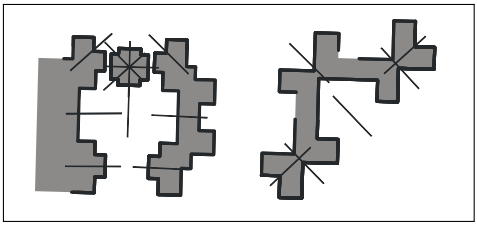


Figure 2.11: Detection of symmetries in building footprints so that these can be taken into account during simplification. The Figure shows the detected axes in which the building footprint is symmetrical (taken from Haunert, 2011).

For a complete generalisation workflow, it is necessary to embed the generalisation algorithms in some orchestrating, overall framework, such as the conceptual framework proposed by Brassel and Weibel (1988), or an expert system framework (such as described by Forrest, 1993). More recent, other fruitful alternatives for orchestration have been proposed, such as simulating annealing (Ware et al., 2003) and multi-agent system (MAS) technology (Lamy et al., 1999). MAS can nowadays be found in a commercially available implementation (*i. e.* 1Spatial Radius Clarity).

Independent from these orchestration processes, the output data sets will have to be stored for later use. For this purpose a multitude of multi-scale data structures have been proposed. This is the topic of the next section.

2.3 Multi-scale hierarchies

Timpf (1999) investigates the connections between abstraction processes, levels of details of objects, and hierarchies in more detail and concludes that 'hierarchies should be used in GIS because they correspond to one way of humans of structuring the world, thus providing a common conceptual mechanism. Hierarchies reduce processing time and they increase the stability of the reasoning system. Hierarchies break down a task into manageable portions, thus allowing for parallel processing. Finally, they allow for efficient reasoning at the level of detail required for a certain task' (Timpf, 1999, p. 137). Therefore, in this section we

will review what terminology is in use and what type of hierarchical approaches are state-of-the-art for managing data at multiple scales (§ 2.3.1 and 2.3.2), show that vario-scale structures take a different approach (§ 2.4) and have potential for being applied for progressive data transfer (§ 2.5).

2.3.1 *Multi-representation, -resolution and -scale terminology*

The simplest (and most commonly used) approach that comes to mind for structuring data for multiple scales (creating a hierarchy of data sets) is to just store the data sets for different map scales as separate sets of objects and together with every data set the scale range for which it is valid. Then, during use it is possible to pick the correct set of objects for the display scale. The advantage of such an organisation is that it is relatively easy to set up. But when it is time to maintain the database, this is not so easy, as the sets have been created as independent copies and there is no explicit knowledge available on which large-scale object relates to which smaller-scale object in the database (*i. e.* no hierarchical set up of relations between objects). This issue of hierarchical, multiple representations therefore gained attention in a research program of the National Center for Geographic Information and Analysis (NCGIA, 1989; Buttenfield and DeLotto, 1989) where it was mentioned that: 'a GIS database must be able to represent objects at different resolution levels and to support modification across resolution levels.' (NCGIA, 1989, p. 125). Since then many researchers have focused on this issue. However, terminology for maintaining multiple object representations for the same real world object is not defined in a rigorous manner and words such as multi-representation, multi-resolution and multi-scale are often used interchangeably in the prevailing literature.

A general concensus on the terminology seems to be that *multi-representation* is not solely intended for the purpose of managing datasets at different scales and thus can be considered a broader, umbrella term. Friis-Christensen et al. (2002) point out, that 'often, the same real-world entity (*e. g.* a river or a building) is represented by different objects in the same or different databases. This phenomenon is called multiple representation, and is a key problem in managing geographic information' (Friis-Christensen et al., 2002, p. 150).

The crux of multiple representation is that the different representations of the same real world object are linked in the database(s), so that it is clear that for one object multiple versions exist. Van Oosterom (2009) identifies that '[a] fundamental question [...] is whether or not an object and its identity change. How one sees the concepts versus the reflection in the physical model (e.g., using

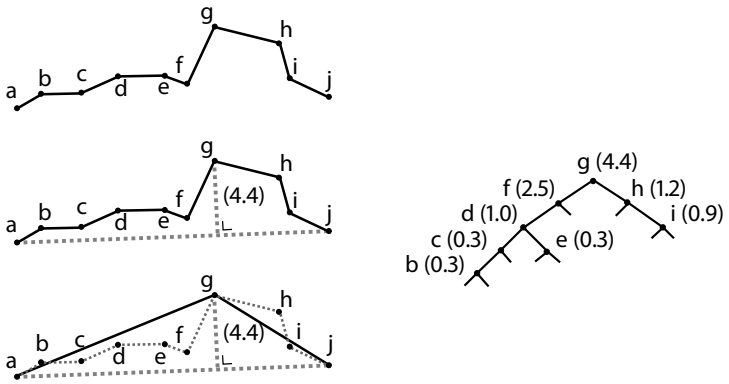
the same identifier, or a link to different identifiers) is important in this instance.' He distinguishes that 'several types of multiple representations can be identified due to: 1. differences in scale; 2. DLM-DCM separation; 3. temporal/lifecycle (plan, realise, modify, remove); and 4. one-object multiple geometric representations (*e. g.* , road area/centre-line, or building a 2D footprint/3D detailed model).' Herewith the term is thus also used for the fact that two kind of geometric representations are stored for the same real world object: *e. g.* for roads one representation is stored for efficient route planning, where roads are represented as polylines representing a graph and one for storage of their physical outline, based on a polygonal description, but both suitable for the same map scale (as mentioned in Zlatanova et al., 2004). Furthermore, these representations are not necessarily targeting visualisation of the information at different map scales, but also at the same scale, which is for example the case with option 3, modelling the temporal differences in the lifecycle of a geographic object.

Another word that is often employed in the literature is *multi-resolution*. Although this is a term that is often regarded as suitable for both geometric and thematic attributes (also thematic attribute values for small scale data sets get a 'coarser' meaning), we regard this terminology to be targeted more at geometric descriptions (common in the field of terrain surface simplification, *e. g.* Hoppe, 1996; Danovaro et al., 2007) and often to be associated with raster data sets (Aplin et al., 1997). Moreover, we see this term more suitable for a map with one level of detail only (where geometric description of the same map objects can be refined or coarsened). Hence, in this respect, we employ the term *multi-scale* to make clear that we intend that there are multiple representations and that these are intended to be used at multiple scales (both for geometric attributes, as well as for thematic attributes).

2.3.2 *Multi-scale data access*

A variety of data structures have been proposed for multi-scale data access. Early attempts focused on making single (line) objects with different resolution available from a single data structure, thereby focusing on storing the result of line simplification. Examples are the Strip-tree (Ballard, 1981), the Multi-Scale Line tree (Jones and Abraham, 1986), the Arc-tree (Günther, 1988, ch. 6), and the Binary Line Generalisation (BLG) tree (van Oosterom, 1990), for which an example is shown in Figure 2.12.

Researchers also started to investigate hierarchical models for feature hierarchies that are strict with respect to the geometric boundary description (*e. g.*



(a) First iteration for the Douglas Peucker algorithm.

(b) Corresponding BLG tree, that stores the result of the line simplification.

Figure 2.12: A polyline and its BLG tree. Note that vertex a and j are not stored in the BLG tree. Taken from van Oosterom and van den Bos (1990).

the boundary of a country is shared with a part of the boundaries of the provinces and the boundary of the provinces is shared with a part of the boundaries of municipalities). The geometric information is shared by all levels in the data structure and no generalisation takes place for the boundaries in this case. Note that the type of mono-scale data sets where the geometry forms a complete subdivision of space is also known as *planar partitions* or *categorical coverages*. Filho et al. (1995) introduce the HPS (hierarchical planar subdivision, illustrated in Figure 2.13) structure, a topological data structure that represents hierarchies of planar subdivisions, providing effcient support for this type of aggregations. Also Rigaux and Scholl (1995) propose a conceptual method to deal with a strict decomposition of the space at multiple scales and partially implement their model with the O_2 DBMS. Plümer and Gröger (1996, 1997) provide a formal data model which allows to establish geometrical and topological integrity for what they term nested maps by the specification of a hierarchical structure. An axiomatic and checkable characterisation of these nested maps is provided, to achieve and maintain integrity for the data structures. Moreover, Frank et al. (2001) show that it is possible to obtain hierarchies from categorical coverages, by focusing on and formalising thematic aggregation through a refinement operator, and that in this way it is possible to produce multi-scale data sets. In current commercial software, the Oracle Spatial Topology product also supports such a

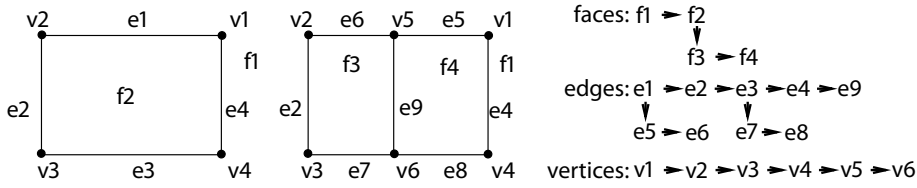


Figure 2.13: An example of the Hierarchical Planar Subdivision (HPS), taken from Filho et al. (1995).

hierarchical topology model, where top features are composed of other features (Kothuri et al., 2007).

Other researchers have looked at descriptions of space at multiple scales, still storing feature hierarchies, but without strict requirements for sharing geometry between the levels stored for the different scales, the so-called MRDBs (where the meaning indeed varies: multiple representation or multiple resolution database). For example, Vangenot et al. (1998) propose a conceptual data model with multi-representation facilities (termed MADS, Modelling of Application Data with Spatio-temporal features, *cf.* Figure 2.14). The important factors that Vangenot (2004) discriminates for multi-representation are a. intended use (which she terms 'viewpoint') and b. resolution. The resolution aspect of data can be subdivided in a spatial component (the geometric description of geographic objects) as well as a semantic component (the desired level of detail, or granularity, for thematic attribute values of the data set).

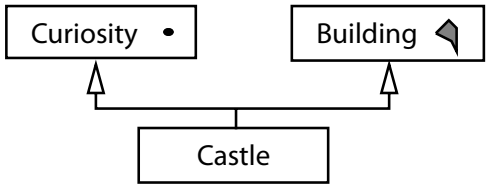


Figure 2.14: An example of a MADS diagram (taken from Vangenot et al., 1998). The conceptual data model supports multiple inheritance: the objects from the class Castle inherit a point geometry from the Curiosity class and an area geometry from the Building class. This way an application can choose between multiple geometry representations. Note that the geometry types are graphically depicted with a small symbol.

A problem with managing several separate representations, stored as independent mono-scale datasets, is the missing links between objects with different

levels of detail (*i. e.* hierarchical relationships are not automatically encoded). Devogele et al. (1996) illustrate how they create these scale-transition relationships between objects by defining a process to build multi-scale databases that consists of three steps: 1. declaring correspondences and conflicts, 2. resolving conflicts and schema merging and, finally, 3. data matching. They tested their approach on two existing mono-scale data sets of road network data. Hampe et al. (2004), who build a MRDB for to support visualisation on mobile devices, acknowledge the approach of linking for building a MRDB, but also show that it is possible to follow another approach by creating new smaller-scale datasets from existing larger-scale ones, which then constitute the new datasets in the MRDB. For this to work, it is necessary that there exists functionality (*i. e.* a generalisation process) that allows the creation of smaller-scale datasets and at the same time immediately establishes the links between the objects for different scales (see Figure 2.15). A commercial solution for creating a MRDB in this way is described by Persson (2004), who employs a quadtree organisation for the different levels of detail.

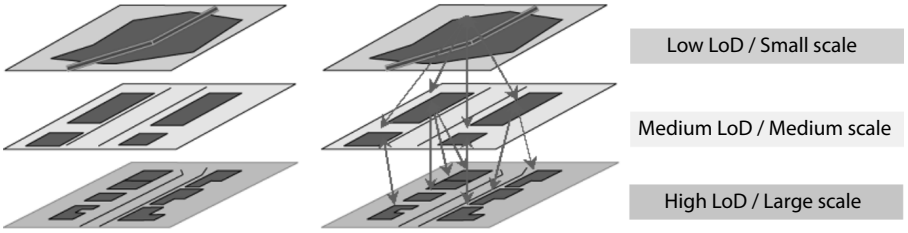


Figure 2.15: An example of a MRDB, taken from Hampe et al. (2004).

The discussion focused hitherto on strategies that produce databases with abrupt changes for discrete levels of detail. Ai and Li (2009) terms these data sets with fixed and discrete levels of detail *scale points* (*i. e.* mono-scale data sets independently stored for every specific map scale). A serious weakness with the approach of datasets for fixed scale points, however, is that these stored datasets produce abrupt changes during interactive use. This is also pointed out by Jones and Ware (2005), who state that 'there are several types of public access map-based websites that allow a user to zoom in and out of a particular region, but at present this is usually based on stepping between independent pre-generalised datasets which may differ markedly in their degree of generalisation. It would be desirable to be able to change the level of detail on such systems in

a smooth and progressive manner rather than the quantum-leap changes that often characterize the current approach.'

2.4 Vario-scale structures

A totally different approach for organising multi-scale data is described in van Oosterom (1990). Here, a variety of data structures (amongst others, the BLG tree and Reactive-tree) for the storage and manipulation of geographic objects at multiple scales are developed. These structures are termed *reactive data structures* and make it possible to perform integrated selections on spatial range and level of detail components of geographic objects stored in large databases, alleviating the 'quantum-leap changes' providing access to the objects at *variable-scale*. Van Oosterom and Schenkelaars (1995) present the development of a system that may be used to interact with a single dataset at a very large range of scales for different detail levels, using these structures. Their system, based on the Postgres DBMS environment, implements three generalisation tools: 1. the BLG-tree for line and area simplification, 2. the Reactive-tree for selection based on importance and location, and 3. the Generalised Area Partitioning (GAP)-tree. Although giving good performance for access, redundancy in this structure is present as the GAP-tree is based on polygons. The non-topological nature also is reflected in the fact that the simplification of the boundaries (obtained by using the BLG-trees) leads to slivers near the boundary of two neighbours.

Inspired by the GAP tree approach, Vermeij (2003) describes the development of a more minimally redundant topological data structure. The data structure consist of two tables: an edge table stores the boundaries of all faces at the level of the input data, and a face table stores thematic information regarding all faces, but holds no geometric description of the objects. Besides the geometric data in the edge table and the thematic information in the face table, both tables also store a 3D bounding box for every object. This bounding box has a 2D extent that encloses the geometric representation of the edge or the geometric representation of the face. The third dimension is used to store information regarding the scale at which an object should be visible in a map. A 2D map is made by taking a slice through this data structure (see Figure 2.16).

Extending this development, van Oosterom (2005) presents an improved variant of the GAP-tree based on topological data structures (called the 'tGAP face tree'). Geometric redundancy between the different levels of detail is avoided as edges are represented by combining BLG-trees. Which edges have to be combined follows from the generalisation process and is stored in a separate

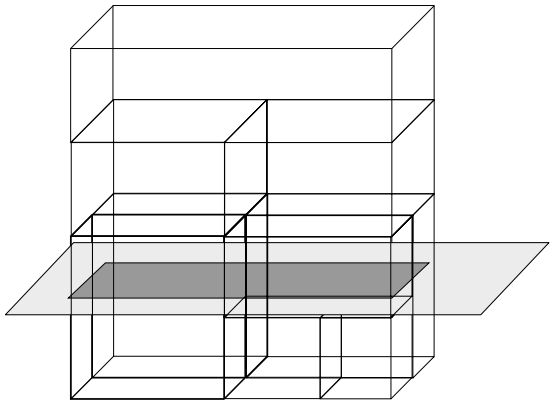
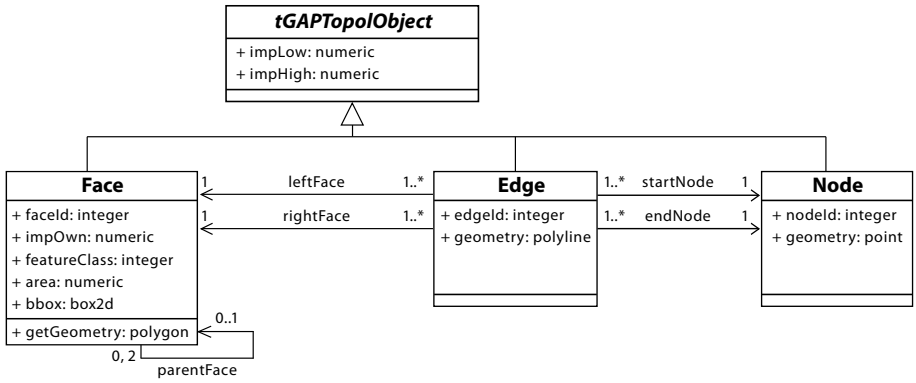


Figure 2.16: The data structure as proposed by Vermeij treats scale as a third dimension (Figure 5.2 of Vermeij, 2003). The plane represents a requested 2D map, and the intersection with the 3D bounding boxes (darker part) then gives the correct edges and faces to display on the map.

data structure. This data structure turns out to be a collection of BLG trees and is therefore called the GAP-edge forest.

As for this thesis the variable-scale tGAP structures (van Oosterom, 2005) are a starting point (see § 1.1), a more detailed explanation follows. The datasets that are currently supported within the tGAP structure have to be modeled as a 2D planar partition, *i. e.* it is a partition of the space in a geometric sense, without gaps and overlaps (the structures are modeled following the space-first principle, § 2.1.1). The physical storage of the data takes place in a database management system (DBMS) using an extended topological node-edge-face data structure. The exact table definitions are given in Figure 2.17.

Each area object of the map is represented by a topological face (this is a one-to-one relation). The level of detail (LoD) can be regarded as third dimension and is represented by the concept of 'importance'. The importance of objects is based on their size and feature classification. E. g. a large forest area can have lower importance than a small city area. A so-called functional spatial index (that is an index defined on the result of a function that runs inside the database system and is applied to one or more columns of some input tables) on a 3D bounding box (bbox) is used to efficiently access the 2D spatial data extended by the third dimension: the importance (or scale) range for which a certain representation is valid.



(a) UML diagram

```
CREATE TABLE tgap_faces (
face_id integer,
parent_face_id integer,
imp_low numeric,
imp_high numeric,
imp_own numeric,
feature_class_id integer,
area numeric,
bbox box2d);
```

(b) Face table

```
CREATE TABLE tgap_edges (
edge_id integer,
imp_low numeric,
imp_high numeric,
start_node_id integer,
end_node_id integer,
left_face_id integer,
right_face_id integer,
geometry geometry);
```

(c) Edge table

```
CREATE TABLE tgap_nodes (
node_id integer,
imp_low numeric,
imp_high numeric,
geometry geometry);
```

(d) Node table

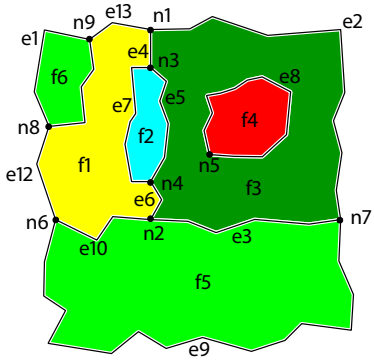
Figure 2.17: An UML diagram and corresponding table definitions in SQL for the classic tGAP structure.

2.4.1 *Filling the face table*

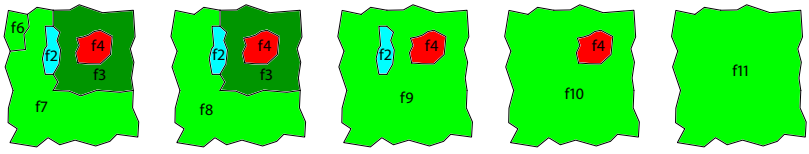
As we want to reduce the LoD for display at smaller scales, we have to generalise our original data. A generalisation process reduces the number of polygonal objects, based on the importance. The object that has the least importance is removed first. Plain removal of the object is not allowed, because a gap would exist after this operation. Therefore we let the most compatible neighbour take the space of the object to be removed. Based on the shared boundary length and the feature class compatibility this neighbour is chosen. The merging operation creates a new object. This new object then has a new identity and is given the feature class of the most compatible neighbour. The importance of this object is recomputed (and several different options have been tested for this: *e. g.* taking the sum of the importance of the two merged objects). This process continues until only one object is left.

During this merging process the importance range for all objects is also created and stored. This range is intimately related to the importance assigned to all faces present at the largest scale. The importance is stored with all the faces as the '`imp_own`' attribute that clearly defines the ordering of the generalisation process. The importance range (stored with an '`imp_low`' and an '`imp_high`' attribute for each face) defines the lifespan of objects in the LoD dimension and allows selection of the right objects at an arbitrary LoD (using an importance level for selection, '`imp_sel`').

The importance range for the objects is created as follows: The objects used as starting point will be assigned an `imp_low` value of 0. The example in Figure 2.18a and Table 2.2 shows that the `imp_low` value of all original faces (1-6) is indeed 0. Then, in each generalisation step, two objects their lifespan will be ended and a new one will be created. In our example, face 1 is the least important face, and is merged with its most compatible neighbour (face 5), a new object (face 7) is formed. Both ended objects are assigned the importance own value of the least important object, named '`imp_remove`', as their importance high attribute (face 1 has an `imp_own` of 150, this is assigned to both face 1 and 5 as `imp_high` value). The new object that is formed in the generalisation step will be assigned the sum of the own importance of the two old objects as `imp_own` value and the `imp_high` value of the removed objects as its `imp_low` value. The resultant of this process is that the sum of all own importance of the original objects is equal to the importance high value of the last remaining object. This means that the sum of importance for all objects valid at any given scale (LoD) in the complete map does not change.



(a) The map of the initial configuration, with `imp_sel` = 0 (note that the nodes, edges, and faces are labeled with their identity)



(b) 150 (c) 325 (d) 395 (e) 505 (f) 610

Figure 2.18: Example map with 6 polygonal regions. Subfigures 2.18b – 2.18f show the map at the `imp_sel` value mentioned in their caption.

Table 2.2: The tGAP face table for the sample data set, which is graphically depicted in Figure 2.18a (note there is a bbox and an area value stored, but this is not shown).

face_id	parent_id	imp_low	imp_high	imp_own	feature_class
1	7	0	150	150	corn
5	7	0	150	750	grass
6	8	0	325	325	grass
3	9	0	395	395	forest
2	10	0	505	505	lake
4	11	0	610	610	town
7	8	150	325	900	grass
8	9	325	395	1225	grass
9	10	395	505	1620	grass
10	11	505	610	2125	grass
11	-1	610	2735	2735	grass

2.4.2 *Filling the node and edge tables*

When merging two faces, the life of the edges between the two old faces is ended by setting their `imp_high` value to the `imp_own` value of the face that is removed (`imp_remove`). The remaining edges are now adjacent to the newly created object, so also these edge versions are terminated (their importance high value is set to `imp_remove`) and new, updated versions for those edges are created (with `imp_remove` as their importance low value). These updated versions get the same identity as before, but with a different left or right face pointer and a new importance low value). In our example edge 10 is removed in the first face merge step, when face 1 is merged to face 5 (the `imp_high` value of edge 10 is set to 150, see table 2.4). Edge 11 is an example of an edge that is changed due to the change of the neighbouring face. This edge was adjacent to face 1, but is after the merge adjacent to face 7. So, a new version of this edge is created.

Furthermore, the nodes that are having a relationship with only two edges after the merge, are as well ended and the incident edges are merged; see the node information from Table 2.3. A new version for those incident edges is created, with merged geometry based on the geometry of the two old edges. This is shown in our example for the edges 9 and 12 (forming a new edge 14) and the edges 3 and 6 (forming the newly created edge 15). In the classic tGAP structure the edge geometry is represented by a BLG-tree. For original (leaf) edges this is a directly stored version. For merged (non-leaf) edges this is a BLG-tree with a new top and references to the two BLG-trees of the child-edges. So there is no

Table 2.3: The tGAP node table. Note that each node has a point geometry, but this is not shown.

node_id	imp_low	imp_high
1	0	2735
2	0	150
3	0	395
4	0	505
5	0	610
6	0	150
7	0	395
8	0	325
9	0	325

redundancy in the storage of geometry, but the result can be having to trace a lot of references during usage of the structure. An alternative therefore is to create a new (redundant) geometric representation of the merged edge (a non-BLG-tree representation). For this new geometry there are two options: 1. keep all original vertices or 2. keep half of the original vertices (after applying line simplification). Both solutions introduce (controlled) geometric redundancy, but will be easier to use.

2.4.3 *Using the structures*

The structures are put to use by providing a spatial extent (for the viewport) and an importance value (for the LoD). The importance value can be derived from a given extent: A smaller extent means more detail to show and finally a lower importance value for querying the data structures with (imagine a user zooming in, more detail can be shown for all objects). Contrary, if a larger extent needs to be shown, due to a user zooming out, a higher importance value needs to be used for selecting less objects. The mapping between importance and spatial extent (map scale) is discussed in detail in § 5.1.

2.5 Progressive data transfer

One of the most important functionalities of a GIS is to *view* stored geographic data. Thus it is important that this interaction type is supported in an effective way and as Shneiderman (1996) states: 'there are many visual design guidelines,

Table 2.4: The classic tGAP edge table with the example content (Note: a. the repeated versions of edges, due to the left/right reference changes, b. The geometry of the edges is stored, but this is again not shown)

edge_id	imp_low	imp_high	left_face	right_face	start_node	end_node
1	0	325	-1	6	8	9
2	0	395	3	-1	7	1
3	0	150	3	5	2	7
4	0	150	3	1	1	3
4	150	325	3	7	1	3
4	325	395	3	8	1	3
5	0	395	3	2	3	4
6	0	150	1	3	2	4
7	0	150	1	2	4	3
7	150	325	7	2	4	3
7	325	395	8	2	4	3
8	0	395	4	3	5	5
8	395	505	4	9	5	5
8	505	610	4	10	5	5
9	0	150	5	-1	6	7
10	0	150	5	1	2	6
11	0	150	6	1	8	9
11	150	325	6	7	8	9
12	0	150	-1	1	6	8
13	0	150	-1	1	9	1
13	150	325	-1	7	9	1
14	150	325	7	3	7	4
14	325	395	8	3	7	4
15	150	325	7	-1	8	7
16	325	395	-1	8	7	1
17	395	505	9	-1	1	1
17	505	610	10	-1	1	1
17	610	2735	11	-1	1	1
18	395	505	9	2	4	4

but the basic principle might be summarized as the Visual Information Seeking Mantra: *overview first, zoom and filter, then details-on-demand*'.

Timpf and Devogele (1997) make an inventory of what kind of new user interface tools can be created in a multi-scale environment, which can help realising this mantra in a geographic viewing setting. An advantage that the MRDBs with a discrete number of scale points bring, is that users can view data at multiple scales at the same time. However, in his thought-provoking keynote held at AutoCarto 2006 Mackaness argues that we have been focusing too much on mimicking the paper map making process, he suggests: 'I think we have built our mapping systems on sand – that sand is the paper map' (Mackaness, 2006, p. 245) and also according to the research plan of NCGIA, 'much of the effort in computer-assisted cartography over the past quarter of a century has been concerned with automating traditional cartographic representations and techniques; there has been too little concern for cartographic methods that go beyond what was possible on the static printed map' (NCGIA, 1989, p. 130). This serious claim is also true for the approach of the MRDBs (with multi-scale datasets at fixed scale points). During interactive use, only abrupt changes can be produced and therefore these solutions can not easily fulfill the criterion of 'details-on-demand' in a smooth manner, because each scale interval requires its own (independent) graphic representation to be transferred. Good examples of progressive data transfer are raster images. When structured correctly these images can be presented to a user relatively quickly (but in a coarse manner) and then can gradually be refined as the user waits longer (see *e. g.* Rauschenbach and Schumann, 1999, where a decomposition scheme for raster images based on wavelets is proposed).

Midtbø and Nordvik (2007) present a user study, that exemplifies that users better preserve their so-called mental map (*i. e.* the image they have in their heads from the objects that are present in a space, *cf.* Misue et al., 1995), when a map is changed in a continuous manner (as opposed to a step-wise manner). For this, progressive transfer is a necessary pre-requisite to reduce waiting time in a distributed environment (such as the Internet). To gain insights in which of these two approaches is best, test persons were asked to localise a point on a digital map after both a step-wise as well as a continuous zoom operation, see Figure 2.19. The results show that continuous zooming creates a better understanding of the geographic space and thus is less confusing for the user.

Van Kreveld (2001) and Ai and Li (2009) also argue that especially for viewing in a digital environment it would be beneficial to have systems supporting multi-scale access together with continuous changes (leading to smooth visualisations).

(a) The point to localise.

(b) Recorded locations after the continuous, smooth zoom operations.

(c) Recorded locations after the step-wise zoom operations.

Figure 2.19: In the experiment presented by Midtbø and Nordvik users were asked to localise a point after a zoom in and out operation had been performed. In the experiment a zoom-in action on the location of the point was shown, the location of the point was then revealed to the user and finally an automatic zoom-out operation was performed by the system. The zoom actions were performed both in a smooth and a step-wise manner. The results show that when a map is changed in a continuous manner, as opposed to changing in a step-wise manner, users can localise the revealed point more accurately (they better preserve their mental image of the map, taken from Midtbø and Nordvik, 2007, p. 295 & 298).

For this to work it is necessary to see generalisation as a process that produces continuous change instead of as one that produces abrupt changes (*e. g.* Danciger et al., 2009). Based on the result of such a process progressive transfer for vector data could be realised. In this respect, the intermediate map scales rather than the predefined scale points in a MRDB, become more important. Recently, some attempts have been made to develop solutions that go in this direction, however, some of them are purely conceptual (*e. g.* Bertolotto and Egenhofer, 2001), and some are only focusing on geometric descriptions of the objects where neither the number of objects nor the type of objects is varied, only the number of coordinates in the geometry, *e. g.* Zhou et al. (2004); Sester and Brenner (2004);

Yang et al. (2007). This shows that progressive data transfer for vector data is still in its infancy.

2.6 Starting points for data at variable scale

This chapter has dealt with the following research question:

1. *What is the state-of-the-art in: 1. multi-scale data management and 2. generalisation of vector data?*

The current state-of-the art of multi-scale data management approaches was reviewed. To summarise the findings: MRDBs are the state-of-the art and store data sets with fixed scale points. That is mono-scale data sets are independently stored for every specific map scale – even when links are used between objects, geometrical attributes are independent for the different scales. Commonly generalisation algorithms are designed such that they can produce these independent data sets.

A serious weakness with the independently stored layers of data in a MRDB is that they produce abrupt changes during interactive use. A totally different approach of organising multi-scale data is using vario-scale data structures that make it possible to perform integrated selections on spatial range and level of detail components of geographic objects, alleviating these abrupt changes and providing access to the objects at variable scale. For this to work it is necessary to have a toolbox of generalisation algorithms that outputs the data at variable scale.

All in all, we can define the starting points for what we see as a vario-scale geo-information environment; such an environment will be one that works under these specific conditions:

1. Enables real time access, based on spatial range selection, to geo-information in a client-server set up in the form of *vector* data as vector data allows interactivity and later binding of exact presentation at the client side;
2. Makes it possible to store, maintain and disseminate data at variable scale — *i. e.* data sets are not only stored for pre-defined scale points, but can provide levels of detail for a whole scale range, at variable scale;
3. Is stored with minimal (geometric) redundancy;
4. Allows progressive transfer and makes continuous zooming possible.

3 FORMALISING VALID VARIO-SCALE DATA

~

As the previous chapter demonstrated starting points for variable-scale data, this chapter first stresses that minimal redundancy also has to do with fundamental choices: it discusses in § 3.1 why for tGAP no specific redundant model is created for creating cartographic output, but that only one model is maintained – the landscape model. Then in § 3.2 we formalise variable-scale data from a mathematical point of view and describe a conceptual model for data storage. This conceptual model paves the way for an implementation of valid variable-scale data in data structures. Furthermore, to realise the conceptual model in practice, there is a need to obtain a valid, single-scale input data set that will be used for starting point to create content for the variable-scale data structures. Therefore, this chapter also proposes a new approach to validate (and automatically repair) 2D input data at the largest available map scale, so that it is suitable as starting point to create content for the variable-scale data structures (§ 3.3). The findings are summarised in § 3.4.

Own publications

This chapter is based on the following own publications:

- Ledoux, H. and Meijers, M. (2010). Validation of planar partitions using constrained triangulations. In *Proceedings of the 14th Joint International Conference on Theory, Data Handling and Modelling in Geospatial Information Science*, pages 51–56, Hong Kong.

- Meijers, M. and van Oosterom, P. (2011). The space-scale cube: An integrated model for 2D polygonal areas and scale. In *28th Urban Data Management Symposium*, volume 38 of *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 95–102.

3.1 A preference for minimal redundancy

In § 2.2.2 it was mentioned that the separation between Digital Landscape Model (DLM) and Digital Cartographic Model (DCM) theoretically is considered as the optimal way of maintaining data sets at multiple scales, but that data producers in practice wrestle with the question what to store explicitly in order to efficiently maintain their geographic databases. The argument we build in this section is that a vario-scale approach facilitates more efficient data production and maintenance, because just a single source data set has to be maintained (in addition to the benefits for end users, such as smooth interaction).

3.1.1 *Four approaches to DLM-DCM*

Geographic data producers are dealing with data capture, data management and visualizations. For creating a digital geographic database (the DLM), objects will be captured in the real world with certain rules applicable for the data capturing process. Besides rules for geometry and topology, like minimum size, geometric accuracy and connectivity of objects, these rules also include object classification and population of attributes carrying thematic semantics. From this database, objects can be selected to produce digital maps. The transformation of objects from the DLM to a visual end-product is described in a DCM. The DCM contains the drawing rules and can be linked to a particular style sheet. To change the looks of the map from for instance the Dutch topographic map into the British or German map style would, in theory, only require a different style sheet. In practice graphic conflicts might still exist and will require corrective action. This action can include generalisation operations such as displacement, but is not directly related to scale change. However, as this transformation process to avoid graphical conflicts is not in all cases straightforward, data producers face the problem of what to store and maintain, only the geographic objects or also the map objects resulting from this transformation? To make the different choices clear, we evaluate the several alternatives that exist to apply the DLM-DCM concept in a multi-scale topographic data environment.

1. TRADITIONAL MAP APPROACH. It is possible to only store the digital map objects. This is not an optimal solution in the sense that it mixes the representation of 'real world' objects with visualization details and makes it difficult to use the objects for geographic analysis. An important disadvantage is also that because of the major (and also interactive) adjustments required to solve cartographic conflicts it is hard to maintain datasets at different scales in an integrated manner. In summary, in a multi-scale environment this approach explicitly stores the DCM at the fixed scales (and there is no DLM or it is somehow 'lost'). In practice, there are no explicit links between the corresponding objects in the various DCM's at relevant scales.

2. PURE DLM/DCM APPROACH. One can store the geographic objects persistently (DLMs) and derive the map objects (DCMs) in an automated way when needed. In this case, the geographic objects are thus not adapted at all for any kind of visualization. Although a lot of research has been carried out, a fully automated solution still has not been reached for such a setup. Operations that are difficult to manage in the transformation from geographic objects to map objects are for example displacement and typification. In the multi-scale environment this approach explicitly stores the various DLM at the fixed scales and the DCM are to be dynamically derived at the corresponding scales. It is attempted to explicitly link the corresponding objects between the various DLMs. However, this is a non-trivial task with complicated correspondences (1-to-many, many-to-many). In practice, it is often omitted making it in turn more difficult to manage consistency.

3. PRE-FABRICATED (DOUBLE) DLM/DCM APPROACH. Another option is to store the geographic objects (DLMs) as well as the map objects (DCMs) explicitly. Both models are thus instantiated and made persistent. This allows for fast access to both the geographic objects and the map objects, but comes with a price of redundancy. In order to maintain consistency more easily, links can be created between the counterpart instances in both DLM & DCM representations. The creation of links and the maintenance of links will become a significant task itself. In the multi-scale environment this results in explicit storage of both the DLM and DCM versions of objects at all relevant scales. In practice this is (nearly) always without explicit links (neither between scales nor between DLM-DCM corresponding objects).

4. IMPROVED DLM/DCM APPROACH. The last option is to store the geographic objects (DLMs) and to adapt them for a default visualization. The map objects (DCMs) are generated on-the-fly by applying relatively simple visualization rules, *i. e.* an 'average' or typical style. This approach differs from the second solution in that non-straightforward visualization aspects, *e. g.* displacements and typification, change the geometry of geographic objects and results of those operations are explicitly stored. This approach has certain inflexibility because it is pre-cooked and if a requested style is far from the 'average' style visualization conflicts might still occur. This change of geographic objects should however only take place within tolerances, specified in the capturing rules with respect to the desired quality of the dataset. Our motivation to allow this kind of distortion is that other geometric distortions take place within smaller scale datasets anyhow; *e. g.* simplification, aggregation, or complete removal of objects. An advantage of this approach is that it is easier to establish links between data sets at different scales which guarantees the consistency optimally. In a multi-scale environment this approach results in explicit storage of the DLM (with controlled DCM influence). The complete DCMs are not explicitly stored but relatively easy derived from corresponding scale DLM. The four options are schematised (in a single scale environment) in Figure 3.1 and classified according to the extent they explicitly represent (*i. e.* store) the DLM and DCM object instances. It becomes clear that the four approaches are complete in the sense that the first three approaches do perfectly fit in one of the four quadrants. The fourth quadrant is empty as it is impossible to have both the DLM and the DCM implicit as there is no data at all. Our fourth option is a kind of balance between options two and three: DLM explicit and a little bit of explicit DCM storage, but with more DCM implicit via rules.

In a multi-scale database the choice how to deal with the geographic objects and map objects is more complicated compared to a single scale only. For example, when the third method is adopted, both models have to be maintained. In a multi-scale setup this doubles the need for maintenance of objects (for each scale, the geographic *and* the map objects). As storage space tends to become cheaper and cheaper this is not a major problem. However, there is also a trend for data producers having to provide higher rates of updates: maintaining more data means more work, so it would be beneficial to minimize the amount of redundancy.

Another problem is that of potential inconsistencies between the different datasets: if maintenance takes place in a not-fully-automated fashion, it is possible for geographic and map objects to become out of sync with each other (and tell

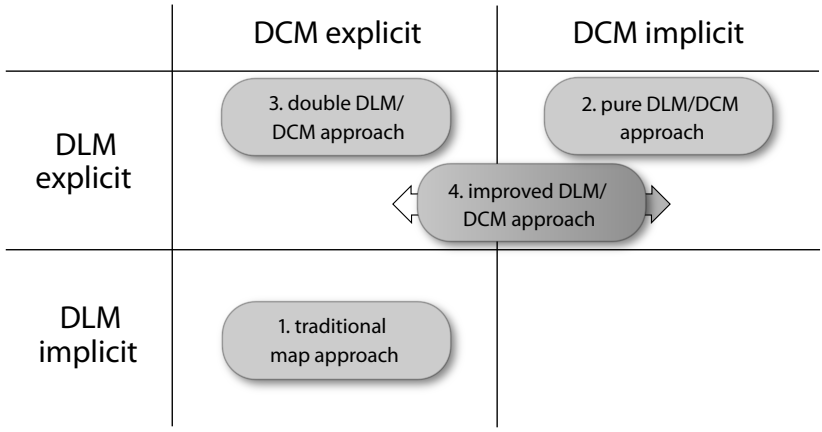


Figure 3.1: Overview of the 4 different DLM/DCM approaches

different stories about the same reality). In a multiple representation environment consistent updating can be easier via the use of explicit links (when present) between corresponding objects. The drawback is that the explicit links have to be maintained as well during the updates.

All in all, the fourth solution we sketched above seems more appropriate for a multi-scale database: Within the quality bounds required for a geographic dataset, the geographic objects should be adapted to make a default visualization easily possible. If a user requires higher accuracy (either geometric or thematic), then the multi-scale setup allows selecting a more detailed and appropriate representation for the application. Therefore to optimally streamline the process of data production for both analysis and map making purposes, we propose to maintain only one geometric model, which also includes a limited number of modifications to apply visualization rules easier, and mention explicitly in data specifications which modifications are allowed. Next, we take up the discussion on managing data with variable-scale structures, as started in § 2.4.

3.1.2 *Variable-scale: an extreme of multi-scale?*

From a data producer's perspective, the multi-scale setup of the fourth option might still not be ideal. We can extend this line of reasoning for the collection of mono-scale data stored in the database: If certain features are present at multiple scales, then why store these representations redundantly? Variable-scale data structures, having no fixed scale data organisation, while delivering

optimal data representations (*e. g.* for visualization) at any requested scale (within the supported range of scales), have been proposed (van Oosterom, 1990; van Oosterom and Schenkelaars, 1995; Vermeij, 2003; van Oosterom, 2005) and tested (Meijers, 2006) to provide an answer to this redundancy problem. Two advantages of variable-scale data structures are: 1. no, or at least very limited, redundancy between scales and 2. also the possibility of 'in-between scales' representations, not only the fixed, stored representations. This clearly brings also benefits from the perspective of a user interacting with the digital data (as demonstrated in § 2.5).

Although variable-scale structures might seem to be an optimal solution, it still can be necessary to include a second representation for certain generalisation events for which the representation in the structure (and/or selection at the required scales) will become too complicated. A few examples: 1. it may be required to store both a road area (at the large scales) and a road centerline (at the smaller) scales and link these representations in the structure to the same real world object, 2. certain generalisation operations require contextual information and are relatively expensive to compute; in these cases it may be more effective to store a second representation, such as a displaced house, and 3. certain types of concepts occur not at the largest scale, but only at smaller scales (*e. g.* roundabout composed by several road areas, or block of buildings composed by individual houses and optionally gardens).

In current map products these decisions when to add extra representations are 'black-white' and bound to the maintained scales: road areas are present on 1:1 000 and road areas and roundabouts on 1:10 000, individual houses on 1:1 000, individual houses and building blocks on 1:10 000 and only building blocks in urban areas on 1:50 000. The classification of objects changes in a continuous way related to geographic scale, and thus there is no reason to take only black-white decisions. It may be far more natural to gradually move from individual houses to building blocks when moving from large to smaller map scales (and thus being able to provide smooth zooming to end users).

It might still be needed to perform cartographic conflict solving, and store the result of such an operation. In this case, a multi-representation database has representations at every scale, as long as an object (or its 'successor') exists, causing significant redundancy and possible causing inconsistency. This is even more true for the approach 'explicit DLM and explicit DCM' (option 3 in Figure 3.1). The vario-scale approach is able to deliver these representations, but without storing them redundantly at all the scales. In fact, the vario-scale approach minimizes redundancy between the DLM and DCM representations (as

in option 4 in Figure 3.1). The vario-scale structure will also have to accomodate multi-representation as there can be different representations of the same object (parts) as argued above; *e. g.* for larger scales a polygon representation and for medium to smaller scales a polyline representation. However, the difference is that the number of multiple-representations is minimised and does not depend on a given number of fixed mono-scale datasets (*i. e.* scale points, § 2.3.2). Therefore inconsistency problems are avoided as much as possible. Moreover, note that the tGAP vario-scale structure has explicit links between multiple-representations; this in contrast to the current practice of most multi-scale and multi-representation solutions. Once operational, the vario-scale approach facilitates most efficient data production and maintenance, because of the fact that just a single source data set is maintained (and this serves both DLM and DCM at all required scales).

3.2 Formalisation of variable-scale partitions

This section introduces the concept of a space-scale partition, which we term the *space-scale cube* (analogous with the space-time cube proposed by Hägerstrand, 1970). Map generalisation of 2D polygonal regions is seen as extrusion into the third dimension (similar to Vermeij, 2003, where this idea was introduced first; *cf.* § 2.4). We formalise what we consider valid data for this cube. To obtain input for a valid 3D space-scale cube (SSC), we first give a formal description of data for a valid 2D planar partition. It is not the intent to redefine the common notion of what is a valid polygon (*cf.* van Oosterom et al., 2003), but to give a formal basis for input data (planar partition) on which we can run a generalisation process for deriving a valid SSC (and later on, in § 3.2.2, the formal description will be extended for the 3D SSC).

The space-scale cube permits us to obtain a conceptual 3D model, where both the dimensions of 2D space and 1D scale (or level of detail) are integrated. From the 3D cube it is possible to extract a consistent 2D map at variable scale (as the cube is one integrated model of space and scale any derived slice from the cube must again be a valid 2D planar partition). Based on the formalisation, we can express what is valid data to be stored in vario-scale data structures.

The cube encodes both a description of space at variable map scale as well as the generalisation process (transitions in the scale dimension). The focus is on maps of polygonal regions in planar partitions, because for a lot of applications polygons are a useful building block for modelling, amongst others, administrat-

ive units, land use maps, land cover maps, soil maps, topographic maps, zoning plans, etcetera.

3.2.1 *A valid 2D partition*

In the formal description that now follows, we use notions from set theory and borrow ideas from the formalisation approach that Erwig and Schneider (1997) describe. Keep in mind that here we aim at a formal and reasonably abstract model, but that this model later will have to be translated in data structures in a computer; an implementation of those data structures are not the main purpose now, but sometimes we will look forward and act if we were already targeting an implementation of the SSC.

SPATIAL BUILDING BLOCKS. We define primitives from which we build a 2D planar partition and a 3D SSC. For the definitions and axioms holds that we only consider cases where the dimension k at maximum is 3 (as we deal with 2D maps and 1D scale).

Definition 1. *Given a k-dimensional Euclidean space $\mathbb{R}^k$, called $\mathcal{X}$.*

Definition 2. *In $\mathcal{X}$, we distinguish $k+1$ distinct types of primitives.*

Definition 3. *We name a i-dimensional primitive p^i, where i is: 0 a node (p^0), 1 a edge (p^1), 2 a face (p^2) and 3 a volume (p^3).*

Definition 4. *The primitives p^i are* non-empty *subsets of points of M^i, that is $p^i \subset M^i$. Here M^i is a* supporting subspace *of dimension i with $M^i \subset \mathcal{X}$ and where for the dimension i holds: $0 \leq i \leq k$. Primitives p^i are* connected *and* open *in M^i (or* relatively open *in $\mathcal{X}$). Open means that for any given point x in a primitive p^i, there exists a real number $\epsilon > 0$, such that, given any other point y in p^i, which has an Euclidean distance smaller than ϵ to x, y also belongs to p (the ϵ distance defines an open ball with infinitesimal small radius). Connected means that for every pair of points $x, y \in p^i$ holds that there is always a path within the interior of p^i that connects the two points.*

What needs to be true for all primitives $\mathcal{P}$ in $\mathcal{X}$ (note that $\mathcal{P}$ is used to name the set with all primitives p that cover $\mathcal{X}$):

Axiom 1. *$\mathcal{X}$ is not empty; $\mathcal{X} \neq \varnothing$.*

Axiom 2. *Every primitive p is part of $\mathcal{X}$; $\forall p \in \mathcal{P} : p \neq \varnothing \wedge \mathcal{X} \cap p = p$.*

Axiom 3. *All primitives are pairwise disjoint, i. e. no points are shared between primitives;* $\forall i, j \in \mathcal{P}, i \neq j : i \cap j = \varnothing$.

Axiom 4. *The union of all primitives* $\mathcal{P}$ *totally covers* $\mathcal{X}$; $\bigcup_{p \in \mathcal{P}} p = \mathcal{X}$.

Based on definitions and axioms, what also has to be true for primitives is that:

Theorem 3.2.1. *There is at least one* k*-dimensional primitive* p *in* $\mathcal{X}$.

Proof. $\mathcal{X}$ is totally covered (Axiom 4) and $\mathcal{X} \neq \varnothing$ (Axiom 1) and for implementation finite primitives are used $\Rightarrow \exists p^k \in \mathcal{X}$ □

Remember that k is the highest dimension, that is, the dimension of the embedding space $\mathcal{X}$. In theory it is possible to cover $\mathcal{X}$ with an infinitely refined space filling curve (but we are targeting a finite implementation, so this here is not relevant).

MAP OBJECTS AND LABELS. To represent real world objects, we now introduce map objects that we term zones. We define the names of the i-dimensional zones as follows.

Definition 5. *We term a* i*-dimensional zone* ω *where* $i = 0$ *a vertex* (ω^0)*,* $i = 1$ *a polyline* (ω^1)*,* $i = 2$ *a polygon* (ω^2) *and* $i = 3$ *a polyhedron* (ω^3)*.*

To discriminate a zone from all other zones, we introduce the concept of labelling zones. A label is meant for giving a proper identity to the zones, *i. e.* a label is a globally unique identifier.

Axiom 5. *All zones have a globally unique label* λ.

Furthermore, we require that zones form a fully bounded and connected entity:

Definition 6. *All zones are closed and connected.*

Apart from all real world objects that are mapped, we also have a zone that represents the unmapped domain, a zone that represents the space outside the mapped domain:

Definition 7. *A zone with label* $\perp$ *represents the space 'outside' the mapped domain.*

Note that the 'outside' zone is allowed to have multiple parts: exactly one part that is not completely bounded (in the direction of infinity) and optionally other parts that are completely bounded (*e. g.* this is the case with exclaves of the 'outside' zone).

To represent a zone in $\mathcal{X}$, we associate the zones with the primitives in $\mathcal{X}$:

Axiom 6. *A zone ω is formed by the union of its associated primitives. These primitives have the same* and *lower dimensions as ω^i. A i-dimensional zone ω is formed by $i \cup i-1 \cup \ldots \cup 0$-dimensional primitives.*

To end up with valid zones, we label all primitives $\mathcal{P}$ based on the labels that the zones have:

Axiom 7. *For a zone ω we require that there is exactly one primitive p with the same label λ and same dimension as the zone.*

And we require that a labeling procedure to give labels to the primitives in $\mathcal{X}$ is carried out as follows:

Axiom 8. *All primitives in $\mathcal{X}$ have to be labelled following this recipe:*

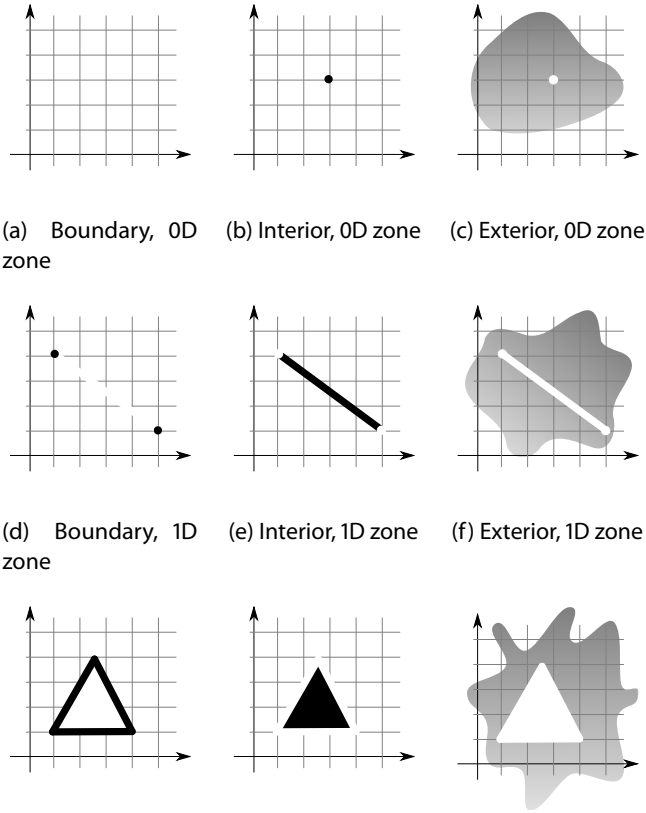
1. *All primitives start with an empty label set.*
2. *For every zone ω^i, add its label λ to all associated primitives of dimension i.*
3. *Add to each remaining primitive p, that has a dimension i smaller than that of its associated zone(s) and has not been labelled, the set of labels of the points that are inside an epsilon disc ϵ centered on the points of p.*

After labelling we can derive the interior, exterior and boundary primitives of an i-dimensional zone ω (Figure 3.2 illustrates these concepts):

Definition 8. *Interior (ω°) of a zone: the associated i-dimensional primitive that has a label set with exactly one label and which is equal to the label of the zone.*

Definition 9. *Boundary ($\partial\omega$) of a zone: the set of $< i$-dimensional primitives that have the label of the zone in its label set.*

Definition 10. *Exterior (ω^-) of a zone: all primitives in $\mathcal{X}$ not having the label of the zone in their label set.*



(a) Boundary, 0D zone (b) Interior, 0D zone (c) Exterior, 0D zone

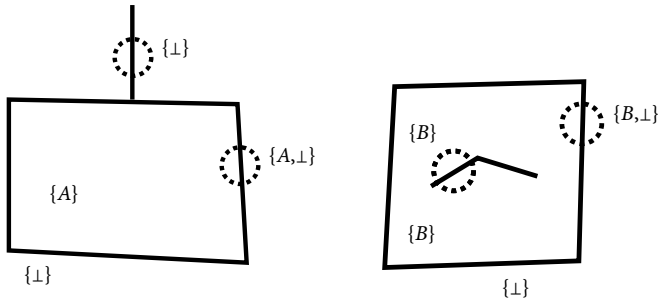
(d) Boundary, 1D zone (e) Interior, 1D zone (f) Exterior, 1D zone

(g) Boundary, 2D zone (h) Interior, 2D zone (i) Exterior, 2D zone

Figure 3.2: Boundary, interior, exterior for 0D, 1D and 2D zones (intent illustrated in black/dark grey). Note that a 0D zone does not have a boundary (by definition there will not be any primitives with dimension <0).

However, we have not yet unambiguously labelled all boundary primitives. For example, what is allowed by the set of axioms and definitions is loose-lying segments inside a polygon, or spikes in the boundary of a polygon. Figure 3.3 illustrates two of those cases. To prevent the degenerate cases, we introduce an additional axiom to restrict valid zones:

Axiom 9. *For a space $\mathcal{X}$ where $k = 2$ the primitives $\mathcal{P} \in \mathcal{X}$ have to be labelled as follows:*



(a) Spike in boundary

(b) Loose-lying segment in interior

Figure 3.3: Degenerate cases will be prevented by correct labelling.

- p^0: *three or more labels*
- p^1: *exactly two labels*
- p^2: *exactly one label*

It has been defined how a zone is represented and is composed by its associated, unambiguously labelled primitives. Loose boundary parts and spikes, such as shown in Figure 3.3, are prevented, as those segments would only have one distinct label and as these degeneracies do not add any extra information (*e. g.* it is already known that that part of the space belongs the polygon) it is useful to prevent these situations. As a final remark, note that this set of statements allows holes to exist in polygonal regions and multi-part polygons are not allowed (Axiom 7).

A VALID 2D PLANAR PARTITION. For a valid 2D planar partition we will not allow zones with lower dimension than k to exist, which means that when $k = 2$ we only allow polygons.

Axiom 10. *For $\mathcal{X}$, we only allow k-dimensional zones.*

We also state that zones are not allowed to overlap each other in their interior:

Axiom 11. *Zones are only allowed to share associated primitives in their boundary and not in their interior: $\forall \omega_1, \omega_2 \in \Omega, \omega_1 \neq \omega_2 : {p^k}_1 \cap {p^k}_2 = \varnothing$ with Ω the set of all zones.*

From the definitions and axioms, we can now derive that the interiors of zones are unambiguously labelled.

Theorem 3.2.2. *The interior of a zone, p^k primitive, is labelled with exactly one label.*

Proof. Following from that primitives are not allowed to overlap (Definition 3), that for every zone there is a labelled primitive having the same dimension (Axiom 7) and that there are no shared primitives between zones (Axiom 11) follows that the interior of a zone has to be labelled with exactly one label. □

TARGETING IMPLEMENTATION. To make it easier to translate the abstract model to a suitable data structure for a computer and to be able to define incidence and adjacency (see next subsection), in addition to Axiom 6 where we state that a zone is a collection of primitives, we add:

Axiom 12. *For every dimension $i \in \{0, \ldots, k\}$ there is at least one primitive associated with zone ω.*

This then translates nicely to data-structures, like DCEL, to encode the incidence relationships of the boundaries (but where the interior point set is not represented explicitly). From the topology point of view a closed ring of a single island zone does not have a node (p^0). From the implementation point of view it is nice if every edge (p^1) starts and ends at a node (two p^0, possibly equal). We will have to add nodes where previously this was not the case and it is necessary to replace Axiom 9 (as such a node has just 2 labels and not 3 or more labels as for topologically defined nodes):

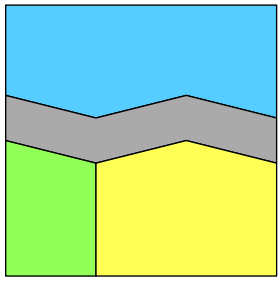
Axiom 13. *For a planar partition where $k = 2$, the primitives $\mathcal{P} \in \mathcal{X}$ have to be labelled as follows:*

- *p^0: two or more labels (two labels only in case of a closed ring, otherwise more than 2 labels)*
- *p^1: exactly two labels*
- *p^2: exactly one label*

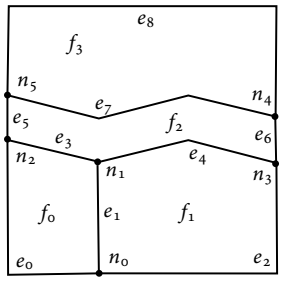
As last requirement, we will define that subsets of points in an edge have to be straight in geometrical sense (not curved).

Definition 11. *Connected subsets of points in an edge (p^1) are on a straight line (in 2D following the equation: $ax + by + c = 0$).*

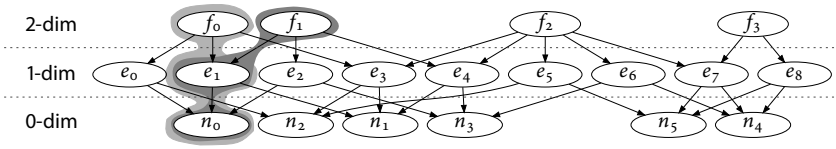
INCIDENCE AND ADJACENCY. From how the primitives are associated with zones, we can obtain a Directed Acyclic Graph (DAG), termed the incidence graph (Lévy and Mallet, 1999). Figure 3.4c shows the incidence graph for the primitives drawn in Figure 3.4b. The highest dimensional primitives will form the top nodes of this directed graph. Edges in the graph represent how a k-dimensional zone is composed of primitives. Primitives having the same dimension are drawn on the same level in the tree structure. Based on the drawing of the DAG, we can define *incidence* relationships of primitives and *adjacency* relationships of zones. With respect to incidence, we can also define the term *degree* of a primitive as the number of incoming graph edges within the DAG.



(a) Planar partition of four polygons



(b) Associated primitives of four 2D zones (polygons).



(c) Incidence graph for the primitives representing the four polygons in (a) and the associated primitives in (b). The 2 polygons for which their interior is represented by f_0 and f_1 are 1-adjacent, as two paths exist that overlap, f_0, e_1, n_0 and f_1, e_1, n_0 (and the highest dimensional primitive in the overlap is at the 1-dim level, therefore 1-adjacent).

Figure 3.4: Incidence and adjacency can be defined by drawing a graph of how the composing primitives for zones are related (after Lévy and Mallet (1999)).

INCIDENCE Two primitives are said to be *incident*, when there exists a path between two primitives in the DAG.

ADJACENCY If two zones have paths in the incidence graph that partly overlap, and the highest dimensional overlapping primitive is at level i, then the

two zones are said to be *i-adjacent*. Furthermore, these two zones will be said to be *strongly connected* when i is exactly one dimension lower than the zone, that is, $i = k - 1$.

3.2.2 *From* 2D *space and* 1D *scale to* 3D *SSC*

As was reviewed in § 2.2.1, map generalisation deals with resolution issues, because the amount of available space is more limited for portrayal of data with a smaller map scale. Here we put forward how we see that the result of a generalisation process of a 2D map can be represented by a description of 3D space and what statements we need to add to the statements of § 3.2.1 to enforce a valid partition of space in 3D.

GENERALISATION OPERATIONS. A generalisation process is often seen as a process, that for an input map outputs a completely new and independent derived map with lower level of detail (*cf.* Mackaness et al., 2007). To make such a generalised representation, we define a set of generalisation operations to derive a representation with less details than the input. For every generalisation operation holds that both its input as well as its output has to be valid according to the set of axioms for 2D partitions (*i. e.* correctly labelled, with no overlaps between the interior of primitives). For the time being, we discriminate 3 types of operations: merge, split and simplification of boundaries.

MERGE Replace the label of a polygonal zone with the label of one other polygonal region that is one of its direct neighbours. Then relabel all primitives, that are not correctly labelled any more (*e. g.* boundary between the two input polygons).

SPLIT Divide a polygonal zone over two or more of its direct neighbours (§ 4.3 gives an example of an algorithm for this operation). Relabel primitives that are not correctly labelled any more (*e. g.* boundary between the two input polygons), introduce new primitives as new boundaries between the direct neighbours and label the primitives with the label of the correct neighbour. A split operation is useful in the case of linear features that are represented at large scale by areas (*e. g.* re-assign different parts of a road or water zone to adjacent neighbours, instead of to one neighbour only, which would have been the case when applying a merge operation).

SIMPLIFICATION (OF BOUNDARIES) Make the geometrical shape of a boundary primitive simpler (*i. e.* less points in the point set). Simplifying the shape has to be carefully performed, without introducing any invalidly labelled primitives, *cf.* § 4.2.

A STEP-WISE SEQUENCE OF GENERALISATION OPERATIONS. Research into multi-representation databases has changed the notion that map generalisation produces independent maps at different scales. With derived and stored links between the objects with different levels of detail the maps are not completely independent. This notion of linking multiple representations is taken a step further by variable-scale data structures by the introduction of a step-wise generalisation process, where a merge operation is iteratively applied and a binary tree structure stores the result of those generalisation operations, *cf.* § 2.3.2 and § 2.4. Storing the sequence of generalisation steps leads to variable-scale data: at every step a progressive reduction of the number of objects to be displayed on the map takes place.

Figures 3.5a to 3.5d show a sequence of generalisation operations. First, a road object is split over its 3 neighbours, then the forest area is merged into neighbouring farmland and finally the boundary between farmland and water area is simplified. Note that the simplification could have taken place as a post-processing of either the merge or split operation, but for clarity this operation is applied on its own. To cope with the results of the split operation we modify the original binary tree structure into a Directed Acyclic Graph (DAG) structure for storing the result. Figure 3.5e illustrates the resulting DAG.

THE SSC AS RESULTING 3D PLANAR PARTITION. We realised that we conceptually can 'stack' all the derived partitions with their primitives on top of each other in a 3D space. We can say that this stacking takes place in an extra 1D dimension, orthogonal to the 2D space, *i. e.* this 1D level-of-detail-dimension describes how 2D map content is reduced, by storing the result of a generalisation process step-by-step. This generalisation process then can be seen as extrusion into the scale dimension: via extrusion the 2D zones in the partition become 3D zones (prisms) living in 3D space. Figure 3.5f gives an illustration of the 3D resulting zones. We can fully describe the resulting SSC with a 3D geometrical approach (where dimension k will become 3) and therefore it is necessary to now replace Axiom 13 for how we label (as this is the only Axiom that is dependent on k):

Axiom 14. *For a planar partition where $k = 3$, the primitives $\mathcal{P} \in \mathcal{X}$ have to be labeled as follows:*

- p^0*: two or more labels*
- p^1*: two or more labels*
- p^2*: exactly two labels*
- p^3*: exactly one label*

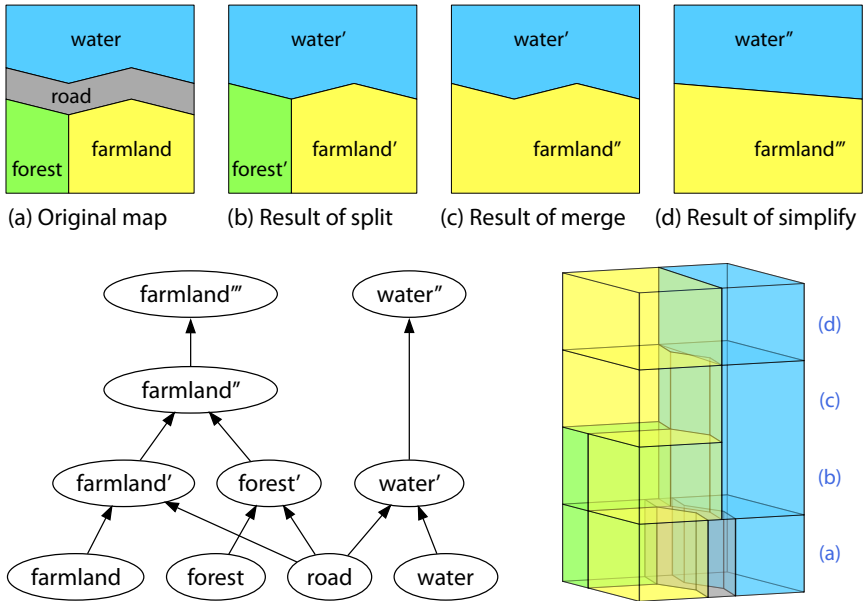
Note that normally in a purely 3D topological setting an edge (p^1) should have three or more labels (and a node four or more). In our case, this is two or more, because we want to preserve the shape of the cube both at the boundary of the domain (*e. g.* corner point of the cube) or at corners of holes (modelled as island shells), extending the reasoning for 2D as mentioned before Axiom 13, p. 65.

Furthermore, in our implementation setting, the labelling is based on the fact that we also want faces in the resulting SSC to be flat (similar to straight subsets in the 2D case):

Definition 12. *Points in a face (p^2) are planar (in 3D following the equation: $ax + by + cz + d = 0$).*

INCIDENCE AND ADJACENCY REVISITED. The definitions and axioms describe what we term a valid space-scale cube in 3D. This cube captures the result of the generalisation process, but from this cube we can also determine what generalisation operations were applied. The split, merge and simplify generalisation operations introduce horizontal and vertical polygons (orthogonal to the space dimension) in the space-scale cube: Extruded boundaries between polygons in 2D (line segments) become vertical (planar) polygons in 3D. As the polyhedrons will have to have a boundary, a 'roof' primitive has to be put on top of a volume – these polygons define the end of the scale range for a polyhedron and will be parallel with the bottom plane of the space-scale cube (see Figure 3.5f). Note that for a single zone, there will be one polyhedron; *e. g.* the water zone extends from top to bottom in the SSC of Figure 3.5f, but for orientation purpose some non-existing interior horizontal faces are depicted.

These parallel polygons define that two volumes are incident with each other in the scale dimension. This means that the top-most volume is the result of



(a) Original map (b) Result of split (c) Result of merge (d) Result of simplify

(e) Generalisation operations applied, describing the lineage of polygons on every map

(f) Space-Scale Cube (SSC). Letters at the right indicate the position of the maps from 3.5a – 3.5d.

Figure 3.5: Illustration of generalisation process and the resulting SSC.

applying a generalisation operation to the other, lower volume. Thus the incidence relationship via horizontal polygons records what generalisation operations were applied, *i. e.* this relationship captures the generalisation process. Based on the incidence relationships *duality* of the volumes can be defined. In the vertical direction, the scale dimension, the duality really reflects the generalisation process ('scale neighbours'), while horizontally 'normal' space neighbours can be obtained.

3.2.3 *Obtaining valid 2D maps from a SSC*

The axioms we have given in § 3.2.1 and 3.2.2 define what we consider a valid 3D space-scale cube. We described the generalisation process as extrusion of a 2D planar partition into a third dimension (the level-of-detail-dimension) leading to a 3D partition of space. Now we want the inverse of this process: deriving a 2D map from the 3D partition. Obtaining this map means to derive a cross-section

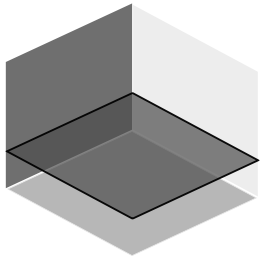
of the 3D cube that is parallel with the bottom plane of the space-scale cube (Figure 3.6a illustrates taking such a slice).

Theorem 3.2.3. *A derived 2D cross-section from a SSC will conform to the axioms for a valid 2D map.*

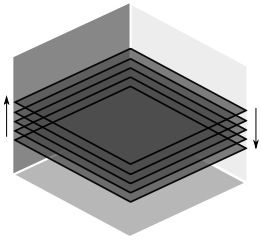
The proof for horizontal slices is straightforward:

Proof. At the bottom of the cube (finest detail, largest scale) the input was already a valid planar partition, every generalisation operation makes sure that the next representation is again a valid planar partition, and in between only simple extrusion in the scale dimension takes place and then slicing is equal to the planar partition just below the slice plane. □

When the cross-section is exactly co-planar with horizontal primitives in the cube, it will be important to be careful when 'slicing' horizontally through the cube at a specific scale. Then the question arises, which of the 2 labels to display in the 2D map. The label to be shown would be the label of the top-most volume (as this then generates a consistent set of 2D maps, *i. e.* also at the bottom plane of the cube data will be shown).



(a) Normal cross-section, parallel to bottom of cube.



(b) Set of cross-sections (moving cross-section up is less detailed data, moving down means more details): replaying steps of the generalisation process.

Figure 3.6: Possible cross-sections.

Figure 3.6b shows another use of the cube: by obtaining a sequence of cross-sections – by moving the plane that defines a cross-section up or down in the cube – it is possible to replay the steps of the generalisation process and thus show how the map changes in the scale dimension. This can be beneficial for

progressive data transfer or smooth display purposes. To efficiently encode differences between two cross-sections might however need different techniques, as data between the first and second cross-section will be mostly similar (this will be investigated in-depth in § 5.3).

3.3 Validation (and repair) of a 2D input partition

This section discusses how to validate an input *planar partition* (*i. e.* make a data set follow the rules given in § 3.2.1 and 3.2.2, so that it is suitable to process with an automated generalisation process to fill the tGAP data structures) and shows that an extension of the approach can be used to automatically repair the data set without manual intervention (see Arroyo Ohori, 2010; Ledoux and Arroyo Ohori, 2011).

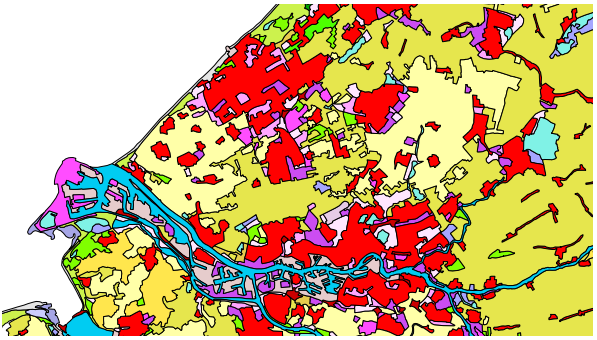


Figure 3.7: Part of the CORINE 2000 dataset for a part of the Netherlands.

As formalised in § 3.2, Figure 3.7 shows that a planar partition is a full tessellation of the plane into non-overlapping zones, which are uniquely labelled. The spatial extent of the mapped domain is thus partitioned into polygons and every location must be covered by one and only one polygon (for certain applications even gaps are not allowed).

The partitions are often represented and stored in a computer as a set of individual polygons to which one or more attributes are attached, and the topological relationships between polygons are not stored (*i. e.* the object-first approach is followed, § 2.1.1). This preferred method of storage follows the Simple Features paradigm, which is an international standard (Herring, 2001, 2006); most databases supporting geometry types (*e. g.* PostGIS) are based on this standard. If a planar partition is modelled via the object-first approach (as a set of individual

polygons), then in practice errors, mistakes and inconsistencies will often be introduced when the planar partition is constructed, updated or exchanged. The inconsistencies most likely to occur are: 1. overlapping polygons (*e. g.* slivers) and 2. gaps between polygons (either in such a way that neighbouring polygons of the gap still touch or that polygons are not connected to the others at all). To validate a partition (and be able to automatically repair it), it is necessary to make a conversion to a model that follows the space-first approach.

Different solutions currently exist, which are based on the construction of the planar graph of the polygons and on the use of geometrical and topological validation rules. The solution we propose, *i. e.* using a constrained triangulation as a supporting structure for validation, is described in § 3.3.1 and has in our opinion several advantages over existing methods. We report in § 3.3.2 on our implementation of the algorithm and on the experiments we have performed. Finally, we discuss the advantages of our method in § 3.3.3.

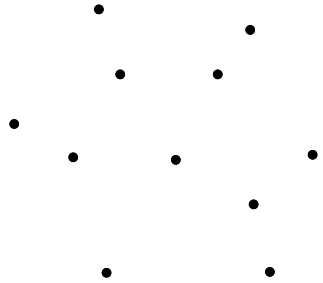
3.3.1 *Validation with a constrained triangulation*

Our approach to validation of planar partitions uses a constrained triangulation (CT) as a supporting structure because CTs are by definition planar partitions (*cf.* Bern and Eppstein, 1992). The workflow of our approach is as follows:

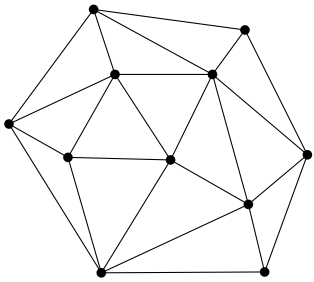
1. the CT of the input segments forming the polygons is constructed;
2. each triangle in the CT is flagged with the globally unique identifier (ID) of the polygon inside which it is located;
3. problems are detected by identifying triangles having no or multiple IDs, and by verifying the connectivity between triangles.

The flagging and the verification of the connectivity of the input polygons is performed by using graph-based algorithms on the dual graph of the CT.

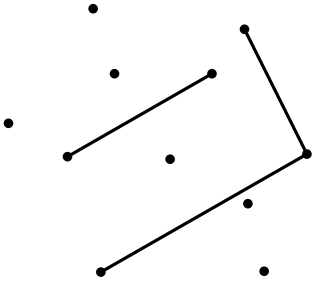
CONSTRAINED TRIANGULATION. A triangulation decomposes an area into triangles that are non-overlapping. As shown in Figure 3.8a and 3.8b, given a set S of points in the plane, a triangulation of S will decompose its convex hull, denoted $\text{conv}(S)$. It is also possible to decompose the convex hull of a set T where points and straight-line segments are present, with a constrained triangulation (CT). In $\text{CT}(T)$ every input segment of T appears as an edge in the triangulation (see Figure 3.8c-3.8d).



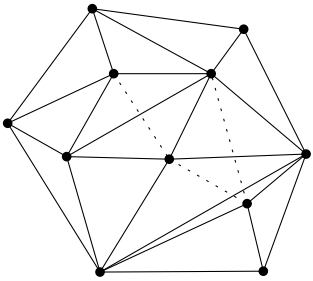
(a) A set S of points in the plane.



(b) A triangulation of S; the union of all the triangles forms conv(S).



(c) The set S with 3 constrained segments.

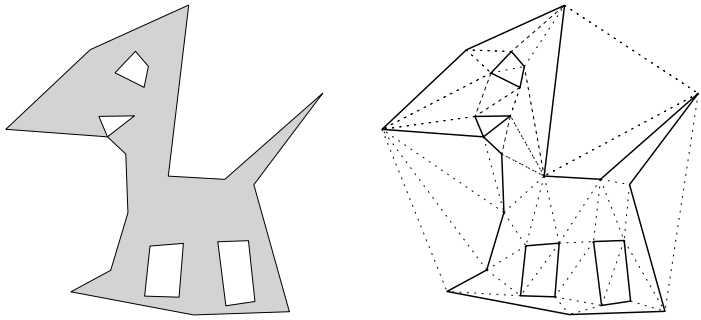


(d) The constrained triangulation of the set of points and segments. The dashed lines are the edges of the triangulation of S that are removed since they are not conform to the input segments.

Figure 3.8: Example of a constrained triangulation (CT)

If T contains segments forming a loop (which defines a polygon), it permits us to triangulate the interior of this loop (*i. e.* a triangulation of the polygon). It is known that any polygon (also with holes) can be triangulated without adding extra vertices (de Berg et al., 2000; Shewchuk, 1997). Figure 3.9 shows an example.

In our approach, the triangulation is performed by constructing a CT of all the segments representing the boundaries (outer + inner) of each polygon. If the set of input polygons forms a planar partition, then each segment will be inserted twice, except those forming the outer boundary of the set of input polygons. This is usually not a problem for triangulation libraries because they ignore points and segments exactly at the same location (as is the case with the solution we use, see § 3.3.2).



(a) A polygon with 4 holes.

(b) The constrained triangulation of the segments of this polygon.

Figure 3.9: A polygon and a possible triangulation. Observe that polygons in GIS can contain holes (many for certain applications) and that one hole touches the outer boundary of the polygon.

FLAGGING TRIANGLES. Flagging triangles means assigning the ID of each polygon to the triangles inside that polygon (the triangles that decompose the polygon). To assign this ID, we first compute one point inside each polygon. This point is what we subsequently call the 'centroid' – observe here that this cannot be always the centre of gravity of the polygon as this could be outside the polygon. Our algorithm therefore finds a location inside the polygon and makes sure that this location is not inside one of the holes of the polygon. This is performed by shooting a virtual ray half way through the polygon – which parts of the ray are inside can be derived by counting the number of intersections of the ray with the segments that form the boundary of the polygon and taking care of rings of the polygon that can touch in one point. The mid point of the largest part of the ray determined to be inside the polygon can now be used as the point on surface ('centroid'). Then for each centroid c we identify the triangle that contains c, and we start a 'walk' on the dual graph of the triangulation, as shown in Figure 3.10.

The walk is a depth-first search (DFS) on the dual graph, and observe that constrained edges in the triangulation will act as blockers for the walk. Observe also that islands are not a problem (see Figure 3.12c).

BIG TRIANGLE. To appropriately flag all the triangles of the CT (those inside the convex hull of the input points/segments but not inside an input polygon) we exploit one particularity of libraries to compute triangulation: the so-called

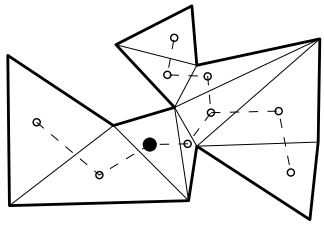


Figure 3.10: One polygon (thick lines) with its triangulation (normal black lines). The dual graph of the triangulation is drawn with dashed lines, and the filled black point is the point on surface ('centroid') of the polygon from where the walk starts.

'big triangle', which is also being called the 'far-away point' (Liu and Snoeyink, 2005). Many implementations indeed assume that the set S of points is entirely contained in a big triangle τ_{big} several times larger than the range of S. Figure 3.11 illustrates the idea.

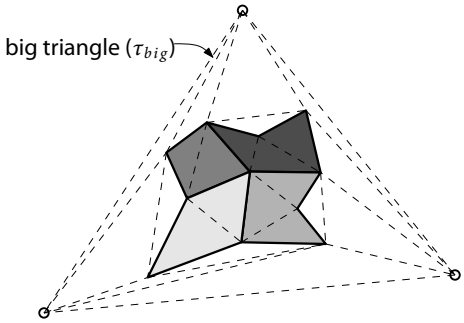


Figure 3.11: The 4 input polygons are triangulated and are inside the big triangle. A walk from one location outside the 4 polygons would appropriately flag as 'universe' the 4 triangles inside the convex hull of the 4 polygons.

With this technique the construction of the CT is always initialised by first constructing τ_{big}, and then the points/segments are inserted. Doing this has many advantages, and is being used by several implementations (Facello, 1995; Mücke, 1998; Boissonnat et al., 2002). To assign an ID 'universe' to the triangles, we simply start at one triangle incident to one vertex of τ_{big} and perform the same walk as for the other polygons.

IDENTIFYING PROBLEMS. If the set of input polygons forms a planar partition then all the triangles will be flagged with one and only one ID. Notice

that because of the big triangle, triangles outside the spatial extent of the planar partitions will be flagged as 'universe'. Notice also that if a polygon contains a hole, then for the planar partition to be valid this hole must be filled completely by another polygon (an island).

If there are gaps and/or overlaps in the input planar partition then some triangles will not be flagged. We can detect these easily by verifying the IDs. Figure 3.12 illustrates one input planar partition that contains 6 polygons; notice that one has an island and that some polygons overlap and that there are also gaps. The walk starting from each centroid is shown in Figure 3.12c, and the resulting flagging of triangles is shown in Figure 3.12d (the grey shadings represent the IDs). When 2 or more polygons overlap then depending on the location of the centroids some triangles will not be flagged (because the constrained edges block the walks) and some triangles can have more than one ID.

Another problem that could arise is when the union of the input polygons forms more than one polygon (*i. e.* the mapped domain is then disconnected). Figure 3.13 shows one example with 5 input polygons: 4 of them form a valid planar partition but one is not connected to the others (thus the 5 polygons do not form a planar partition, strictly speaking). We solve that problem by starting a walk from any centroid, but that walk is not stopped by the constrained, only by the triangles flagged as 'universe'. The connectivity problem simply boils down to ensuring that all the triangles flagged with an ID other than 'universe' can be reached.

3.3.2 *Implementation and experiments*

We implemented the described algorithm. The implementation reads as input either a shapefile (ESRI, 1998) or geometries in well-known text representation (Herring, 2006), and tells the user what problems are present in the input polygons (if any).

For the constrained triangulation, we rely entirely on the implementation of CGAL (to be more precise, we used the Python language bindings of CGAL[1]). Each segment of the input polygons is inserted incrementally in the CT. When 2 segments are identical, the second one is simply ignored. Since the input is formed of individual polygons, it is faster (and simpler) to rely on the spatial indexing scheme of CGAL to detect the duplicate edges than to pre-process them with an auxiliary data structure. We also rely on CGAL for ensuring that

[1] http://cgal-python.gforge.inria.fr

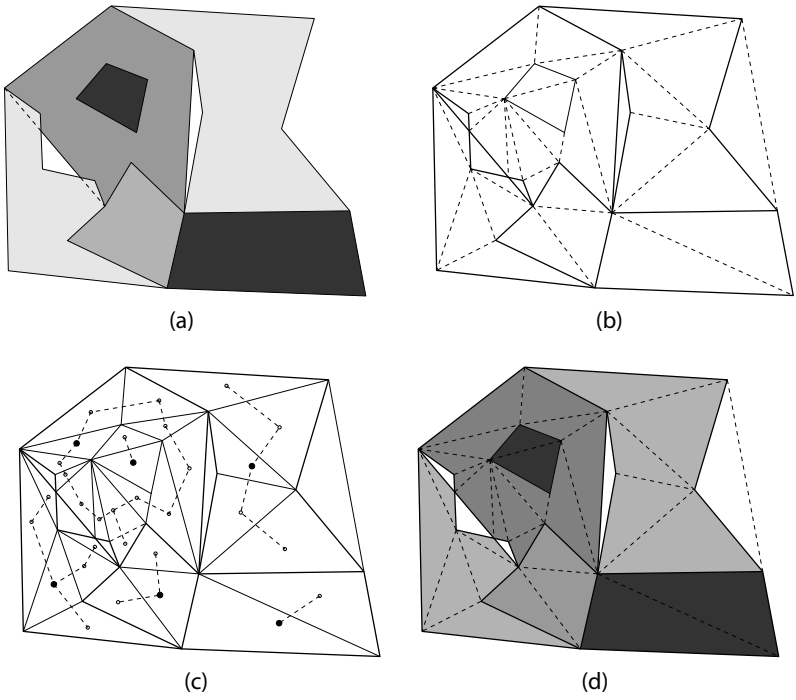


Figure 3.12: (a) Six polygons form the input planar partition. (b) The constrained triangulation of the boundaries of the input polygons. (c) The dual graph of the triangles is drawn with dashed lines; the dark points are the points from which the walk in each polygon starts. (d) The result contains triangles that are not flagged (white triangles). The white triangle on the right is not a problem since it is a 'universe' triangle.

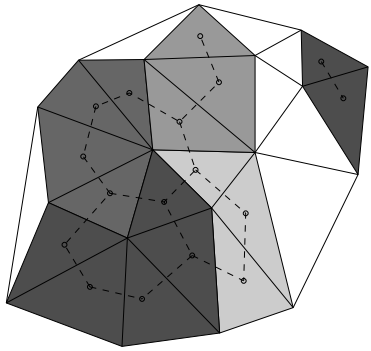
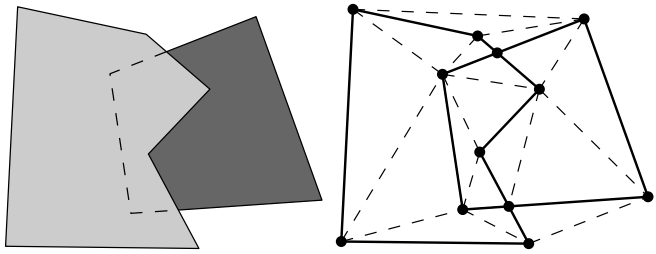


Figure 3.13: Five polygons, with one unconnected to the other ones. The dual graph for the flagged triangles is shown in with dashed lines.

a valid triangulation is formed when 2 or more polygons overlap. As shown in Figure 3.14, if 2 polygons overlap their segments will intersect (which would not be a valid planar graph). However, CGAL has built-in operations to calculate the intersection of 2 segments and to create new sub-segments.

We have tested our implementation with different parts of the CORINE2000 dataset. The dataset is divided into tiles and each tile can be downloaded as a shapefile. Although the specifications of CORINE2000 state that the polygons form a planar partition and that validation rules are used, we found several errors. One example that we found is a case where a polygon had been obviously 'shifted' manually by a user (see Figure 3.15).



(a) Two overlapping polygons. (b) Constrained triangulation of the polygons.

Figure 3.14: Ensuring a valid triangulation means creating new sub-segments in case of intersecting segments.

Figure 3.15: A polygon manually shifted (from CORINE2000 tile E41 N27) – it is overlapping with neighbours on one side and gaps are present on the opposite side (the size of the width of these gaps and overlaps is in the order of approximately 10 to 15 centimeter).

3.3.3 *Advantages (and automated repair)*

The problem of validating a planar partition modelled via the object-first approach is theoretically a simple one: construct the planar graph of the input, and define a set of geometric and topological validation rules. Unfortunately, the implementation of a planar graph construction algorithm and of the validation rules is far from being trivial (especially when the input polygons contain holes) plus no tools exist that automatically try to repair found errors. This section has presented a new algorithm and a successful implementation of it. The approach solves most of the current problems and has in our opinion several advantages:

1. The algorithm is simple and can be implemented easily over a CT library such as CGAL. The only things needed are: 1. to be able to add attributes to triangles (for the IDs); 2. having access to the internal data structure of the triangulation. All the validation rules simply boil down to flagging triangles and graph-based searches.

2. The holes/islands inside polygons are easily handled by the CT. No additional data structure or special mechanisms are necessary, as is the case with planar graph approaches.

3. The implementation can be built over well-known and optimised CT implementations, which are fast and can handle millions of objects. It is known that triangulations of several millions points can be managed in main memory (Amenta et al., 2003; Blandford et al., 2005).

Next to these 3 advantages, Arroyo Ohori (2010) and Ledoux and Arroyo Ohori (2011) show as an extension to this work that if problems are present in the input, the CT can also be used to *automatically repair* the planar partition: different repair operations can be defined to successfully fix gaps and overlaps, which simply involves re-flagging the IDs of problematic triangles (based on some user-defined rules). By 'following' the boundaries between zones with different IDs in the triangulation, it is straightforward to reconstruct the polygons and give them back to a user following the object-first approach (*e. g.* simple features), or to obtain a description encoded in a graph based data structure (such as the node-edge-face structure).

3.4 Closing remarks

In this chapter we have studied the following research questions:

2. *How can we formally describe what is variable-scale geo-information?*

3. *How can we create valid 2D input data as much automated as possible?*

For the formal description we have introduced in § 3.2 the concept of a space-scale partition, termed the space-scale cube (SSC), for which minimal redundancy and consistency are starting points (§ 3.1). We have taken a view of 'map generalisation is extrusion of 2D data into a third, orthogonal dimension'. With an axiomatic approach we have formalised the validity of the partition of space in three dimensions (2D space plus 1D scale). Insights were provided into how to:

1. obtain initial valid data to create the cube representation (by means of a constrained triangulation, *cf.* § 3.3);

2. perform a step-wise generalisation process extruding features into the scale dimension (leading to non-overlapping prism-shaped vario-scale objects);

3. get a valid 2D polygonal map at variable scale from the cube.

The space-scale cube acts as a conceptual and formal framework, guaranteeing valid data at any level of detail present in the cube. As the 3D representations are non-overlapping inconsistencies in the derived 2D maps are also prevented. The space-scale cube thus provides provable consistent representations.

4 IMPROVING VARIABLE-SCALE DATA STRUCTURES

To this chapter the generalisation algorithms to create vario-scale data and the storage of this data into the data structures are central. Firstly, it shows in § 4.1 that some changes in the design are necessary to obtain leaner structures in terms of needed storage space, *i. e.* the theoretical worst case performance of the structures is greatly improved for a merge operation.

Secondly, for the initial tGAP data structures it was proposed to use the BLG-tree to represent edge geometry. No topological consistency guarantees were given, as the BLG-tree is based on the standard Douglas Peucker algorithm and this algorithm does not provide out-of-the-box topological guarantees. Therefore, § 4.2 proposes an algorithm to simultaneously simplify a set of polylines to obtain a topologically consistent result and keep the 'weight' of edge geometry under control (in terms of the number of vertices to be stored in a merged and simplified edge version).

Thirdly, this chapter investigates possibilities to use a triangulation for splitting polygons (§ 4.3) – merging as the only generalisation operator is suboptimal in some cases. Merging is an all or nothing decision (not really fair) and is less suited for certain types of geographical feature classes, *e. g.* elongated features, such as roads and canals. The devised method allows to specify which neighbours get a share of the object to be split, by introducing a weighted split, where also fixed boundaries can be specified, *e. g.* at the outer boundary of the domain. The chapter also proposes a way of evaluating the result of the split operation, to obtain a measure for evaluating the result. This measure can also be used nicely in the implementation of the algorithm to steer the generalisation process (*i. e.* such measures give a goal and are a good tool to formalise the generalisation problem).

The measure reveals weaknesses of the possible solution space imposed by the triangulation. Furthermore, the implications of the proposed algorithms for the data structures (*e. g.* the theoretical worst case of number of edges to be stored and a need for a separate face hierarchy table) are analysed.

Finally, § 4.4 provides an answer to the relevant subquestions and closes with a summary of the results.

Own publications

This chapter is based on the following own publications:

- Meijers, M., van Oosterom, P., and Quak, W. (2009). A storage and transfer efficient data structure for variable scale vector data. In Sester, M., Bernard, L., and Paelke, V., editors, *Advances in GIScience*, Lecture Notes in Geoinformation and Cartography, pages 345–367. Springer Berlin Heidelberg.
- Meijers, M. (2011b). Simultaneous & topologically-safe line simplification for a variable-scale planar partition. In Geertman, S., Reinhardt, W., and Toppen, F., editors, *Advancing Geoinformation Science for a Changing World*, Lecture Notes in Geoinformation and Cartography, pages 337–358. Springer Berlin Heidelberg.
- Meijers, M., Savino, S., and van Oosterom, P. (2011). SplitArea: An algorithm for splitting faces in the context of a hierarchical data structure. Manuscript submitted for review to an academic journal.

4.1 Minimally redundant data storage

This section studies how minimal geometry redundancy influences the layout of the tGAP data structures. It shows that some design changes are necessary to obtain leaner structures in terms of needed storage space, and that the theoretical worst case storage performance of the structures in this respect is greatly improved. It provides empirical evidence that this indeed is the case for real world data.

4.1.1 *Design alternatives for a lean structure*

Here we explore a number of different alternatives for the design of a more data storage and transfer efficient version of the tGAP structure. In van Oosterom

(2005) it was mentioned that fewer columns in the table structure directly implies less storage (a column less to store), but also indirectly saves additional storage space. If scale changes are reflected only in a column that is removed then there is no need for a new row with the new value. This was explained by showing how the tGAP edge table which has four edge-to-edge references (wings) could be reduced in size by removing two edge-to-edge references and only keeping 2 of those references (`edge_lr` and `edge_fl`). In the example data sets this resulted in fewer columns and fewer rows. In the implementation reported in (van Oosterom et al., 2006) all edge-to-edge references were removed, but even in that case the tGAP edge table for a realistic data set still did have up to 15 times more rows than the original edge table (and the theoretic worst case is even $O(n^2)$ with n the number of edges at the largest scale, *cf.* § 4.1.2 – a 'row explosion of edges'). This was mainly due to the changing references to the left and right faces after merging two neighbor faces (and not so much due to merging two existing edges into one new edge or edge-edge references). One of the approaches followed was splitting the edge table into two parts: one part with attributes that do not change in the tGAP structure (*e. g.* geometry, and references to start node and end node) and attributes that do change for different scales/importance values (*e. g.* left and right face references). However, for the changing part of the edges the number of rows is still the same factor higher, only the fixed part is not repeated, saving some storage space. So the aim is to further reduce the required storage, but without loosing performance during the most relevant operations. The most important operation is spatial range selection and visualising a part of the data set at a certain scale. Another important operation is selecting refinement differences between two scales (for a given part of the map) – realising progressive transfer, see § 5.3.

The selection and visualisation of a part of the map at a certain scale, called `imp_sel`, functions as follows: select all faces and edges that overlap the selection rectangle and that have their `imp_low` to `imp_high` range containing `imp_sel`. Note that these are efficient queries (assuming proper 3D spatial clustering and indexing) and this is all the interaction needed with the database server. Then at the client side some topology processing is done: for every face the relevant edges are selected (based on the left/right face references they contain) and rings are created (and if needed inner-rings are properly included in the outer ring). Due to the fact that the edges are selected based on their bbox overlap, not all edges needed to complete the rings of faces partly included in the search rectangle may be present. This is solved by first clipping the selected edges against the selection rectangle (and also splitting the selection rectangle at the intersection points

and creating temporary edges). Together, the clipped edges and the temporary edges created from the selection rectangle are sufficient for forming closed loops, which together cover the whole selected area. For sure every ring contains at least a part of an original edge. The left/right information of such an edge provides a reference to the face which can then be colored according to its classification. This is the setting of the use of the tGAP structure and it is clear that the left and right references are used (for classifying and colouring the faces) despite the fact that it is storage expensive; the 'row explosion of edges'. Now we are going to discuss our three alternatives, `no_lr`, `abox` and `use_tree`, to make the structure more storage efficient.

NO_LR. We started out with a very lean topology data structure: no left/right references (as these caused most of the storage overhead), only edge geometry and a point inside a face region ('spaghetti with meatballs'-approach, *cf.* Figure 4.1); Tables that are stored:

- Nodes (`id`, `location`, `imp_low`, `imp_high`)
- Edges (`id`, `geometry`, `imp_low`, `imp_high`)
- Faces (`id`, `mbr`, `point_on_surface`, `imp_low`, `imp_high`)

The rings are formed based on topology processing without left/right information. There are three steps: 1. clipping and creating rings, 2. assigning island rings to their parent and 3. association of the right identifier with the area (outer ring). Step 1: First edges are clipped against the bounding box of the area that is retrieved (as described for the standard approach above). Then the procedure starts with an arbitrary edge and then starts forming rings by finding all edges incident with the end (node) coordinates (using the geometry of edges), sorting all incident edges based on angle and then takes the first edge left (for counter-clockwise orientation), this process is repeated until the start edge is reached again and the ring is closed. This procedure is then repeated with the next unused edge and a new ring is formed. The ring production terminates when all edges are used twice (once in forward and once in backward direction). Step 2: some of the rings do not have the expected counter-clockwise orientation, and these correspond to islands in the face. The parent outer-ring can be found by a point-in-polygon test (use arbitrary point from inner-ring and finding the smallest outer ring that contains this point). Step 3: Now all faces with holes are created and have to be assigned an identifier. This is done again with a point-in-polygon

test (the point now being the point on surface from the Faces table). For both step 2 and 3 the use of an R-tree (or other type of spatial index) will speed up the point-in-polygon test, building the R-tree once takes $O(n \log n)$ time and then the repeated searches take $O(\log n)$ time.

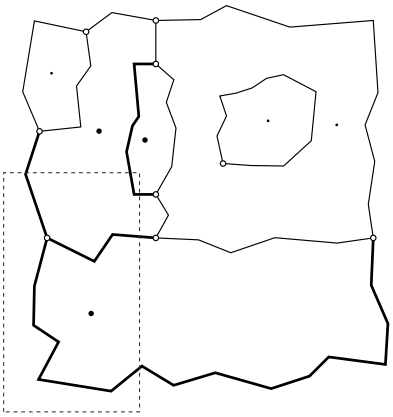


Figure 4.1: The 'spaghetti with meatballs' approach. The retrieved edges (overlapping with the selection rectangle in dashed lines) are given with the thickest lines. After clipping, 3 rings are formed, but the two rings at the top of the selection rectangle can not be labelled with the correct face information as the point on surface for these faces is outside the formed ring.

This approach does work for having a complete extent of area partition within the viewport while visualising. It does not work well when clipping the data: areas can not be reconstructed any more, without having a complete set of edges. An option is to clip the selected edges again (as described above). The result is that now areas can be created covering the selection rectangle. However, faces crossing the boundary might have their point on surface outside the rectangle (and therefore the area can not be identified). There might be some solution to go back to the database server for each unidentified area (*e. g.* by sending the union of the boxes of the unidentified edges to the server, at the server retrieve all faces for which the minimum bounding box overlaps the union of these boxes, then query for all edges inside the bounding boxes of these faces and form polygons based on the retrieved edges and then communicate back which edges belong to which polygon), but this is both a non-trivial query and time expensive.

ABOX. In an attempt to solve the identification of the clipped areas, the adjacency box (van Oosterom and Vijlbrief, 1994), or abox for short, instead of the bbox of edges was proposed for selection. The abox of an edge is the union of

the bbox of the faces left and right of the edge, as illustrated in Figure 4.2. The result is that more edges are selected based on the abox, but for sure these are enough to completely reconstruct all faces in the selected rectangle. However, in order to have the aboxes available in the edge table they have to be maintained (stored). Due to merging of faces in the tGAP structure also the aboxes have to be updated. Actually this is then exactly the same increase in rows as what would be obtained by maintaining the left and right face references. So, there is no real storage reduction, rather the opposite as the abox will take more storage space than the left and right reference. Another drawback is that in some cases the selection based on the abox will retrieve a lot of additional data that is not visible within the retrieved extent at all, *e. g.* long road or river polygons with a large extent for which the abox does overlap with the selection rectangle, but no part of the objects is inside the rectangle. This then will have a negative impact on performance (more data needs to be retrieved, as well as processed client side). The advantage of the abox solution is that it allows easier reconstruction of faces at the client side resulting in full unclipped areas. In theory the explicit storage of aboxes might be avoided by introducing them in a view which uses a function to compute the abox. But again this is non-trivial without the left and right references. Therefore we concluded that this was also not the ideal solution and continued investigating another alternative with fewer drawbacks.

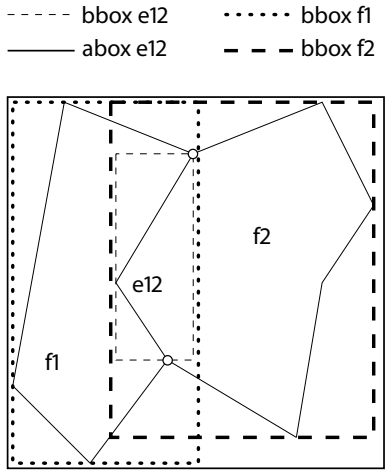


Figure 4.2: Adjacency box (abox)

USE_TREE. Looking at edges that are changed due to changes in the left and right side information (and not in the edge geometry), we might consider merging the rows related to the same edge in one row. This results in no change for the geometry, start and end nodes, and id attributes. The `imp_low` and `imp_high` attributes contain the union of all imp ranges of the edge (which are per definition adjacent ranges). The next question is what to do with the differences in left and right references? Store the left/right reference corresponding to the lowest imp range or to the highest imp range? Take for example edge 4 in Table 2.4 (p. 49) and 4.1 (p. 92): storing the right face reference corresponding to the lowest imp range $[0 - 150)$ would imply a reference to face 1, and storing it related to the highest imp range $[325 - 395)$ would result in a reference to face 8. It was decided to store the left and right face references related to the lowest imp-range, for reasons that will be explained below when assigning the proper identity to the created areas. Anyhow, just storing only rows for edges that are really new (because these edges are merged) saves a lot of storage (rows) as will be explained in § 4.1.2 (the number of rows is for sure always below a factor 2 as edges are merged pairwise). The left/right information and the tGAP face-tree can then be exploited to properly identify the areas at a certain importance level (scale). With this solution we have combined both the requirement to be storage efficient (as the factor 15 of records, as found in earlier tests with real world data, in the edge table is solved), while still having an efficient solution for the most relevant operation (visualisation).

The identification of areas in a given search rectangle of a specified importance level `imp_sel` proceeds as follows. All edges are retrieved a. based on a selection rectangle and b. having an imp range that includes `imp_sel`. The faces are also selected based on these two criteria. Then the clipping is applied to the edges and rings are created as described above and inner-rings are again assigned to outer-rings. During the creation of rings the left/right information is used to find the identity of the face. As the edges carry the left/right information of the lowest imp-range (which may be below the requested `imp_sel`) not all edges directly have a pointer to the correct face (that is at the requested `imp_sel` level). In many cases however there will be at least one edge with the proper (with respect to `imp_sel`) left/right information and this is then indeed the identity of the area. In some cases this information is not present (1. when this edge is outside the selection rectangle, 2. when an island is not yet merged with its parent). In these cases the referred face (and the corresponding edge) with the highest `imp_low` level is used as start in the tGAP face-tree and the tree is traversed upwards until

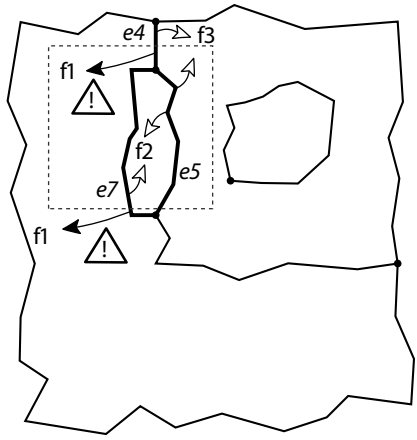


Figure 4.3: Rewriting of face-id's with the use_tree variant. Edges are retrieved, at `imp_sel` = 330, based on their bounding box and the selection rectangle (dashed). After clipping, only the thickest lines are used for forming rings. The most-left ring is formed based on edge 4, 7 and temporary edges stemming from the selection rectangle. Both edges 4 and 7 do not point to the correct neighboring face and rewriting has to take place (face 1 is rewritten using the tGAP face tree as face 8).

the face identifier at the right imp level is found. Figure 4.3 show an example of a case for which it is necessary to lookup the correct face identifier.

The final layout of data structure is (again) based on topology and has the following tables:

- Nodes (`id`, `geometry`, `imp_low`, `imp_high`)
- Edges (`id`, `start_node`, `end_node`, `left_face_lowest_imp`, `right_face_lowest_imp`, `geometry`, `imp_low`, `imp_high`)
- Faces (`id`, `feature_class`, `mbr`, `imp_low`, `imp_high`, `imp_own`)

See Table 2.2, 2.3 (see p. 48) and 4.1 (p. 92) for the sample data set in Figure 2.18a (p. 46), the sample map, and Figure 4.4 (p. 91), a visual representation of the tGAP face-tree. Note that already for this simple 'toy' example the number of rows in the edge table was reduced from 29 to 18 (in § 4.1.3 real world data is used).

The drawback of using the tGAP face-tree is that this tree is not present at the client side (after the two face and edge selection queries). An efficient solution is to send a third query to the server requesting the 'rewriting' of the face-id's which

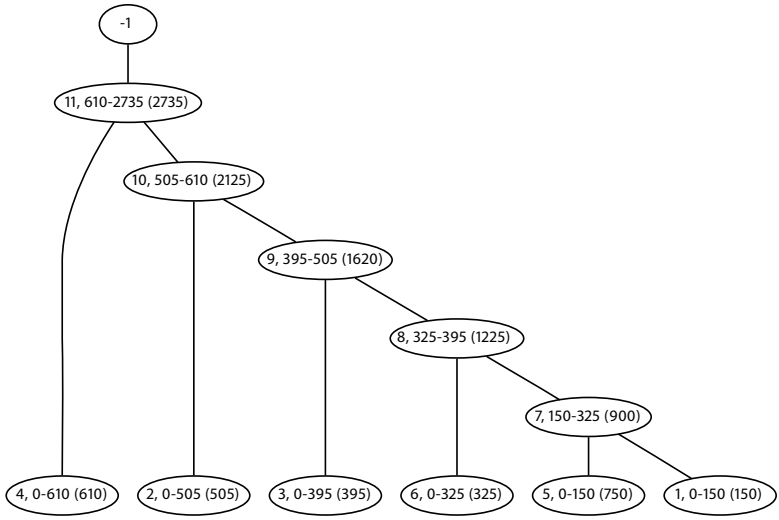


Figure 4.4: The tGAP face-tree, corresponding to the data set of Figure 2.18a, encoding how area features are progressively merged.

correspond to a too low imp level and get back face-id's that correspond with `imp_sel`. An easier solution is not to draw these faces at all: the drawback is of course that white spots will occur on the map (most often near the boundaries of the selection rectangle). A more difficult solution could be to build the relevant parts of the tGAP face tree gradually at client side, and when enough data has been retrieved, colour the white spots.

4.1.2 *Theoretical numbers for faces and edges*

In the previous section we sketched a more optimal solution for storing data in the edge table. Here, we continue our investigations by finding the theoretical upper bounds after filling the data structures for both the classic and the lean variant. These bounds are expressed in numbers of edges (e) and faces[1] (f) present in the original dataset.

Lemma 4.1.1. *The number of total faces stored in the tGAP structure is, after the generalisation process, equal to:*

$$2 \cdot f - 1$$

[1] Numbers for faces here do *not* include the concept of a universal face

Table 4.1: The lean tGAP edge table with the example content (note the geometry/line is not displayed but present in the structure)

edge_id	imp_low	imp_high	lf_low_imp	rf_low_imp	start_node	end_node
1	0	325	-1	6	8	9
2	0	395	3	-1	7	1
3	0	150	3	5	2	7
4	0	395	3	1	1	3
5	0	395	3	2	3	4
6	0	150	1	3	2	4
7	0	395	1	2	4	3
8	0	610	4	3	5	5
9	0	150	5	-1	6	7
10	0	150	5	1	2	6
11	0	325	6	1	8	9
12	0	150	-1	1	6	8
13	0	325	-1	1	9	1
14	150	395	7	3	7	4
15	150	325	7	-1	8	7
16	325	395	-1	8	7	1
17	395	2735	9	-1	1	1
18	395	505	9	2	4	4

Proof. The generalisation process starts with f original faces. Merging can be executed until we have only one face left. This means we can merge u times, with $u = f - 1$. Each time we merge two faces, we add 1 new face to f. In total we add u times a face to f. The total number of faces will thus be $u + f$, or, expressed differently:

$$2 \cdot f - 1$$

□

Lemma 4.1.2. *The total number of edges in the classic tGAP structure, that is, filled with the original method (generating all intermediate edge versions), is at most:*

$$\sum_{i=0}^{f-1} e - i$$

Proof. Faces are merged in $f - 1$ steps. Faces that are neighbors are adjacent in, at least, one edge (due to the planar map criterion). With each merge step thus at least one edge will disappear. The worst case is that in every generalisation step

all remaining edges will be duplicated due to new left/right references. These observation lead to Lemma 4.1.2. □

Corollary 4.1.3. *The total number of edges in the classic tGAP structure, that is, filled with the original method (generating all intermediate edge versions), can be quadratic:*

$$O(e^2)$$

Proof. Assume a configuration (similar to the one shown in Figure 4.5) with one big face (described by one big edge) containing many small islands (small faces, each one described by one edge). Then in the summation of Lemma 4.1.2 it is clear that $f = e$ and this results in a total of $e \cdot (e + 1)/2 = O(e^2)$ edges. □

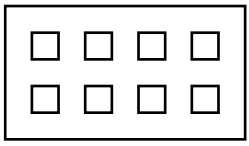


Figure 4.5: A worst case initial configuration

Our new, lean approach performs significantly better in this respect:

Lemma 4.1.4. *The total number of edges stored in the tGAP structure, filled with the new 'use_tree' method, is dependent on the number of original edges and faces and is at most:*

$$2 \cdot e - f$$

Proof. All original edges will be present once in the output. The merging of edges is what brings new edge versions.

Suppose this edge merging is performed with all start edges as input, as follows: two edges will be merged at a time, until 1 edge is left. The resultant of this process is then one large polyline with self-intersections. The total number of edges in the output will then be at most two times the original number of edges minus 1 (*cf.* Lemma 4.1.1).

However, in each generalisation step, to merge two faces, at least one edge has to be removed, *i. e.* the number of edges to be removed is the number of faces minus 1 (as that is the amount of merges that will take place). Taking both steps into account, results in a number of edges that is equal to:

$$(2 \cdot e - 1) - (f - 1) = 2 \cdot e - f$$

This is a worst case estimate, as in each merge step more than one edge might be removed. □

4.1.3 *Experiment and results*

To judge whether our theoretical investigations described above would yield valid results in practice, we implemented both variants (classic and lean) of filling the tGAP structures. Table 4.2 highlights the number of faces and edges for the original data, the amount of data after using the classic variant and for the lean variant of filling the structures.

Table 4.2: Number of faces and edges for the different test data sets. Numbers are shown for the original data, the data after using the classic variant of filling (*i. e.* edge version duplication) and for the lean variant (only each first edge version is stored). Both the Sample data set and the Archipelago set with 2 500 islands were created artificially. The other data sets contain real world data. The Buchholz data set contains land cover data. The Cadastral data set consists of parcels and the Amsterdam data set contains topographical data.

data set	original faces	tGAP faces	original edges	tGAP edges classic		tGAP edges lean	
Artificial data							
1. Sample	6	11	13	29	2.2×	18	1.4×
2. Islands	2 501	5 001	2 501	3 128 751	1251×	2 501	1.0×
Real world data							
3. Buchholz	5 537	11 073	16 592	77 585	4.7×	26 787	1.6×
4. Parcels	50 238	100 475	178 815	2 663 338	15×*	264 950	1.5×
5. Amsterdam	173 187	346 373	426 917	3 544 232	8.3×	630 944	1.5×

* The factor 15 as mentioned at the start of § 4.1.1.

To verify the lemma's from § 4.1.2, we started by creating two artificial test data sets (1 and 2). It is clear that the number of faces follows Lemma 4.1.1 in all cases, independently from which filling variant is used. Further, it is also clear that our concerns with respect to the duplication of edge rows are valid: To see whether the upper bound for the number of edges could exist in practice, we created a data set (set 2) consisting of one polygon containing 2 500 islands polygons. Each polygon was described with one line, resulting in 2 501 faces and 2 501 edges. In practice, this data set can occur when an archipelago is mapped and in which all islands are merged to the surrounding ocean. The factor for

the classic variant of filling is an abominable result (on average each edge is duplicated 1 251 times, that indeed is $O(e^2)$), especially compared to the lean version (in which only the original edge versions are present once).

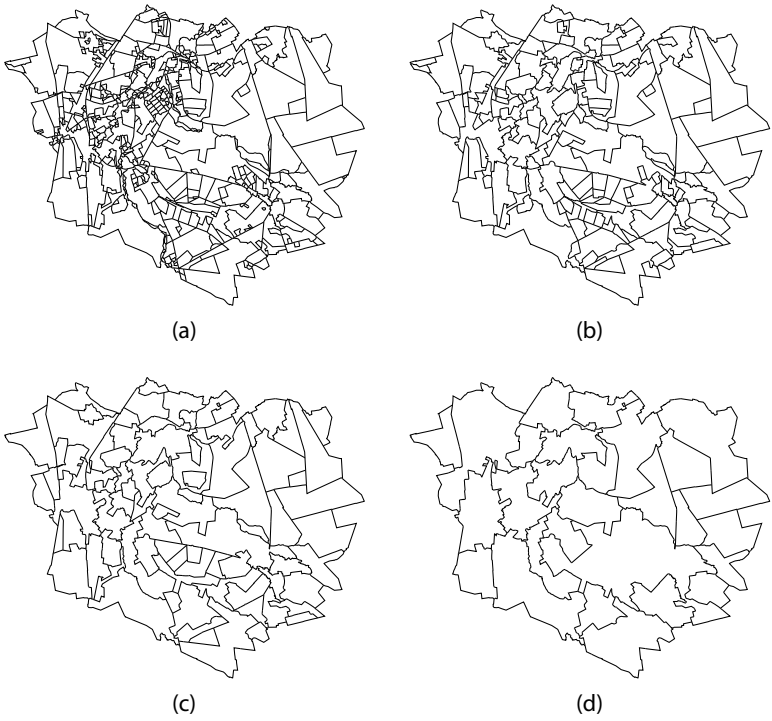


Figure 4.6: A clip of the Buchholz data set, visualised with different `imp_sel` values.

Besides artificial data sets we also used some data sets containing real world data. That the factors are higher for the sets 4 and 5 compared to the factor for 3, is explainable by the fact that set 3, shown in Figure 4.6, does contain only few island polygons, while set 4 and 5 do contain some polygons with a few hundred islands (although these datasets are still relatively small); Filling the structures in the classic way leads then to even more duplicated edge rows. Although the theoretical upper bounds are, by far, not met by these data sets, the factors of the classic filling variant are still high (and we suspect that this will even be worse for larger data sets), while our new variant significantly performs better (clearly below the theoretical upper bound of factor 2).

4.2 Simultaneous, topologically-safe line simplification

This section introduces a line simplification method, based on the method proposed by Kulik et al. (2005), that does not introduce any topological errors for a valid input planar partition.

4.2.1 *A brief review of known methods*

In literature a multiplicity of methods is known to simplify (cartographic) lines. Saalfeld (1999) gives a classification of polyline simplification methods:

IN VACUO modify one polyline in isolation, possibly leading to topological conflicts that have to be resolved by post-processing;

EN SUITE modify a single polyline in context (looking at topological relationships with nearby features);

EN MASS modify the complete collection of polylines and all other features of a map, taking the topological relationships into consideration during adjustment.

Apart from the classification given by Saalfeld, the algorithms can be divided in two main groups: 1. using *refinement* (*i. e.* an approach from coarse to fine, starting with a minimal approximation of a polyline and then adding the most significant points, until a prescribed tolerance is met) or 2. using *decimation* (*i. e.* an approach which starts with the most detailed version of a polyline and then eliminates the least important points first, thus going from fine to coarse).

The best known algorithm for simplifying lines, *in vacuo* using a refinement approach, is the Douglas-Peucker line simplification (Ramer, 1972; Douglas and Peucker, 1973). It was modified by Saalfeld (1999) to work on a polyline *en suite*. Da Silva and Wu (2006) argued that topological errors could still occur and gave an extension to the suggested approach. However, their approach is not explicitly designed for keeping a planar partition valid as they cannot ensure that polygonal areas keep size (*i. e.* do not collapse so that their area becomes 0). Another en suite algorithm was developed by de Berg et al. (1998). The core of the algorithm is also used for simplifying polylines in a planar subdivision (en mass), but each polyline in the main loop of their algorithm is still simplified en suite (so the simplification outcome depends on the order of processing the polygonal chains).

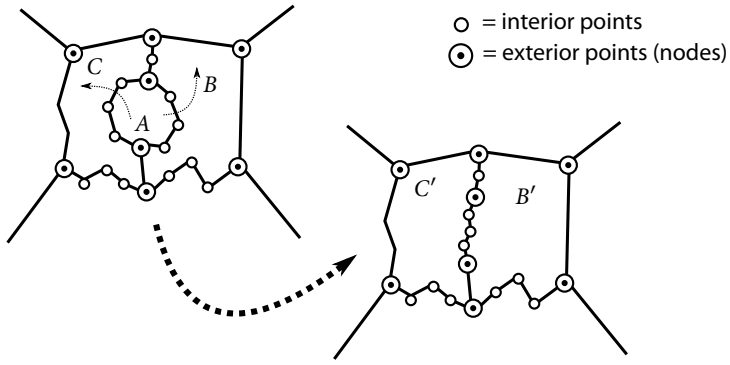
A better approach in this respect is the one given by Kulik et al. (2005), which simplify the polylines simultaneously (thus not one after the other). The basis for their recipe is the algorithm described by Visvalingam and Whyatt (1993). It uses decimation for simplifying lines in vacuo. The algorithm of Visvalingam and Whyatt was extended by Barkowsky et al. (2000) using different criteria for the order in which points will be removed (leading to different shapes as output). Kulik et al. (2005) developed the approach for simplifying polylines en mass, but they consider only a connected graph for the topology aware simplification (the algorithm described in this section also deals with an unconnected graph, in case of islands in the polygonal areas). Furthermore, in their description of the algorithm they show that it is necessary to check after every simplification step whether points that could not be removed before, are now allowed to be simplified. It appears that their algorithm in this case can lead to quadratic running times. Also, it is not clear in their description how near points that might influence the simplification can be obtained efficiently in an implementation.

As a last remark, it must be noted that none of the methods described above discuss line generalization in a stepwise generalization process, thus intermingled with other generalization operations, such as merging and splitting of polygonal areas (aggregation) in a planar partition for a variable-scale context.

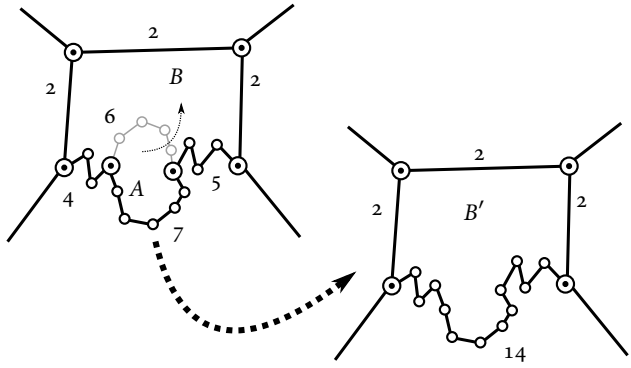
4.2.2 *The need for line simplification*

To generate generalised data for the tGAP structures, we employ a stepwise map generalisation process: this process records all states of the planar partition after applying a generalisation operator in the tGAP face tree structure. With the obtained hierarchy the average number of polygonal areas shown on a user's screen can be kept roughly equal, independent of the size of the user's view port, by varying the level of detail when a user zooms in or out, thus showing objects closer to or further from the top of the hierarchical structure. The removal of area objects (by merging or splitting them, § 4.3) leads to fewer objects per region. A user zooming out leads to an enlarged view port and ascending the hierarchy can supply an equal number of more generalised (thus larger) area objects to an end user, similar to the number before the zoom action.

However, the related boundaries of the polygonal objects will get more coordinates (per object) if the original boundary geometry is kept and not simplified. As can be observed in Figure 4.7a, a split operation, *e. g.* implemented using triangulation, as described in § 4.3, can lead to unnecessary nodes in the topology graph (nodes where degree = 2). This also happens when an area merge



(a) Splitting of polygonal areas leads to unwanted nodes.



(b) Unwanted nodes also result from merging two polygonal areas. Furthermore, the average number of coordinates per boundary increases.

Figure 4.7: Both Figure 4.7a and 4.7b show that unwanted nodes can exist after a split or a merge operation. Furthermore, it is illustrated that not simplifying merged edges leads to an increased average number of coordinates per boundary.

operation is performed (see Figure 4.7b). Therefore, we merge the boundaries that are incident to those nodes. However, this merging leads to boundaries with more coordinates.

The increase in the number of coordinates is illustrated by the example shown in Figure 4.7b. Polygonal areas A and B are merged. This leads to two nodes with *degree* = 2. On average the number of coordinates before the area merge operation in the boundaries is $(2 + 2 + 2 + 4 + 7 + 5 + 6)/7 = 28/7 = 4$. After the merge, we can remove the two nodes having *degree* = 2 and thus merge the boundaries which leads to: $4 + 7 + 5 - 2 = 14$ coordinates for this new boundary. On average the number of coordinates of all the boundaries is: $(14 + 2 + 2 + 2)/4 = 20/4 = 5$, which is more than before the merge operation. The polylines thus have to be simplified.

According to our rule of thumb, that we want to keep the amount of information (density) shown per viewport on average constant (see first paragraph of this section and, for more details, § 5.1), we also will have to keep the number of vertices per polyline roughly equal. By merging edges (after the generalisation operation on the polygonal areas), we will try to keep the number of vertices in the new edges approximately the same as in the old edges (before the polyline merge).

4.2.3 *An overview of the simplification approach*

We employ a decimation approach for simplifying the selected boundary polylines. The order of removing points is determined by a weight value w, which we calculate for each interior point of the polylines to be simplified. For calculating the weight values, we get 3 consecutive points, p_{i-1}, p_i, p_{i+1} from a polyline forming a triangle τ. In our implementation the weight is calculated based on the area of the associated triangle τ, *i. e.* $\triangle(p_{i-1}, p_i, p_{i+1})$, and therefore completely based on geometry (*cf.* Visvalingam and Whyatt, 1993). There could be more geometrical criteria, like sharpness of turn angle, length of sides, ratio of sides, etcetera (alternatives are discussed in Barkowsky et al., 2000). Note that Kulik et al. (2005) also assign a 'semantic' weight per point (next to the 'geometric' weight), which they base on the feature class of the polyline, where the point belongs to and is also dependent on the user's application domain.

The exterior points of the polylines (forming a node in the planar partition) can not be removed. At each step, the interior point p_i having the overall lowest weight value will be removed, leading to a 'collapse' of triangle τ into a *short cut* $\overline{p_{i-1}, p_{i+1}}$. Our simplification strategy has to obey the requirements that we have

given for a planar partition, thus not all short cuts will be allowed. We observed that a spatial configuration that leads to a problem at first, might be changed later, because another point has been removed (that was preventing a collapse). The algorithm re-tries removal of the blocked point in this case.

4.2.4 *Dynamic and static polylines*

Figure 4.8 shows that we distinguish two types of polylines that play a role in the simplification: 1. *dynamic* polylines that will be simplified, *i. e.* interior points can be removed as long as no intersections or degeneracies in the planar partition requirements are caused by this removal and 2. *static* polylines that will not be simplified and for which all points are fixed (these points can forbid certain short cuts in lines that are simplified). Points of the first type are termed dynamic points and points of the second type are termed static points. Points that eventually will be removed by the simplification algorithm have to be interior and dynamic points.

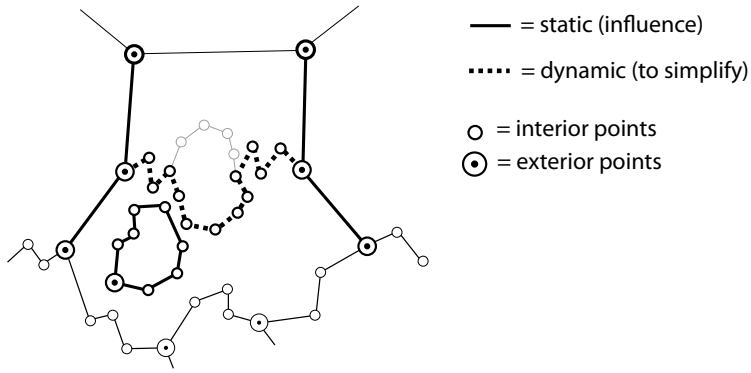


Figure 4.8: Dynamic polylines will be simplified (only one in this Figure), static polylines can have an influence on the simplification. Note that the alternative is illustrated, in which only the polylines that are incident to a merge boundary will be simplified.

After a merge or split generalisation operation is finished we have to chose which lines to simplify (thus select the *dynamic* polylines). Two viable alternative approaches, in which polylines are simplified en mass, are:

1. Simplify the polylines that are (in case of an area merge operation) incident to the common merge boundaries, or (in case of an area split operation) simplify the new boundaries that stem from the split operation;

2. Simplify all polylines of the resulting new area(s).

As the simplification should be topology-aware, the *static* polylines in the neighbourhood also have to be selected as input for our algorithm as these can influence the outcome of the simplification. For this purpose, we can use the topology structure to select the lines that are in the vicinity of the lines that we want to simplify. We use the topology structure as an initial spatial filter (going from neighbouring areas to their related boundaries), then with a second pass we can select the related boundaries based on bounding box overlap with the union of the bounding box of all dynamic polylines. An alternative approach is to keep an auxiliary data structure (such as an R-tree or quad-tree) for fast selection of the polylines in the vicinity. The downside of this approach is that an auxiliary structure needs to be kept, while the topology structure is already present. However, the initial filtering step using the topology structure can be expensive if the new polygonal area is at the border of the domain (leading to a selection of *all* edges at the border of the domain that have to be filtered on their bounding box).

4.2.5 *A stop criterion for the simplification*

We iteratively remove points from all dynamic input polylines, until a certain optimisation goal is reached. We have two main choices for defining this optimisation goal (to stop the simplification):

EPS-STOP Use a geometric measure as a threshold ϵ (all points having their weight $w < \epsilon$ should be removed, where w is based on the size of the triangle of 3 consecutive points in the polyline).

Using this approach, we could use a fixed epsilon throughout the whole process of building the variable-scale hierarchy. This is a not very realistic option, as the number of polygonal areas (and thus the level of detail) decreases when more generalisation operators have been applied (when more polygonal areas have been merged or split, the remaining boundaries should also be simplified more). A better option is to determine dynamically the value of ϵ with every generalisation step. For this we can:

- Take the average or median value of all weight values as ϵ (all points having a weight value smaller than this have to be removed, removing about half of the points);

- Set an ϵ based on other criteria, like the smallest segment length of all polylines taking part in the simplification. Such an alternative choice for ϵ also means that the weight values w for all interior points have to be calculated accordingly.

COUNT-STOP Use the number of points that we want to see removed.

Using the number of output points as optimisation goal, we can count the number of points in the input and try to remove a certain percentage. Two similar, but somewhat different, options in this respect are:

- Take a local approach: *e. g.* per input polyline try to remove half of the points (but do not remove more points from a polyline than half of its original points);
- Take a regional approach: for all polylines being simplified, count the total number of points and keep removing points, until in total half of these points have been removed.

Note that both approaches can leave more points as a result than wished for, as some of the points can be blocked by others (because topological errors must be prevented), although they fulfil the condition for removal (*e. g.* $w < \epsilon$). Note also that, with both approaches we can vary the percentage of points that we want to remove (instead of half of the points), depending on how far we want to 'push' the generalisation. In an extreme case, we could set the percentage to such a value, that the algorithm will try to remove all points, leading to straight lines as much as possible (only topological 'problematic' points are remaining).

4.2.6 *To prevent topological errors*

THE ALGORITHM. An outline of the procedure is depicted in Algorithm 1. For all dynamic polylines a doubly-linked list is created (storing the points in the order in which they are present in the original polyline, *cf.* Algorithm 1, line 1). Further, for all interior points of these polylines a weight w is calculated (line 2). Important points get a higher weight than less important ones.

All dynamic and interior points are inserted in a priority queue Q, ordered by their weight values w (line 3). In our implementation we use a red-black tree (Guibas and Sedgewick, 1978) for the priority queue. Points with equal weights are dealt with in the order of insertion. In-order traversal of the red-black tree Q allows now to find the point with the smallest weight value, which

Algorithm 1 Simplification, while keeping the planar partition valid

Require: A set of dynamic and a set of static polylines
Ensure: A set of simplified polylines {pre-processing}
Create doubly-linked list for each dynamic polyline
Compute weights w for all interior points of dynamic polylines
Add dynamic, interior points to priority queue Q based on weights
Create pointers between points of static polylines with only 2 points
Create kd-tree of all points of both dynamic and static polylines {simplifying}
while Q not empty **do**
    Pop least important p_i from Q {stop criterion, see section 4.2.5}
    if stop criterion met for p_i **then**
        break
    allowed ← True
    if p_i part of loop edge with 4 points **then**
        allowed ← False {no more 'tries' for this point}
    Retrieve τ (using p_{i-1} and p_{i+1} from linked list)
    vicinity ← search kd-tree for points near p_i using box of τ
    for all $b \in$ vicinity **do**
        if $b \notin (p_{i-1}, p_i, p_{i+1})$ and b part of segment $\overline{p_{i-1}, p_{i+1}}$ **then**
            allowed ← False {no more 'tries' for this point}
    if allowed **then**
        for all $b \in$ vicinity **do**
            if $b \notin (p_{i-1}, p_i, p_{i+1})$ and b on τ **then**
                allowed ← False
                Append b to p_i.blocked_by list
                Append p_i to b.blocks list
    if allowed **then**
        Remove p_i from linked list
        Adjust weights for p_{i-1} and p_{i+1}
        Check whether p_{i-1} and p_{i+1} are still blocked, otherwise add to Q
        Mark p_i as removed in kd-tree
        for all $u \in p_i$.blocks **do**
            Remove p_i from u.blocked_by list
            if u.blocked_by list empty **then**
                Add u to Q
{output}
return Simplified polylines (traverse doubly-linked lists)

is then removed from Q. For point p_i its neighbours, p_{i-1} and p_{i+1}, can be retrieved from the polyline doubly-linked list. The three points together form the triangle τ (see Figure 4.9).

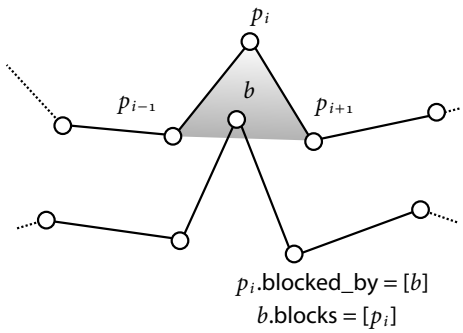


Figure 4.9: As b blocks the removal of p_i, the blocks and blocked by lists are filled accordingly.

The short cut that will be taken is $\overline{p_{i-1}, p_{i+1}}$. Such a short cut is only allowed if it does not lead to an *invalid* planar partition, *i. e.* violates one of the requirements, as described in § 3.2.1. Any intersections of the new short cut with other polylines or another segment of the polyline itself (*i. e.* a line between two consecutive points of the polyline) have to be prevented. As the partition is valid to begin with (which can be ensured by using a constrained triangulation, see § 3.3), the polylines of the planar partition do not contain any (self-)intersections. An intersection of the short cut can only be created when a segment σ 'enters' τ via the open side $\overline{p_{i-1}, p_{i+1}}$ (as it is not allowed to enter or leave the area of τ via either $\overline{p_{i-1}, p_i}$ or $\overline{p_i, p_{i+1}}$; this immediately would lead to an intersection). A point of σ must thus be interacting with τ for an intersection to happen and it is sufficient to check whether such a point exists, to prevent this. Points that can influence the collapse are termed *blockers*. These blockers stem from:

1. the polyline itself (self-intersection);
2. other polylines in the vicinity of τ (both static and dynamic).

To efficiently find those points, we use a kd-tree (not just a regular kd-tree, but one following Bentley (1990), for which the tree does not need to be re-organised after removal of points, but to which no extra points can be added after initial organisation of the tree). All (interior as well as exterior) points of all polylines taking part in the simplification are inserted in this kd-tree (algorithm 1, line 5). The bounding rectangle around the triangle τ is used to query the kd-tree to find

all points in the neighbourhood of this triangle, to see if there are any blockers for creating the short cut $\overline{p_{-1}, p_{i+1}}$. potential blockers b are checked whether they lie on the triangle τ. If a blocker is found, the short cut is not taken and as p_i was removed from Q it will not turn up in the next iteration.

As the kd-tree contains the points of all dynamic polylines, a potential blocker b can be a point that forms the triangle τ. If this happens we do not check whether b blocks p_i (*i. e.* p_{i-1}, p_{i+1}, nor p_i itself can block removal of p_i), or no simplification could take place at all. Since a blocker b can be removed itself later on and then a short cut for this vertex p_i might be allowed, a cross reference is set up between p_i and b (b is registered in the 'blocked by'-list of p_i and p_i is registered in the 'blocks'-list of b).

If no blockers were found, p_i can be removed from the doubly-linked polyline list it belongs to (creating a short cut in this polyline). The point is also *marked* as removed in the kd-tree. If the removed point p_i was a blocker itself (having one or more points in its 'blocks' list), it removes itself from the 'blocked by' list of these particular points. If for a point u its 'blocked by' list becomes empty (because of the removal of p_i) u is placed back again in Q, so it has a chance of being short cut in the next iteration (if then not blocked by any other point and still not having fulfilled the condition for removal, *e. g.* having a weight $w < \epsilon$). This is an improvement over the approach suggested by Kulik et al. (2005), who potentially have to check in every iteration step again whether a point is allowed to be removed. If one of the two neighbouring points p_{i-1} or p_{i+1} was blocked, it is also checked whether this is still the case (the shape of their related triangle also has changed, because of the short cut operation).

The algorithm ends when the chosen criterion has been met, *i. e.* there are no more points that can fulfil the criterion to reach the optimal goal, and the new polylines are returned.

MORE CASES FOR VALIDITY. Apart from intersection prevention by testing near points, more specific situations have to be taken into account, because of the validity requirements of the planar partition. Two other conditions also have to be checked (illustrated in Figure 4.10), to prevent occurrence of zero-sized polygonal areas:

1. Same polyline (see Algorithm 1, line 11): A special check is performed when p_{i-1} or p_{i+1} is the endpoint of a so-called 'loop' polyline (a special case where the 2 exterior points of the polyline are at the exact same location, *cf.* Figure 4.10-left). We now have to check whether there will still be enough

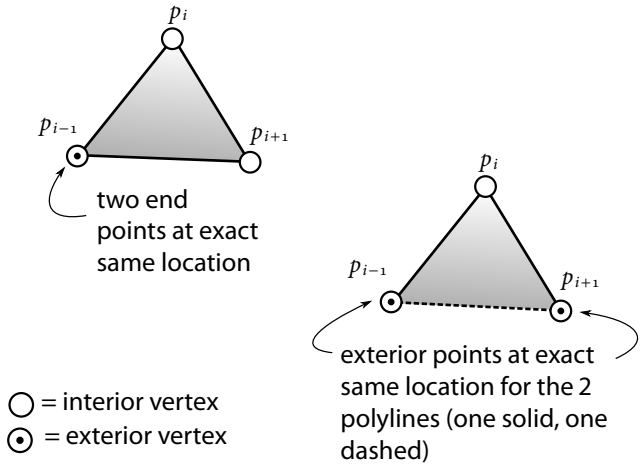


Figure 4.10: If taking a short cut leads to a polygonal area that has no area, we put the end points as blockers for p_i. The result is that p_i is not removed, as the endpoints of the polylines will never be removed.

points in the polyline when we take away p_i (because no zero-size area is allowed). We can do this, by traversing the linked list and check when p_{i-1} is a loop endpoint, whether p_{i+2} is also such an endpoint (similar with p_{i-2} for p_{i+1}). Note that this is a rare case (only the two last interior points for a triangular face).

2. Different polyline (see Algorithm 1, line 16): Another check is performed on whether $\overline{p_{i-1}, p_{i+1}}$ is already connected by another polyline (by allowing twice such a polyline, a zero-size area would be created, Figure 4.10-right). To prevent this, it is necessary to check if a potential blocking point b returned by the kd-tree is part of such a polyline between p_{i-1} and p_{i+1}: a. for a dynamic point returned by the kd-tree it is possible to use the doubly-linked list to navigate to the next vertex and check whether p_{i-1} and p_{i+1} are fixed, and b. for a static point returned by the kd-tree we put an extra pointer to the other endpoint of the static polyline, if the line only consists of two points. This allows checking whether a static point blocks the collapse of τ.

4.2.7 *Experiments*

We implemented the line simplification algorithm in our tGAP test environment. With the implementation we tried different alternatives. In total, we tested 7 alternatives – with only merge operations (so not yet the collapse/split operation of § 4.3) applied to the polygonal areas – for which the symbolic names are shown in Table 4.3.

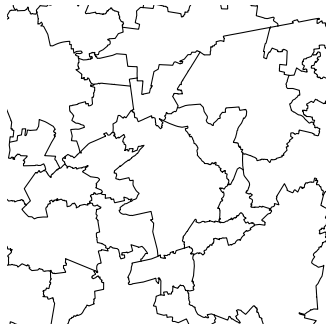
Table 4.3: Symbolic names of the alternatives that were tested.

	Simplify which edges?	
Stop criterion?	Merged edges only at nodes with degree = 2	All edges of new area (including merged ones)
No simplification at all	none	none
Count stop (regional, half # interior points)	m_ct	a_ct
Eps stop (median of all weights)	m_eps	a_eps
As far as possible	m_full	a_full

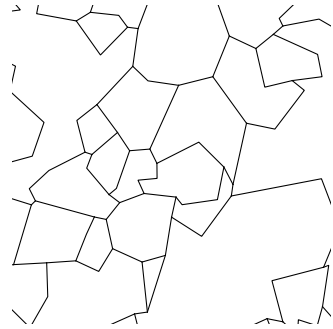
The first alternative (labelled 'none' in Table 4.3) we tested, was merging edges at nodes with *degree* = 2, but not applying any simplification. This was meant as reference test as we already knew that this would lead to too many coordinates per boundary, but used to measure the improvements. The remaining strategies come from varying two alternatives: 1. which lines to simplify (only the merged boundaries, prefixed with 'm_' in Table 4.3, or all boundaries of the new area) and 2. when to stop the generalisation (based on the median ϵ-value for all boundaries being simplified – dynamic eps-stop, based on the number of points, the regional count-stop approach, or simplify as far as possible, respectively labelled 'a_eps', 'a_ct' or 'a_full').

Table 4.4 shows the number of polylines and their average number of coordinates for the datasets we used in our experiment. We tested with three data set fragments, representing different types of geographic data. We used 1. a topographic, urban dataset (Amsterdam, city centre), 2. a topographic, rural dataset (Colchester), and 3. a land use dataset (Buchholz in der Nordheide). Both topographic datasets represented infrastructure objects (*e. g.* roads and waterways), which were not present in the land use dataset.

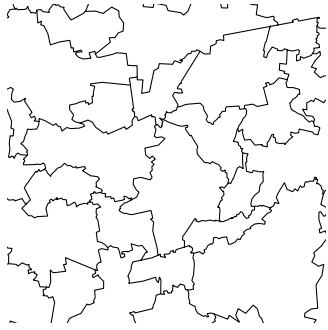
Figure 4.11 shows graphically some results of a few of the alternatives tested for the land use dataset. Figure 4.11a shows the result of keeping all original



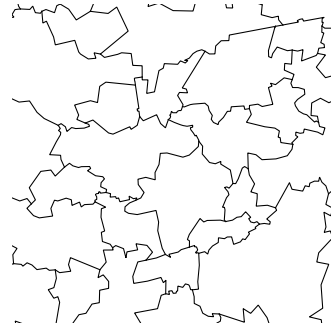
(a) No simplification (none)



(b) Count stop for all edges of new area (a_ct)



(c) Epsilon stop, using only merged edges (m_eps)



(d) Count stop, using only merged edges (m_ct)

Figure 4.11: From the Buchholz in der Nordheide dataset: 'Slices' of variable-scale data that show the result of the different alternatives for the line simplification, plotted at the same map scale (within brackets the symbolic name of the tested alternative). Note that the simplification of the boundaries changes the size of the areas and influences the order in which the areas are merged, therefore the boundaries on the 4 maps do not exactly correspond to each other.

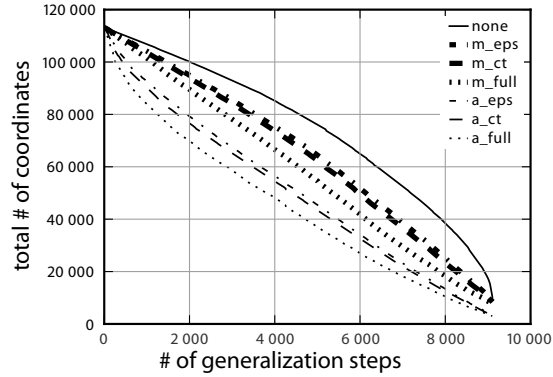
Table 4.4: For the datasets used in our experiment, the number of polygonal areas, polylines and average number of coordinates per polyline at start.

Dataset fragment (original map scale)	# of areas at start	# of polylines at start	avg # coords per polyline	total # coords
Amsterdam, urban (1:10K)	9 381	24 528	4.6	112 828
Colchester, rural (1:5K)	3 286	8 212	10.6	87 047
Buchholz, rural (1:25K)	5 537	16 592	7.2	119 462

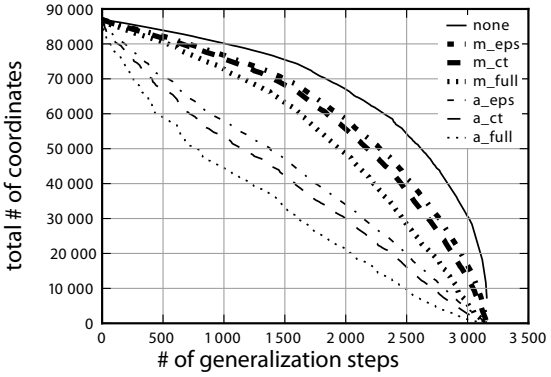
coordinates of the boundaries, thus not simplifying them. Tiny details and too many coordinates in the boundaries are the result. It can be seen in Figure 4.11b, that the count-stop approach (preserve half the number of points) applied on *all* boundaries of the new area leads to a very simplified and coarse version. Both alternatives in which only the merged boundaries are simplified leave more details (see Figure 4.11c and 4.11d), where the count-stop approach is a bit more 'aggressive' than the eps-stop approach. This is caused by the fact that the epsilon values of adjacent vertices are changed when a vertex is removed (these adjacent epsilon values most of the time increase), and therefore the eps-stop approach is less likely to remove exactly half of the number of the vertices, but somewhat less (while the count-stop approach is exactly instructed to hold on to the fixed number).

That the eps-approach is more gentle than the ct-approach is also illustrated by the graphs in Figure 4.12. In each graph, it is shown how many coordinates there are left for the total map, after every generalisation step. As expected, the line at the top of the graph is the reference situation, where no coordinates are weeded. It is also clear, as already visually illustrated in Figure 4.11, that the approach where only the merged boundaries play a role in the simplification is more gentle in removing coordinates, compared to when all edges of the new area will be simplified. Main cause for this is that if all edges of the new area are simplified, they will be simplified more often, compared to the situation where only the merged edges are simplified (*i. e.* for every generalisation step in which a polygonal area is the area to which a neighbour is merged, its boundary edges will again be simplified).

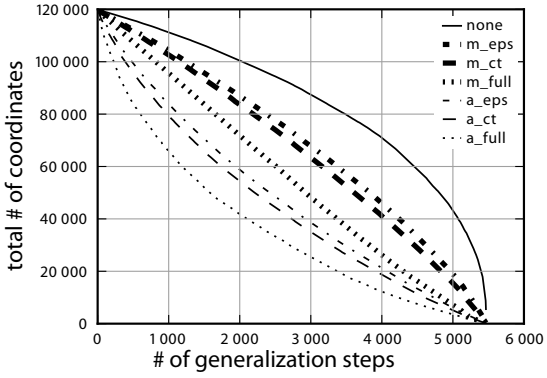
Table 4.5 illustrates the fact that simplifying the boundaries over-and-over again also has a negative effect on the contents of the hierarchy. Although the graphs from Figure 4.12 show that there are less coordinates on average on every



(a) Topographic, urban (Amsterdam)



(b) Topographic, rural (Colchester)



(c) Land use (Buchholz)

Figure 4.12: For each dataset, the graph shows the total number of coordinates for the complete map, in each generalisation step (*i. e.* the number of coordinates in a 'slice' of variable-scale data).

Table 4.5: Resulting number of polylines and coordinates with varying types of stop criteria for the simplification. Note that, although the 'a_' variants look much more generalised (*cf.* Figure 4.11b) these are much more expensive to be stored (*i. e.* there are *many* more rows to be stored (total # polylines), hence also the bigger total number of coordinates in the hierarchy).

(a) Amsterdam dataset (average # coords: 4.6)

simplify type	total # polylines	avg # coords per polyline	total # coordinates in hierarchy
none	36 447	7.1	256 969
a_ct	60 390	4.3	260 777
a_eps	62 006	4.6	284 289
a_full	55 084	3.7	205 870
m_ct	36 449	4.6	167 431
m_eps	36 438	4.8	176 350
m_full	36 403	3.8	139 187

(b) Colchester dataset (average # coords: 10.6)

simplify type	total # polylines	avg # coords per polyline	total # coordinates in hierarchy
none	12 347	22.4	276 335
a_ct	23 553	8.5	200 860
a_eps	24 539	10.1	247 767
a_full	19 640	6.7	131 538
m_ct	12 345	11.1	136 940
m_eps	12 343	13.0	160 066
m_full	12 349	7.8	96 665

(c) Buchholz dataset (average # coords: 7.2)

simplify type	total # polylines	avg # coords per polyline	total # coordinates in hierarchy
none	26 771	15.4	413 250
a_ct	54 166	5.8	312 394
a_eps	55 603	6.3	348 118
a_full	45 040	4.8	216 174
m_ct	26 770	7.5	200 132
m_eps	26 768	8.4	223 623
m_full	26 769	5.3	141 019

'slice' derived from the variable-scale structures when all boundaries of a new face are simplified, the opposite is true for the contents of the data structures. More coordinates need to be stored, because for every line that is simplified also a new version, with the simplified geometry, has to be stored in the data structures (*e. g.* compare alternative 'm_ct' with 'a_ct' – in all cases more coordinates are stored for the 'a_ct' alternative). Therefore, as a rule of thumb, simplifying only the merged edges is to be preferred over simplifying all the edges of a new area.

After this work was finished, we realised that the described simplification approach still leaves room for improvement: When all edges after a high level generalisation operation are used for simplification, the edges are over-simplified. This could be fixed by not reducing to half the number of points per edge, but by reducing the number of vertices by a smaller amount, *e. g.* by taking the number of edges and setting up a reduction factor for every edge as 1 / # of edges; However, it is likely that this still will lead to many edge rows. Fortunately, the same can also be done, when only merged edges are simplified: also here not always simplify to half the number of points, but to a factor that is based on the number of edges that are merged together into one new edge (and set up the reduction factor per new edge). Eventually the number of edges per polygonal area (and the average number of coordinates per area) – before the high level generalisation operation has taken place – could also be taken into account in setting up the reduction factor. This way the optimisation goal to keep the map density on average constant might be reached even better (compared to just taking always half the number of input points): whether this indeed is the case (and which approach is then best) requires more research. (*cf.* § 7.2).

4.3 Collapsing areas: splitting over multiple neighbours

This section investigates a generalisation operation, by which polygonal areas are divided over their neighbours and for which the result should fit in the tGAP data structures. Merging as the only generalisation operator in practice is not always 'optimal' or 'fair', as this is an all or nothing decision where one neighbour 'wins' and is not sufficient to deal with all different classes of geographical objects. For example, linear area features, such as roads and canals, deserve different treatment. The section also proposes a way of evaluating the result of the split operation, to obtain a measure for judging results so that the measure is objective, instead of subjective, as often the case with generalisation rules. To implement the split operation, the section investigates possibilities of using a triangulation. Further, the measure used reveals weaknesses in the possible solution space

imposed by the triangulation. The section also discusses the implications of the split operations for the data structures, such as the face hierarchy in a separate table.

4.3.1 *SplitArea: an explanation*

This section gives an overview of the SplitArea algorithm. The purpose of the algorithm is to split a face of a planar partition over its neighbouring faces. Taking into account the tGAP structure, we formulated a set of requirements that should be present in such a splitting operation:

1. Assign larger pieces of the area object to be split to more compatible neighbours and smaller pieces of the area to less compatible ones.
2. Prevent certain merges, *i. e.* completely disallow merges between faces having extreme incompatible feature classes. A special variant of this requirement is that the extent of our geographic domain covered by the data set should not be changed, which means that the objects should not be merged to the universe (the space surrounding the dataset), otherwise the domain would become smaller.
3. Generate resulting line work that fits in with and connects to the rest of the planar partition.
4. Deal with holes (filled by one or more island faces, that can also get a share of the face to be split).
5. Work on vector data stored in a topological data structure (such as the well-known node-edge-face data structure, *cf.* Worboys and Duckham (2004)) and generate the topological references (*e. g.* incidence relations between edges and faces) for the edges being outputted.

The algorithm uses a constrained Delaunay triangulation to find a skeleton – *i. e.* an *approximation* of the medial axis (Blum, 1967) in our case approximated by straight lines – inside the face to split, prunes parts of this skeleton, and connects the obtained line work to the rest of the planar partition. The following definitions will be used:

FACE A polygonal object, possibly with holes, representing one object of the planar partition. In the tGAP structure the geometry of a face is not

explicitly represented as a polygon, but its geometry has to be obtained from the set of edges related to it.

TOPOLOGICAL CHAIN An edge of the planar partition. Every topological chain lies exactly between a left and right face. Topological chains in the tGAP structure are modelled as polylines, having a start node and end node reference as well as two face references (one for the left neighbouring face and one for the right).

SPLITTEE The face F for which the area has to be split over the neighbouring faces, *cf.* Figure 4.13a.

PERIMETER CHAIN A topological chain having the face F as its left or right face.

EXTERNAL CHAIN A topological chain that is incident with at least one perimeter chain of the face F. An external chain is *not* part of the boundary of the splittee F (but touches the boundary of F in one or two points).

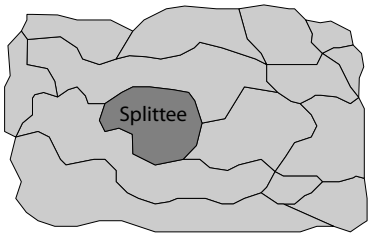
SKELETON EDGE The skeleton is the collection of line work that is created inside the splittee. The skeleton consists of a set of edges (line segments having two points). The new boundaries for the neighbours of the splittee will be formed by the set of skeleton edges and the connections to the external chains (see Figure 4.13c).

The input of the SPLITAREA algorithm is:

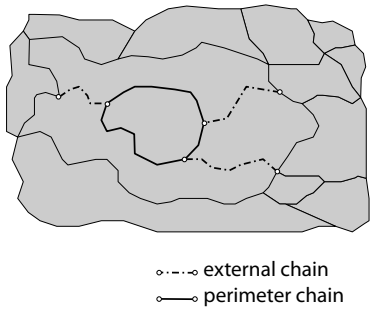
- The id of the face F, the splittee.
- The set of topological chains forming the boundary of the face F, *i. e.* the perimeter chains.
- The set of topological chains incident with the boundary of the face F in one or two nodes, the external chains.

The output of the algorithm consists of a set of new topological chains representing the new boundaries of the splittee's neighbouring faces: some of these topological chains can be completely new, others are an extended version of the external chains. Both the input and output topological chains contain their left and right references; as we will see in § 4.3.2, the input chains may have also a weight value attached to them.

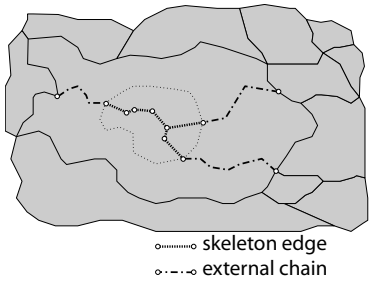
The algorithm performs the following steps:



(a) The splittee: the object that has to be subdivided over its neighbours.



(b) The perimeter chains and the external chains (chains incident with the perimeter chains).



(c) Inside the splittee skeleton edges will be created. The new boundaries will be formed by these edges and the external chains.

Figure 4.13: An example showing `SplitArea` at work.

1. Triangulation
2. Selection of internal triangles
3. Creation of the skeleton
4. Creating connectors
5. Edge labelling and Skeleton pruning
6. Obtaining the final boundaries

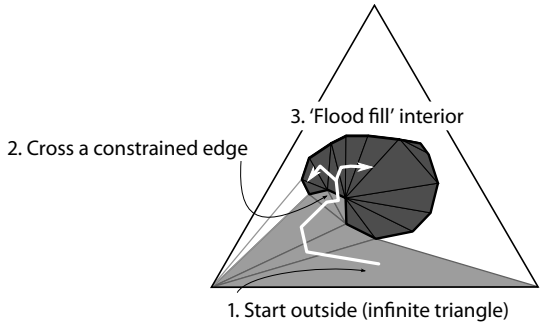
After these steps are described in more detail, we give an overview of how to handle holes in the splittee (as a more difficult case).

STEP 1. TRIANGULATION. The first step of the algorithm is to build a constrained Delaunay triangulation of the perimeter chains of the splittee. A constrained triangulation is a triangulation of a set of points that has to include a given set of segments between these points (Bern and Eppstein, 1992). As constrained edges are not necessarily Delaunay edges, a constrained Delaunay triangulation tries to fulfil the empty circumcircle property as much as possible, but in case a constrained edge is present, the empty circle criterion is weakened. The set of constrained triangulation edges is the boundary between the interior and the exterior of the splittee.

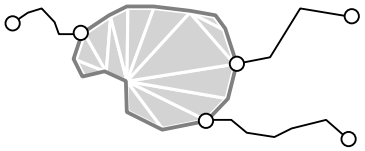
STEP 2. SELECTION OF INTERNAL TRIANGLES. The triangulator produces a mesh of triangles both in the interior and exterior of the splittee, bounded by a triangle with points at infinity (Liu and Snoeyink, 2008); to select only the interior triangles, the algorithm performs a walk on the triangles (see Figure 4.14).

The walk starts from one of the triangles having one vertex at infinity. While walking from one triangle to an adjacent one, the triangulation edge that the two triangles have in common is crossed: if this edge is a constrained edge, then we know that we are on the interior of the splittee. Once inside, we can flag the triangles we are looking for as internal – by avoiding to cross a constrained edge again, the walk stays inside. This search can be done fast (in linear time on the number of triangles) and reliably using the data structure already existing, whereas using a point-in-polygon test would require an algorithm to find one internal point robust to the presence of holes in the polygon, a spatial index to find efficiently the triangle in which the point is and then, again, a walk on the neighbouring triangles to flag them.

STEP 3. CREATION OF THE SKELETON. The creation of the skeleton edges is performed individually for each triangle, following the technique described in Uitermark et al. (1999). Uitermark et al. also use a CDT algorithm in which they input a graph $G = (V, E)$ where V is a set of vertices, the endpoints of the input edges, and E is this set of edges, which they termed *G-edges* (constrained triangulation edges). Looking at the resulting CDT, two kinds of edges can be discriminated: G-edges, edges that were already present in the input graph, and D-edges, created by the CDT algorithm. Triangles in the CDT can now be



(a) Walk on internal triangles.



(b) Triangulation of the splittee, where all internal triangles have been selected. Also external chains are shown.

Figure 4.14: Selection of internal triangles.

classified by determining the number of G-edges in the boundary of a triangle. In this way, four types of triangles can be distinguished: 0-triangles, 1-triangles, 2-triangles and 3-triangles, where the number is the number of G-edges. A skeleton of the triangulated object can be found by forming new line segments based on connecting the geometric midpoint of the *D-edges* for the triangles (see Figure 4.15 and Figure 4.16a for an illustration).

STEP 4. CREATING CONNECTORS. To complete the construction of the skeleton, we need to assure that it is connected with the existing line work outside the boundary of the splittee, *i. e.* it has to be connected with the external chains, see Figure 4.16b.

Every external chain touches the splittee in one or two nodes: if such a node is not already connected to the skeleton then a special skeleton edge is drawn from this node to the rest of the skeleton. These special edges are called connectors and are generated according to the scheme shown in Figure 4.17a (as this scheme prevents topology errors). Figure 4.17b shows that by generating the connectors

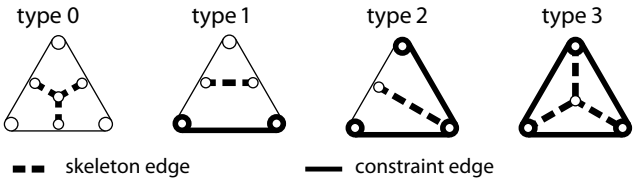
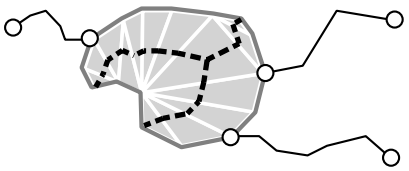
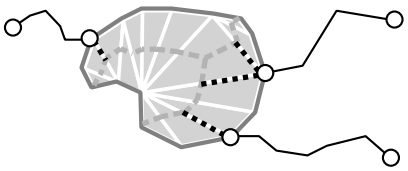


Figure 4.15: Four types of triangles can be distinguished by looking at the number of G-edges: 0-triangles, 1-triangles, 2-triangles and 3-triangles. By connecting the midpoints of the D-edges (unconstrained edges) a skeleton edge can be obtained (skeleton edges are visualised with dashed lines).



(a) Obtaining skeleton edges using the triangulation.

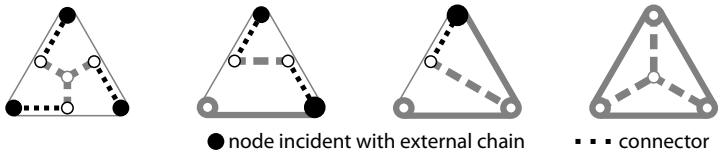


(b) Adding extra skeleton edges (connectors) to guarantee a connection between the existing external chains (those need to be preserved) and the skeleton. Note that a connector has to be chosen with the node at the boundary at the top right.

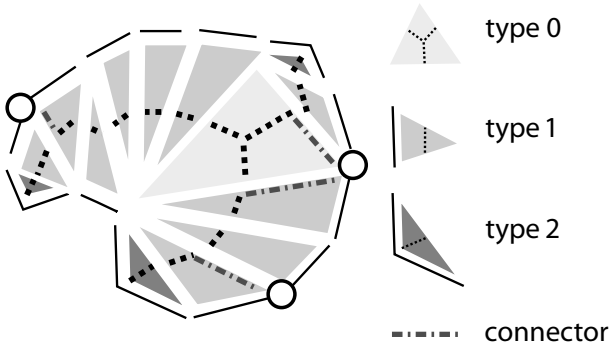
Figure 4.16: Creation of the Skeleton.

only going outward of a node when going counter-clockwise around a triangle, we can guarantee that this operation is local (so no need to look at neighbouring triangles, when generating skeleton edges and connectors) and that duplicate connectors in neighbouring triangles are prevented.

Depending on the triangulation, there can be many unconstrained triangulation edges between the skeleton and the node of an incident external chain: in such a case the connector to be used has to be chosen. In our implementation we decided to choose the connector for which the angle is the most collinear with the direction of the external chain, but other choices could be preferred (*e. g.* the longest or the shortest one).



(a) The 4 different triangle types and how connectors are generated per type

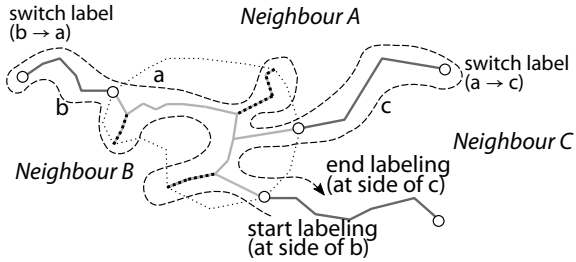


(b) Connectors and skeleton edges are created locally, *i. e.* generated for each triangle individually. Which connectors and skeleton edges are generated is dependent on the type of triangle. To prevent duplicate connectors for neighbouring triangles, the connectors are only generated going outward of a node (where an external chain is incident) when going counterclockwise around a triangle.

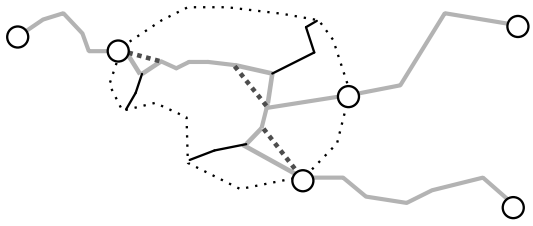
Figure 4.17: Connectors are created if a corner vertex of a triangle also is a node in the planar partition (skeleton edges are visualised with dashed lines).

STEP 5. EDGE LABELLING AND SKELETON PRUNING. The next step of the algorithm is to propagate the left and right face reference values from the external chains to the skeleton edges. For this purpose a graph in which all skeleton edges, chosen connectors and external chains are represented, is built. This graph can be represented by a winged-edged structure (Baumgart, 1975). The labelling starts at one side of an external chain, propagating the correct neighbour reference value onto all skeleton edges, until another external chain is encountered. Then, the neighbour reference is switched, propagating the other reference of the external chain, and the labelling continues, until the external chain where the labelling

started is encountered. An illustration of this labelling procedure is found in Figure 4.18a.



(a) Labelling of the skeleton edges (light grey), starting from an external chain (dark grey). Note: edges that have the same neighbouring reference value on both sides are dashed.



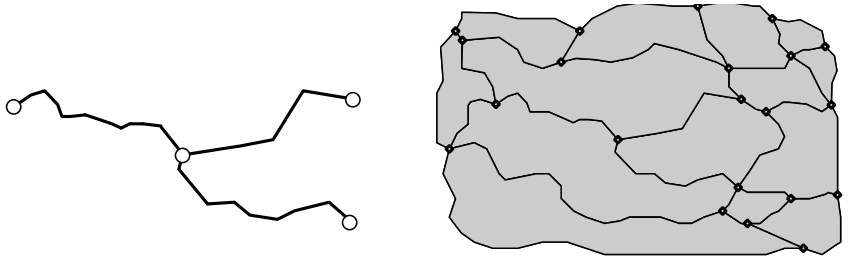
(b) Parts of the graph that have the same labels on both sides (black) are removed. Short-cuts (dashed) replace the two skeleton edges that are incident with the branches that are removed.

Figure 4.18: Labelling the skeleton edges with the correct neighbour reference and pruning parts of the skeleton that are enclosed completely by one neighbour.

At the end of the labelling some of the skeleton edges can have the same neighbour reference on both their sides. This means that in the planar partition these parts will be completely enclosed by one neighbouring face, therefore these edges are removed. Removal of these edges can be seen as pruning branches in the graph. The winged-edge structure makes it easy to delete those edges from the graph, while keeping the connectivity between the remaining edges in the graph. The removal of the branches leads to some artifacts (spikes, where the skeleton makes a sharp turn). To compensate for that, some short-cut edges are – when possible – introduced, replacing the two skeleton edges that make the sharp turn and are incident with the branch removed (this is shown in Figure 4.18b). The

edges that are removed only belong to type-0 triangles, thereafter the shortcut edges that replace them do not affect the topology, as they are inside triangles that already belong to the original area.

STEP 6. OBTAINING THE FINAL BOUNDARIES. As a final step, the new topological chains for use in the tGAP structure are obtained. All skeleton edges and external chains that have the same neighbour reference values are merged into one topological chain. When this process has finished, all the perimeter edges of the splittee can be removed from the tGAP structure, the new boundaries have been completely built and will fit in with the remaining part of the line work of the planar partition, and the area of the splittee F has been subdivided over its neighbouring faces, *cf.* Figure 4.19.



(a) All skeleton edges and external chains are merged into the longest topological chains possible (having the same neighbour references).

(b) The line work obtained fits in with the rest of the planar partition.

Figure 4.19: Result obtained with SplitArea.

HANDLING HOLES IN THE SPLITTEE. One of our requirements was that the algorithm also should handle holes (*i. e.* island faces within the outer boundary of the splittee). Irrespective of whether the input contains holes, all segment geometry for the skeleton is obtained correctly by our approach from the classification of triangles (as described in Section 4.3.1 and illustrated in Figure 4.17). However, the generation of topological references needs attention when the input face contains a specific type of islands in its interior. For such islands, we term them 'lonely' islands here, one face is completely contained in a hole of the splittee.

Figure 4.20 shows three of such lonely island faces: face B, C and D. As shown, the resulting skeleton for the splittee, face A, will not be a tree structure, but will result in a graph having one or more cycles (a cycle per island). This has consequences for the labelling step described, as not all edges in this graph can be labelled on both sides — the inside of the cycles can not be reached automatically. No external edges will be present at the interior side of the cycle of a lonely island. This is only the case for lonely islands, but not for holes containing more than one island face. In the example, the cycles for face E and F will be automatically labelled from the boundary edge between the two (as this edge will be inserted as external edge in the skeleton graph).

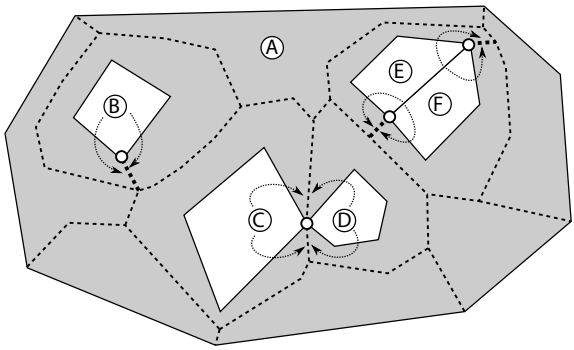


Figure 4.20: Handling islands. No extra steps have to be taken for label propagation, when a hole is filled by more than one face (*cf.* face E and F). However care has to be taken, when holes are filled by only one face (face B), or when island faces are tangent (face C and D).

To solve the labelling problem, one node from a ring around a lonely island will be attributed with the face identifier that lies inside the ring. Subsequently, generated skeleton segments incident at those nodes will be marked as being a seeding segment and as such will be treated as external segments from where the label propagation can take off. Note that only one face pointer will be set on both sides of these external segments, so that at the end of the labelling step they will be removed (see face B in the example).

Another typical case that happens in real world datasets is the following: Islands can be tangent in one point (think of such islands as looking as a bow tie). In this case this tangent point will be chosen as the node to generate the face pointers for the cycles from. In this particular case, care has to be taken when propagating the adjacent face information to the incident seeding segments. This can be solved by taking into account how the original faces are incident to the

node, thus looking at which face is incident to which sector around the node. Note that more than two faces can be tangent in one point. An angle comparison of the new segment with this sector information reveals which face pointers have to be propagated onto the newly created segment. Two different face pointers will be set on the two sides of the external segment (see face C and D in the example, one side of the segment will be labelled with C, the other side with D).

4.3.2 *A weighted split*

In Ai and van Oosterom (2002) it was mentioned that setting weights could help to get a fairer split, *e. g.* objects having a similar feature class can get a larger share of the splittee than less compatible features. With the normal approach, the vertices of the skeleton edges to be created are normally positioned exactly at the middle of an unconstrained edge. This changes with the weighted approach.

INTRODUCING WEIGHTS. To obtain a weighted solution, weight values are given as input together with the perimeter chains (because the weights are based on the neighbouring faces). For each vertex that is part of the triangulation at least one weight value will be present. The vertices of the skeleton edges can be slided along the unconstrained edges of the triangulation, respecting the weights. Figure 4.21 illustrates this. The weights make it possible to move the vertices of the skeleton edges further away from one perimeter chain (and closer to the opposite one).

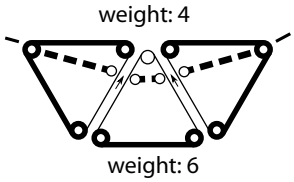


Figure 4.21: Weights are set on the perimeter chains and are entered in the triangulation at the vertices. The new vertices of the skeleton edges are moved along the unconstrained triangulation edge accordingly. The vertices where the external chains are incident have two weights set.

The weights are determined by the compatibilities of the neighbours. If the compatibility of a neighbour and the splittee is high, then the neighbouring face should get a large share of the face to be split. In this case a higher weight is set on a perimeter chain compared to when the neighbour is less compatible with

the splittee. A higher weight 'pushes' the newly created skeleton edge further away from the original perimeter chain, obtaining a larger share.

In the case that a vertex in the triangulation is also a topological node (an external chain is incident with the boundary of the splittee at this vertex) more than one weight will be associated with a vertex. Different choices can now be made on how to handle this multiplicity of weight values (we take the average, but also the minimum or maximum of the weight values could be chosen).

HANDLING ZERO-WEIGHTS. Putting weights on the perimeter chains in the input gives more flexibility on how to subdivide the area of the face to be split. To prevent one of the neighbours from getting a share at all zero-weights are introduced. This is useful in two cases: 1. prevent a neighbour that is highly incompatible with respect to the feature class of the splittee from getting a share and 2. when the feature to be split is at the border of the domain — we can then fix the border at its position (see Figure 4.22). Zero-weights are in accordance with how weights are set up in the tGAP structure. If two faces have two incompatible feature classes, a value of zero will be given in the compatibility matrix. Therefore the resulting weight set on the boundary between two such faces will also be zero ($ngb_{compatibility} = 0$).

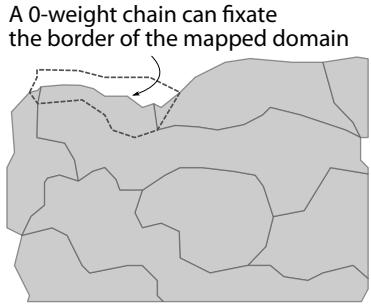


Figure 4.22: Fixating the border of the domain can be performed with 0-weights. Note that there should at least be 1 non-zero weight neighbour, otherwise no split can take place.

When allowing zero-weights as input, care has to be taken, when generating the skeleton edges based on the classification of the internal triangles. The basic classification (0- to 3-triangle) has to be extended to deal with one, two or three vertices having a 0-weight.

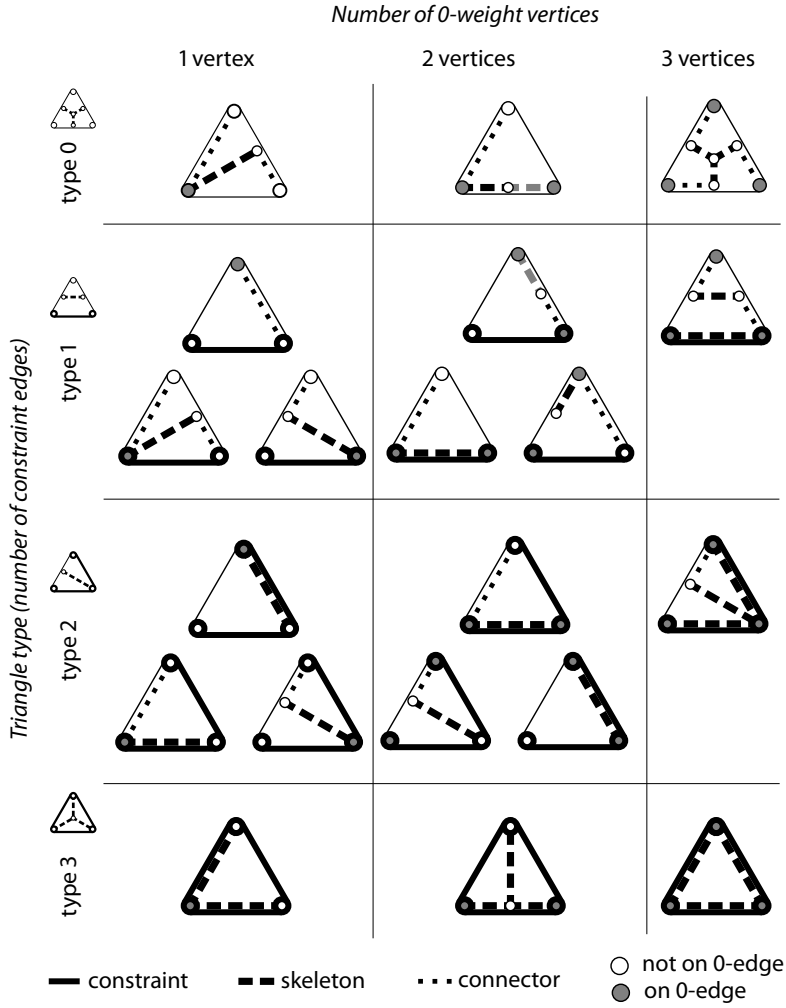


Figure 4.23: Triangles with zero-weight vertices. Unmovable vertices, having a weight = 0, are coloured gray. Skeleton segments that will be created are in black, skeleton segments that will not be created are visualised with gray (*e. g.* triangle type-0-2-vertices).

Figure 4.23 shows all possible situations that can occur with 0-weight vertices and Figure 4.24c shows some of these classified triangles in a real world example. There are 5 cases that need some attention:

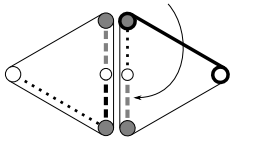
1. Sometimes no skeleton edge has to be generated. This is the case with triangles with two vertices being on a constrained edge and the third vertex being a zero-weight vertex (Figure 4.23 triangle type-1-1-vertex-top, *i. e.* the topmost triangle of the three triangles in the cell belonging to triangle type-1 – row 1 in the Figure – having one 0-weight vertex – column 1 in the Figure). See also Figure 4.24c, triangle labelled with 't-1-1-v-top'.

2. When two zero-weight vertices are opposite of each other on an unconstrained triangulation edge, a skeleton vertex is generated at the middle of this edge, similar to when two weights are equal and non-zero (*e. g.* triangles type-1-3-vertex, type-2-2-vertex-bottom-left).

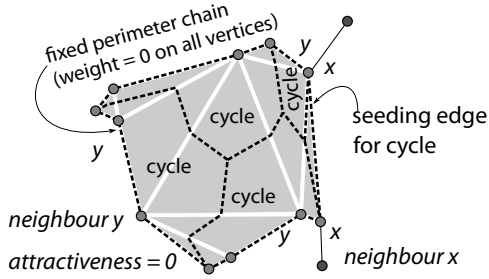
3. Figure 4.24a shows another case that only happens, when 0-weights are allowed. When a type-0-2-vertex and a type-1-2-vertex-top triangle are adjacent, duplicate edges will be generated. To prevent this, only the black skeleton edges are generated. Here the same reasoning is followed while adding connectors (going counter-clockwise around a triangle, only the edges that depart from an incident node are generated).

4. As mentioned before, vertices in the triangulation will have more than 1 weight associated if an external chain is incident (*i. e.* nodes in the planar partition topology). In an unweighted situation it is possible to choose how to deal with the weights in such a situation. In case one of these weights is zero, this 0-weight has to be set as *the* weight of this vertex. This ensures that the vertices of the original perimeter chain are not moved.

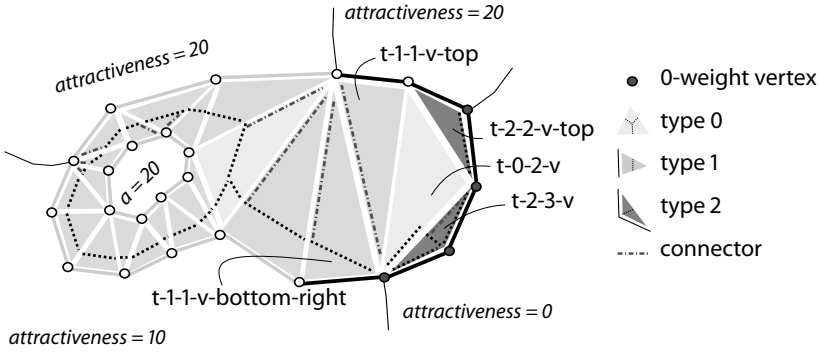
5. Introduction of 0-weight vertices can lead to cycles in the skeleton edge graph (see Figure 4.24b for an illustration). Therefore in the labelling step, when left- and right-references are set, an extra check has to be performed. If after labelling skeleton edges are encountered that are only labelled on one side and which were not entered in the triangulation as a edge with 0-weight, these edges can act as a seeding edge from which the label can be propagated to the other side (on the side of the edges that are facing the interior of the cycle). This label propagation continues visiting all cycle edges until all these edges have been labelled on both sides. Note that



(a) If not careful with generating skeleton edges having 0-weight vertices, duplicates can arise in the resulting skeleton edge graph.



(b) Zero weight edges can lead to cycles in the skeleton graph. The labelling step thus should be modified. In this example x will be propagated onto the edges between two cycles. These edges then will be labelled with x on both sides and removed. The result is that neighbour x gets the complete area of the splittee.



(c) Also with zero-weight vertices triangles are processed locally (independent from all other triangles). However more possibilities have to be taken into account (*cf.* Figure 4.23). Note that triangles that have a zero-weight vertex have their constraints visualised in black and are labelled with their position in Figure 4.23.

Figure 4.24: Fixed perimeter chains (*i. e.* zero-weight vertices) bring some special implications.

edges that have the same label on both sides will be removed from the skeleton graph (just as in the unweighted case).

4.3.3 *Judging the fairness of a split*

In the original GAP tree (van Oosterom, 1993, 1995) the compatibility between a specific face and one of its neighbours, the i-th neighbour – ngb_i, is defined (at instance level) by three terms:

1. Type compatibility. This takes into account the semantic distance between the feature class of the neighbour and the splittee. For example, urban fabric can be considered more compatible with industrial area than with grassland. Objects having similar feature classes are thus considered more compatible. This is encoded by a value between 0 and 1, where 0 means not compatible at all, and 1 means very compatible (equal). The compatibility values can be stored in a square matrix for all feature classes, which is termed the compatibility matrix.
2. Boundary length between the splittee and ngb_i.
3. The relative importance (or weight) of the feature class of ngb_i. Dependent on the use of the dataset, different feature classes can be given a higher weight, which means they are less considered for being generalised, compared to equal sized features having a lower weight.

Equation 4.1 contains these terms for an attraction value for ngb_i of the splittee s:

$$\text{attract}(\text{ngb}_i) = \text{compat}(\text{s}, \text{ngb}_i) \times \text{length}(\text{s}, \text{ngb}_i) \times \text{imp}(\text{ngb}_i) \qquad (4.1)$$

Instead of assigning the face to the most compatible neighbour (as in the original GAP-tree) or by splitting the face 'in the middle' (as in § 4.3.1), we could define our 'ideal split' as one that assigns areas proportional to the attraction values of the various neighbours.

The objective of defining these attraction values is to let each neighbour obtain a fair share of the splittee and, as described in § 4.3.2, this measure will influence the position of the newly generated skeleton segments. In the same line of reasoning, we have designed a methodology for the evaluation of the split algorithm.

Equation 4.2 shows that we expect a neighbour i to get a share of the splittee s, that is in accordance with the sum of all attraction values of the n neighbours around the splittee.

$$\text{share}(\text{ngb}_i)_{ideal\_target} = \frac{\text{attract}(\text{ngb}_i)}{\sum_{j=0}^{n} \text{attract}(\text{ngb}_j)} \times \text{area}(\text{s}) \tag{4.2}$$

The share that a neighbour i actually obtains is easily measured by subtracting the size of the neighbour from before the split, from the size of the neighbour after the split.

$$\text{share}(\text{ngb}_i)_{obtained} = \text{area}(\text{ngb}_i)_{post} - \text{area}(\text{ngb}_i)_{pre} \tag{4.3}$$

We now can obtain the absolute difference of the share that one neighbour should obtain in theory and the share that it gets assigned by SplitArea (Equation 4.4). This difference is the error that is made in appointing a share to one of the neighbours.

$$\text{error}_i = abs(\text{share}(\text{ngb}_i)_{obtained} - \text{share}(\text{ngb}_i)_{ideal\_target}) \tag{4.4}$$

Equation 4.5 shows that summing all error values of the neighbours involved in the split operation, normalized against the size of the splittee s, leads to a value that expresses the total error made with subdividing the feature.

$$\frac{\frac{1}{2} \times \sum_{j=0}^{n} \text{error}_j}{\text{area}(\text{s})} \tag{4.5}$$

Because for each error the value is recorded twice (what one neighbour gets too much, is automatically not appointed to any of the others, but also counted) the total error value is divided by two, resulting in an error value between 0 and 100%. Subsequently, these error values can be visualised in a histogram.

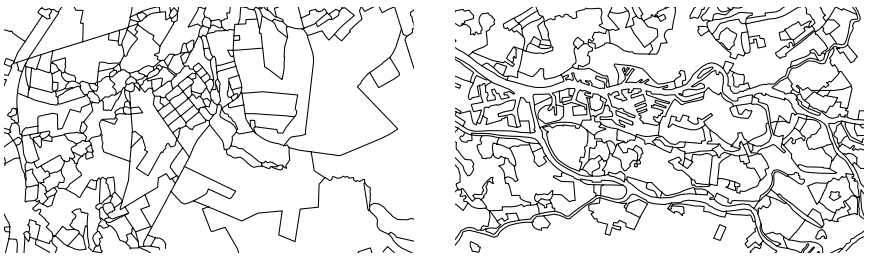
4.3.4 *Results and Discussion*

To test the behaviour of SplitArea we implemented the algorithm in the context of the tGAP data structure. The algorithm was tested with two land cover datasets. Table 4.6 shows some characteristics for both datasets and Figure 4.25 gives a graphical impression.

Table 4.6: Characteristics of the test data sets used in the experiments.

	Buchholz in der Nordheide (clip) (intended for 1:50K use)	CORINE 2000 (clip) (intended for 1:100K use)
edges	1 564	6 635
faces	525	2 471
neighbours per face		
- average	5.6	4.5
- median	5	3
- maximum	27	439*
feature classes	12	29

* In this dataset, some polygons have a relatively large extent (rivers) or have a lot of islands, which explains the high maximum number of neighbours.

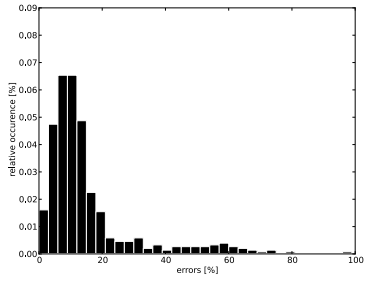


(a) German land cover dataset (b) European land cover dataset

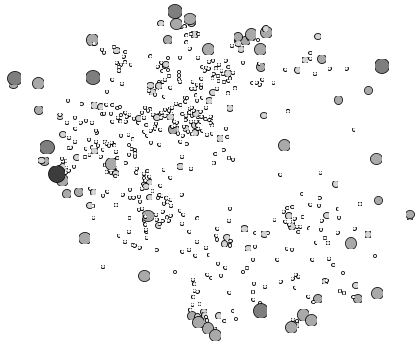
Figure 4.25: Map fragments from both test datasets.

In § 2.4 it was explained that constructing data for the tGAP data structure is an off-line process, preparing the data for on-line use, *e. g.* in a web environment. In every iteration of this batch process the least important face is replaced by the result of a generalisation operator (either a merge or, now with the availability of SplitArea, a split operation). At the end of every iteration the obtained topological chains are put back into the planar area partitioning. From the topological chains, the polygon geometry is obtained to calculate the new area size of the neighbouring faces. In our experiment, no failures were found in reconstructing the geometry – this is an indication that all topological references were correctly set by our implementation of SplitArea. Note that we only performed split operations this experiment (no merge operations).

EVALUATING SPLITS AND IMPROVING FAIRNESS. We first ran SPLITAREA in unweighted mode on the German land cover dataset. In this case, the attraction values are only determined by the boundary length between the splittee and its neighbours (*cf.* Equation 4.1, for which in this case compatibility values and weights were set to 1, thus effectively only taking the length into account). For each split, the error per neighbour was recorded, according to the methodology described in § 4.3.3 and post-processed into the total error value per split. Figure 4.26 shows an histogram and the spatial distribution of the total errors being made for all splits while building the tGAP structure for this dataset.



(a) Histogram of the errors occuring while splitting faces. An error is the difference between the expected and actual share obtained by a neighbouring face.



(b) Distribution of the errors over the domain. The larger and darker a circle is, the larger the total error in the outcome of the performed split is.

Figure 4.26: Results of first, unweighted run of SplitArea.

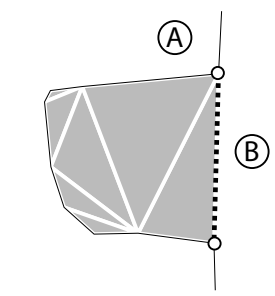
The errors stored per split allowed us to investigate which input was the basis for the splits with the highest error (the tail of the histogram which we tried to analyse). The larger circles on the map in Figure 4.26b show those errors. It appeared that three types of input resulted in large differences between what we set as ideal target and what the algorithm obtained:

1. The splittee is completely surrounded at one side by another face, which therefore devours the splittee and leaves no part for the other neighbour at all (*e. g.* Figure 4.27a).

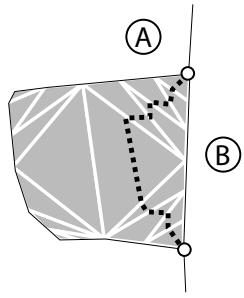
2. A splittee having unevenly distributed vertices in its boundary, preventing one of the neighbours of getting a fair share (thin and long triangles will

appear at one side, preventing a face at this side to get a fair amount of the triangles on the interior of the splittee). See Figure 4.27c for an example.

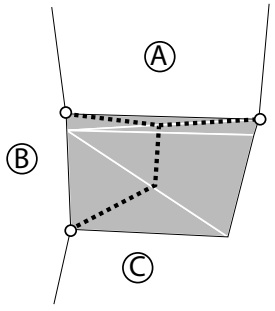
3. Faces at the border of the domain will give a share to the universe (as we did not instruct the algorithm to not extend the universe, *i. e.* 0-weights were not used).



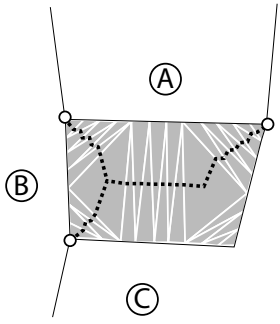
(a) Completely surrounded splittee, face A gets all of the splittee.



(b) Improved split.



(c) Unevenly distributed vertices prevent face A from getting a share.

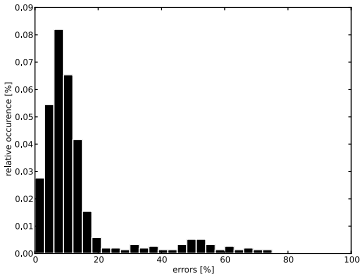


(d) Improved split.

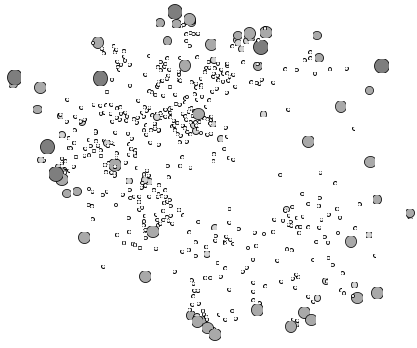
Figure 4.27: (a) & (b): Face A completely surrounds the splittee and gobbles it up. (c) & (d): On the boundary of the splittee, the vertices are distributed unevenly, leading to small triangles in front of face A: these triangles prevent face A from getting its fair share of the splittee.

To improve on the first two cases, we came up with the following strategy to compute the improved split (the edges are only modified as input for the split algorithm, but as SplitArea will obtain new boundaries, these modified input

edges are not stored in the tGAP data structure): first, delete the vertices that are unevenly distributed by simplifying the boundaries using a small tolerance value, and second, densify the boundaries, by placing new vertices regularly into the simplified boundaries. For simplification we used the approach described in § 4.2 (now with a tolerance as stop criterion) and as a tolerance value we used twice the smallest segment length that is present in all the boundary edges of the splittee.



(a) Histogram of the errors occuring while splitting faces.



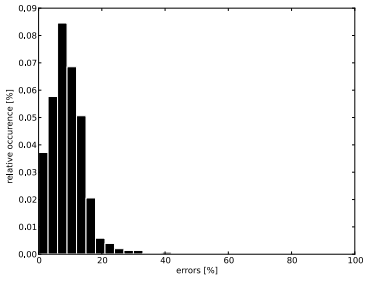
(b) Distribution of errors over the domain. Errors remain at the border of the domain.

Figure 4.28: Results of *unweighted* run of `SplitArea`, using simplification and densification on the input.

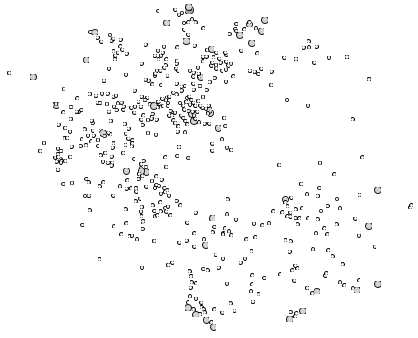
Figure 4.27b and 4.27d show the result of two splits that clearly benefit from the strategy of simplifying and densifying. Overall, the total amount of error has decreased, compare the error distribution of Figure 4.28b with the one in Figure 4.26b and it is evident that some of the larger errors have gone away.

The errors at the boundary of the domain can only be solved by fixing the borders of the splittee that are incident to the universe. Thus, to prevent the universe face from 'eating' objects, we placed zero weights on the edges incident to the universe. The result of this is shown in the third histogram in Figure 4.29, which is clearly the best result we obtained so far.

The problem of the algorithm not being able to get an even fairer split (an 'error free' result would result in an histogram with only one bar adjacent to the y-axis) is to be searched in the limits imposed by the triangulation. Using a triangulation has clearly advantages: it brings good control over maintaining correct topology, enables relatively easy computation of an 'approximate' skeleton, and good and robust (triangulation) implementations are freely available.



(a) Total errors made.



(b) Distribution of errors over the domain. Errors at the border of the domain have been solved.

Figure 4.29: Results of a *weighted* run of SplitArea, using 0-weights (with simplification and densification) to fixate the border of the domain.

However, the densification step that was shown to be necessary, also shows that our solution space for finding the ideal split is limited by the way the triangles are placed based on the spatial configuration of the input (*i. e.* no better solution is possible with the way triangles are placed).

EFFECTS FOR THE TGAP DATA STRUCTURE. To compare the number of topological chains and faces that are the result of SplitArea with the number of primitives stored in the tGAP structure we ran another experiment. In the classic set-up of the tGAP structure only a merge operation was available and duplicate edge rows are avoided as much as possible, *cf.* § 4.1. For both test datasets a lean tGAP structure was built, as well as a structure in which we applied SplitArea as the only generalisation operation (without weights set on the input perimeter chains) and the structure was filled with as lean as possible content: only a new row in the edge table is recorded if an edge has new or changed edge geometry. Furthermore, we input for the compatibility values a matrix with ones, which means that the merging and splitting operations are purely driven by geometric criteria (area and boundary length), but that thematic attributes (*i. e.* looking at compatible feature classes) are not taken into account and therefore has little effect on the number of edges and faces to be stored.

Table 4.7 shows the resulting number of topological chains (edges) and faces that are stored in the tGAP structure, depending on which generalisation

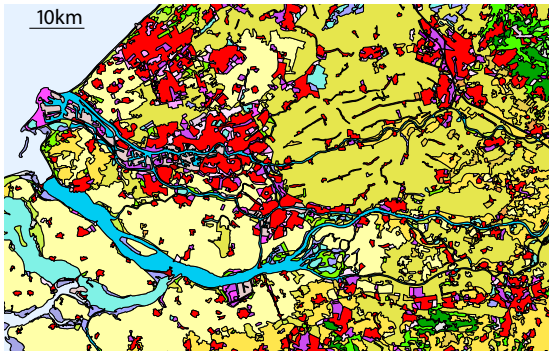
Table 4.7: Empirical results: comparing merge operation with split operation

		Buchholz (clip)		CORINE 2000 (clip)	
original	edges	1 564		6 635	
	faces	525		2 471	
merge	edges	2 533	1.6×	9 795	1.5×
	faces	1 049	2×	4 941	2×
split	edges	4 913	3.1×	15 535	2.3×
	faces	2 498	4.8×	8 840	3.5×

operation is applied. The second row in Table 4.7 shows the result of applying only the merge operation, while the third row shows the results of applying SplitArea. In the case of applying a merge operation, a new face (having a new identifier) was introduced for the face that replaces the two old faces being merged. Therefore, the number of face rows is exactly 2 times the original number of faces minus 1 (total number of nodes in a binary tree). When applying only the split operation, we also get 1 face less per operation, so we can conclude that the total number of faces could be equivalent to the number of faces as a result when merging. However, the difference of the number of split and of merged faces is caused, because faces, which played a role in the split operation of one of their neighbours (not because they were split themselves), were stored in the tGAP structure (needed because geometry of the face has changed).

When looking at the number of edges, new edge records are only stored when these edges are merged (for more details, see § 4.1). In case of applying SplitArea, the lengthened external chains, together with new edge geometries as result from the split, are stored in the tGAP structure. All in all, the number of primitives to be stored *has* to be higher with SplitArea than when applying a merge operation, mainly because new boundary geometry is generated.

Figure 4.30 shows 2 map series as result of applying the two generalisation operations. Although the differences between the two map series are subtle, they give an impression of what can be accomplished with both operations (without tweaking compatibility values much). Merging leaves original boundaries of objects untouched, while SplitArea introduces new geometries. Because of this, the merge result seems to be a more 'all or nothing approach', specially when compared to the split operation, which can operate more subtle (*e. g.* see urban areas in the Northwest part of the maps). However, in this experiment the



(a) Most detailed representation of European land cover dataset.

Figure 4.30: A series of maps derived from the tGAP structure. The series illustrate the (subtly) differences between the merge and split operations (continues on p. 137).

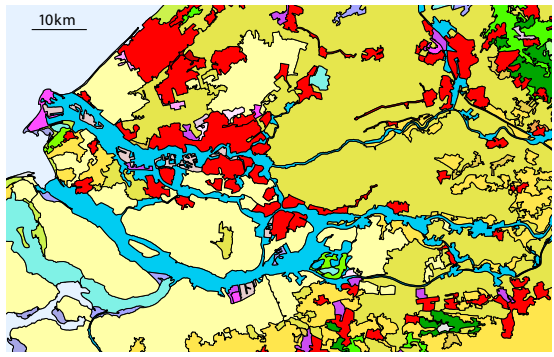
merge operation is hindered by the fact that the compatibility values were not tweaked much (and only were based on geometric criteria). Therefore, in some cases land features and water features are not prevented from merging. Therefore, when making the generalisation process more optimal, it should in future work be considered to : 1. tweak the compatibility matrix, and 2. investigate how to determine when to apply the split and when the merge operation (*i. e.* take more thematic attributes of the polygons into account, for example only split linear infrastructure objects, while using the merge operation for other objects) and 3. represent linear features explicitly in the structure (linking these features to a set of edges – then linear networks, as often found in topographic data, can play a role in generalising the area partition as input for the tGAP structure).

4.4 Closing remarks

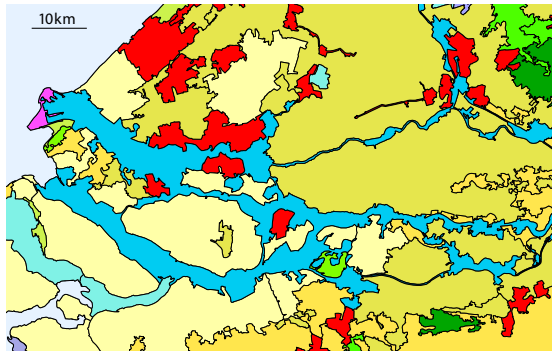
In this chapter we studied 3 generalisation algorithms (area merge, area split and line simplification) and their impact on the data structures, scrutinising the following research questions:

4. *How does minimal geometric redundancy influence the design of the data structures?*

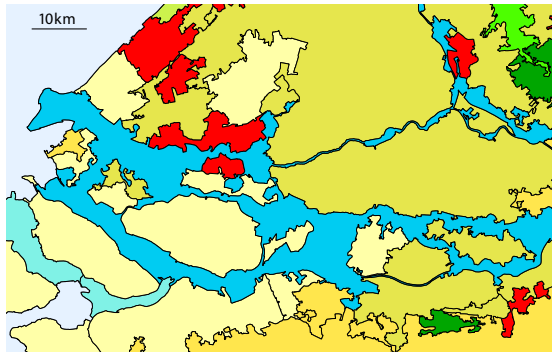
5. *How can we simultaneously simplify edges so that the result is topologically consistent?*



(b) Merge I (360 areas)

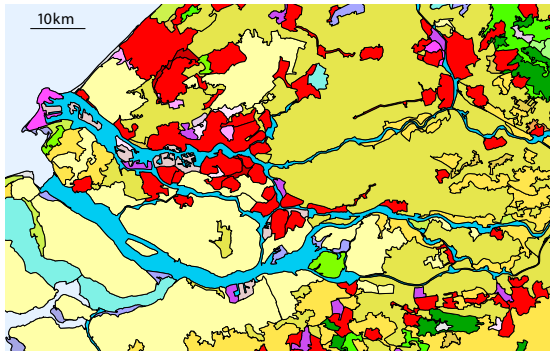


(c) Merge II (120 areas)

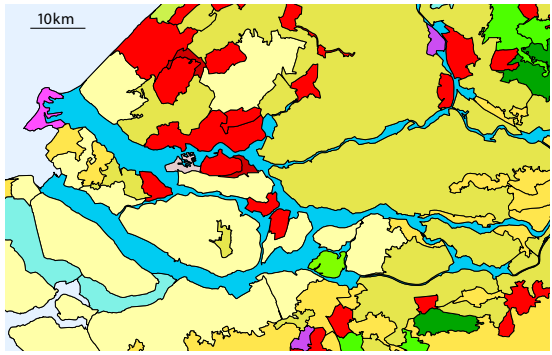


(d) Merge III (60 areas)

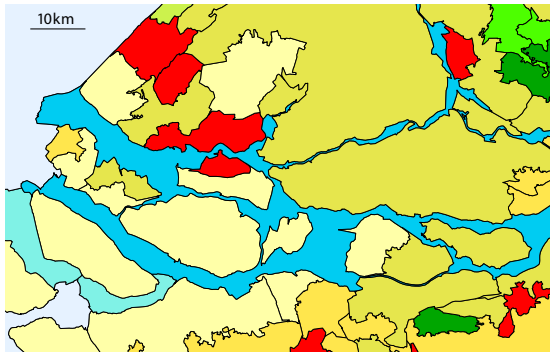
Figure 4.30: A series of maps derived from the tGAP structure. The series illustrate the (subtly) differences between the merge and split operations – merge operations are shown here (continues on p. 138).



(e) Split I (360 areas)



(f) Split II (120 areas)



(g) Split III (60 areas)

Figure 4.30: A series of maps derived from the tGAP structure. The series illustrate the (subtly) differences between the merge and split operations – split operations are shown here (continued from p. 137).

6. *How can we split linear features over their neighbours, instead of merging to one of their neighbours?*

During the design of a more data storage (and transfer) efficient version of the tGAP structure different alternatives were explored (§ 4.1). It was shown that the number of elements to be stored not only depends on the schema of the data structures, but that also choices have to be made when certain elements are stored in the data structures and when not, *i. e.* information is implicitly stored yet can still be derived from the stored information. This leads to a better trade off between storage and calculation-when-needed than before (much less data is to be stored and transferred, but with our lean alternative sometimes it is necessary to perform a lookup operation of the correct neighbouring face). Note that left/right references of the lean tGAP structure (references at the lowest importance value) have different meaning from the classic left/right references (therefore these attributes are stored in a column with a different name).

Secondly, we described an algorithm, based on the approach by Kulik et al. (2005), to simplify simultaneously a subset of polylines in a planar partition in a vario-scale context (§ 4.2). We improved the approach in two ways: better running time and preventing introduction of any topological error (when using a planar partition). Furthermore, we gave a description of the options that we have when employing this algorithm in practice. Another contribution is that we analysed how much the average number of points in the boundaries of the polygonal areas would grow without simplification, to choose the best simplification strategy, also from the perspective of the amount of data to be stored in the data structures.

Thirdly, SplitArea, the algorithm presented in § 4.3, is a useful tool to divide a polygonal area over its neighbours or to obtain a thinned representation of a face (*i. e.* collapse). It was shown that the algorithm can be modified to handle weighted and unmovable edges as input, making a weighted split possible. Based on the weights, the attractiveness of individual neighbours can be steered and measured (beforehand a goal can be set how to fairly divide an area). As a tool to use in the context of the tGAP data structure, SplitArea opens new possibilities for fine tuning the generalisation process: it is now possible to choose when to split/collapse (*e. g.* in case of linear road or water features) or to merge (other features). Note that SplitArea with all neighbours except one having a zero weight, is again the classic merge of the tGAP so SplitArea is more generic compared to the merge. In an implementation however this case can be

dealt with more efficiently by using the classic merge operation, *e. g.* no need to triangulate.

5 IMPROVING VARIO-SCALE DATA DISSEMINATION

This chapter investigates (and improves) the data structures by using them for transmission of the stored vector data over a network, such as the Internet. Where the previous chapter discussed mostly the layout of the data structures, this chapter tests how and if a 2D map can be derived from data structures (with the lean design of the data structures from the previous chapter), targeting constant number of map objects for delivery, independent from the map scale chosen for display. As map scale is not explicitly encoded – the level of generalisation is coded via the concept of 'importance' (introduced in § 2.4) – it is necessary to perform a mapping between map scale (which is dependent on device characteristics) and importance. This mapping is discussed in § 5.1. Then follows a description of how a map at a specific scale point can be derived by a stateless, thin client (§ 5.2). Further, it is investigated whether the lean structures make more dynamic map solutions possible by using the structures for retrieval of data for a thick client by incrementally adding additional details to an already sent map. As Haunert et al. (2009) described how to use the tGAP structures for progressive streaming of data, it is tested in § 5.3 whether their method proposed and the way the data structures are now composed (as described in the previous chapter) are capable of this type of use (by developing and studying a prototype implementation). Next to the test results, § 5.4 proposes the use of an additional data structure, so that the variable-scale approach can become more cache-friendly. This brings then the following benefits: less redundant data transfer during interactive use (compared to a stateless approach) and possibilities for offline operation by (partially) priming a cache at client-side. Finally, § 5.5 summarizes the results of this chapter.

Own publications

This chapter is based on the following own publications:

- Meijers, M. (2008). Retrieving tGAP data with a stateless client for visualization. RGI Project Report 233-03, Delft University of Technology, Delft.
- Meijers, M. (2011a). Cache-friendly progressive data streaming with variable-scale data structures. In *Proceedings of 14th ICA/ISPRS Workshop on Generalisation and Multiple Representation*, pages 1–19.

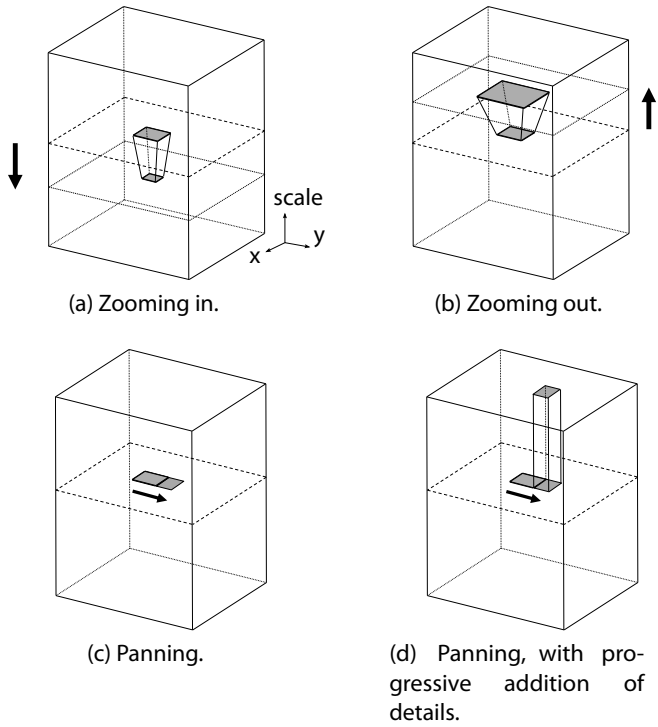
5.1 Quantitative importance-setting approach

A vario-scale planar partition to be stored with the tGAP structures will be created by an automatic generalisation process before online use. As shown in the previous chapters, the tGAP structures use explicit topological data structures for storage. The input data is validated. When the input data is 'clean' and 'valid', it is easier to implement generalisation operations. The three generalisation operators, that are implemented for the automatic generalisation are merge and split (of polygons) and simplification (of boundary lines). The merge and split generalisation operators (§ 4.1 and § 4.3) are applied as 'global optimisation': the least important object (according to some criteria) present in the complete dataset is being generalised.

A simplification operation (§ 4.2) is performed as post-processing step after one of the merge and split operations (*i. e.* after a merge or split the boundaries of the new area feature(s) will be simplified). Furthermore, all operators have been implemented in such a way that they do not introduce topological errors.

In an online usage scenario where a 2D map is retrieved from the tGAP structures, the amount of vector information to be processed has an impact on the processing time for display on the client. Therefore, as a rule of thumb, we strive to show a fixed number of (area) objects on the map, independent from the level of detail the objects have, in such a way that the optimal number of objects is displayed (*i. e.* optimal information density). This number is termed here the *optimal number* of map objects and will be used for retrieving data in such a way, that the amount of objects, *i. e.* faces and edges (with certain number of coordinates), to be retrieved on *average* remains constant per viewport (independent from which level of detail is retrieved) and thereby the transfer and processing time stay within limits.

The optimal number can be realised, because the generalisation procedures that create tGAP data incrementally lead to less and less data in the hierarchy, *i. e.* less data is stored near the top of the space-scale cube (SSC) and the extent of area objects near the top of the cube is considerably larger (with a limited number of coordinates in their boundaries) than at the bottom (with more coordinates in their boundaries). A cross section in this cube leads to a 2D map, see § 3.2.3. The extent of the viewport (*i. e.* the window through which the user is looking at the data) also implies that it is necessary to take a clip of data out of such a slice: when a user is zoomed out, the viewport of a user will lead to a big extent (the area to be clipped is large) and when a user is zoomed in this extent will become considerably smaller. For a user that performs a panning action it is necessary to move the extent of the clip within the slice. Figure 5.1 gives an illustration.



(a) Zooming in.

(b) Zooming out.

(c) Panning.

(d) Panning, with progressive addition of details.

Figure 5.1: Zooming & panning with vario-scale data explained with the SSC (after van Oosterom and Meijers, 2011a).

Positioning the height of the slice in the cube, together with taking the clip, should lead to a constant number of objects to be visualised. To realise the position of the cross section means that the question to be answered is 'which importance value corresponds to the map scale at the client?' In practice this will mean that a thin client will only have to report the current extent (plus device characteristics) to a server and then can be sure to receive the right amount of data for a specific level of detail as the server can translate this extent to a suitable importance value to query the data structures. Make note that this on *average* is the right amount of information, as there may be regions with more dense content than on average (*e. g.* rural vs. urban area). For a thick client, that has more capabilities and where it is possible to perform more advanced processing, it should be possible to first receive a coarse map, and then to incrementally receive additional details when a user waits longer, leading to a more detailed map (with additional map objects). The mapping of map scale to importance in this scenario is necessary to determine when to stop sending additional details. Independent from whether a thin or thick client is used: it is key to know what the importance value is that is implied by the current map scale of the client (*i. e.* positioning the height of the slice).

We will first show that a filled tGAP data structure will be able to supply data for a specific scale range. As we will see, this range is dependent on the optimal number of objects we want to show together with device characteristics. Second, we will show how to translate a map scale into an importance value, with which it is possible to take a cross section of the SSC, encoded by the tGAP structures. This cross section then leads to a map with the desired amount of detail. The focus with which the generalisation process is executed makes that this selection is possible.

VALID VARIO-SCALE RANGE. Clearly, the cube has a bottom (maximum scale) and a top (minimum scale) and in between the optimal amount of data can be supplied: Depending on device characteristics and how the optimal number is set up, these minimum and maximum scale values will be different. By our rule of thumb of average constant data density, we set up the optimal number of objects, O_v as a constant value for a given dataset and viewport[1], *e. g.* 1 000 objects:

[1] This value can, when the characteristics of the viewport on which the dataset is displayed are also known, be derived from the intended map scale specification of a data set: how many objects will on average be shown on the given viewport, when the viewport is set to the intended map scale.

$$O_v = \text{required optimal number of objects for viewport} \tag{5.1}$$

For the sake of simplicity, in the following discussion let us assume that the base map (the planar partition that we used as input for creating the tGAP data), its objects and the viewport are square and that the base map has an approximately regular distribution of data over its domain. Furthermore, we will express all units in meters.

First, we define how many objects we have on the base map:

$$n = \text{total number of objects} \tag{5.2}$$

Together all these objects cover a certain area:

$$a = \sum_{i=1}^{n} \text{area}(o_i) = \text{total area of } n \text{ objects in planar partition} \tag{5.3}$$

As we assume that the area is square, we can get the size of its diagonal:

$$a_d \approx \sqrt{2a} = \text{diagonal of area in meters} \tag{5.4}$$

Based on this size we can give an estimate for the average diagonal size of an object (assuming that all objects are square):

$$\bar{a}_d \approx \frac{a_d}{n} = \text{average diagonal size of an object} \tag{5.5}$$

The characteristics of the viewport are determined by its size in pixels and the physical properties of the screen on which the viewport is depicted (*i. e.* the number of pixels per inch, PPI, for the underlying raster). The second last Styled Layer Descriptor (Lalonde, 2002) standard puts this as follows: 'the "standardized rendering pixel size" is defined to be 0.28mm × 0.28mm (millimeters). Frequently, the true pixel size of the final rendering device is unknown in the web environment, and 0.28mm is a common actual size for contemporary video displays' (Lalonde, 2002, p. 27). This calculation is based on a default PPI of 90, as 1 inch at 90 PPI means that the size of a raster pixel approximates 0.28 mm. We now can determine the size of the viewport diagonal in real world units, V_d (note that we use meters):

$$V_p = \text{viewport diagonal in pixels} \tag{5.6}$$

The real size of the diagonal V_d is then:

$$V_d = V_p \times \frac{0.0254 \text{ m } (= 1 \text{ inch})}{\text{PPI}} = \text{viewport diagonal in meters} \tag{5.7}$$

From the viewport diagonal in meters (the screen PPI combined with viewport size), the applied generalisation process to create tGAP data and a fixed, optimal number of objects to show on the map we can get to a scale range for which the tGAP data structure can produce a map (*i. e.* a clip out of the correctly positioned slice), that fulfills having the optimal number of objects O_v.

If a user is viewing the base map, with all the original details, no additional details can be shown (as the base map is already the most detailed version) – this data is stored near the bottom of the SSC. There is thus a minimum scale denominator, after which the tGAP structures will not be able to supply O_v objects (the original objects will just be 'blown up' by zooming, which leads to fewer than O_v objects on the screen). When the user sees exactly O_v objects with the level of detail of the base map, this means that in this case along the diagonal of the viewport we will see approximately $\sqrt{O_v}$ objects, which have an average diagonal size $\bar{a}_d$. Therefore an approximation of the minimum scale denominator in this case will be the ratio of the real world size of the diagonal of the objects and the real world size of the diagonal of the viewport:

$$\text{minimum denominator} \approx \frac{\sqrt{O_v} \times \bar{a}_d}{V_d} \tag{5.8}$$

On the other side of the SSC (near the top), a similar effect takes place. If a user fits the whole dataset exactly in the viewport, we can get the denominator of this situation as follows:

$$\text{maximum denominator} \approx \frac{a_d}{V_d} \tag{5.9}$$

The user now views the dataset in such a way that the full extent exactly fits in the viewport. The tGAP data structures will supply O_v objects for the full extent. Let us look at what happens when a user will zoom out. The ultimate generalisation that is present in the tGAP structure is the object in the root node of the face tree, in which the initial base map is generalised to 1 object. It is thus clear that if a user zooms out further than the full extent of the dataset, that in the end only 1 object would be returned by the tGAP data structure, which is clearly less than the wanted number of O_v objects. Hence the approximation of the

maximum denominator, as this is the denominator for which it still is possible to supply O_v objects. If a user zooms out more, it is likely that the structures will not supply O_v objects, but less (*i. e.* taking a slice between the root of the tGAP face tree and the place where a slice leads to O_v objects will always generate less than O_v objects).

All in all, by definition outside the scale range defined by the minimum and maximum denominator, less than the optimal number of objects will be produced (the user is zoomed in or out too much, both leading to less than the optimal number of objects) and it will be necessary to clamp values between the minimum and maximum scale.

SCALE-TO-IMPORTANCE MAPPING. Now that we know what the valid scale range is for which it is possible to use this tGAP data set and that we want to retrieve the optimal number of objects O_v on average, we need to carry out a mapping in which we translate the scale denominator of the current viewport of a user to an importance value in the tGAP structure (so that we position the slice at the correct height in the cube). In § 5.2.2 an experiment will be described that shows that we can query the data with hypothetical viewports and thereby empirically validate the theoretical mapping function described here.

If we assume that the user wants to see O_v objects, this is the case when a user is exactly zoomed to full extent (*i. e.* when the viewport shows exactly the area a from Equation 5.3). If a user now zooms in, *e. g.* by reducing the viewport width to half of this original size (zooming in by powers of two), this means that we need $2^2 = 4$ times more objects on a *complete* slice of tGAP data (the reduction of displayed area follows quadratic relationship). To the user only $\frac{1}{2^2} = \frac{1}{4}$ of this slice will be shown by clipping the relevant area based on the new location of the viewport. It is thus evident that the ratio of the current viewport size and the size of the full extent of the data set determines (together with the optimal number O_v) where a complete slice of data needs to be taken, so that clipping of this slice then leads to the optimal number for the current viewport.

A complete slice of data with the correct number of total objects present is relatively straightforward to obtain: 'replay' in reversed order from the top of the data structure this amount of generalisation operations and the `imp_low` value of the object that is generalised is the importance value for producing O_v. Here we use the knowledge of how the generalisation procedure for tGAP produces stepwise less data (in every step either an area object is merged with its neighbour, or it is split over all its neighbours, which frequently produces a unique `imp_low`

value for the resulting objects); ordered `imp_low` values describe the order in which generalisation operations have been applied.

The mapping can thus work as follows:

1. Take the scale denominator d and real world size of the viewport diagonal V_d;
2. Clamp d between the minimum and maximum scale denominator, this leads to d';
3. Calculate the real world area b of the viewport based on d' and V_d;
4. Calculate scaling factor f:
$$f = \frac{a}{b}$$
5. Derive the total number of objects that should be on a new and *full* slice of data, *i. e.* the count c to query data structures with: $c = O_v \times f$;
6. With this count c it is possible to query the face table of the data structures to get an ordered list of all generalisation events at their importance values and taking the correct value by looking at how many there should be skipped (the offset clause in the following query):

   ```
   SELECT DISTINCT
   imp_low
   FROM
   <dataset>_face
   ORDER BY imp_low DESC
   LIMIT 1 OFFSET <c>
   ```

   This query leads to an importance value, with which a slice of tGAP data can be retrieved and for which the clipped area then – on average – leads to the O_v optimal number of objects.

Note that clamping based on scale denominators, as shown here, can also be performed on the count value that is calculated after applying the scale factor: this value should lie between O_v (optimal number of objects, Equation 5.1) and n, being the total number of objects in the base map (Equation 5.2). Also note that f can be determined based on the current scale denominator d and the maximum scale denominator (Equation 5.9).

5.2 2D MAP FOR A THIN CLIENT

This section provides details on retrieval and transfer of the data from the topological data structures at a server to a thin client. Communication between the thin client and a server is here performed in a stateless way, which means that the client requests data after each user action (zoom in, out or pan) and does not maintain state of which part of the data is already retrieved (each request for data is independent from the other requests).

5.2.1 *Retrieving a 2D map*

At the client-side a topological structure of a 2D map is kept – the `TopoMap` object. The `TopoMap` object contains all retrieved topological primitives. These primitives will have to be transformed into geometries, which then can be visualised to an end user. As interaction is stateless, for every new request for data with the server a new `TopoMap` object is created (and a previous one is abandoned).

Figure 5.2 shows that the `TopoMap` object at the client-side is implemented as a Doubly-Connected Edge List (DCEL, *cf.* de Berg et al., 2000), extended with a Ring class to handle Faces that have holes in their interior. Relations are implemented as memory pointers. From the topological primitives Simple Feature geometries (polygons) will be formed (when all pointers are set correctly this means that rings can be formed, and when there are multiple rings for a face that the rings having the largest bounding box has to be the outer ring of a 2D polygon).

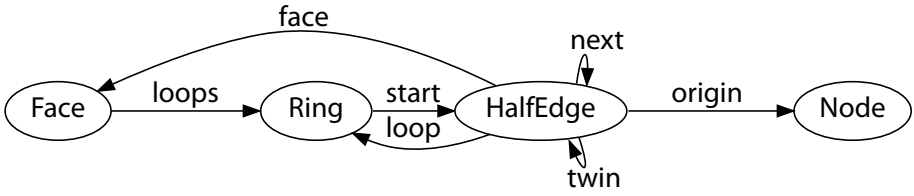
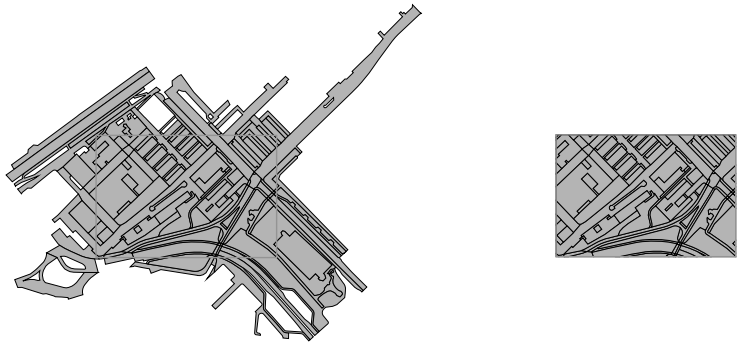


Figure 5.2: Structure of the `TopoMap` – which objects are kept in memory, plus their pointers. Basically the `TopoMap` object represents a Doubly-Connected Edge List (DCEL) extended with Rings to handle Faces that have islands in their interior.

We will discuss 2 possible options for data retrieval for a `TopoMap` for a thin client. Common for both options is that they need to: a. perform a mapping from map scale to an importance value (as discussed previously in § 5.1) and

b. have access to a function (implemented in the database server) to translate left/right pointers to the correct neighbouring face (as was shown in § 4.1 it is necessary to encode edges more compactly – preventing excessive numbers of edge records to be stored). Figure 5.3 illustrates the resulting map of the two options, after translating map scale to an importance value:



(a) Retrieval based on faces and administratively on edges (option 1).

(b) Retrieval based on faces and edges their bounding box (option 2). Edges are clipped before face geometries are reconstructed.

Figure 5.3: Data retrieval for a viewport (rectangle). Note that clipping was also discussed in § 4.1.

1. Select faces for the correct importance value (based on bounding box overlap) and then join edges administratively on a value that is implemented in the DBMS that returns the translated face pointer. This option leads to complete areas retrieved (*i. e.* also edges outside the extent of the viewport are retrieved).

2. Select faces and edges (both selections are performed on bounding box overlap, see § 4.1), then translate all face pointers for selected edges to the correct face pointer. In this option the information that is retrieved will be incomplete, *i. e.* implicit edges at the rim of the viewport need to be found. The relationship that the retrieved edges have with the rim of the viewport is used to find this missing information. After retrieval of the edge data, the edges are processed based on their bounding box: If an edge is completely inside, it is just added, ready to form polygons. Otherwise an edge likely intersects with the rim that consists of 4 edges, although an edge still might be completely outside as well (*i. e.* having

no interaction with the viewport at all). Those edges can be processed with an extended Liang-Barsky algorithm for line clipping (see *e. g.* Hearn and Baker, 2003, chapter 6). Such a clipping algorithm works as follows: All edges their geometries that possibly intersect the query window, are processed segmentwise from start node towards end node. During this process information is gathered on which parts of the geometry of the edge are inside the query window.

For each part of the geometry that lies inside the query window a completely new edge is created, with information for the start and end node and the faces that are adjacent to the edge part, plus the geometric part that is inside the query window. New start and end nodes are recorded if this is necessary: an identifier is created based on the coordinate where the edge is clipped. These clipped coordinates are stored in a clipped nodes list, together with the four corners of the query window, which are also added. This clipped nodes list is sorted on the angle the nodes have with respect to the viewport centre. The newly created edges are added to the `TopoMap`, as well as all edges that lie between all pairs of clipped nodes on the rim of the viewport.

5.2.2 *Experiment*

We conducted a small experiment where we produced a series of viewports for which we retrieved data via Option 2. A hypothetical viewport size of 640 × 640 pixels at 90 DPI was used (approximately 18 × 18 cm), together with a optimal number of $O_v = 250$. We utilised a quadtree-like organisation for creating extents which mimic a user zooming in on, and for each level panning, the data set. Figure 5.4 gives an illustration of the viewports (and their position within the SSC). The viewports were set up as follows: start with full extent of the whole dataset as initial square, then recursively split the extent of a quadrant in 4 sub-quadrants, creating hierarchical organisation of quadrants until quadrants will lead to less than O_v objects. Each quadrant now is used as a viewport by means of which we will query the tGAP data set. Figure 5.5 illustrates the amount of data that we want to retrieve for the viewports. To position a level of viewports at their correct height in the SSC, *i. e.* perform the mapping from scale to an importance value, we employed the approach described in § 5.1.

Figure 5.6 highlights the results for the Buchholz data set. For every 'slice' (*i. e.* a level consisting of a set of viewports) both the number of faces and number of coordinates inside all viewports were aggregated into a boxplot. Although the

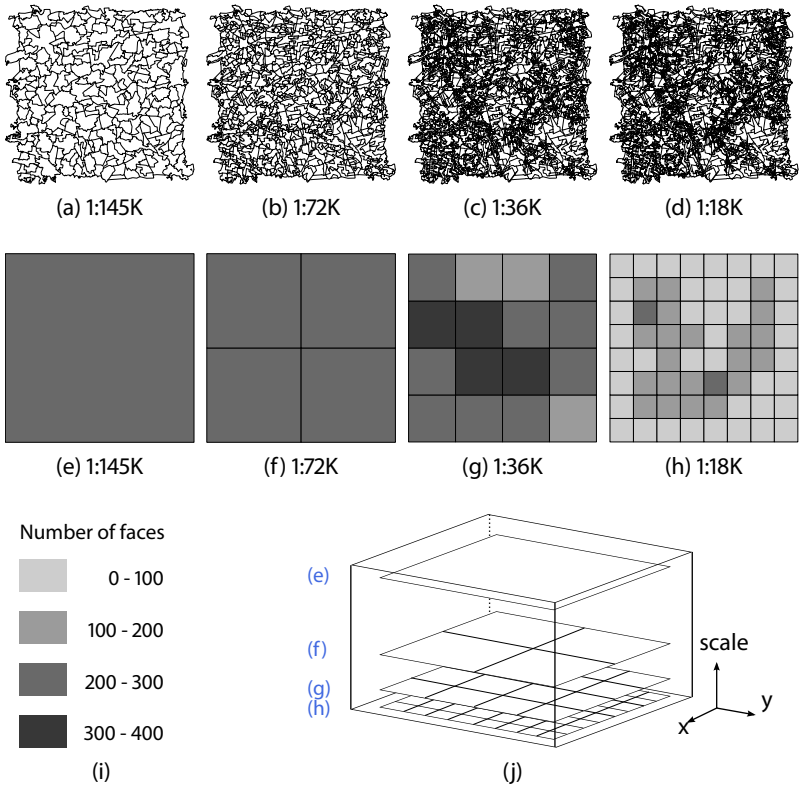
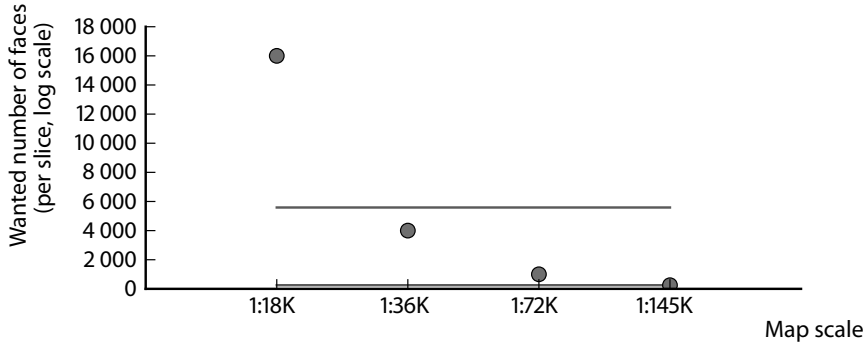
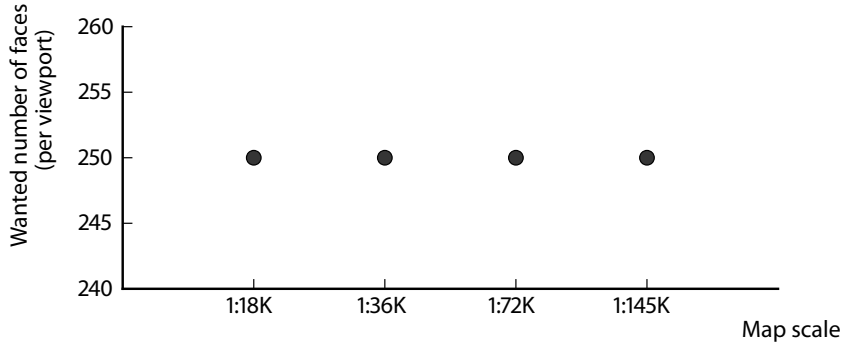


Figure 5.4: Viewports used in experiment for retrieving data. Figure 5.4a, 5.4b, 5.4c and 5.4d show the slices of data (after mapping scale to importance). Figure 5.4e, 5.4f, 5.4g and 5.4h show the viewports used; the colour of a viewport indicates how many faces are retrieved (Figure 5.4i shows the legend). An indication of the position in the ssc of the used viewports is shown in Figure 5.4j.

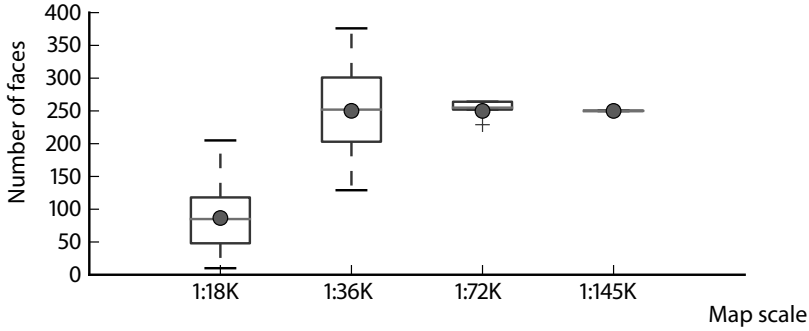


(a) The number of faces that we will try to retrieve for a whole slice of tGAP data. The horizontal lines in the graph depict the minimum (*i. e.* the lowest line is the optimal number set up for a viewport = 250) and the maximum number of faces (*i. e.* the highest line is the total number of faces present in the dataset = 5587) which will be retrieved (the required number of faces for a whole slice is clamped between these two values).

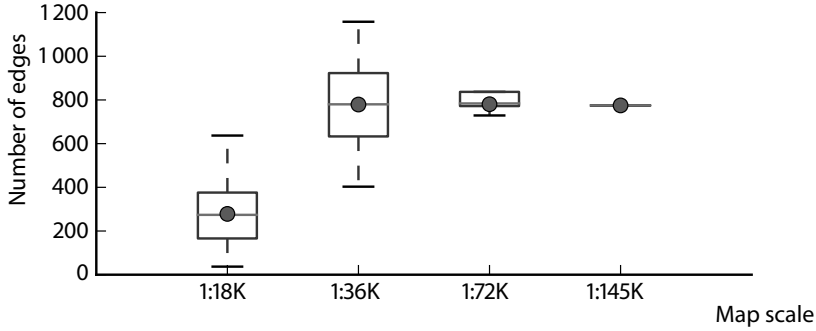


(b) The number of faces that are set up for data retrieval of the maps in our experiment for 4 different map scales.

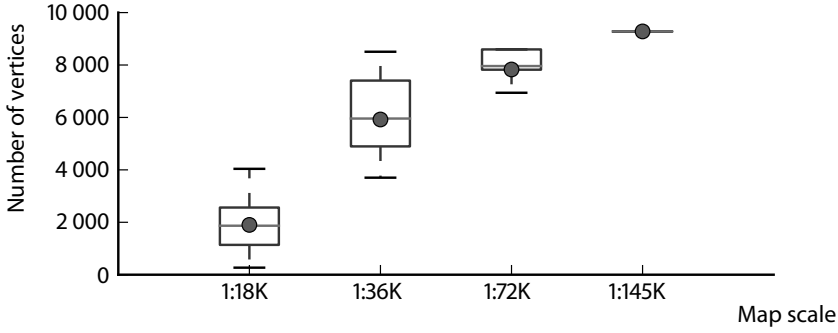
Figure 5.5: Settings for data retrieval. The number of maps which have been retrieved in the experiment vary with the map scale: 64 different viewports are used to create maps for the map scale 1:18k, 16 for 1:36k, 4 for 1:72k and 1 for 1:145k. The expected result is that the obtained map density is constant on average.



(a) Spread in the retrieved number of tGAP faces. This plot shows that indeed about 250 objects are retrieved per viewport. Note that sometimes more, and sometimes less, data is retrieved, however, on average – the dot on top of the boxplot – the resulting number of faces fulfills the wanted number of 250, except when we are zoomed in too far (namely at the map scale 1:18k).

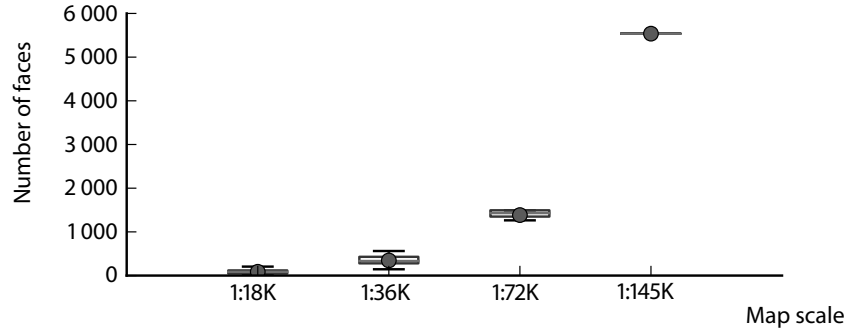


(b) Spread in the retrieved number of tGAP edges.

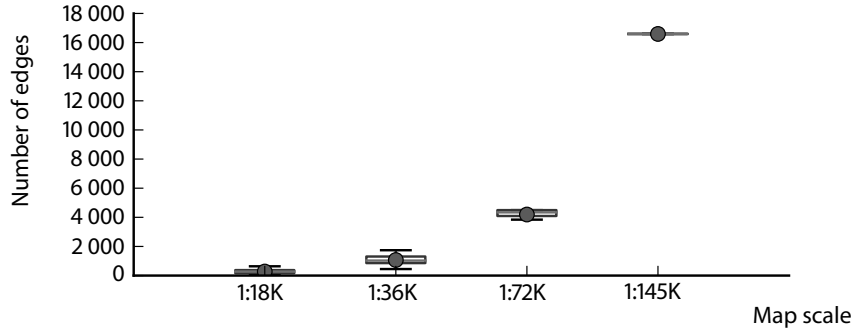


(c) Spread in the retrieved number of tGAP vertices.

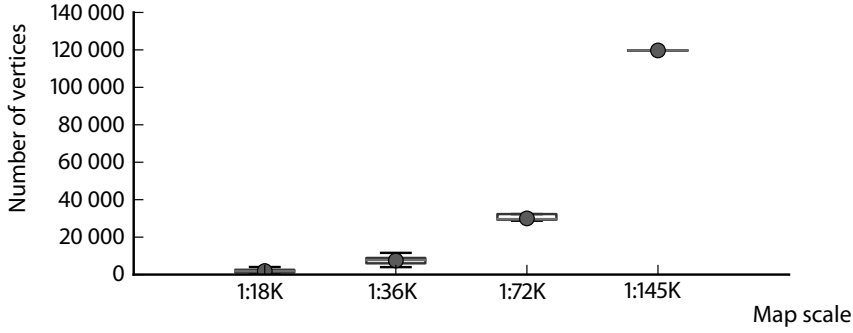
Figure 5.6: Spread of actually retrieved number of data elements, when the tGAP data structures are used for data retrieval.



(a) Spread in the retrieved number of original faces.



(b) Spread in the retrieved number of original edges.



(c) Spread in the retrieved number of original vertices.

Figure 5.7: Spread of actually retrieved number of data elements, when the original data set is used for data retrieval (*i. e.* no generalization takes place and original data set is queried for retrieval).

number fluctuates, it is shown that the average number – the dot in every boxplot – approximates fairly well the optimal number that was set (*i. e.* O_v was set to 250). As reference, Figure 5.7 illustrates what happens when just the original data is queried in the same way (*i. e.* when no generalisation is taking place).

The boxplots provide empirical evidence that it is indeed possible to keep the number of objects, but also the number of coordinates, to be retrieved under control, albeit for a specific scale range and depending on the viewport size, device characteristics and the optimal number of objects set. That the number of coordinates is not excessive is accomplished by the line simplification procedure (see § 4.2), that was used to produce output geometries with only a limited number of coordinates (*i. e.* the criterion for resulting output of line simplification was focused on the amount of reduction of data – only keeping half of the original amount of vertices for a merged polyline).

5.3 Progressive data streaming

Progressive data streaming can make more dynamic map solutions possible by using the tGAP structures for incremental data retrieval, *i. e.* adding additional details to an already sent map. This way an end user can rapidly get an initial coarse overview, which is then refined with additional data. This section discusses step by step a possible architecture for progressive data streaming.

Figure 5.8 shows the visualisation pipeline, from pre-processing steps to the client that processes incremental updates to show first a coarse map, then gradually refined with additional detail until the required amount is reached. Subsequently, we will now discuss every step, highlighted with a number, in the architecture.

5.3.1 *Server-side*

This section discusses how the data is structured at the server-side and what steps are needed to transform the data from these structures into data packages (Figure 5.8, step 1 and 2) that are transferred over the network (Figure 5.8, step 3 and 4, see § 5.3.1).

database tables. Figure 5.9 shows the database tables that are needed for progressive data transfer. Edges (polylines) have a prominent place in the database: the edge table is the 'centre of gravity' in the structure. The `imp_low` and `imp_high` attributes define the range of map scales for which a topological

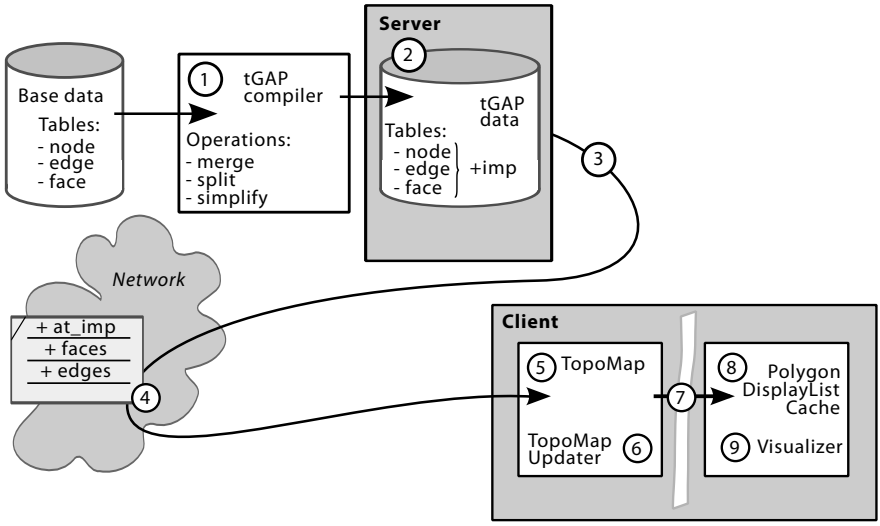


Figure 5.8: Visualisation pipeline for progressive data streaming.

entity is valid and has to be shown. Face references that are stored are limited to the neighbours that are adjacent at the start of such a scale range (`left_face_lowest_imp` and `right_face_lowest_imp`) and which faces are neighbouring at the end of this range (`left_face_highest_imp` and `right_face_highest_imp`). This way it is prevented that a lot of duplicate edge records have to be stored, while the only thing that changes (due to a generalisation operation) is a neighbouring face (see § 4.1). Note that the two extra face pointers (at the high end of the scale range) are added compared to § 4.1, as this is useful for replaying generalisation operations progressively, while traversing the tGAP DAG from top to bottom (in a strict sense, these extra pointers are not necessary, but this takes away the necessity to perform a translation of the face pointers using the tGAP face tree hierarchy to the high end of the scale range, before sending the edge). Because the Collapse/Split operation splits Faces over multiple neighbours, the resulting hierarchy is not necessarily a tree structure any more, but a Directed Acyclic Graph structure (DAG), which is reflected in the separate `face_hierarchy` table, where per parent-child combination a record is stored.

The geometry of the edges can be merged after a merge or a split operation, if after application of a generalisation operation nodes are present in the topology of degree 2. These merged geometries will then be simplified with the modified Visvalingham-Whyatt line simplification algorithm, taking into account the

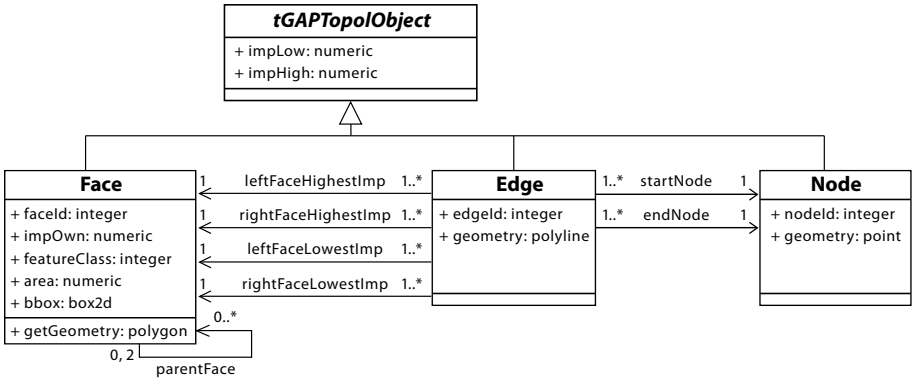
topological correctness of the result (see § 4.2 for more details). Note that the line simplification leads to a new edge record (as the new geometry has to be stored). However, the average number of coordinates per edge will remain constant, as the simplification tries to remove half of the points of the lines that were merged (therefore edges remain similar in complexity in terms of vertices) – this is important for transfer over a network, because if all original vertices were kept, too much unnecessary detail has to be transferred from server to client.

RETRIEVAL QUERIES. As tGAP data in our prototype is stored in an object-relational database, depending on whether standardised features are available in such a database system, queries for progressive data retrieval can be formulated in one of the two following ways. Option 1 is to use a hierarchical query for face record retrieval (using the `parent_face_id` attribute of the faces) and a sorted set of edge records. The Structured Query Language (SQL) standard specifies hierarchical queries by way of recursive common table expressions (CTEs). Figure 5.10 shows the CTE for retrieval of faces, using the hierarchical relationship for faces stored in the face table. Note that the `union all` part of the query references itself via a self-join. Option 2, when the database system does not have CTEs implemented, is both using sorted sets for face records as well as for edge records. Figure 5.11 illustrates this. Note that the query for retrieval of edges for both options is the same.

5.3.2 *Network*

COMMUNICATION CHANNELS. A client will need 2 channels for communicating with the server: a communication control and a data channel. Over the communication control channel the client will give commands to the server: *e. g.* start retrieval for zoom-in or zoom-out, pause retrieval and stop retrieval. Data packages – that contain the incremental updates for the client-side – will be transferred over the data channel. The data packages and the relationship with the queries will be explained next.

PACKAGE LAYOUT. Incremental updates are realised by reading the data packages at the client-side and processing the updates contained in the data package. The whole stream consists of multiple packages, which are send in order defined by the `imp` value. The structure of *one* data package is the following:



(a) UML diagram

```
CREATE TABLE tgap_faces (
face_id integer,
imp_low numeric,
imp_high numeric,
imp_own numeric,
feature_class_id integer,
area numeric,
bbox box2d);
```

(b) Face table

```
CREATE TABLE tgap_face_hierarchy (
face_id integer,
parent_face_id integer,
imp_low numeric,
imp_high numeric);
```

(c) Face hierarchy table

```
CREATE TABLE tgap_edges (
edge_id integer,
imp_low numeric,
imp_high numeric,
start_node_id integer,
end_node_id integer,
left_face_lowest_imp integer,
right_face_lowest_imp integer,
left_face_highest_imp integer,
right_face_highest_imp integer,
geometry geometry);
```

(d) Edge table

Figure 5.9: UML diagram and the database tables at server-side. Note that the DAG is stored in a separate face hierarchy table: this table holds multiple records per face when it is split, *i. e.* for every parent one record is stored.

```
with recursive hierarchy as (
select
face_id, imp_low, imp_high, parent_face_id
from
<dataset>_face_hierarchy
where parent_face_id = -1
union all
select
t1.face_id, t1.imp_low, t1.imp_high, t1.parent_face_id
from
<dataset>_tgap_face_hierarchy t1
join
hierarchy as h on t1.parent_face_id = h.face_id
) select * from hierarchy;
```

Figure 5.10: Hierarchical query. This query retrieves faces in the face hierarchy in level-order.

AT IMP an `imp` value, that describes the scale point where the change has to be applied for (`imp_high` for zoom-in, `imp_low` for zoom-out).

FACES faces to be added and faces to be removed. The parent-child relationship of the faces encodes this, together with the direction of the user action (zoom-in or out determines how to traverse the relationship).

EDGES the edges that have to be added: geometry (polyline) plus face- and node-references.

PACKAGES AND THEIR RELATION WITH QUERIES. Tables 5.1 and 5.2 show the first part of the resultset that is retrieved by the queries. Packages will contain data from both resultsets, grouped by `imp_low` (in case of a zoom-out action) or `imp_high` (zoom-in) attribute values.

Table 5.1: Set of face records, sorted descending by `imp_high` – zoom-in. First three records encode that Face 6756 is split into Faces 6746 and 6646

face_id	feature_class	parent_id	imp_low	imp_high
6756	4107	1	1354522168	17421338713
6746	4107	6756	1109020325	1354522168
6646	2101	6756	178039195	1354522168
6736	4107	6746	786370390	1109020325
6576	4107	6746	130670112	1109020325

```
SELECT
fh.face_id::integer,
f.feature_class::integer,
fh.parent_face_id::integer,
fh.imp_low::bigint,
fh.imp_high::bigint,
ST_AsBinary(f.mbr_geometry::geometry) as mbr_geometry
FROM
<dataset>_tgap_face_hierarchy fh
JOIN
<dataset>_tgap_face f
ON
fh.face_id = f.face_id
ORDER BY
imp_high DESC,
parent_face_id ASC

SELECT
edge_id::integer,
imp_low::bigint,
imp_high::bigint,
start_node_id::integer,
end_node_id::integer,
left_face_lowest_imp::integer,
right_face_lowest_imp::integer,
left_face_highest_imp::integer,
right_face_highest_imp::integer,
ST_ASBINARY(geometry)
FROM
<dataset>_tgap_edge
ORDER BY
imp_high DESC
```

Figure 5.11: Queries for both edges and faces (sorted sets). Network packages can be realised by executing these two queries from which the resultsets are 'intermingled' by a process at the server-side into packages per `imp` value.

Table 5.2: Set of edge records retrieved from database at server-side, sorted descending by `imp_high` – zoom-in. Note that each edge record also contains an associated geometry (polyline), but that this is not shown. Edge 9183 forms the boundary of Face 6756 (compare imp_high of Face and Edge records) and when Face 6756 is split in 2 Faces, Edges 9171, 9172 and 8680 form the boundaries of these 2 Faces, replacing the old boundary 9183.

id	imp_low	imp_high	sn	en	lf_lo	rf_lo	lf_hi	rf_hi
9183	1354522168	17421338713	343	343	6756	1	6756	1
9171	1109020325	1354522168	896	343	6746	6646	6746	6646
9172	1109020325	1354522168	343	896	6746	1	6746	1
8680	44573839	1354522168	896	343	6336	1	6646	1
9147	776041941	1109020325	435	5	6726	6576	6736	6576
9160	786370390	1109020325	5	896	6736	1	6736	1
8968	130670112	1109020325	435	343	6576	6506	6576	6646
8967	130670112	1109020325	343	5	6576	1	6576	1
9159	786370390	1109020325	896	435	6736	6646	6736	6646

Creation of the packages can be realised by opening two cursors to the database, that query the database and from which the resultsets are 'intermingled' by a process at the server-side into packages per `imp` value. Most of the time, database cursors can be implemented in such a way that not the whole resultset needs to be pulled into memory of the server-side process, leading to relatively low memory requirements at the server-side. The server-side process iterates over both resultsets at the same time — first face records and when a change in `imp` value is detected then switch over to edge records. The process outputs a package when enough relevant content is gathered (*e. g.* both face and edge records with same `imp_high` value in case of zoom-in operation are obtained from the 2 cursors). Based on this data package, the client will update its local data structures, which is discussed next.

5.3.3 *Client-side*

TOPOMAP. At the client-side again a topological structure of a 2D map is kept – the `TopoMap`. This map here (in contrast with the thin client from § 5.2) is incrementally being updated when a data package arrives over the network, by a component called the `TopoMapUpdater` (Figure 5.8, step 5 and 6). The task of the `TopoMapUpdater` is to unpack the data packages arriving from the network and to perform incremental updates on the `TopoMap` structure.

TOPOMAPUPDATER. The `TopoMapUpdater` updates the `TopoMap` object incrementally. It performs updates based on the incoming packages in 5 steps:

STEP 1 Unpack data package into new faces and edges.

STEP 2 Remove unneeded Topo primitives: the client can deduct this from the face hierarchy in the incoming package — which faces are not valid any more (because these will be replaced by more refined faces) is in the package, navigate from these faces to their edges and remove edges that are not valid any more, *i. e.* `at-imp` does not overlap their `imp` range. In the process of removing edges it is necessary to keep track of broken and orphaned Loops.

STEP 3 Add new primitives from package to the `TopoMap` (faces and edges).

STEP 4 Put back broken and orphaned loops to faces where they belong — geometric searching might be required, as not all intermediate face pointers are kept explicitly.

STEP 5 Reconstruct polygon geometries (of new faces, but also of their neighbours if simplify had changed boundaries of neighbours) and put polygons into the visualization queue together with instructions which polygons to delete from the Display List cache.

POLYGON DISPLAY LIST CACHE. Figure 5.8, step 8 and 9 show that in the client OpenGL display lists are used for caching drawing instructions. Per polygon (identified by the `face_id` attribute) a Display List is created (Shreiner et al., 2005). Updates placed in the visualization queue allow Display Lists that are not active any more to be removed and new drawing instructions for Polygons have to be placed into the Display List cache. This entails triangulation of the incoming polygons, as OpenGL can only handle convex objects. Once placed in the Display List cache the visualization loop will execute the drawing instructions. The planar partition that will be drawn (with a certain frame rate) is at this stage in the visualization pipeline only a set of low-level OpenGL drawing instructions.

5.3.4 *Experiments*

This section gives details about 2 experiments that were run: a demonstrator for progressive data streaming was built and an analysis was performed on how

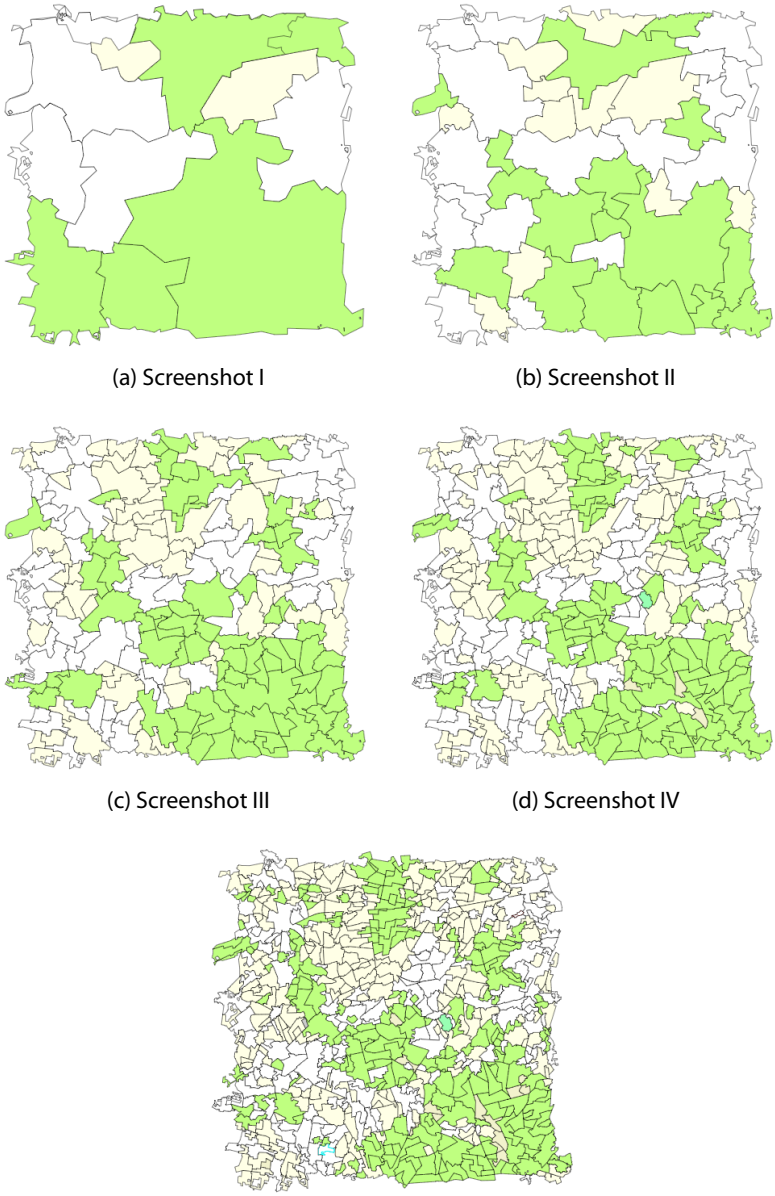
much data a progressive scenario requires compared to a map requested with the highest level of detail.

A DEMONSTRATOR FOR PROGRESSIVE DATA STREAMING. A demonstrator has been implemented, as example of a fat client that retrieves the tGAP data from the database incrementally (Figure 5.12). However, this retrieval is not bound by the current viewport, *i. e.* the process starts at the top of the tGAP structures and traverses these downwards (thus zooming in), progressively streaming additional data to the user interface. Also, the process does not automatically stop when enough data is retrieved (although this can be realised by performing a mapping from the display scale to an importance value). WxPython[2] is used for creating the user interface, together with PyOpenGL[3] and a wrapper to the OpenGL triangulation library, created with Cython[4]. The client processes the received packages and visualises the result. The decoupled components (data retrieval and visualization are running in two separate processes) keep the client responsive (*i. e.* the map can be moved while data is streaming in) and the demonstrator gives a good initial idea of what progressive data streaming entails.

Furthermore, from the implementation exercise the following was learned:

- The initial data model that Haunert et al. (2009) used at the server-side has to be modified to support both zoom-in as well as zoom-out operations: the data structures also need left and right face pointers at the `imp_high` level (at the end of the scale range for which an edge is valid) – these two attributes can either be provided at the server-side by dynamically translating the face pointers when needed or by adding them explicitly to the edge table – then these are rich enough to be used for progressive data streaming, *i. e.* to perform incremental updates to the `TopoMap` object.
- Holes in polygons can be dealt with, but it is sometimes only possible to know where islands (holes) belong by using a geometrical check (*e. g.* by using a bounding box check). This is due to the efficient encoding of the edge records (only first scale and last scale, but no intermediate face pointers stored for edges).
- With the incremental updating approach, it is necessary to address topological primitives explicitly (based on their identifier in the `TopoMap`

[2] http://www.wxpython.org/
[3] http://pyopengl.sourceforge.net/
[4] http://www.cython.org/



(a) Screenshot I

(b) Screenshot II

(c) Screenshot III

(d) Screenshot IV

(e) Screenshot V

Figure 5.12: Progressive data streaming demonstrator user interface.

structure), a dictionary data structure has been used for that (in the C++ programming language one could use the `map` type from the standard template library for this).

SIZE OF ORIGINAL 2D MAP VERSUS PROGRESSIVE PACKAGES. Another experiment was conducted to see how much data needs to be transferred in the progressive scenario, compared with a 2D map having the original amount of detail (formed by topological primitives). The difference between the two is the price that has to be paid for additional network traffic of progressive data streaming with the tGAP structures (in exchange for the advantage that a coarse overview can rapidly be provided).

Table 5.3 shows the sizes of the 2D maps (that is, their topological primitives serialized into a text format) that were used as input for the tGAP structure versus the total size of all packages to be transferred over the network for the complete tGAP structure in a progressive streaming scenario (*i. e.* for all scales, all packages containing updates from coarse to detailed, serialized into a plain text format). From the table it is clearly visible that progressive data streaming with the tGAP structures can be realised within less than 2 times the size of the original dataset (which is in line with our theoretical analysis of § 4.1).

It must be said that the data for the tGAPs has been created with line simplification where the optimal number of points that should be preserved (in the polylines being merged) was set to half of the original input vertices at maximum. As illustration that it is important to perform this line simplification, Table 5.3 also shows the size of packages when no line simplification is performed in the compilation of the tGAP data. It is evident that this leads to more data that needs to be streamed (in our tests around a factor 3).

5.4 A CACHE-FRIENDLY AND STATEFUL SOLUTION

This section explores an alternative for making the progressive data streaming more cache-friendly as in the demonstrator the data were just streamed without taking into account a bounding box for requests (*i. e.* no spatial range selection based on the viewport was performed); this is not very realistic for larger datasets. Furthermore, the advantages of a cache-friendly solution are two-fold: 1. it leads to a faster user experience when the same area is visited again – no need to stream the same data again, as it already is available at the client-side and 2. it leads to possibilities to operate the solution even when no network access is

Table 5.3: Size of the original 2D map (serialization to text of topological primitives, *i. e.* edges and faces, that form the map) compared with size for full hierarchy, progressive data streaming (serialization to text of packages). The table shows that within a factor of 2 of the original size progressive data streaming can be realised with the tGAP structures. To reach this factor, it is necessary to perform line simplification; This is illustrated by the last column, which shows how much space the edge records take when they are not simplified.

Dataset + type of data	Size 2D map (kB)	Size Progressive (kB)	Increase (factor)	Size Progressive (non-simplified) (kB)	Increase (factor)
Hamburg (rural)	477	822	1.72×	1 515	3.17×
Colchester (rural)	3 377	5 366	1.59×	9 722	2.88×
Buchholz (rural)	5 044	8 597	1.70×	15 257	3.02×
Delft (urban)	8 369	13 802	1.65×	19 582	2.34×

available by priming the cache, *i. e.* placing the data (partly) in the cache on the client beforehand. With 'plain' tGAP structures it is difficult to communicate in a client-server environment which parts of the tree structures (that store free form vector objects with arbitrary shapes and geographic extents) have been already retrieved and is complex to administer.

Haunert et al. mentioned in their list of future work to investigate the use of a more regular block pattern for data retrieval. To obtain a regular block pattern, here the use of a Fieldtree (Frank and Barrera, 1990) is explored. The Fieldtree exists in different forms, but we will use the so-called Partition Fieldtree. This tree is a hierarchical data structure, composed of Fields (actually a Directed Acylcic Graph, a DAG, as each Field can have up to 4 parents). Fields are grid cells with a certain width and height. Each level of Fields covers the whole domain, has a different resolution (coarser Fields to the top of the hierarchy) and a different displacement which is a nice property for our problem. In this case the Fieldtree is *not* used as usual, where all features of one 2D map at one map scale are distributed over different Fields based on the smallest Field which totally contains an object's geographic extent. Here the Fields of the tree are used as the more regular blocks suggested by Haunert et al. (2009). Figure 5.13 shows that the Fields will get a height in the scale dimension and will therefore become 'real' 3D blocks.

The normal approach of creating tGAP data is to search for the least important object over the complete domain and apply a generalisation operation to this object. Here we modify this approach to limit the search within the extents of one Field (as suggested in van Putten and van Oosterom, 1998, § 5.2). A Field is

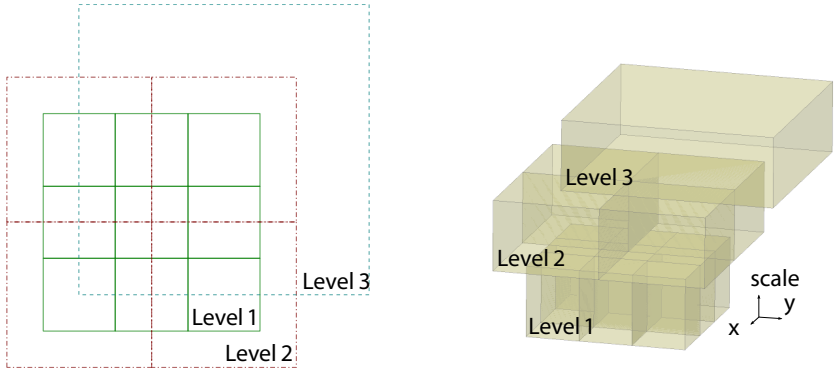


Figure 5.13: Fieldtree with 3 levels in 2D and 3D. Note that extents of Fields at a level higher in the hierarchy are twice the size of previous level and shifted.

generalised enough if a certain percentage of the objects that are falling within it have been generalised. Objects in a Field that do not completely fit (*i. e.* their bounding box overlaps with the border of a Field) can not be touched – *i. e.* these objects are '*locked*' and can not be merged, will not get a share of a split/collapse operation and their boundaries will not be simplified, in short, they will not be candidates for generalisation this round. When all data for the Fields of the most detailed level in the Fieldtree have been generalised enough, the next level in the Fieldtree will be used for continuing the generalisation process: Fields for this next level will be displaced and their extents will be larger this round. Because each level of Fields has a different displacement, the boundaries of the Fields are not fixed at one location in space. This also means that it is not very likely that objects that are locked at one level will also be locked at the next round of processing (*i. e.* objects that could not be generalised the first time, will most likely be generalised the next time, or when the Fields are large enough for the object to fit in).

When tGAP data is obtained with the help of the Fieldtree and the fields are also recorded (in the structure), the fields can also be used in a progressive data streaming scenario as initial filter step to obtain only a part of the tGAP data from the server-side. The characteristics of the Fieldtree (as it is a very regular structure) can be coded compactly. These Fieldtree characteristics are firstly transmitted from server to client. In subsequent steps, the client can then progressively retrieve data from the server by requesting the Fields. A client will have to start with a slice of data at the top or bottom of one (or more) Field(s). It

is therefore necessary to map scale to a level of the Fieldtree. Once the slice of data is retrieved, both 'locked' and not locked polygons for this slice are available at the client. As each Field is associated to a part of the tGAP structure, this part of the tGAP structure can now progressively be transferred from server to client. With this approach there is thus no need to split the polygons that form a polygonal coverage of the whole domain into parts, as polygons that are interacting with the boundary of a Field are valid for the whole scale range, *i. e.* the height of the Fields at this level. It is enough to only retrieve once a slice of data and then data for every Field can be streamed independently from the other Fields.

Fields also allow to purge retrieved data from memory, which is useful when a user is navigating somewhere else (*e. g.* zooming in to complete different region). To appreciate the effects, it is necessary to test the tGAP-Fieldtree approach with a real large data set (having many faces and preferably large in geographic extent) and a small amount of memory available at client (which can be 'faked' by setting a constraint how much data may be cached at client-side). Experiments should be conducted in which not only zooming is supported, but also panning.

A different cache 'solution' to compare to the Fieldtree solution can be to implement a cache with only a limited amount of available memory, *i. e.* only cache X previous requests: when formulating a new request specify not only 1. a new request (a spatial range for which tGAP data needs to be retrieved from the SSC) but also 2. send together with this new request a fixed number X of previous requests (*i. e.* selections of tGAP data that have been retrieved and remembered by the client already). As the client has remembered the data that is retrieved for these X requests, this data does not need to be retrieved again (this can be done by putting the X previous requests as not to be retrieved into the where clause of the query which retrieves edges and faces, this way saving on data transfer). Tuning of the number of previous requests (X) can take place and question then is how large X should be so that a server will still be responsive (so that it does not lead to too complex queries, again back to free form vector objects with arbitrary shapes and geographic extents, which are complex to administrate).

To obtain tGAP data for a very large dataset (thus generalising a large dataset), it may be possible to use the same approach with the Fieldtree: Fields at the same level in the tree can be processed into a part of the whole tGAP structure, and only objects that are completely inside a Field its extent are allowed to be 'touched' by the generalisation process (this can also be beneficial to perform the generalisation work on the whole dataset in parallel, speeding up the process). Probably it is also possible to *partly* retrieve the associated tGAP data of a Field.

When the client remembers how much data in which direction (zoom-in or zoom-out) for a Field is already retrieved, the progressive data streaming of a Field can be paused and later continued. Other advantages that the Fieldtree might bring: Now we are no longer looking for the globally least important object, but this search is more local (within one Field only). This makes more local updates possible (and could lead to a dynamic structure that can be edited). The structure will most-likely have somewhat different data content, but the expectation is that this content will not be of noticeably lower quality.

5.5 Closing remarks

In this chapter an attempt was made to answer these research questions:

7. *How can we query the data structures to retrieve a 2D map from the structures?*

8. *How should progressive data streaming in a client-server setup look with respect to increments, communication and data structures (both at the client- and at the server-side)?*

As this chapter has shown, a filled tGAP data structure is able to supply vario-scale data for a specific scale range (§ 5.1). This range is dependent on the optimal number of objects that is set up to be shown on a map, the viewport size in pixels and the device characteristics (pixels per inch). To determine where to take a cross-section of tGAP data as basis for a 2D map, it was demonstrated how to carry out a mapping in which a scale denominator is translated to an importance value to query the tGAP structures. It was also shown that for making a map it is necessary to translate the edge-to-face pointers by using the hierarchy of faces (because intermediate edge records are not stored as a consequence of the design decisions in § 4.1).

An experiment provided empirical evidence that it is indeed possible to keep the number of faces to be retrieved under control (§ 5.2). This is the main accomplishment of the preceding generalisation process that generates the tGAP data which is stored within a factor of 2 of the original size of the dataset. For this process, the very focus has been on the 'weight' of the amount of data to be stored (mainly the number of edges and the number of vertices in the edge geometry). This is thus not only beneficial for lean data storage, but more important for the topic in this chapter, also has a severe impact on the amount of data to be transferred in a client-server set-up.

Further, this chapter has presented an experiment with regards to progressive data streaming (§ 5.3). The goal was to see whether the theory sketched in Haunert et al. (2009) was complete. The demonstrator that was built has shown that with some modifications this indeed is the case with explicit storage of 2 additional face pointers at the `imp_high` scale point of every edge (again this is needed as a result of leaving out the intermediate edge records in the lean tGAP implementation, § 4.1). Moreover, the following was learned:

- With the designed data structures, it is possible to transmit packages containing additional topological primitives (faces and edges) that allow updating and reconstruction of the polygonal geometry of a 2D map at the client-side at vario-scale, leading to a more, or less, detailed map at a different map scale.
- It is an absolute requirement to output valid and clean topological update packages: when processing incrementally updates for data in the scale dimension all earlier updates have to be correct. Validity of tGAP data becomes very important to realise progressive data streaming (*e. g.* all topological references have to be stored correctly, otherwise errors may occur).
- It is necessary to simplify the geometry of the edges (the polylines) – otherwise more than a factor 2 for storage size is needed.
- In case of a larger dataset, the streaming of all updates from top to bottom can lead to information overload; after some time too many details will be shown if the transmission of additional data is not stopped. However, this is not really a valid scenario – increments should be requested, filtered and streamed by using a geographic extent and scale range, *i. e.* linked to the zoom and pan action of a user (employing mapping from importance to map scale) and using Fields of the proposed, additional Fieldtree structure (§ 5.4).
- Moreover, the proposed cache friendly approach may both be helpful to process big data sets with the tGAP compiler, even in parallel (*i. e.* providing a divide-and-conquer approach to the generalisation process by using the Field tree as a way to partition the data set in manageable chunks) as well as making the structure dynamic and more suitable for 'local' updates.

6 A NEW ERA: SMOOTH VARIO-SCALE DATA

This chapter first brings together the insights in § 6.1 from Chapters 3, 4 and 5. By doing that, it re-examines the final design of the data structures that are minimally redundant, can store simplified edge geometry, support merge as well as split operations and are suitable for progressive transfer. Due to the proposed progressive demonstrator and the line simplification algorithm we realised that the data structures are not pushed to their limits in terms of continuous generalisation, such that morphing between representations at different map scales can take place as the key to smooth zooming. Therefore this chapter proposes in § 6.2 a new way of obtaining data for the exact same conceptual model proposed in Chapter 3 (the space-scale cube, SSC), but where horizontal planes are removed as much as possible from the SSC. This leads to continuously generalised vario-scale data and makes true smooth zooming possible, such that during use a delta change in the map scale represents a small change (delta) in the map. In addition, taking a different type of cross-section (by modifying the slice plane) leads to radial generalisation (*cf.* Figure 6.7), which is performed automatically and is useful for application in 3D virtual worlds. Then, § 6.3 discusses some drawbacks of the proposed solution and § 6.4 summarises the introduction of the smooth SSC.

Own publications

This chapter is based on the following own publications:

- van Oosterom, P. and Meijers, M. (2011a). Method and system for generating maps in an n-dimensional space. Dutch patent application 2006630, filed April 19, 2011, expected to be published October 2012.

- van Oosterom, P. and Meijers, M. (2011b). Towards a true vario-scale structure supporting smooth-zoom. In *Proceedings of 14th ICA/ISPRS Workshop on Generalisation and Multiple Representation*, pages 1–19, Paris.

6.1 Lessons learnt: a synthesis

Recall the list of requirements for a variable-scale environment (p. 52), that concluded Chapter 2:

1. Enables real time access to geo-information in the form of *vector* data;
2. Makes it possible to store, maintain and disseminate data at variable scale;
3. Is stored with minimal redundancy;
4. Allows progressive transfer and makes smooth zooming possible.

In the foregoing chapters we have scrutinised different aspects that are important for creating such an environment. Now, we will summarise the main results per chapter in relation to these requirements.

Chapter 3, by following an axiomatic approach, has formalised the validity of a vario-scale partition (in three dimensions) where we have combined 2D space plus 1D scale, leading to the space-scale cube (SSC) that can store vario-scale data. We have shown how to obtain input data for a generalisation process that can create the data for the SSC. The conceptual model establishes what a vario-scale partition entails. As an important starting point of a vario-scale environment is that data is stored with minimal redundancy, therefore we have opted to not store an additional, separate and specific model (DCM) for drawing a 2D map (as is sometimes the case in workflows for making paper maps, where the derived and improved geometric attributes are very important in the production process).

The proposed SSC concept does not enforce the way in which it is encoded in data structures and how the structure is populated with 'good quality' generalised content. Therefore, Chapter 4 has focused on 3 specialised generalisation operations, area merge, area split and line simplification, and their impact on the tGAP data structures, which store 2D geometry plus 1D importance attributes, encoding the SSC. The very focus of this chapter was: 1. data volume – limiting the number of composing elements for map objects to be stored, leading to a better balance between calculation and storage and 2. operations that focus on

topological consistency. It was proven that it is possible to prevent an excessive amount of edge records when a merge operation is applied. This was accomplished by storing some of the information implicitly (face pointers). Based on this result, the lean filling variant of the data structures was proposed, leading to a better trade off between storage and computation, *i. e.* information is more implicitly stored, meaning that attributes need to be derived on the fly (translation of face pointers for edges). This has shown to be a balancing act whether correct working functionality can indeed be provided or that additional elements need to be stored. This is not immediately evident; *e. g.* the clipping operation for making a 2D map has thrown a spanner in the works for the leanest investigated alternative (the `no_lr` or 'spaghetti-with-meatballs' alternative), where it was not always possible to associate the thematic attributes to all the map objects on the 2D map. The solution was to rewrite the edge-to-face pointers using the tGAP face tree.

It was also shown that once data is topologically valid, it is necessary to guarantee topological consistency under generalisation operations. For example, the developed line simplification algorithm (which simultaneously simplifies multiple input polylines) is aware not to introduce any topological errors. Initially, for storing the geometry of the boundaries the use of a forest of Binary Line Generalisation (BLG) trees was proposed (van Oosterom, 2005), with the advantage that the data structure would contain as little geometric redundancy as possible. However, the Douglas-Peucker algorithm does not give any guarantees on topological correctness and we noted that the use of the BLG trees would create communication overhead in a situation where a client application retrieves the data from a server, when the trees are only partially transmitted to obtain the right amount of vertices in the polylines. An alternative we investigated was not to simplify the boundaries at all, but to keep the original geometry of the boundaries we started with. We quickly noticed that during use of the resulting variable-scale planar partition in a network context, the number of vertices in the boundaries was too high, especially for the smaller map scales, leading to a slow performing user interface. Therefore, we turned back to simplify the boundaries, but now store the result of the simplification explicitly (thus *not* deriving the geometry dynamically from a special reactive data structure, as with the BLG trees, as this lead to administrative overhead), and allow some redundancy in the data structure (but preferably as little as possible). The line simplification has to be performed without violating any of the requirements for a valid variable-scale planar partition. The use of explicit topological data structures can be 'exploited' to define influence regions where topological errors

might occur, so that these can be prevented. The stored relations have provided handholds for technical implementation of the algorithms. Moreover, as paper map making often relies on a geometry-first way of modelling, together with a painter's algorithm, these topological relations have not always been on the priority list for designing generalisation algorithms. For the usage of maps the topological relations might however be more important than exact geometrical descriptions – as also argued by Mackaness (2006): humans tend to deal very well with abstract graphic conceptualisations of space (*e. g.* illustrated by the route maps generated by the LineDrive system, described in Agrawala and Stolte, 2001).

With the SplitArea algorithm it is possible to split polygons over their neighbours. The result of this operation clearly has an impact on the face tree structure, as a split introduces the need to model this structure as a Directed Acyclic Graph (DAG). This change is reflected by storing a separate face hierarchy table, so that the one-to-many relationship for parent faces can be stored. The resulting empirical evidence from the experiments, in which only a split operation was performed for creating vario-scale data, does not suggest that splitting leads to an excessive amount of records to be stored, although it leads to new geometries to be stored. However, no theoretical investigation was performed that proves that excessive edge records are prevented in all cases by this operation. From the minimal redundancy perspective however it would be good to have guarantees on a worst case bound of the number of resulting edge geometries.

Furthermore, the split operation is useful for differentiating in the generalisation process between types of feature classes, for example paving the way for representing linear features (polygons collapsed to line representations can be represented in the scale dimension of the SSC by vertical faces). In a sense, even a merge operation can be seen as a split with only one possible neighbour, while keeping the rest of the boundaries fixed, thus when a client can process the result of a split operation with weighted and zero weight edges (*e. g.* while incrementally processing additional data), a merge operation is automatically supported. Furthermore, the split operation can be a building block for other generalisation operations performed in a gradual way: *e. g.* typification, 'transforming a group of features without defacing the group's appearance' (Hangouët and Lamy, 1999), can be expressed as a series of splits and merges, whereby the individual members of the group will be changing gradually.

Chapter 5 has first focused on the relationship between scale and the concept of importance, which is associated with every face and also drives the order in which map objects are generalised. Mapping a map scale to a correct

importance value is key for obtaining a 2D map with the desired amount of map objects (following our rule of thumb to keep the average number of map objects the same – independent from the map scale for which the information is retrieved). This mapping has made clear that this rule of thumb *together with* physical device constraints (*i. e.* viewport size in pixels and pixels per inch of the underlying display) defines reasonable limits in the scale dimension, *i. e.* a valid vario-scale range, for which the data structures can indeed supply the desired amount of data. An experiment has delivered empirical evidence that a correct mapping can deliver the desired number of objects, while working with real world data sets.

Secondly, the architecture of a thin client was discussed, with which it is possible to obtain via stateless communication a 2D map for the correct spatial range and map scale from the structures. The use in the online scenario shows that taking a minimal redundant approach for data storage is a good basic principle. This might seem strange at first, as generalisation not necessarily deals with creating less, but certainly also different information for smaller map scales, especially when making large steps in the scale dimension, going from large scale to very small scale (Mackaness, 2006). However, this minimal redundancy is a critical aspect in times where bandwidth for mobile devices still can be considered a limited asset (for example, even while 3G mobile networks are present, using a mobile device abroad can be relatively expensive). Next to the fact that less data needs to be transferred, this frequently means faster performance (as data serialization is often an expensive step in network communication).

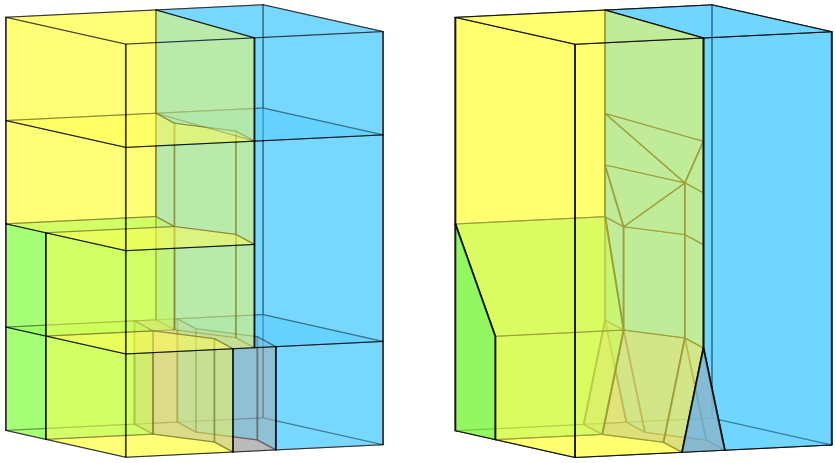
Thirdly, it was shown that progressive data streaming is driven by the data organisation that the tGAP data structures provide (both the ordering of data, as well as the number of data elements is important from a progressive transfer perspective). Both topics have been the focus of the generalisation process that was performed beforehand. Moreover, the structures define a hierarchical format that is very suitable for progressive data streaming, such that data can be retrieved in a progressive manner – coarse overview first, more details later. However, additional data structures may be needed for efficient caching and making it possible to quickly check which part of the data structures in a client server set up already have been retrieved. To summarise, the structuring of the information as a pre-process is the key to making progressive transfer possible.

All in all, the design of the structures as shown in Figure 5.9 (p. 159) is the final design of the tGAP structures to encode a SSC. This design allows for storing the merge, split and simplification results and is suitable for progressive transfer. However, we realised that the data that we store is still not optimal,

as the approach can be made more smooth. We did realise this due to the progressive streaming demonstrator and the line simplification approach that were implemented: objects that replace other objects still lead to popping effects, *i. e.* discrete changes (jumps), similar to the popping effects from the layers in a MRDB, but with a more local effect. How smooth vario-scale data should be obtained in a way that it alleviates also these remaining popping effects leading to smooth display capabilities (*cf.* van Kreveld, 2001; Nöllenburg et al., 2008; Danciger et al., 2009) is the topic of the next section.

6.2 Smooth data for the space-scale cube

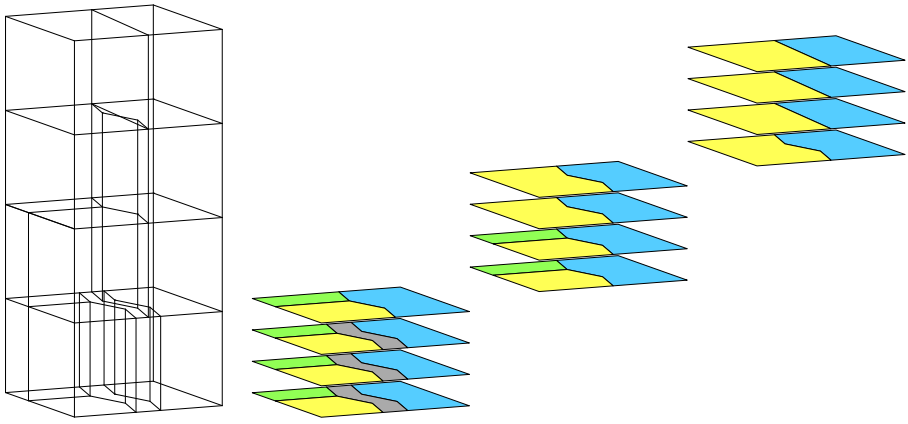
Recall Figure 3.5 (p. 70), which shows 4 maps fragments and the corresponding tGAP structure. The tGAP structure shown is a DAG, as the split operation (introduced in § 4.3) causes the road object to have several parents; see Figure 3.5e. In our current implementation the simplify operation on the relevant boundaries is combined with the remove or collapse/split operators and is not a separate step (although illustrated this way to visualise these operators separately). The scale has been depicted as third dimension – the integrated space-scale cube (ssc) representation, which has been formalised in § 3.2. Figure 6.1a repeats this 3D representation for the example scene of Figure 3.5e.



(a) ssc for the *classic* tGAP structure (b) ssc for the *smooth* tGAP structure

Figure 6.1: The space-scale cube (ssc) representation in 3D

Though many small steps (from most detailed to most coarse representation – in the classic tGAP $n - 1$ steps exist, if the base map contains n objects), this could still be considered as many discrete generalisation actions approaching vario-scale, but not true *smooth* vario-scale. Split and merge operations do cause a sudden local 'shock': a small scale change results in a not so small geometry change; *e. g.* leading to complete objects disappearing at once and a small delta in scale thus does not lead to a small delta in the map; see Figure 6.2. In the space-scale cube this is represented by a horizontal face; a sudden end or start of the corresponding object. Furthermore, polygon boundaries define faces that are all vertical in the cube, *i. e.* the geometry does not change at all within the corresponding scale range (resulting in the collection of fitting prism shapes, together forming a full partition of the space-scale cube).

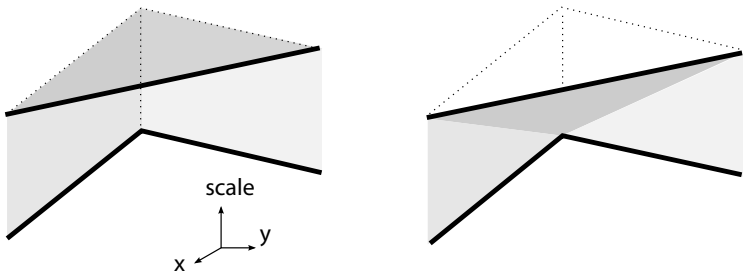


(a) Wireframe of (classic) space-scale cube (b) Slices for Step 1 (c) Slices for Step 2 (d) Slices for Step 3

Figure 6.2: The map slices of the classic tGAP structure: (b) step 1 (collapse), (c) step 2 (merge) and (d) step 3 (simplify). Note that nothing changes until a true tGAP event has happened (*i. e.* a generalisation operation has been applied).

In order to obtain more gradual changes when zooming, *i. e.* in a morphing style, we first realised that the line simplification operation could also output non-vertical faces for the space-scale cube and that this has a more true smooth vario-scale character; *e. g.* when replacing two neighbouring line segments by a single new line segment (omitting the shared node), this can be represented by three triangular faces in the space-scale cube; see Figure 6.3. Note that both the sudden-change line simplification and the gradual-change line simplification have both 3 faces in the SSC: sudden-change has 2 rectangles and 1 triangle and

gradual-change has 3 triangles. When slicing a map (to 'slice' means taking a cross-section of the cube) at a certain scale, a delta in scale leads to a delta in the derived map. That is, a small change in the geometry of the depicted map objects and no sudden change any more, as was the case with the horizontal faces parallel with the bottom of the cube, which were the results of the merge or split operations. Note that the more general line simplification (removing more than one vertex of a polyline) can be considered to consist of several smaller sub-steps: one step for the removal of each of the intermediate vertices.



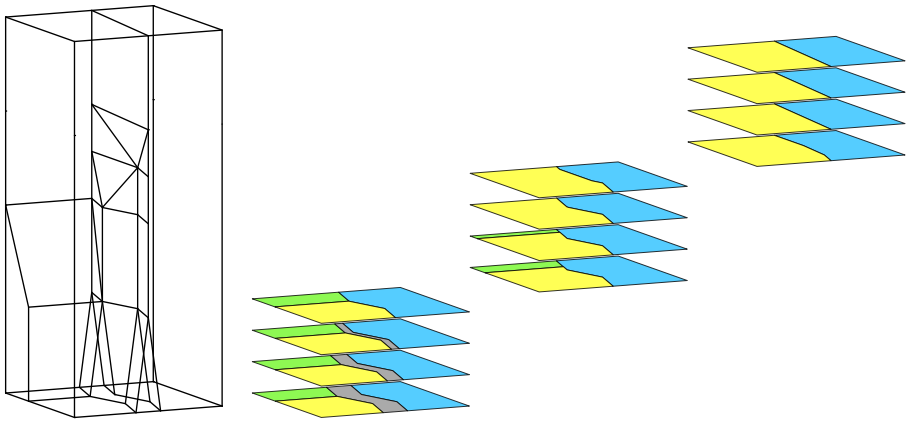
(a) Sudden-change line simplification: 2 rectangles and 1 triangle

(b) Gradual-change line simplification: 3 triangles

Figure 6.3: Line simplification in the SSC: (a) sudden removal of node, (b) gradual change. The dashed lines in (b) only illustrate the difference with the sudden-change variant.

SUPPORTING SMOOTH ZOOM. The split and merge operations can, similar to the gradual line simplification operation as sketched above, also be redefined as gradual actions supporting smooth zoom. For example in case of the merge of two objects: one object gradually grows and the other shrinks – in a space-scale cube this corresponds to non-vertical faces (and there is no more need for a horizontal face, *i. e.* a suddenly disappearing feature); see Figure 6.1b. All horizontal faces in the cube are now gone, except the bottom and top faces of the cube. Note that faces belonging to the same object are merged into one larger face if they can, *e. g.* the big front-right face in Figure 6.1b corresponds to four faces in Figure 6.1a. The same is true for the involved edges, several smaller edges on straight lines are merged, and the shared nodes are removed. This can be done because they carry no extra information. Perhaps the most important and elegant consequence is that the merging of the different polyhedral volumes belonging to the same real world object is that also the number of volumes is reduced: there is a one-to-one correspondence between a single object and its smooth

tGAP polyhedral representation, valid for all relevant map scales. The benefit of a smaller number of primitives, the nodes, edges, faces and volumes, is that there are also less topology references needed to represent the whole structure. In previous investigations it was reported that the storage requirements for topology structure may be as high, or even higher, than the storage requirements for plain geometry (see previous tests, described in Louwsma et al., 2003; Baars et al., 2004; Penninga, 2004). This is even more true for topology based vario-scale data structures (*cf.* § 4.1): lighter structures are more suitable for (progressive) data transfer and high(er) performance (§ 5.3).

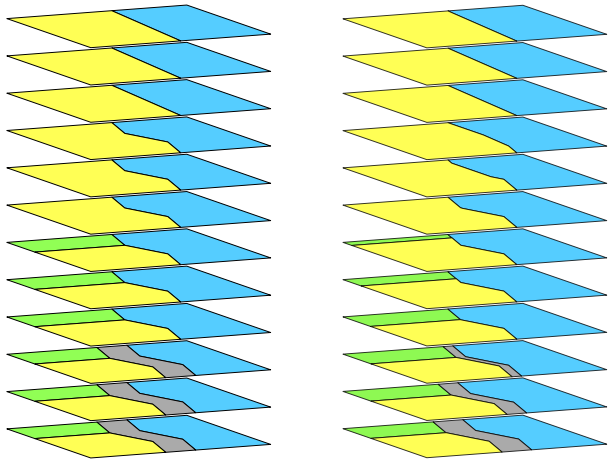


(a) Wireframe of (smooth) space-scale cube (b) Slices for Step 1 (c) Slices for Step 2 (d) Slices for Step 3

Figure 6.4: The map slices of the smooth tGAP structure: (b) step 1 (collapse), (c) step 2 (merge) and (d) step 3 (simplify). Note the continuous changes, also in between the 'true' tGAP events.

Figure 6.4 illustrates the resulting smooth vario-scale structure: small deltas in scale will give small deltas for map areas. Figure 6.5 shows that if all slices of the classic tGAP and the smooth tGAP space-scale cubes are compared, the differences and the benefits of the latter become clear.

So far, only horizontal slices parallel to the bottom and top of the cube were discussed and used for creating 2D maps. It is not strictly necessary to do parallel slices, nothing prevents taking *non-horizontal* slices. Figure 6.6 illustrates a mixed-scale map derived as a non-horizontal slice from the SSC. What does such a non-horizontal slice mean? More detail is shown at the side where the slice is closer to bottom of the cube and less detail at the side where slice is closer to the top. Note that the slice plane in this case is still flat (planar), but tilted.



(a) Slices for the classic tGAP structure

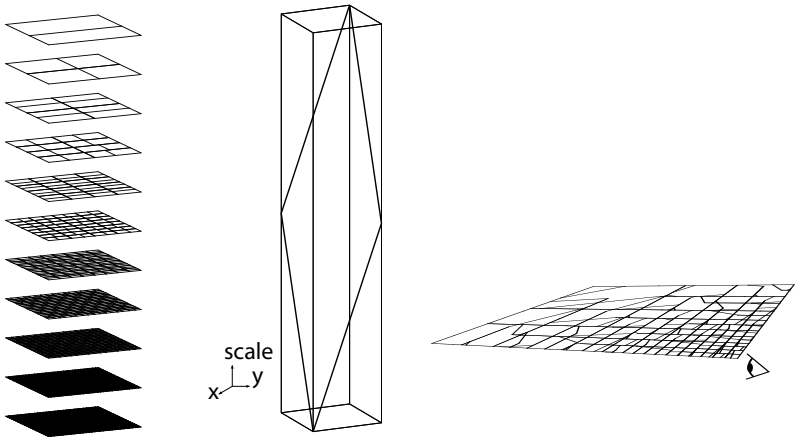
(b) Slices for the smooth tGAP structure

Figure 6.5: The maps – slices of (a) the classic and (b) the smooth tGAP structure – compared.

Compare to 3D visualizations, where close to the eye of the observer lots of detail is needed, while further away not so much detail. Such a slice leads to a *mixed-scale* map, as the map contains more generalised features far away (intended for display on small scale) and less generalised features close to observer (large scale). This type of map generalisation has been implemented by Harrie et al. (2002) (Figure 6.7 shows an example) and is termed radial generalisation by Reichenbacher: 'radial generalisation will radially simplify the map from the centre … towards the edges' (Reichenbacher, 2004, p. 51). Note that the slicing surface here will not be planar.

6.3 Exploring possible drawbacks

This section explores possible drawbacks of the presented smooth tGAP. Four potential issues are presented in the subsections below: 1. will slivers occur when slicing for a 2D map (§ 6.3.1), 2. can use of a sequence of non-horizontal delta-slices lead to less gradual changes than expected (§ 6.3.2), 3. can multiple generalisation operations be performed in parallel (§ 6.3.3) and 4. can square split and merge operations (horizontal faces) always be transformed into their smooth counterparts with non-horizontal faces (§ 6.3.4)?



(a) A set of smooth slices derived from the SSC.

(b) How the non-horizontal slice of (c) is taken.

(c) Corresponding *mixed-scale* map (non-horizontal slice): the imaginary eye of the viewer is at front right side, where most detail is, further away less detail is shown.

Figure 6.6: Checkerboard data as input: each rectangular feature is smoothly merged to a neighbour. Note that all merge operations have been executed in parallel, see § 6.3.3. Figure 6.6a: a stack of horizontal slices, 6.6b: taking a non-horizontal slice leads to a 'mixed-scale' map and 6.6c: one mixed scale slice (non-horizontal slice plane).

6.3.1 *Slivers*

The first possible drawback of the smooth tGAP structure might be that if at certain scale a slice is taken, then one could get a sliver: just before a object to be merged is disappearing. If this tGAP structure is used for static 2D maps (and not smooth zoom), then such a sliver should be removed; either by finishing the operation or going back to the start state of the operation. The correct state before or after the operation works like a magnet in this case: This corresponds to moving the slice slightly up or down in the cube. So, this is no real problem.

6.3.2 *Bad luck can happen…*

Imagine that we have a smooth tGAP structure (so non-horizontal faces), then horizontal slices and their movement, up or down the scale-dimension, will give delta map changes. Now for the same structure imagine that we want to have a

(a) (b) (c)

Figure 6.7: A 'mixed-scale' map. Note that Harrie et al. term this type of map a 'vario-scale' map, while we term this a 'mixed-scale' map. Furthermore it is clear that there is a need for extra generalisation closer to the borders of the map, which is not applied in 6.7b, but is applied in 6.7c. With our solution, this generalisation would be automatically applied by taking the corresponding slice (bell-shaped, curved suface) from the SSC. Illustrations 6.7a and 6.7b taken from Harrie et al. (2002) and 6.7c from Hampe et al. (2004).

mixed-scale map using a non-horizontal slice plane and we also want smooth mixed-scale zoom (whatever this may be, doubtful if useful). If the slice plane has the same angle as one of the object faces, then a delta slice plane movement could result in a sudden big map change: a complete object disappearing at once and other objects reappearing. As this is an exotic use case and unlikely that exact same angles will ever occur (probability near 0% if there is no systematic preference for angles of object faces and/or slice plane), this drawback is not really considered a problem.

6.3.3 *Parallel execution, instead of 1 by 1*

Despite the fact that the proposed solution results in a true vario-scale structure, it has still an (old) tGAP drawback and that is the 1 by 1 sequencing of all generalisation operations. This has the positive effect that it can be proven that all operations are validly represented in the SSC (giving a good topological structure): both a start and end scale of an operation, but also in between. But all gradual changes are local when looking at the big picture: first one operation is (gradually) finished and then the next local operation is started, and so on. This might give a suboptimal smooth-zoom effect – more experiments with end users are needed to verify whether this is indeed suboptimal.

A solution for the 1 by 1 sequencing is not implementing the steps in the structure in a sequential manner, but to group them (to group size N_g) and

then let all members in the group transform in parallel. Danger is now that neighbouring actions (each creating a valid part of the structure when executed alone), may together result in an invalid representation, that is, intersecting planes. This can be efficiently detected: prepare resulting faces of this gradual group step and put a 3D R-tree on the new faces. For every new face check for conflicts, that is, intersection with faces already present in the group. Through the use of the R-tree this takes $O(\log n)$ time with n the number of faces in the R-tree group). In case of any conflict, then undo the action that belongs to the last (smaller scale) action of the sequence. In total this takes this takes $O(n \log n)$ time, the time needed for the creation of the R-tree for the faces in the group. If no intersection is found, then the total result is correct, in next group of N_g actions, the undone actions gets a new change and then have highest priority, so will not be undone this time.

An alternative approach, not based on spatial searching (3D R-tree) is to exploit the topology structure when creating the parallel groups. The normal tGAP creation approach is followed: select the least important object, process this object, then select the next least important object, process this object and so on until enough objects in the group are found. However, when an object is selected, then it will be temporarily marked and also the neighbours will be marked. If a to be removed object or one of its neighbours is already in the set of marked objects, then skip this object and continue with next least important object (and in next grouping the skipped object will be first in line). By finding non-neighbouring objects, the local smooth zoom actions (faces) will not interfere.

Below a list of related research questions:

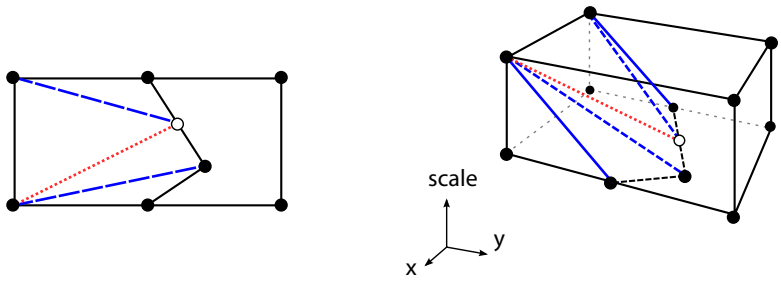
- With one of the approaches, can deadlocks occur in undoing intersections of the N_g actions?

- What is a good group size N_g? Too small approaches the normal tGAP, while too big N_g might result in a number of discrete stages (end of many non-vertical walls) at the same time. Also a too big N_g increases the change on conflicts between neighbour actions. Having a larger group size N_g might also give a strange artificial effect during the smooth zoom: if many disappearing features are visible in the image, then some kind of artificial climax moment is introduced, when they all disappear together. Tests should be conducted whether this is noticeable (because only very few actions will be visible at same time in one window; *e. g.* 1 or 2; and others are outside display window).

- Should the size of N_g vary due to: 1. stage in process (more to the top, smaller N_g; *e. g.* always be a percentage of the total number of objects at a given scale stage) or 2. relative to accumulated area change by actions in group?

6.3.4 *Smooth zoom with real data*

The example data set (Figure 3.5) only shows very simplistic shapes. It is easy to imagine that when there are two neighbouring equal rectangles, how one rectangle gradually has to take the space of the other rectangle and that the resulting non-horizontal faces in the smooth tGAP structure will be flat. The question that arises: Is this always possible for any pair of strangely shaped neighbours or configurations with island polygons included? Answer: Yes. Proof: it is possible for strictly convex parts[1] to be 'removed', using the following algorithm (see Figure 6.8):



(a) Top-view (b) Space-scale view

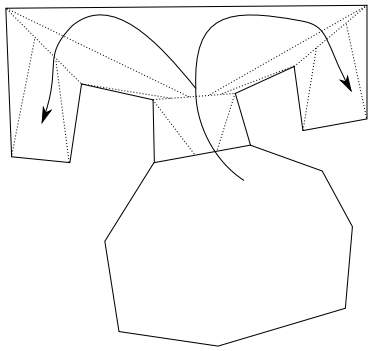
Figure 6.8: The simple neighbour merge: one more or less rectangular feature is smoothly merged into neighbour feature. Note that the plane that forms the boundary between the two features is composed of 2 triangular and 1 quadrilateral faces (these faces can be dissolved by post-processing into 1 face, as they are planar).

- Count the number of interior nodes on the boundary to be removed and on the boundary to be moved to (minimum number is 1, otherwise neighbour to be removed would have no area).

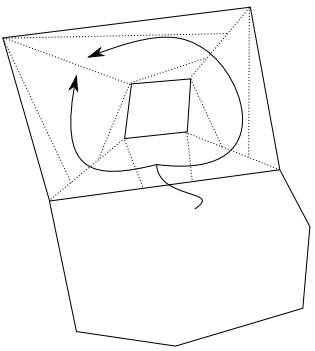
[1] A simple polygon is strictly convex if every internal angle is strictly less than 180 degrees (so not equal to 180 degrees).

- If unequal, add the missing number of (fake) nodes fairly distributed along the boundary with the smaller number. The number of intermediate nodes is called I and is equal in both boundaries.
- Now both boundaries have an equal number of nodes and add edges between pair of corresponding intermediate nodes (note – at least one edge is added).
- This results in two faces with three nodes and $I - 1$ faces with four nodes (and also four) edges. If a face is not flat add an additional diagonal edge and the resulting two triangles are per definition flat.

Note that this 'simple' algorithm may add some unneeded (temporary) nodes. Imagine two equal shaped neighbour rectangles, then a single diagonal face is sufficient. However, our algorithm would add two intermediate nodes (on the shared boundary) and create two triangles and one 4-node face. In a planarity check it may be detected that these faces are co-planar and can be merged (and same for split edges and added node may be removed). So, the final result is equal after this post-processing.



(a) The processing of a m-shape neighbour, with growing area attached to middle leg of 'M'



(b) The example of neighbour with island: decompose in strictly convex parts.

Figure 6.9: The processing of complex shapes into vario-scale representations. Note that quadrilaterals will not be planar and will have to be decomposed into triangular faces by adding an extra diagional.

Because of the convex shape, there will never be intersecting edges or faces. If the to be merged shape is concave, then decompose it in convex parts and

treat the convex parts one by one. The order in which this should be done is to start with a direct neighbour part of the growing area (and repeat until all parts are processed); see Figure 6.9a. Note that this algorithm also works when the to be merged neighbour has an island: creating the strictly convex parts and processing these with the algorithm above will give correct results in the SSC; see Figure 6.9b. Furthermore, this approach will also work as post-processing of a split operation: the split operation delivers boundaries to move to. Each part of a split polygon can, following the sketched recipe, result in a gradual merge with its neighbour.

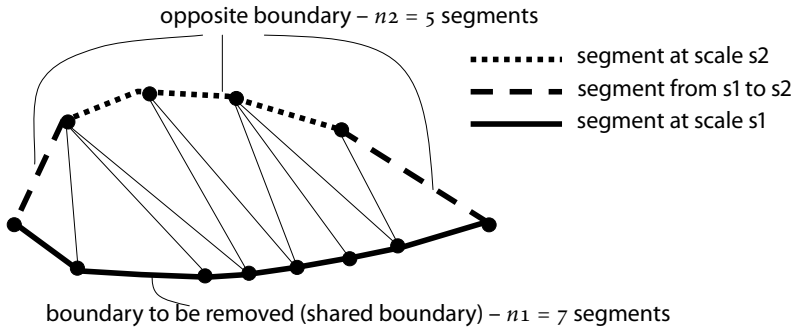


Figure 6.10: Alternative way of constructing non-horizontal faces. In this example 7 triangles have their base in the shared boundary and $5 - 2 = 3$ triangles have their base in the opposite boundary.

A second approach, which does not require the 1-to-1 connection between intermediate nodes (but still requires convex parts), to construct the faces between the boundary to be removed (shared boundary) and the boundary to be moved to (opposite boundary) is as follows (illustrated in Figure 6.10): Count the number of segments in the shared ($n1$) and the opposite boundary ($n2$). Now there will be $n1$ triangles constructed, which will have their base in the shared boundary (and the remaining vertex on the opposite boundary) and $n2 - 2$ triangles which will have their base in the opposite boundary (and the remaining vertex on the shared boundary). This approach does not require post-processing.

6.4 Closing remarks

First, this chapter has brought in § 6.1 a synthesis of results obtained. This synthesis has shown that the vario-scale concept needs some specialised generalisation operations, but that these are feasible to implement in practice. One difficulty

is that not all generalisation operations devised for paper maps can directly be used, as vario-scale has a different perspective, *viz.* topological consistency is very important. It is possible to progressively stream the vario-scale data and for this functionality the ordering imposed by the generalisation process, but certainly also the minimal redundancy, is key. However, we realised that the continuous generalisation aspects are not optimal yet, *i. e.* a delta in scale can lead to a 'jump' in the map, so the concept of vario-scale data can be improved in this respect.

Therefore, this chapter introduced the *smooth* SSC for geographic information in § 6.2: a delta in scale leads to a delta in the map (and smaller scale deltas lead to smaller map deltas until and including the infinitesimal small delta) for all scales. The smoothness is accomplished by removing all horizontal faces of the classic tGAP structure. Recipes were given how to obtain data for the smooth tGAP structure: 1. performing generalisation operations in such a way that the output given gradually changes the boundaries between the features being generalised and 2. grouping generalisation operations for parallel execution while still guarding topological consistency. The resulting smooth tGAP structure then delivers true vario-scale data and can be used for smooth zoom. It is one integrated space-scale partition, and when using a non-horizontal slice plane for taking a cross-section the resulting 2D maps will be a valid, mixed-scale planar partition: this is useful for use in 3D computer graphics; radial generalisation is thereby automatically performed – just choosing the correct surface for slicing is enough.

7 CONCLUSIONS AND FUTURE WORK

This chapter highlights the main achievements of the thesis as well as the insights gained into variable-scale geo-information and provides an answer to the main research question (§ 7.1). It also gives a number of suggestions for future research (§ 7.2).

7.1 Conclusions

The use of digital maps is changing by the advent of new mobile devices (such as tablets), that harness a lot of computing power, *cf.* § 1.1. It is therefore crucial to investigate the new technological possibilities that variable-scale data structures can bring, to make better digital 2D map solutions possible. By taking a step back from the regular map generalisation process, which reduces the information on a map for a smaller map scale for a whole map in one go and using a step wise generalisation process at its heart, as proposed by van Oosterom (1993, 2005), this work has clearly contributed to the technological foundations for data structures for producing 2D maps at variable-scale (*e. g.* for use on the Internet).

7.1.1 *Answer to the research question*

The objective we had with this research is expressed in the main question, which was formulated as:

How can we realise improved vario-scale geo-information having minimal redundancy?

To answer this main question the following 8 subquestions were defined:

1. What is the state-of-the-art in: 1. multi-scale data management and 2. generalisation of vector data?
2. How can we formally describe what is variable-scale geo-information?
3. How can we create valid 2D input data as much automated as possible?
4. How does minimal geometric redundancy influence the design of the data structures?
5. How can we simultaneously simplify edges so that the result is topologically consistent?
6. How can we split linear features over their neighbours, instead of merging to one of their neighbours?
7. How can we query the data structures to retrieve a 2D map from the structures?
8. How should progressive data streaming in a client-server setup look with respect to increments, communication and data structures (both at the client- and at the server-side)?

Within the concluding sections of the foregoing chapters answers were given to the subquestions, leading to the synthesis performed in § 6.1, after which a recipe was given for obtaining smooth vario-scale data (§ 6.2). In summary, we have identified in our design the following key factors extending the current state-of-the-art (subquestion 1) for realising vario-scale geo-information:

1. With the concept of the proposed space-scale cube (SSC) we have formalised what vario-scale vector data entails (subquestion 2). In a sense, the improved design of the tGAP data structures can be seen as a lossless encoding of the data that is captured for a SSC.
2. To make vario-scale geo-information operational, we need specific generalisation operations. These vario-scale generalisation operations need valid input (subquestion 3) and should be designed carefully to be able to give guarantees on the amount of data to be stored and output topologically consistent vario-scale data (subquestions 4, 5, 6). Empirical evidence was

provided that the tGAP structures are able to store the result of area merge, area split and line simplification operations, while keeping the amount of data to be stored under control. Moreover, a lot of the other cartographic operations (such as displacement and typification) can be expressed in a continuous way by first splitting the geometric description of an object and secondly merging the parts to neighbours, this way it is then possible to gradually change from one situation to the next in a continuous form.

Throughout this thesis, specific measures have surfaced as an important concept to steer the generalisation process, mostly with respect to the weight of the geometric description, but for example also to evaluate the result of the split operation. Where normally the generalisation process is seen as a subjective matter, these measures give control over the generalisation process from an algorithmic point of view (giving an optimisation goal). In a sense, the cartometric analysis that we have performed every time in this thesis – keeping the information density on average constant after every generalisation step – works well; the number of objects and vertices to be retrieved can be guaranteed during use. These measures also exemplify that cartography is still changing (Kraak, 1998) and that such measures should be evaluated for given solutions, as they are giving guarantees on the amount of resulting data and thus in a sense also on how fast a user can interact with a system. All in all, when these measures are correctly evaluated in the generalisation stage, end users will benefit later.

3. The design of the tGAP data structures after this research is such that 2D maps can be efficiently derived from the structures and progressive transfer is possible (subquestions 7 and 8). The resulting ordering of output data, which is imposed by the *step-wise* generalisation process, is key to this type of transfer. Thus, it is important that the generalisation process is delivering output that is broken up in small steps, which can be replayed so that a 2D vector map can be gradually refined, first providing a coarse overview, to a more detailed one.

   Also in this respect having lean structures is beneficial, and therefore the focus on minimal redundancy during design is a satisfactory one. However, minimal redundancy is not automatically obtained, this certainly means performing trade offs and also testing with real world datasets, which can reveal cases not thought of earlier.

4. Although we have criticised the MRDB approach for not being able to supply data at vario-scale, the progressive demonstrator and the proposed line simplification approach made us realise that we still store discrete steps – albeit smaller and more local. Therefore we proposed how smoothness of the vario-scale data can be improved. This fits within the very same formal model of the SSC, but leading to a *smooth* SSC. Next to the fact that this SSC can supply true and efficient smooth zoom, also mixed-scale maps can be derived and gradual changes in the level of detail for data, based on the observer's viewpoint, can automatically obtained. This means that the only prerequisite for obtaining radially generalised data during use is a vario-scale generalisation process that provides smooth data.

7.1.2 *Contributions*

Over the course of this research, we have made the following contributions to the design of a vario-scale geo-information environment. In the longer run these contributions can help to realise a paradigm shift from managing data at (multiple) fixed map scales towards true vario-scale geo-information with minimal redundancy. We have:

- formalised the concept of variable-scale data as a conceptual, 3D geometric model (the space-scale cube, SSC), where 2D space and 1D scale is integrated (§ 3.2);
- shown how to obtain valid 2D input data for the generalisation process that creates data for the tGAP structures (§ 3.3);
- shown how minimal data redundancy can be obtained when applying a merge operation (§ 4.1);
- shown how to perform a parallel simplification of lines, without introducing unwanted intersections and other topological errors (§ 4.2);
- proposed a split operation and have shown what are the impacts on the designed data structures (§ 4.3);
- shown how to solve the mapping of map scale to an importance value (§ 5.1);

- described two possible approaches for creating both a thin and a thick client, in which the thick client can benefit from the way the data is correctly ordered, so that progressive data transfer can take place (§ 5.2 and 5.3);
- proposed a method in § 5.4 to deal with retrieval of very large data sets by means of partitioning the input – *i. e.* a divide-and-conquer approach (without introducing explicitly stored or visible artificial boundaries in the dataset);
- proposed an improved way of generating data so that even smoother graphic transitions can be derived for visualisation (§ 6.2).

7.2 Recommendations for further research

Although this work has made a contribution to further the development of variable-scale geo-information (now beyond the doubt that it is useful and feasible), still many avenues for future research remain. The following topics seem interesting to explore in future research and can form a (partial) road map for the 2 research projects – the STW project 'Vario-scale geo-information' and the NWO VIDI project 'Modelling geographic information in 5D' – that are being carried out at the GIS technology group of Delft University of Technology (some of these were already mentioned as being out of scope for this thesis, *cf.* § 1.3):

OPTIMAL NUMBER. We have shown, that we can keep the average amount of data per viewport to be constant, but: what is an optimal number that users prefer? How is this influenced by the physical screen size (*i. e.* what is the influence of the type of the device, flat screen monitor or cell phone screen) and the map scale (should this indeed be a constant number)? What influence has data delivery speed (*e. g.* determined through network speed)) on this choice? Is a map derived with this amount of data still clearly readable and does this depend on the type of application? In this respect user testing is needed.

THE IMPACT OF THE SPLIT OPERATION. No empirical evidence was found that the split operation, as introduced in § 4.3, generates a very large amount of edges. However, as the operation introduces new geometry, it is less minimal redundant than only applying a merge operation. Therefore the split operation deserves to be investigated into its worst case behaviour, similar to the analysis of § 4.1.2.

Another aspect of the split operation is that after the collapse of an area object to a line (or point) object, the same object can live on in the scale dimension. In the SSC this object is then represented by a polyhedral volume to which a vertical surface is connected at the top (in case of collapse to a point object then in the SSC the polyhedral object is extended with a vertical line). All attributes are attached to the same object, which is represented in the SSC by connected multiple parts of respectively dimension 3, 2 in case of collapse to a line and 1 in case of collapse to a point. However, storage of objects not equal to the highest dimension of the SSC, *i. e.* line/edge and point/node objects in 2D, requires additions (a place to store these objects) to the data structures that should be researched.

TUNING THE LINE SIMPLIFICATION APPROACH. The line simplification approach can be tuned further by specifying differently how many vertices should be kept, *cf.* end of § 4.2.7.

MORE ADVANCED GENERALISATION PROCESS. To create a more advanced generalisation process it should be considered with the current approach to: 1. tweak the compatibility matrix, 2. investigate how to determine when to apply the split and when the merge operation (*i. e.* take more thematic attributes of the polygons into account, for example only split linear infrastructure objects, while using the merge operation for other objects) and 3. represent linear features explicitly in the structure (linking these features to a set of edges – then linear networks, as often found in topographic data, can play a role in generalising the area partition as input for the tGAP structure).

A possible alternative where map specifications are expressed as geometric and topological constraints could also be used to generate vario-scale data, *i. e.* using constraint-based techniques, such as also used in the AGENT project (Lamy et al., 1999). These techniques require less explicit specification and make intelligent solutions possible. Problems with these techniques can be that a complete and formal description of all constraints is lacking, that the combination of constraints is sometimes too restrictive and that the importance of certain constraints differs per specific geographic location (Stoter et al., 2009a) or that obtaining a solution is computationally expensive (Neun et al., 2009). Alternatively, it may be an idea to generate vario-scale data with the current approach, retrieve a dataset at arbitrary map scale and then use constraints together with a solver for placement of cartographic symbols, texts and so on (following a hybrid

approach to dynamically style the resulting map, but for which the content is already reduced).

GROUPING OF GENERALISATION OPERATIONS. Is it possible to replace the 1-by-1 sequence of generalisation operations to obtain a more optimal smooth-zoom effect, as described in § 6.3.3: How to form groups of objects to perform a generalisation operation on? What would be a good group size? Should the group size change during the generalisation process?

BEST ROUTE FOR CREATING VARIO-SCALE DATA. What is the best route to create vario-scale data content (*cf.* Figure 7.1)? Two options come to mind: 1. integrate all themes at the base scale, then build a vario-scale data set, or 2. build a vario-scale data set per theme, and integrate the themes when used. Do these 2 routes lead to the same result in the end?

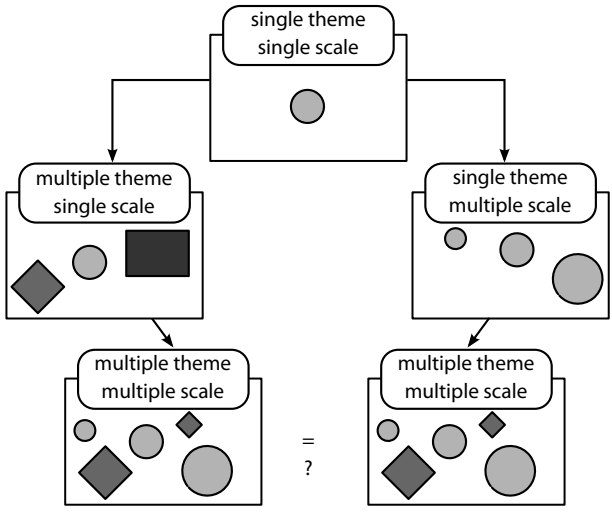


Figure 7.1: Two routes for creating vario-scale data: 1. First perform an integration step of data, then generalise all data for the integrated themes all at the same time (left), or 2. First generalise data separately for different themes (*e. g.* only roads or only buildings), then integrate generalised data sets (right).

NECESSITY OF ALL GENERALISATION OPERATIONS AND THE INFLUENCE OF VISUALISATION. As demonstrated in § 2.2.2, there is a divide of operators

in model and cartographic generalisation. From these operators (such as merging, splitting/collapsing, typification, displacement, amalgamation, etcetera) it would be useful to know which are crucial for map making with tGAP data on a digital user interface. Cartographic quality of paper maps has focused on geometric resolution, *e. g.* the displacement operation is very important for paper maps, but problems herewith can to some extent – as a last resort when data is reasonably generalised – be alleviated in a digital environment by zooming in; This also depends on the styling rules that are applied to the data, *e. g.* symbolising roads with very thick lines might lead to problems again. Question are: to which extent can a default visualisation be applied without problems to the tGAP data (*i. e.* when will a certain set of styling rules again cause problems for display) and how can you be sure to alleviate all these problems when creating tGAP data? Hypothesis: the diverse generalisation operations have different importance for digital map making than for making paper maps and the result of other generalisation operations, such as displacement will also fit in the tGAP structures.

LARGE DATA SETS. Dealing with very large data sets that do not fit in main memory – not during the generalisation process and, retrieved as vario-scale data, not during visualisation – deserves attention. First, these large data sets need to be generalised. When such a data set does not fit in main memory, it needs to be split up in smaller, more managable chuncks of data. Question here is whether the Fieldtree approach (§ 5.4) is useful and whether the use of such a chunking approach brings noticeable – *i. e.* visible for an end user – side-effects when using the resulting vario-scale data set. Secondly, large data sets result in large cubes and for visualising a complete slice near the bottom will contain a lot of data (takes time) and is not what a user wants. So slicing should be combined with other (spatial) selection criteria; *e. g.* the bounding box (bbox). The bbox is most likely smaller at the bottom and larger near the top for 'sane' applications. For non-horizontal slices the lower edges of the bbox should be shorter than the higher edges of the bbox. This can be compared to the use of frustums in 3D computer graphics for perspective views. Techniques to page in some parts of the dataset into memory, while removing others become then essential. Test datasets that could be worked with are: CORINE, with land cover data for the whole of Europe (not only clips) and large scale datasets for a specific country, *e. g.* IMGEO or TOP10NL data sets in The Netherlands.

DYNAMIC STRUCTURES. Make the structures dynamic: currently the tGAP structure (including the new smooth tGAP) is a static structure. When an update of the most detailed data takes place, the structure has to be recomputed. Due to global optimisation criteria (globally find the least important object at every step of creation), the impact of a local change is not guaranteed to have a local effect. The grouping approach of § 6.3.3 might be helpful for making more local updates possible. Also techniques for viewing large data sets efficiently might be helpful for controlling the locality.

IMPROVED GENERALISATION OPERATIONS. Improved generalisation operations that contribute to 3 elements: a. smooth transitions, b. provide big steps in the scale dimension (such that the definition of phenomena – or put differently, the set of classes in the legend – changes), c. also work for linear features (*e. g.* start with topographic data as an integrated theme, where everything is represented by a polygonal area, but gradually move towards line based representations for feature classes such as rivers and roads – this type of features are often represented on topographic, smaller scale maps – challenge is to keep the connectivity of these networks in tact, while generalising): thus integration of diverse themes (houses, land use, but also linear networks, such as river and road networks) and how to generalise these at the same time for a vario-scale partition is another relevant question here.

IMPLEMENTING THE SMOOTH SSC. The smooth tGAP has the same building challenge as the classic tGAP with respect to applying the right sequence of generalisation operators (remove or merge, collapse or split, simplify) to obtain cartographic quality. This has to be well tuned, otherwise the maps will be of (too) low cartographic quality despite the fact that they are perfect in topological sense and 100% consistent between scales. One option for this might be the constrained tGAP (*cf.* Haunert et al., 2009; Dilo et al., 2009). It is also clear that this requires 'understanding' (semantics) of the different types of object classes involved (and the map needs of the end-users). Questions to be answered: What is a good series of small scale steps, will it be the same as a big scale step performed at once (*e. g.* comparing with current available data sets)? Do these steps need to be performed one by one, or can we bundle set of operations to be performed at once (proposed grouping approach)? Moreover, this type of modifications may also have an impact on the mapping of map scale to importance, so this mapping should be taken into account here as well.

INTERACTING WITH TRUE SMOOTH DATA. A user interface should be developed where data retrieval is linked to user actions and memory management (*e. g.* purge data from memory that is not needed any more) is automatically performed. Progressive streaming capabilities (either directly to database, or over network) should be used and interaction should be giving a very smooth feeling (where consistency of stored data is emphasised). How should these techniques for interacting with the data look like? Having a big data set at hand is a prerequisite to test the scalability of such a system. Another interesting question in this respect is whether the simple operations (SOs) as proposed by Sester and Brenner (2009) can be used as a wire format for serialization of vario-scale data into progressive data streaming packages for network transmission.

CACHING AT CLIENT-SIDE. Related to interaction: Explore caching techniques for retrieval of vario-scale data, *e. g.* using a Field tree based approach vs. keeping the results of a number of previous requests in cache memory of the client and communicate that these do not need to be transferred again (as described in § 5.4).

OPPORTUNITIES FROM NEW WEB TECHNOLOGIES. New web technologies are being implemented in the new generation of mainstream Internet browser engines – WebGL, WebSocket, Web Workers and the HTML5 Canvas element are potentially very useful technologies for implementation of variable-scale structures in a mainstream IT environment, client-side (a possible benefit can be that a large user group, *e. g.* without the need to install additional plugins, can use vario-scale data). It is worth to test this, also to see whether these emerging mainstream IT standards are rich enough to implement this type of geo-applications (or that specific geo-extensions to these standards are necessary).

SMOOTH INTERACTIONS. It is of importance to know how users perceive the smooth interactions: is it indeed beneficial in the way tGAP can provide them (extending the work of Midtbø and Nordvik, 2007) and how fast should these smooth actions be displayed? This could be tested as follows: Suppose we have focused on a large city in Europe, say Amsterdam, and we want to pan to another European city, *e. g.* Rome. If the system performs combined zooming and panning between the points, a pre-defined path can be followed, leading to a zoom out first, then a pan across all intermediate countries and a zoom in on the destination city. Van Wijk and Nuij (2003; 2004) give a mathematical recipe for

how such a combined zooming and panning action should look and perform a user evaluation in which they let users vary properties of how to set up the path followed by the system. The integrated zooming-panning action is automatically performed by the system between the two fixed points. The result should be a smooth animation, where the frame rate is high enough to show interactively the map between the two points at various levels of details at different locations (*i. e.* variable-scale data can be used). Such an implementation can then be used to: 1. technically evaluate the system (can a client-server architecture with variable-scale data keep up with data requests, while visualising at a specific frame rate, *e. g.* while increasing the number of clients that request data) and 2. perform a user study in which the vario-scale approach (perhaps combined with computer graphics 'tricks', such as fade-in, out or ease effects) is compared against the current state-of-the-art MRDB, *e. g.* a tile and raster based solution, where an equivalent path can also be created: which of the two approaches is found to be better from the end user perspective? Note that for such an experiment to work, cartographic quality of the generalisation should be under control first, *i. e.* at least comparable to graphic quality of the MRDB solution.

SAME SOURCE DATA, DIFFERENT APPLICATIONS. Build different tGAP data structures based on same source data, but making the resulting data sets suitable for different user groups and applications (by taking different decision in the generalisation process when creating the vario-scale data; this can be compared to having multiple indices on the same database table). A useful tool in this respect could be a viewer that has synchronised views, so that a person that creates the tGAP structures can compare and inspect visually the results of the different generalisation processes.

ANALYSIS PERFORMED AT VARIO-SCALE. Focus of this research has been on viewing data. Analysis with vario-scale data is another interesting aspect. For example, vario-scale data could be of help to data integration. For this it is necessary to test overlay processing with two (or more) independent space-scale cubes – this 3D overlay resembles data integration: it is possible to geometrically overlay the two space-scale cubes and carry over the attribute information to the newly segmented space-scale partition. Note that before the actual overlay, the scale-dimension has to be first well aligned: only intersect the corresponding representations. However, for data integration this will not be enough, *e. g.* one of the difficulties will be to harmonise semantically the attribute values. Using

space-scale cubes might give more clues for a data integration process than integrating just two separate 2D map sheets (*e. g.* which do not have the same reference scale) and can be helpful for performing both horizontal as well as vertical conflation at the same time. An example application could be creating a smooth tGAP based on a soil map 1:50K and a land cover map 1:100K. Intersect the two SSC and use the result to answer the request to find the areas that are forest on sandy soils at scale 1:250K. Another aspect is that progressive streaming can also be used for progressively finding an answer (first coarse approximation, then later more detailed answer). Specific types of querying (*e. g.* route planning using different levels of detail as described by van Bemmelen et al., 1993) might benefit from this.

INVESTIGATE MIXED-SCALE SLICES. Cross-sections for mixed-scale data retrieval are non-planar; *e. g.* supporting radial generalisation and fish-eye type of visualizations (see Figure 6.7, p. 184). What are in this case useful slicing surface shapes? Folding back surfaces seem to be non-sense as this will give two representations of the same object on the same location in one visualisation.

HIGHER DIMENSIONALITY OF SMOOTH, VARIO-SCALE DATA. If instead of a 2D base map we start with a 3D base map (model) and then create in a similar manner a 4D space-scale hypercube, then this might be used for good perspective view visualizations by taking non-horizontal scale slices: near a lot of detail (low in scale) far not so much detail (high in scale). The intersection of this 4D hypercube with the hyperplane gives a perfect 3D topology: all representations do fit without gaps or overlaps. This solves a big problem as often the case in the transition from one Level of Detail (LOD) to the next LOD in computer graphics. Interesting 'implementation' issues will arise: How can the slicing in the 4D hypercube be done efficiently? Is this efficient enough for interactive performance (100 times per second)? The slice is a 3D model and still has to be rendered on a 2D display (or 3D stereo device). How often to reslice (every frame or re-use a taken hyperslice for multiple frames)? Would it be possible to combine the above two steps in a single operation on the 4D hypercube (selection and transformation for display). What steps can be done in hardware and what needs to be done in software? Making the structure dynamic also might result in a 5D hypercube (van Oosterom and Stoter, 2010). Again slicing issues arise when we want to create visualizations: slice from 5D to 4D with hyperplane (*e. g.* select a specific moment in time or alternatively select a specific scale).

FULL 3D (OR EVEN 4 OR 5D) VERSUS 2D PLUS 1D. As the conceptual model does not mandate 2D vario-scale data to be strictly stored as 3D information and we started our research investigations from the classic tGAP structures, we serialised the SSC as 2D topological primitives (faces and edges) together with 1D scale attributes (importance). This is useful for storing a vario-scale structure in a current state-of-the-art DBMS (PostGIS in our case), which does not offer full 3D capabilities: the choice for this 'split implementation' is thus both practical – many implementations allow storage of 2D geometry – as well as sufficient, *i. e.* no explicit need for real 3D storage for the applications we tested: viewing a 2D map slice, either single slices, or a progressive set up of retrieving 2D data is possible. However, this might change for other types of applications (*e. g.* 3D computer graphics with 2D vario-scale data in a 3D environment). Here it may be more efficient when selections can be made on all dimensions integrated, with specific indexing methods capable of this task. Question remains how to encode the space-scale (hyper)cube in an efficient manner? What is more efficient: an implementation with a split implementation, but then with specific data structures for encoding smoothness of the boundaries (*e. g.* a slightly modified BLG tree structure in which the importance range for vertices is also stored) or an implementation based on full 3D primitives, *e. g.* using 3D nodes, edges, faces and volumes (a polyhedral 3D topology structure), or using for example a Tetrahedronized Irregular Network (TEN) or regular polytopes? Note that for progressive streaming being able to get a sensible ordering is important (which is provided by the importance values in the tGAP face structure, where these values are increasing near the top of the structure) and that minimal redundancy has been a crucial aspect for the structures that are based on 2D space plus 1D scale. Hypothesis is that these aspects are equally important for going to higher dimensions, although it is more difficult – or even impossible – to directly visualise this 4D and 5D information.

BIBLIOGRAPHY

Agrawala, M. and Stolte, C. (2001). Rendering effective route maps: improving usability through generalization. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 241–249, New York, NY, USA. ACM. (Cited on page 176).

Ai, T. and Li, J. (2009). The lifespan model of GIS data representation over scale space. In *17th International Conference on Geoinformatics*, pages 1–6. (Cited on pages 41 and 50).

Ai, T. and van Oosterom, P. (2002). GAP-tree extensions based on skeletons. In Richardson, D. and van Oosterom, P., editors, *Advances in Spatial Data Handling, 10th International Symposium on Spatial Data Handling*, pages 501–513. (Cited on page 123).

Amenta, N., Choi, S., and Rote, G. (2003). Incremental constructions con BRIO. In *Proceedings 19th Annual Symposium on Computational Geometry*, pages 211–219, San Diego, USA. ACM Press. (Cited on page 81).

Aplin, P., Atkinson, P. M., and Curran, P. J. (1997). Fine spatial resolution satellite sensors for the next decade. *International Journal of Remote Sensing*, 18(18):3873–3881. (Cited on page 38).

Arroyo Ohori, K. (2010). Validation and automatic repair of planar partitions using a constrained triangulation. Master's thesis, Delft University of Technology, The Netherlands. (Cited on pages 72 and 81).

Baars, M., Stoter, J., van Oosterom, P., and Verbree, E. (2004). Rule-Based or Explicit Storage of Topology Structure: a Comparison Case Study. In Toppen, F. and Prastacos, P., editors, *Proceedings of the 7th Conference on Geographic Information Science (CD-ROM)*, pages 765–769. Heraclion: Crete University Press. (Cited on pages 22 and 181).

Bader, M. (2001). *Energy Minimization Methods for Feature Displacement in Map Generalization*. PhD thesis, University of Zürich, Switzerland. (Cited on page 35).

Bakker, N. J. (2005). Developing a new geographical object database. experiences from idea to delivering datasets. In *ICC 2005: Proceedings of the 22nd International Cartographic Conference: Mapping approaches into a changing world*, A Coruña, Spain. International Cartographic Association (ICA). (Cited on page 21).

Ballard, D. H. (1981). Strip trees: a hierarchical representation for curves. *Communications of the ACM*, 24(5):310–321. (Cited on page 38).

Barkowsky, T., Latecki, L. J., and Richter, K. F. (2000). Schematizing maps: Simplification of geographic shape by discrete curve evolution. In *Spatial Cognition II*, volume 1849 of *Lecture Notes in Computer Science*, pages 41–53. Springer Berlin Heidelberg. (Cited on pages 97 and 99).

Baumgart, B. G. (1975). A polyhedron representation for computer vision. In *AFIPS '75: Proceedings of the May 19-22, 1975, National computer conference and exposition*, pages 589–596, New York, NY, USA. ACM. (Cited on pages 26 and 119).

Bentley, J. L. (1990). K-d trees for semidynamic point sets. In *SCG '90: Proceedings of the sixth annual symposium on Computational Geometry*, pages 187–197, New York, NY, USA. ACM. (Cited on page 104).

Bern, M. and Eppstein, D. (1992). Mesh generation and optimal triangulation. In Du, D.-Z. and Hwang, F., editors, *Computing in Euclidean Geometry*, volume 4 of *Lecture Notes Series on Computing*, pages 23–90. World Scientific, Singapore, 2nd edition. (Cited on pages 73 and 116).

Bertolotto, M. (1998). *Geometric Modeling of Spatial Entities at Multiple Levels of Resolution*. PhD thesis, Department of Computer Science, University of Genova. (Cited on page 24).

Bertolotto, M. and Egenhofer, M. J. (2001). Progressive Transmission of Vector Map Data over the World Wide Web. *GeoInformatica*, 5(4):345–373. (Cited on page 51).

Blandford, D. K., Blelloch, G. E., Cardoze, D. E., and Kadow, C. (2005). Compact representations of simplicial meshes in two and three dimensions. *International Journal of Computational Geometry and Applications*, 15(1):3–24. (Cited on page 81).

Blum, H. (1967). A transformation for extracting new descriptors of shape. *Models for the perception of speech and visual form*, 19(5):362–380. (Cited on page 113).

Boissonnat, J.-D., Devillers, O., Pion, S., Teillaud, M., and Yvinec, M. (2002). Triangulations in CGAL. *Computational Geometry—Theory and Applications*, 22:5–19. (Cited on page 76).

Brassel, K. and Weibel, R. (1988). A review and conceptual framework of automated map generalization. *International Journal of Geographical Information Systems*, 2(3):229–244. (Cited on page 36).

Brönnimann, H. (2001). Designing and implementing a general purpose halfedge data structure. In Brodal, G., Frigioni, D., and Marchetti-Spaccamela, A., editors, *Algorithm Engineering*, volume 2141 of *Lecture Notes in Computer Science*, pages 51–66. Springer Berlin Heidelberg. (Cited on page 26).

Burghardt, D. and Schmid, S. (2010). Constraint-based evaluation of automated and manual generalised topographic maps. In Gartner, G. and Ortag, F., editors, *Cartography in Central and Eastern Europe*, Lecture Notes in Geoinformation and Cartography, pages 147–162. Springer Berlin Heidelberg. (Cited on page 35).

Buttenfield, B. P. and DeLotto, J. S. (1989). Multiple representations: Scientific report for the specialist meeting. Technical report, National Center for Geographic Information and Analysis. (Cited on page 37).

Chaudhry, O. and Mackaness, W. A. (2007). Utilising partonomic information in the creation of hierarchical geographies. In *Proceedings of 10th ICA Workshop on Generalisation and Multiple Representation*, Moscow. (Cited on page 35).

Cromley, R. G. (1991). Hierarchical methods of line simplification. *Cartography and Geographic Information Science*, 18:125–131. (Cited on page 34).

da Silva, A. C. G. and Wu, S. T. (2006). A robust strategy for handling linear features in topologically consistent polyline simplification. In Monteiro, A. M. V. and Davis, C. A., editors, *GeoInfo*, VIII Brazilian Symposium on Geoinformatics, 19-22 November, Campos do Jordão, São Paulo, Brazil, pages 19–34. (Cited on page 96).

Danciger, J., Devadoss, S. L., Mugno, J., Sheehy, D., and Ward, R. (2009). Shape deformation in continuous map generalization. *Geoinformatica*, 13(2):203–221. (Cited on pages 51 and 178).

Danovaro, E., De Floriani, L., Puppo, E., and Samet, H. (2007). Out-of-core multiresolution terrain modeling. In Belussi, A., Catania, B., Clementini, E., and Ferrari, E., editors, *Spatial Data on the Web: Modeling and Management*, pages 43–64. Springer. (Cited on page 38).

de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. (2000). *Computational geometry: Algorithms and applications*. Springer-Verlag, Berlin, second edition. (Cited on pages 26, 74, and 149).

de Berg, M., van Kreveld, M., and Schirra, S. (1998). Topologically correct subdivision simplification using the bandwidth criterion. *Cartography and Geographic Information Science*, 25:243–257. (Cited on page 96).

de Hoop, S., van Oosterom, P., and Molenaar, M. (1993). Topological querying of multiple map layers. In Frank, A. and Campari, I., editors, *Spatial Information Theory: A Theoretical Basis for GIS*, volume 716 of *Lecture Notes in Computer Science*, pages 139–157. Springer Berlin Heidelberg. (Cited on page 22).

De Maeyer, P., De Vliegher, B. M., and Brondeel, M. (2004). *De spiegel van de wereld: fundamenten van de cartografie*. Academia Press. In Dutch. (Cited on page 29).

Dettori, G. and Falcidieno, B. (1982). An algorithm for selecting main points on a line. *Computers & Geosciences*, 8(1):3–10. (Cited on page 34).

Devogele, T., Trevisan, J., and Raynal, L. (1996). Building a multi-scale database with scale-transition relationships. In *Proceedings of the 7th International Symposium on Spatial Data Handling*, pages 337–351, Delft, Netherlands. (Cited on page 41).

Dilo, A., van Oosterom, P., and Hofman, A. (2009). Constrained tGAP for generalization between scales: The case of Dutch topographic data. *Computers, Environment and Urban Systems*, 33(5):388–402. (Cited on page 199).

Douglas, D. (1974). It makes me so CROSS. Unpublished manuscript from the Harvard Laboratory for Computer Graphics and Spatial Analysis. Reprinted in Peuquet D. J. and Marble, D. F., editors, *Introductory Readings in Geographic Information System*, pages 303–307. Taylor & Francis, London. (Cited on page 8).

Douglas, D. H. and Peucker, T. K. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122. (Cited on pages 34 and 96).

Dyken, C., Dæhlen, M., and Sevaldrud, T. (2009). Simultaneous curve simplification. *Journal of Geographical Systems*, 11(3):273–289. (Cited on pages 34 and 35).

Erwig, M. and Schneider, M. (1997). Partition and conquer. In Hirtle, S. and Frank, A., editors, *Spatial Information Theory A Theoretical Basis for GIS*, volume 1329 of *Lecture Notes in Computer Science*, pages 389–407. Springer Berlin Heidelberg. (Cited on page 60).

ESRI (1998). ESRI shapefile technical description white paper. Technical report, Environmental Systems Research Institute. (Cited on pages 8 and 77).

European Parliament (2011). More frequencies for mobile internet by 2013. Press release. http://www.europarl.europa.eu/en/pressroom/content/20110510IPR19123/html/More-frequencies-for-mobile-internet-by-2013. (Cited on page 2).

Facello, M. A. (1995). Implementation of a randomized algorithm for Delaunay and regular triangulations in three dimensions. *Computer Aided Geometric Design*, 12:349–370. (Cited on page 76).

Filho, W. C., de Figueiredo, L. H., Carvalho, P. C., and Gattass, M. (1995). A topological data structure for hierarchical planar subdivisions. Technical Report CS-95-53, University of Waterloo. (Cited on pages 39 and 40).

Flato, E., Halperin, D., Hanniel, I., and Nechushtan, O. (1999). The Design and Implementation of Planar Maps in CGAL. In Vitter, J. and Zaroliagis, C., editors, *WAE'99*, volume 1668 of *Lecture Notes in Computer Science*, pages 154–168. Springer-Verlag Berlin Heidelberg. (Cited on page 26).

Foerster, T., Stoter, J., and Köbben, B. (2007). Towards a formal classification of generalization operators. In *ICC 2007: Proceedings of the 23rd International Cartographic Conference: Cartography for everyone and for you*, Moscow, Russia. International Cartographic Association (ICA). (Cited on pages 33 and 34).

Foerster, T., Stoter, J., and Kraak, M.-J. (2010). Challenges for Automated Generalisation at European Mapping Agencies: A Qualitative and Quantitative Analysis. *The Cartographic Journal*, 47(1):41–54. (Cited on page 33).

Forrest, D. (1993). Expert systems and cartographic design. *The Cartographic Journal*, 30(2):143–148. (Cited on page 36).

Frank, A. and Barrera, R. (1990). The Fieldtree: A data structure for Geographic Information Systems. In Buchmann, A., Günther, O., Smith, T., and Wang, Y.-F., editors, *Design and Implementation of Large Spatial Databases*, pages 29–44. Springer Berlin Heidelberg. (Cited on page 167).

Frank, A., Volta, G. S., and McGranaghan, M. (2001). Formalization of families of categorical coverages. *International Journal of Geographical Information Science*, 11(3):215 – 231. (Cited on page 39).

Friis-Christensen, A., Skogan, D., Jensen, C., Skagestein, G., and Tryfona, N. (2002). Management of multiply represented geographic entities. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS'02)*, pages 150–159. (Cited on page 37).

Gold, C. M. (1988). PAN graphs. An aid to GIS analysis. *International Journal of Geographical Information Systems*, 2(1):29–41. (Cited on pages 26 and 27).

Goodchild, M. F. (2001). Metrics of scale in remote sensing and GIS. *International Journal of Applied Earth Observation and Geoinformation*, 3(2):114–120. (Cited on pages 27 and 29).

Grünreich, D. (1985). Computer-assisted generalisation. In *CERCO Cartography Course, Session data Manipulation*, pages 1–16, Frankfurt am Main, Germany. Institut für angewandte Geodäsie. (Cited on page 33).

Grünreich, D. (1992). ATKIS – a topographic information system as a basis for GIS and digital cartography in Germany. In Vinken, R., editor, *From digital map series to geo-information systems*, volume 122 of *Geologisches Jahrbuch Reihe A*, pages 207–216. (Cited on page 33).

Guibas, L. J. and Sedgewick, R. (1978). A dichromatic framework for balanced trees. In *19th Annual Symposium on Foundations of Computer Science, 1978*, pages 8–21. (Cited on page 102).

Günther, O. (1988). *Efficient structures for geometric data management*. Number 337 in Lecture Notes in Computer Science. Springer-Verlag, Berlin. (Cited on page 38).

Hägerstrand, T. (1970). What about people in regional science? *Papers in Regional Science*, 24(1):6–21. (Cited on page 59).

Hampe, M., Sester, M., and Harrie, L. (2004). Multiple representation databases to support visualisation on mobile devices. In *Proceedings of the XXth ISPRS Congres*, volume 35 of *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 135–140. (Cited on pages 3, 41, and 184).

Hangouët, J.-F. and Lamy, S. (1999). Automated cartographic generalization: Approach and methods. In *ICC 1999: Proceedings of the 19th International Cartographic Conference*, Ottawa, Canada. (Cited on page 176).

Harrie, L., Sarjakoski, L. T., and Lehto, L. (2002). A variable-scale map for small-display cartography. In *Proceedings of Symposium on Geospatial Theory, Processing and Applications*, volume 34 of *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 237–242, Ottawa, Canada. ISPRS. (Cited on pages 31, 182, and 184).

Haunert, J.-H. (2011). Detecting symmetries in building footprints by string matching. In Geertman, S., Reinhardt, W., and Toppen, F., editors, *Advancing Geoinformation Science for a Changing World*, Lecture Notes in Geoinformation and Cartography, pages 319–336. Springer Berlin Heidelberg. (Cited on pages 35 and 36).

Haunert, J.-H., Dilo, A., and van Oosterom, P. (2009). Constrained set-up of the tGAP structure for progressive vector data transfer. *Computers & Geosciences*,

35(11):2191–2203. Progressive Transmission of Spatial Datasets in the Web Environment. (Cited on pages 141, 164, 167, 171, and 199).

Haunert, J.-H. and Sester, M. (2008). Area collapse and road centerlines based on straight skeletons. *Geoinformatica*, 12(2):169–191. (Cited on page 34).

Haunert, J.-H. and Wolff, A. (2010). Optimal and topologically safe simplification of building footprints. In Abbadi, A. E., Agrawal, D., Mokbel, M., and Zhang, P., editors, *Proceedings of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (ACM-GIS'10)*, pages 192–201. (Cited on page 35).

Hearn, D. D. and Baker, M. P. (2003). *Computer Graphics with OpenGL*. Pearson Prentice Hall. (Cited on page 151).

Herring, J. (2001). The OpenGIS Abstract Specification, Topic 1: Feature Geometry (ISO 19107 Spatial Schema), version 5. (Cited on pages 12, 24, and 72).

Herring, J. (2006). Implementation Specification for Geographic information - Simple feature access - Part 2: SQL option. (Cited on pages 12, 24, 72, and 77).

Hevner, A. and Chatterjee, S. (2010). *Design Research in Information Systems: Theory and Practice*, volume 22 of *Integrated Series in Information Systems*. Springer. (Cited on page 11).

Hevner, A. R., March, S., J., P., and Ram, S. (2004). Design science in information systems research. *Management Information Systems Quarterly*, 28:75–105. (Cited on pages 10, 228, and 232).

Hoel, E. G., Menon, S., and Morehouse, S. (2003). Building a robust relational implementation of topology. In Hadzilacos, T., Manolopoulos, Y., Roddick, J., and Theodoridis, Y., editors, *Advances in Spatial and Temporal Databases*, volume 2570 of *Lecture Notes in Computer Science*, pages 508–524. Springer Berlin Heidelberg. (Cited on page 24).

Hoffmann, C. M. (1989). The problems of accuracy and robustness in geometric computation. *Computer*, 22(3):31–39. (Cited on page 8).

Hoppe, H. (1996). Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 99–108, New York, NY, USA. ACM. (Cited on page 38).

Imhof, E. (1972). *Thematische Kartographie*. Walter de Gruyter. (Cited on page 30).

ISO/TC 211 (2003). ISO 19107:2003 Geographic information - Spatial schema. (Cited on pages 24 and 26).

Jakobsson, A. (2002). Data Quality and Quality Management – Examples of Quality Evaluation Procedures and Quality Management in European National Mapping Agencies. In Shi, W., Fisher, P., and Goodchild, M. F., editors, *Spatial Data Quality*, pages 216–229. Taylor & Francis, London, first edition. (Cited on page 32).

Jenks, G. F. (1989). Geographic logic in line generalization. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 26(1):27–42. (Cited on page 34).

Jones, C. B. and Abraham, I. M. (1986). Design considerations for a scale-independent cartographic database. In Marble, D., editor, *Second International Symposium on Spatial Data Handling*, pages 384–398, Seattle, Washington. (Cited on page 38).

Jones, C. B., Bundy, G. L., and Ware, J. M. (1995). Map generalization with a triangulated data structure. *Cartography and Geographic Information Science*, 22(4):317–331. (Cited on page 34).

Jones, C. B. and Ware, J. M. (2005). Map generalization in the web age. *International Journal of Geographical Information Science*, 19(8 & 9):859–870. (Cited on page 41).

Jongeneel, C. (2011). Draadloze datashuffle. *Technisch Weekblad*, 11:8–9. In Dutch. (Cited on page 2).

Kennedy, M. and Kopp, S. (2001). *Understanding Map Projections*. Esri Press. (Cited on page 29).

Kettner, L. (1999). Using generic programming for designing a data structure for polyhedral surfaces. *Computational Geometry*, 13(1):65–90. (Cited on pages 25 and 26).

Kettner, L., Mehlhorn, K., Pion, S., Schirra, S., and Yap, C. (2008). Classroom examples of robustness problems in geometric computations. *Computational Geometry*, 40(1):61–78. (Cited on page 8).

Kothuri, R., Godfrind, A., and Beinat, E. (2007). *Pro Oracle Spatial for Oracle Database 11g*. Apress. (Cited on pages 22 and 40).

Kraak, M.-J. (1998). The cartographic visualization process: From presentation to exploration. *The Cartographic Journal*, 35(1):11–15. (Cited on page 193).

Kulik, L., Duckham, M., and Egenhofer, M. (2005). Ontology-driven map generalization. *Journal of Visual Languages & Computing*, 16(3):245–267. (Cited on pages 34, 96, 97, 99, 105, and 139).

Lalonde, W. (2002). Styled Layer Descriptor Implementation Specification. (Cited on page 145).

Lam, N. S.-N. and Quattrochi, D. A. (1992). On the issues of scale, resolution, and fractal analysis in the mapping sciences. *Professional Geographer*, 44(1):88–98. (Cited on pages 27 and 28).

Lamy, S., Ruas, A., Demazeu, Y., Jackson, M., Mackaness, W., and Weibel, R. (1999). The application of agents in automated map generalisation. In *ICC 1999: Proceedings of the 19th International Cartographic Conference*, Ottawa, Canada. (Cited on pages 36 and 196).

Ledoux, H. and Arroyo Ohori, K. (2011). Edge-matching polygons with a constrained triangulation. In *Proceedings of Symposium GIS Ostrava 2011*. (Cited on pages 72 and 81).

Ledoux, H. and Meijers, M. (2010). Validation of planar partitions using constrained triangulations. In *Proceedings of the 14th Joint International Conference on Theory, Data Handling and Modelling in Geospatial Information Science*, pages 51–56, Hong Kong. (Cited on pages 15 and 53).

Lévy, B. and Mallet, J.-L. (1999). Cellular modelling in arbitrary dimension using generalized maps. Technical report, Gocad Consortium. (Cited on page 66).

Lie, H. W. (2005). *Cascading Style Sheets*. PhD thesis, University of Oslo, Norway. (Cited on page 7).

Liu, Y. and Snoeyink, J. (2005). The "far away point" for Delaunay diagram computation in $\mathcal{E}^d$. In *Proceedings 2nd International Symposium on Voronoi Diagrams in Science and Engineering*, pages 236–243, Seoul, Korea. (Cited on page 76).

Liu, Y. and Snoeyink, J. (2008). Faraway point: A sentinel point for Delaunay computation. *International Journal of Computational Geometry and Applications*, 18(4):343–355. (Cited on page 116).

Louwsma, J., Tijssen, T., and van Oosterom, P. (2003). Topology under the microscope. GeoConnexion. (Cited on page 181).

Lüscher, P., Weibel, R., and Burghardt, D. (2009). Integrating ontological modelling and bayesian inference for pattern classification in topographic vector data. *Computers, Environment and Urban Systems*, 33(5):363–374. (Cited on pages 9 and 36).

Mackaness, W. A. (1994). An algorithm for conflict identification and feature displacement in automated map generalization. *Cartography and Geographic Information Science*, 21(4):219–232. (Cited on page 34).

Mackaness, W. A. (2006). Automated cartography in a bush of ghosts. *Cartography and Geographic Information Science*, 33(4):245–256. (Cited on pages 50, 176, and 177).

Mackaness, W. A. and Mackechnie, G. A. (1999). Automating the detection and simplification of junctions in road networks. *GeoInformatica*, 3:185–200. (Cited on page 35).

Mackaness, W. A., Ruas, A., and Sarjakoski, L. T. (2007). *Generalisation of geographic information: cartographic modelling and applications*. Elsevier Science Ltd. (Cited on page 67).

March, S. and Smith, G. (1995). Design and natural science research on information technology. *Decision Support Systems*, 15:251–266. (Cited on page 10).

Matijević, H., Biljecki, Z., Pavičić, S., and Roić, M. (2008). Transaction processing on planar partition for cadastral application. In *Proceedings FIG Working Week 2008--Integrating Generations*, Stockholm, Sweden. (Cited on page 26).

McAllister, M. and Snoeyink, J. (April 2000). Medial axis generalization of river networks. *Cartography and Geographic Information Science*, 27:129–138. (Cited on page 34).

McMaster, R. B. (1987). Automated line generalization. *Cartographica: The International Journal for Geographic Information 2and Geovisualization*, 24(2):74–111. (Cited on page 34).

Meijers, M. (2006). Implementation and testing of variable scale topological data structures: Experiences with the GAP-face tree and GAP-edge forest. Master's thesis, Delft University of Technology. (Cited on pages 4 and 58).

Meijers, M. (2008). Retrieving tGAP data with a stateless client for visualization. RGI Project Report 233-03, Delft University of Technology, Delft. (Cited on pages 15 and 142).

Meijers, M. (2011a). Cache-friendly progressive data streaming with variable-scale data structures. In *Proceedings of 14th ICA/ISPRS Workshop on Generalisation and Multiple Representation*, pages 1–19. (Cited on pages 15 and 142).

Meijers, M. (2011b). Simultaneous & topologically-safe line simplification for a variable-scale planar partition. In Geertman, S., Reinhardt, W., and Toppen, F., editors, *Advancing Geoinformation Science for a Changing World*, Lecture Notes in Geoinformation and Cartography, pages 337–358. Springer Berlin Heidelberg. (Cited on pages 15 and 84).

Meijers, M., Savino, S., and van Oosterom, P. (2011). SplitArea: An algorithm for splitting faces in the context of a hierarchical data structure. Manuscript submitted for review to an academic journal. (Cited on pages 15 and 84).

Meijers, M. and van Oosterom, P. (2009). Applying DLM and DCM concepts in a multi-scale environment. In Mustière, S., Sester, M., van Harmelen, F., and van Oosterom, P., editors, *Dagstuhl Seminar Proceedings 'Generalization of Spatial Information (09161)'*. (Cited on pages 15 and 20).

Meijers, M. and van Oosterom, P. (2011). The space-scale cube: An integrated model for 2D polygonal areas and scale. In *28th Urban Data Management Symposium*, volume 38 of *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 95–102. (Cited on pages 15 and 54).

Meijers, M., van Oosterom, P., and Quak, W. (2009). A storage and transfer efficient data structure for variable scale vector data. In Sester, M., Bernard, L., and Paelke, V., editors, *Advances in GIScience*, Lecture Notes in Geoinformation and Cartography, pages 345–367. Springer Berlin Heidelberg. (Cited on pages 15, 20, and 84).

Meyer, B. (1992). Applying 'Design by Contract'. *Computer*, 25(10):40–51. (Cited on page 8).

Meyer, B. (1997). *Object-oriented software construction*. Prentice-Hall, second edition. (Cited on page 8).

Midtbø, T. and Nordvik, T. (2007). Effects of animations in zooming and panning operations on web maps: a web-based experiment. *The Cartographic Journal*, 44(4):292–303. (Cited on pages 50, 51, and 200).

Misue, K., Eades, P., Lai, W., and Sugiyama, K. (1995). Layout adjustment and the mental map. *Journal of Visual Languages & Computing*, 6(2):183–210. (Cited on pages 31 and 50).

Molenaar, M. (1989). Single valued vector maps: a concept in Geographic Information Systems. *Geo-Informations-Syteme*, 2(1):18–26. (Cited on page 22).

Molenaar, M. (1998). *An introduction to the theory of spatial object modelling for GIS*. Research Monographs in GIS. Taylor & Francis, London. (Cited on pages 22 and 23).

Mücke, E. P. (1998). A robust implementation for three-dimensional Delaunay triangulations. *International Journal of Computational Geometry and Applications*, 8(2):255–276. (Cited on page 76).

Muller, D. E. and Preparata, F. P. (1978). Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7(2):217–236. (Cited on page 26).

NCGIA (1989). The research plan of the National Center for Geographic Information and Analysis. *International Journal of Geographical Information Systems*, 3(2):117–136. (Cited on pages 37 and 50).

Neun, M. (2007). *Data Enrichment for Adaptive Map Generalization Using Web Services*. PhD thesis, University of Zürich, Switzerland. (Cited on page 35).

Neun, M., Burghardt, D., and Weibel, R. (2009). Automated processing for map generalization using web services. *Geoinformatica*, 13:425–452. (Cited on page 196).

Nöllenburg, M., Merrick, D., Wolff, A., and Benkert, M. (2008). Morphing polylines: A step towards continuous generalization. *Computers, Environment and Urban Systems*, 32(4):248–260. Geographical Information Science Research - United Kingdom. (Cited on page 178).

Ordnance Survey (2010). A guide to coordinate systems in Great Britain. D00659 v2.1. (Cited on page 29).

Penninga, F. (2004). Oracle 10g Topology: Testing Oracle 10g Topology using cadastral data. Technical report, Delft University of Technology, Delft. (Cited on pages 13, 26, and 181).

Persson, J. (2004). Streaming of compressed multi-resolution geographic vector data. In Brandt, S. A., editor, *Proceedings of 12th International Conference on Geoinformatics Geospatial Information Research: Bridging the Pacific and Atlantic*, pages 765–775. University of Gävle. (Cited on page 41).

Peuquet, D. J. (1984). A conceptual framework and comparison of spatial data models. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 21(4):66–113. (Cited on pages 20, 22, 23, and 24).

Plümer, L. and Gröger, G. (1996). Nested maps—a formal, provably correct object model for spatial aggregates. In *Proceedings of the 4th ACM international workshop on Advances in geographic information systems*, page 83. (Cited on page 39).

Plümer, L. and Gröger, G. (1997). Achieving integrity in geographic information systems—maps and nested maps. *Geoinformatica*, 1(4):345–367. (Cited on page 39).

Ramer, U. (1972). An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256. (Cited on page 96).

Rauschenbach, U. and Schumann, H. (1999). Demand-driven image transmission with levels of detail and regions of interest. *Computers & Graphics*, 23(6):857–866. (Cited on page 50).

Reichenbacher, T. (2004). *Mobile Cartography – Adaptive Visualisation of Geographic Information on Mobile Devices*. PhD thesis, University of Technology, Munich, Germany. (Cited on page 182).

Rigaux, P. and Scholl, M. (1995). Multi-scale partitions: Application to spatial and statistical databases. In Egenhofer, M. and Herring, J., editors, *Advances in Spatial Databases*, volume 951 of *Lecture Notes in Computer Science*, pages 170–183. Springer Berlin Heidelberg. (Cited on page 39).

Robinson, A. H. (1953). *Elements of cartography*. Wiley. (Cited on pages 28, 29, and 31).

Saalfeld, A. (1999). Topologically consistent line simplification with the douglas-peucker algorithm. *Cartography and Geographic Information Science*, 26:7–18. (Cited on page 96).

Schirra, S. (1997). Precision and robustness in geometric computations. In van Kreveld, M., Nievergelt, J., Roos, T., and Widmayer, P., editors, *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, pages 255–287. Springer Berlin Heidelberg. (Cited on page 8).

Schmid, S. (2008). Automated constraint-based evaluation of cartographic generalization solutions. Master's thesis, University of Zürich, Switzerland. (Cited on page 35).

Seljebotn, D. (2009). Fast numerical computations with Cython. In Varoquaux, G., van der Walt, S., and Millman, J., editors, *Proceedings of the 8th Python in Science Conference*, pages 15–23. (Cited on page 12).

Sester, M. (2000). Generalization based on least squares adjustment. *International Archives of Photogrammetry and Remote Sensing*, 33:931–938. (Cited on page 35).

Sester, M. (2005). Optimization approaches for generalization and data abstraction. *International Journal of Geographical Information Science*, 19(8):871–897. (Cited on page 35).

Sester, M. and Brenner, C. (2004). Continuous generalization for visualization on small mobile devices. In Fisher, P. F., editor, *Developments in Spatial Data Handling: 11th International Symposium on Spatial Data Handling*, pages 355–368. Springer Berlin Heidelberg. (Cited on page 51).

Sester, M. and Brenner, C. (2009). A vocabulary for a multiscale process description for fast transmission and continuous visualization of spatial data. *Computers & Geosciences*, 35(11):2177–2184. (Cited on page 200).

Shewchuk, J. R. (1997). *Delaunay Refinement Mesh Generation*. PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburg, USA. (Cited on page 74).

Shneiderman, B. (1996). The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings IEEE Symposium Visual Languages '96*, pages 336–343, College Park, Maryland 20742, U.S.A. (Cited on page 48).

Shreiner, D., Woo, M., Neider, J., and Davis, T. (2005). *OpenGL Programming Guide: the official guide to learning OpenGL*. Addison-Wesley. (Cited on page 163).

SSC (2005). *Topographic Maps - Map Graphics and Generalisation*. Swiss Society of Cartography, Bern. Cartographic Publication Series, volume 17, CD-ROM. (Cited on page 29).

Stoter, J. (2005). Generalisation within NMA's in the 21st century. In *ICC 2005: Proceedings of the 22nd International Cartographic Conference: mapping approaches into a changing world*, pages 1–11, A Coruña, Spain. International Cartographic Association (ICA). (Cited on page 32).

Stoter, J., Burghardt, D., Duchêne, C., Baella, B., Bakker, N., Blok, C., Pla, M., Regnauld, N., Touya, G., and Schmid, S. (2009a). Methodology for evaluating automated map generalization in commercial software. *Computers, Environment and Urban Systems*, 33(5):311–324. (Cited on pages 35 and 196).

Stoter, J., Meijers, M., van Oosterom, P., Grünreich, D., and Kraak, M.-J. (2010). Applying DLM and DCM concepts in a multi-scale data environment. In Buttenfield, B., Brewer, C., Clarke, K., Finn, M., and Usery, L., editors, *Proceedings of GDI 2010: Symposium on Generalization and Data Integration*, pages 1–7, Boulder, USA. University of Colorado. (Cited on pages 15 and 20).

Stoter, J., Morales, J., Lemmens, R., Meijers, M., van Oosterom, P., Quak, W., and Uitermark, H. (2007). Considerations for the design of a semantic data model for a multi-representation topographical database. In Kremers, H., editor, *Proceedings of the 2nd ISGI 2007: International CODATA symposium on generalization of information, Geneva, Switzerland, 1-3 October 2007*, Lecture notes in Information Sciences, pages 53–71, Berlin. CODATA. (Cited on pages 15 and 19).

Stoter, J., Morales, J., Lemmens, R., Meijers, M., van Oosterom, P., Quak, W., Uitermark, H., and van den Brink, L. (2008). A data model for multi-scale topographical data. In Ruas, A. and Gold, C., editors, *Headway in Spatial Data Handling: Proceedings of the 13th international symposium on Spatial*

Data Handling, SDH 2008, Lecture Notes in Geoinformaton and Cartography, pages 233–254, Berlin. Springer. (Cited on pages 15 and 20).

Stoter, J., van Smaalen, J., Bakker, N., and Hardy, P. (2009b). Specifying map requirements for automated generalization of topographic data. *The Cartographic Journal*, 46(14):214–227. (Cited on page 9).

Stoter, J., Visser, T., van Oosterom, P., Quak, W., and Bakker, N. (2011). A semantic-rich multi-scale information model for topography. *International Journal of Geographical Information Science*, 25(5):739–763. (Cited on page 32).

Theobald, D. M. (2001). Topology revisited: representing spatial relations. *International Journal of Geographical Information Science*, 15(8):689–705. (Cited on pages 23, 24, and 26).

Tichy, W. F. (1997). Should computer scientists experiment more? - 16 excuses to avoid experimentation. *IEEE Computer*, 31:32–40. (Cited on page 11).

Timpf, S. (1999). Abstraction, levels of detail, and hierarchies in map series. In Freksa, C. and Mark, D. M., editors, *Spatial Information Theory - cognitive and computational foundations of Geographic Information Science*, volume 1661 of *Lecture Notes in Computer Science*, pages 125–140, Stade, Germany. Springer-Verlag. (Cited on page 36).

Timpf, S. and Devogele, T. (1997). New tools for multiple representations. In Ottoson, L., editor, *ICC1997: Proceedings of the 18th International Cartographic Conference*, pages 1381–1386, Stockholm. (Cited on page 50).

Tobler, W. R. (1987). Measuring spatial resolution. In *Proceedings, Land Resources Information Systems Conference*, pages 12–16, Beijing. (Cited on page 29).

Töpfer, F. (1974). *Kartographische Generalisierung*. VEB Hermann Haack, Geographisch-Kartographische Anstalt Gotha, Leipzig. (Cited on page 28).

Uitermark, H., Vogels, A., and van Oosterom, P. (1999). Semantic and geometric aspects of integrating road networks. In *INTEROP '99: Proceedings of the Second International Conference on Interoperating Geographic Information Systems*, volume 1580, pages 177–188, London, UK. Springer-Verlag. (Cited on page 116).

van Bemmelen, J., Quak, W., van Hekken, M., and van Oosterom, P. (1993). Vector vs. raster-based algorithms for cross country movement planning. *Auto-Carto*, 11:304–317. (Cited on page 202).

van Kreveld, M. (2001). Smooth generalization for continuous zooming. In *ICC 2001: Proceedings 20th International Cartographic Conference*, pages 2180–2185, Beijing, China. (Cited on pages 50 and 178).

van Oosterom, P. (1990). *Reactive Data Structures for Geographic Information Systems*. PhD thesis, Leiden University. (Cited on pages 8, 38, 42, and 58).

van Oosterom, P. (1993). The GAP-tree, an approach to "on-the-fly" map generalization of an area partitioning. In *Proceedings of GISDATA Specialist Meeting on Generalization*, pages 1–15, Compiènge, France. (Cited on pages 128 and 191).

van Oosterom, P. (1995). The GAP-tree, an approach to "on-the-fly" map generalization of an area partitioning. In Müller, J., Lagrange, J., and Weibel, R., editors, *GIS and Generalization, Methodology and Practice*, pages 120–132. Taylor & Francis. (Cited on page 128).

van Oosterom, P. (2001). De geo-database als spin in het web. Delft. In Dutch. (Cited on page 2).

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*, 32:331–346. (Cited on pages 4, 5, 42, 43, 58, 84, 175, 191, 227, and 231).

van Oosterom, P. (2009). Research and development in geo-information generalisation and multiple representation. *Computers, Environment and Urban Systems*, 33(5):303–310. (Cited on page 37).

van Oosterom, P., De Vries, M., and Meijers, M. (2006). Vario-scale data server in a web service context. In *Proceedings of the ICA Workshop on Map Generalisation and Multiple Representation*, pages 1–14, Vancouver, USA. (Cited on pages 4 and 85).

van Oosterom, P. and Lemmen, C. (2001). Spatial data management on a very large cadastral database. *Computers, Environment and Urban Systems*, 25(4-5):509–528. (Cited on page 21).

van Oosterom, P. and Meijers, M. (2011a). Method and system for generating maps in an n-dimensional space. Dutch patent application 2006630, filed April 19, 2011, expected to be published October 2012. (Cited on pages 15, 143, and 173).

van Oosterom, P. and Meijers, M. (2011b). Towards a true vario-scale structure supporting smooth-zoom. In *Proceedings of 14th ICA/ISPRS Workshop on Generalisation and Multiple Representation*, pages 1–19, Paris. (Cited on pages 15 and 174).

van Oosterom, P., Quak, W., and Tijssen, T. (2003). Polygons: the unstable foundation of spatial modeling. In *Proceedings ISPRS joint workshop on spatial, temporal and multi-dimensional data modelling and analysis, Quebec, Canada*. ISPRS. (Cited on page 59).

van Oosterom, P. and Schenkelaars, V. (1995). The development of an interactive multi-scale GIS. *International Journal of Geographical Information Science*, 9(5):489–507. (Cited on pages 42 and 58).

van Oosterom, P. and Stoter, J. (2010). 5D data modelling: full integration of 2D/3D space, time and scale dimensions. In Fabrikant, S., Reichenbacher, T., van Kreveld, M., and Schlieder, C., editors, *Geographic Information Science*, volume 6292 of *Lecture Notes in Computer Science*, pages 310–324. Springer Berlin Heidelberg. (Cited on page 202).

van Oosterom, P. and van den Bos, J. (1990). An object-oriented approach to the design of geographic information systems. In Buchmann, A., Günther, O., Smith, T., and Wang, Y.-F., editors, *Design and Implementation of Large Spatial Databases*, volume 409 of *Lecture Notes in Computer Science*, pages 253–269. Springer Berlin Heidelberg. 10.1007/3-540-52208-5_31. (Cited on page 39).

van Oosterom, P. and Vijlbrief, T. (1994). Integrating complex spatial analysis functions in an extensible gis. In *Proceedings of the 6th International Symposium on Spatial Data Handling*, pages 277–296, Edinburgh, Scotland. (Cited on page 87).

van Putten, J. and van Oosterom, P. (1998). New results with Generalized Area Partitionings. In *8th International Symposium on Spatial Data Handling*, pages 485–495. (Cited on page 167).

van Smaalen, J. (2003). *Automated Aggregation of Geographic Objects: A New Approach to the Conceptual Generalisation of Geographic Databases*. PhD thesis, Wageningen University. (Cited on page 9).

van Wijk, J. J. and Nuij, W. A. A. (2003). Smooth and efficient zooming and panning. In *9th IEEE Symposium on Information Visualization (InfoVis 2003), 20-21 October 2003, Seattle, WA, USA*. IEEE Computer Society. (Cited on page 200).

van Wijk, J. J. and Nuij, W. A. A. (2004). A Model for Smooth Viewing and Navigation of Large 2D Information Spaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(4):447–458. (Cited on page 200).

Vangenot, C. (2004). Multi-representation in spatial databases using the MADS conceptual model. In *Proceedings of ICA Workshop on Generalisation and Multiple representation*, Leicester. (Cited on page 40).

Vangenot, C., Parent, C., Spaccapietra, S., Zimanyi, E., Donini, P., and Plazanet, C. (1998). Modeling spatial data in the MADS conceptual model. In *Proceedings of the International Symposium on Spatial Data Handling*, Vancouver, Canada. (Cited on page 40).

Vermeij, M. (2003). Development of a Topological Data Structure for On-the-Fly Map Generalization. Master's thesis, Delft University of Technology, The Netherlands. (Cited on pages 42, 43, 58, and 59).

Visvalingam, M. and Whyatt, J. D. (1993). Line generalisation by repeated elimination of points. *The Cartographic Journal*, 30(1):46–51. (Cited on pages 34, 97, and 99).

Ware, J. M., Jones, C. B., and Thomas, N. (2003). Automated map generalization with multiple operators: a simulated annealing approach. *Internation Journal of Geographical Information Science*, 17:743–769. (Cited on page 36).

Weibel, R. (1991). Amplified intelligence and knowledge-based systems. In Buttenfield, B. and McMaster, R., editors, *Map Generalization: Making Rules for Knowledge Representation*, pages 172–186. Longman, London. (Cited on page 32).

Weibel, R. (1997). Generalization of spatial data: Principles and selected algorithms. In van Kreveld, M., Nievergelt, J., Roos, T., and Widmayer, P., editors,

Algorithmic Foundations of Geographic Information Systems, volume 1340 of *Lecture Notes in Computer Science*, pages 99–152. Springer Berlin Heidelberg. (Cited on page 34).

Westra, E. (2010). *Python Geospatial Development*. Packt Publishing. (Cited on page 11).

Wiggins, J. C., Hartley, R. P., Higgins, M. J., and Whittaker, R. J. (1987). Computing aspects of a large geographic information system for the European Community. *International Journal of Geographical Information Science*, 1(1):77–87. (Cited on page 13).

Worboys, M. F. and Duckham, M. (2004). *GIS: A computing perspective*. CRC Press, second edition. (Cited on pages 26 and 113).

Yang, B., Purves, R., and Weibel, R. (2007). Efficient transmission of vector data over the internet. *International Journal of Geographical Information Science*, 21(2):215–237. (Cited on page 52).

Zhou, S. and Jones, C. B. (2004). Shape-aware line generalisation with weighted effective area. In Fisher, P. F., editor, *Developments in Spatial Data Handling: 11th International Symposium on Spatial Data Handling*, pages 369–380. Springer Berlin Heidelberg. (Cited on page 34).

Zhou, X., Prasher, S., Sun, S., and Xu, K. (2004). Multiresolution spatial databases: Making web-based spatial applications faster. In Yu, J., Lin, X., Lu, H., and Zhang, Y., editors, *Advanced Web Technologies and Applications*, volume 3007 of *Lecture Notes in Computer Science*, pages 36–47. Springer Berlin Heidelberg. (Cited on page 51).

Zlatanova, S., Stoter, J., and Quak, W. (2004). Management of multiple representations in spatial DBMSs. In Toppen, F. and Prastacos, P., editors, *Proceedings of AGILE 2004: 7th conference on Geographic Information Science*, pages 269–278, Heraklion. Crete University Press. (Cited on page 38).

SUMMARY

~

Variable-scale Geo-information

The use of geo-information is changing by the advent of new mobile devices, such as tablet-pc's that harness a lot of computing power. This type of information is more and more applied in mainstream digital consumer products, in a net-centric environment (*i. e.* dissemination takes place via the Internet) and the advances in mobile hardware also have changed the way people can interact with the geographic information at hand, compared to 'old-fashioned' paper maps.

However, current state-of-the-art solutions for storing, maintaining and disseminating digital maps still mimic the analogue map-series concept in the sense that for every map scale in the serie (*e. g.* 1:25k, 1:50k, 1:250k) a different digital copy with independent data is kept and maintained at the producers site. The challenge of this work was to get to a representation of the real world with gradually changing level of detail, instead of representations with discrete levels of detail (organised in multiple, independent layers, each layer representing only one resolution level).

Vario-scale data structures try to avoid this redundancy of the geometric description of the map by storing references to composing map elements of the highest level of detail for any other element of a lower level of detail. An example of variable-scale data structures are the tGAP data structures (van Oosterom, 2005). In addition to the geometry and references, an importance value for every object is stored and based on this importance value different representations (where the level of detail is gradually changing) can be derived on the fly from these structures according to the needed level of detail.

The overall aim of this research has been to investigate variable-scale geo-information, by defining theoretical underpinnings of vario-scale geo-information

and improving the initial tGAP structures. The objective we had with this research is expressed in the main question, which was formulated as:

How can we realise improved vario-scale geo-information having minimal redundancy?

The overall outline of the research design draws heavily upon the paradigm of design research (Hevner et al., 2004). In an iterative fashion we performed theory building, prototype developments and experiments with real world data sets. Over the course of this research, we have made the following main contributions to the design of a vario-scale geo-information environment. We have:

- formalised the concept of variable-scale data as a conceptual 3D model (the space-scale cube, SSC), where 2D space and 1D scale is integrated;
- shown for the tGAP data structures how minimal data redundancy can be obtained when applying a merge operation, how to perform a parallel simplification of lines, without introducing unwanted topological errors and proposed a split operation, for which it was analysed what the impacts are on the designed data structures;
- shown how to derive a 2D map from the structures with a particular number of objects, as well as investigated progressive data streaming;
- proposed an improved way of generating data so that even smoother graphic transitions can be derived for visualisation.

The main conclusions that can be drawn from these contributions:

1. With the concept of the proposed space-scale cube (SSC) we have formalised what vario-scale vector data entails. In a sense, the improved design of the tGAP data structures can be seen as a lossless encoding of the data that is captured for a ssc.
2. To make vario-scale geo-information operational, we need specific generalisation operations. These vario-scale generalisation operations should be designed carefully to be able to give guarantees on the amount of data to be stored and output topologically consistent vario-scale data.
3. Although the improved tGAP structures are capable of providing a smooth zooming end user experience, we still store and visualise discrete steps –

albeit smaller and more local than is common with current state of the art solutions. Therefore we proposed how smoothness of the vario-scale data can be improved (where the smooth SSC taking a small step in scale leads to a small change in the 2D derived map). A novelty of this approach is that, as it is one integrated space-scale partition, using a non-horizontal slice plane leads to a valid, mixed-scale planar partition: this is useful for use in 3D computer graphics (far away from an observer having less detail than close by).

Although this research has generated some knowledge for a vario-scale environment, it also paves the way for future research. The main recommendations for future work are:

- Investigate how to deal with very large data sets that do not fit in main memory (during the generalisation process or during visualisation) deserves attention.

- The smooth encoding of the SSC has the same building challenge as the classic tGAP with respect to applying the right sequence of generalisation operators (remove or merge, collapse or split, simplify) to obtain maps with sufficient cartographic quality.

  Another point for further research is the smooth interactions: it is of importance to know how users perceive these. The same holds for mixed-scale slices (in a 3D world).

- Focus of this research has been mostly on obtaining and viewing vario-scale data. Performing analysis with vario-scale data is another interesting aspect that deserves attention, *e. g.* vario-scale data could be of help in data integration.

- Investigate how to make the structures dynamic: currently the tGAP structure (including the new smooth variant) is a static structure and has to be re-built if the source data changes. Being able to perform incremental updates (partially re-generalising data for a new situation) would be beneficial if the data volume increases.

  Related to this is higher dimensionality of smooth, vario-scale data (*e. g.* 3D data) leading to integrated 5D data management (integrating dimensions of space (2D or 3D), time (updates, 1D) and scale (level of detail, 1D).

SAMENVATTING

~

Variabele-schaal Geo-informatie

Het gebruik van geo-informatie is de laatste tijd sterk veranderd door de opkomst van nieuwe mobiele hardware, zoals tablet pc's die over veel rekenkracht beschikken. Daarnaast wordt geo-informatie, zoals topografische kaarten, meer en meer toegepast in consumentenproducten (denk aan mobiele telefoons) binnen een netwerk-georiënteerde omgeving (*i. e.* de verspreiding van de informatie vindt plaats via het internet) en de nieuwe mobiele hardware biedt tevens nieuwe mogelijkheden, ten opzichte van 'ouderwetse' papieren kaarten, voor de manieren waarop mensen interacties kunnen hebben met de geografische informatie.

Echter, de huidige state-of-the-art oplossingen voor het opslaan, onderhouden en verspreiden van digitale kaarten bootsen nog steeds de analoge kaartseries na, in de zin dat voor elke kaartschaal in de serie (bijvoorbeeld 1:25k, 1:50k, 1:250k) een aparte digitale kopie met onafhankelijke gegevens wordt bewaard en onderhouden door de producent. De uitdaging van dit onderzoek is om tot één representatie van de echte wereld met geleidelijk veranderende mate van detail te komen, in plaats van meerdere voorstellingen met hun eigen discrete detailniveaus (die ook nog georganiseerd zijn als aparte en onafhankelijke kaartlagen).

Vario-schaal datastructuren proberen om de redundantie van de geometrische beschrijving te vermijden door de kaarten op te slaan als verwijzingen in een structuur, waarbij elementen van een laag detailniveau verwijzen naar elementen van het hoogste detailniveau. Een voorbeeld van deze zogenaamde variabele-schaal datastructuren zijn de tGAP datastructuren (van Oosterom, 2005). In aanvulling op de geometrie en referenties, wordt voor elk element een

zogenaamde belangrijkheidswaarde opgeslagen en op basis van deze waarde kunnen 2D kaarten on-the-fly worden afgeleid (waarbij de mate van detail geleidelijk verandert).

Het algemene doel van dit onderzoek is het definiëren van de theoretische onderbouwing van variabele-schaal geo-informatie en het verbeteren van de initiële tGAP structuren. Dit doel komt tot uitdrukking in de onderzoeksvraag, die werd geformuleerd als:

Hoe kunnen we verbeterde vario-schaal geo-informatie realiseren, waarbij minimaal redundante data-opslag gewaarborgd is?

De onderzoeksopzet leunt op het paradigma van design research (Hevner et al., 2004). In een iteratieve manier hebben we theorievorming, prototype ontwikkelingen en experimenten met data sets uit de praktijk afgewisseld. In de loop van dit onderzoek zijn de volgende bijdragen aan het ontwerp van een vario-schaal geo-informatie omgeving gerealiseerd. We hebben:

- het concept van variabele-schaal gegevens geformaliseerd als een conceptueel 3D model (de ruimte-schaal kubus, Engels: space-scale cube SSC), waar 2D ruimte en 1D schaal in is geïntegreerd;
- weergegeven hoe voor de tGAP datastructuren een minimale redundante opslag van gegevens kan worden verkregen na de toepassing van een samenvoeg-actie, hoe parallel lijnen te versimpelen, zonder dat er ongewenste topologische fouten optreden en een splits operatie voorgesteld, waarvoor werd geanalyseerd wat de effecten zijn op de ontworpen data structuren;
- laten zien hoe een 2D kaart uit de structuren met een ingesteld gemiddeld aantal objecten kan worden afgeleid, evenals dat we progressieve data overdracht hebben onderzocht;
- een verbeterde manier van het genereren van vario-schaal gegevens voorgesteld, zodat nog geleidelijkere grafische overgangen kunnen worden afgeleid voor visualisatie doeleinden.

De belangrijkste conclusies die getrokken kunnen worden uit deze onderzoeksbijdragen:

1. Met het concept van de voorgestelde ruimte-schaal cube (SSC) hebben we geformaliseerd wat vario-schaal vector data inhoudt. In zekere zin kan het verbeterde ontwerp en vullen van de tGAP data structuren gezien worden als een exact omkeerbare (Engels: lossless) codering van de gegevens die worden vastgelegd voor een SSC.

2. Om vario-schaal geo-informatie operationeel te maken is er behoefte aan specifieke generalisatie operaties. Deze operaties moeten zorgvuldig ontworpen worden om de hoeveelheid van de gegevens die worden opgeslagen te beperken en de topologische consistentie te waarborgen.

3. Hoewel de verbeterde tGAP data structuren in staat zijn om een geleidelijke overdracht van gegevens aan een eindgebruiker aan te bieden (van grof naar fijn) worden nog steeds discrete stappen opgeslagen – alhoewel het kleinere en lokalere stappen zijn, dan wat gebruikelijk is voor de huidige state-of-the-art oplossingen (multi-representatie databases, MRDBs). Daarom hebben we voorgesteld hoe de geleidelijkheid van vario-schaal gegevens kan worden verbeterd (met de geleidelijke SSC, waarbij het nemen van een delta in schaal ook leidt tot een delta in het 2D afgeleide kaartbeeld).

   Het bijzondere van deze aanpak is dat, omdat het een geïntegreerde ruimte-schaal partitie is, het gebruik van een niet-horizontale doorsnede ook leidt tot een valide vlakkenpartitie, maar dan met een gemengde kaartschaal (Engels: mixed-scale): dit is nuttig voor gebruik in 3D computer grafiek toepassingen (waarbij ver weg van een waarnemer minder detail nodig is dan dichtbij).

Hoewel dit onderzoek heeft geleid tot nieuwe kennis over een vario-schaal omgeving, zijn er genoeg vragen onbeantwoord gebleven voor verder toekomstig onderzoek. Als belangrijkste punten voor toekomstige onderzoek onderscheiden we:

- Onderzoek hoe om te gaan met zeer grote data sets die niet meer in het hoofdgeheugen van de computer passen (noch tijdens het generalisatie proces, noch tijdens de on-the-fly visualisatie).

- De geleidelijke SSC heeft dezelfde uitdagingen als de klassieke tGAP data structuren met betrekking tot de toepassing van de juiste volgorde van

generalisatie operatoren (samenvoegen, splitsen, lijnversimpeling) om kaarten te verkrijgen met voldoende kartografische kwaliteit.

Een ander punt voor verder onderzoek is de interactie mogelijkheden die de geleidelijkheid brengt: het is van belang om te weten hoe de gebruikers dit ervaren. Hetzelfde geldt voor afbeeldingen met een gemengde kaartschaal (mixed-scale, voor toepassing in een virtuele 3D wereld).

- Nadruk van dit onderzoek heeft vooral op het verkrijgen en bekijken van vario-schaal gegevens gelegen. Het uitvoeren van analyses met vario-schaal gegevens is een ander interressant aspect dat aandacht verdient, *e. g.* vario-schaal gegevens zouden kunnen helpen bij het proces van data-integratie.

- Onderzoek hoe de structuren dynamisch gemaakt kunnen worden: op dit moment is de tGAP structuur (inclusief de nieuwe geleidelijke variant) een statische structuur en moet opnieuw worden gebouwd, indien de brongegevens veranderen. In staat zijn om incrementele wijzigingen uit te voeren, *viz.* het gedeeltelijk kunnen hergeneraliseren van de (gewijzigde) startgegevens voor een nieuwe situatie, is nodig (vanuit praktisch oogpunt) als het volume van de opgeslagen gegevens toeneemt.

  Gerelateerd aan het dynamisch maken is het verhogen van de dimensionaliteit van de geleidelijke, vario-schaal partitie. Geïntegreerd data management (met de volgende geïntegreerd dimensies: ruimte (2D of 3D), tijd (updates, 1D) en schaal (niveau van detail, 1D) zou kunnen worden uitgevoerd op een 5D data model (waarbij gelijktijdig historie bijgehouden wordt).

CURRICULUM VITAE

Martijn Meijers was born on September 24, 1981 in Delft. In 1999 he obtained his high school diploma (in Dutch: gymnasium) from Interconfessionele Scholengemeenschap het Westland (ISW) in Naaldwijk. After this, he studied Geodesy and Geoinformatics at the University of Applied Sciences Utrecht (Hogeschool van Utrecht), where he specialised in Geographic Information Systems and got a Bachelor's degree (ing.) in 2003. He continued his studies at Delft University of Technology, where he obtained his Master's degree (ir.) in Geomatics in 2006. Next, he started to work for Royal Dirkzwager, a maritime information provider in the port of Rotterdam. Here, he worked amongst other ICT related topics, on a web application for tracking seagoing vessels. Since July 2007, Martijn has been a PhD candidate at Delft University of Technology, OTB Research Institute for the Built Environment, department of GIS technology, researching vario-scale geo-information, under the supervision of Prof. dr. ir. P. J. M. (Peter) van Oosterom and Prof. dr. M. J. (Menno-Jan) Kraak. During his PhD he contributed to the successful RGI-233 project 'MobiMaps: usable and well-scaled mobile maps for consumers', which won the Dutch Geo-Innovation Award 2009, category Science. In the summer of 2011 he completed his PhD project and this dissertation. Currently, Martijn is employed as a postdoc at the same group where he conducted his PhD research. He performs research in the STW project 11185 'Vario-scale geo-information' and the NWO VIDI project 11300 'Modelling geographic information in 5D'.