3D Topography

A Simplicial Complex-based Solution in a Spatial DBMS

3D Topography

A Simplicial Complex-based Solution in a Spatial DBMS

Friso Penninga

Publications on Geodesy 66

NCG Nederlandse Commissie voor Geodesie Netherlands Geodetic Commission

Delft, June 2008

3D Topography. A Simplicial Complex-based Solution in a Spatial DBMS
Friso Penninga
Publications on Geodesy 66
ISBN: 978 90 6132 304 4
ISSN 0165 1706

Published by: NCG, Nederlandse Commissie voor Geodesie, Netherlands Geodetic Commission, Delft, The Netherlands
Printed by: Optima Grafische Communicatie, Optima Graphic Communication, Rotterdam, The Netherlands
Cover illustration: Friso Penninga

NCG, Nederlandse Commissie voor Geodesie, Netherlands Geodetic Commission
P.O. Box 5058, 2600 GB Delft, The Netherlands
T: +31 (0)15 278 28 19
F: +31 (0)15 278 17 75
E: info@ncg.knaw.nl
W: www.ncg.knaw.nl

The NCG, Nederlandse Commissie voor Geodesie, Netherlands Geodetic Commission is part of the Royal Netherlands Academy of Arts and Sciences (KNAW).

Acknowledgements

Each PhD project has its own particular history. My story starts on a Thursday night, late in 1999, the evening of the 'oliebollenbar'*. At that time, I was still a Geodesy student and, together with many other students, I paid a visit to the bar. During that year, staff members from each research section were asked to act as bartenders to stimulate the interrelationships between students and staff members. The GISt staff members were not only acting as bartenders, but also decided –in advance of the approaching holiday season– to bake some 'oliebollen'. A few weeks earlier it was announced that Peter van Oosterom would become the new professor of GIS Technology early in 2000. Early in the evening, he arrived to see his future group in action. It was the first night that I spoke to him, discussing his ideas for the group. He was wearing a chef's hat (the one from Tjeu, who was baking the 'oliebollen'), holding a bottle of beer in his hand and appeared to be a no-nonsense guy. Later that night I decided to choose GIS as my MSc specialization.

Eight years have past since that particular evening in 1999. In the meantime I obtained my MSc under Peter's supervision, and he played an important role in my search for a job afterwards. While pursuing another job, we agreed on a temporal position as a researcher, early in 2004. The three month position would cover the period that my future employer needed to fix a budget problem. However, life is what happens to you while you're busy making other plans, so after six weeks Peter offered me a PhD position that I accepted. That's how things started.

Today I am at the end of this project. Although only my name is printed on the cover of this book, I could not have achieved this result without the indispensable support of many people. First of all I have to thank Peter van Oosterom. Without his enthusiasm, open communication and down-to-earth attitude this project could not have been successful. Furthermore I like to thank the 2004 GIS-t group for welcoming me (Axel, Edward, Elfriede, Henri, Marian, Sisi, Theo, Tjeu, Wilko), with special thanks to Edward for his involvement in the start of the project, all staff members

*An 'oliebol' is a typical Dutch treat, traditionally eaten on New Year's Eve, and can be described roughly as a doughnut ball. The 'oliebollenbar' was a theme night at the Snelliusbar (the bar of the students association Snellius of the faculty of Geodesy, Delft University of Technology), organised by staff members of the GIS Technology group.

that joined the group since (Arta, Frank, Hugo, Maarten, Martijn, Safiza, Swati, Wei, Wiebke, Yahaya) and our guest researchers (Chen, João, Ludvig, Rod). I really enjoy working with this young, dynamic and enthusiastic group of researchers.

Other OTB colleagues also contributed to the pleasant atmosphere in which I have worked for four years. Thanks to the 12 o'clock lunch crew, the 20 year OTB committee, the PhD council and board (I enjoyed both the work and the drinks) and all other colleagues that I met during joint projects, off-sites, OTBorrels and coffee breaks. Thanks to Itziar Lasa for the cover design of the dissertation version.

A lot of people outside OTB also contributed in one way or the other. The week Peter and I spent at Oracle USA was invaluable (thanks to Siva Ravada, Ravi Kothuri, Baris Kazar, John Herring, Han Wammes), even though the annual 'A Quilters Gathering' created the most surreal atmosphere in the Nashua Sheraton. Discussing spatial data types during the day and being surrounded by hundreds of quilters with humongous trolleys with sewing kits and machines by night; it's quite a gap. Furthermore I like to thank the 3D Topography consortium partners (Garmt Zuidema, George Vosselman, Han Wammes, Hans Nobbe, Marc van der Eerden, Nico Bakker, Sander Oude Elberink, Stefan Flos), the 3D Topography use case interviewees (Bram Verbruggen, Hans Nobbe, Irwin van Hunen, Nico Bakker, Paul van Asperen, Stefan Flos) and the 3D Topography international top-up partners (Aiden Slingsby, Andrew Frank, Chris Gold, Farid Karimpour, Hang Si, Jonathan Raper, Klaus Gärtner, Ludvig Emgård, Pawel Boguslawski, Rod Thompson). A special word of gratitude goes to Hang Si for his TetGen software, which was used for all tetrahedronisations in my research.

Special thanks to Dave Houben for his work on the DUT campus data set and Sijmen Wesselingh for his work on a web-based viewer for tetrahedronised data. Furthermore I have to acknowledge Rien Elling; without his course the writing process would have been much more chaotic and less on schedule. John Herring, Hugo Ledoux, Hang Si and Edward Verbree provided me with very valuable and highly appreciated suggestions for further improvements of the final dissertation text.

Finally a big thank you to all my friends and family, for providing a life outside university. A last word of gratitude goes to Brechtje. I started this project without you, but at the end it is so obvious that your love is more important than a PhD.

Friso Penninga
February 2008

Contents

Chapter 1

Introduction

More than 350 years ago, the famous Dutch cartographer Willem Blaeu (1571-1638) created a beautiful map of the city of Vlissingen (see figure 1.1). In his map, every house, church and windmill is depicted as a volume with a side-view, which increases the map's readability. This early 3D mapping method fits with the human perception of the world, which is often based on an oblique view. In the following centuries the orthogonal projection became the new standard and the resulting maps describe the world in only two dimensions.

With the introduction of geographic information systems (GIS), these computerised 2D maps served no longer only as visualisation tool, but as basis for storage, calculations and analysis as well. In the last decade several steps towards 3D GIS

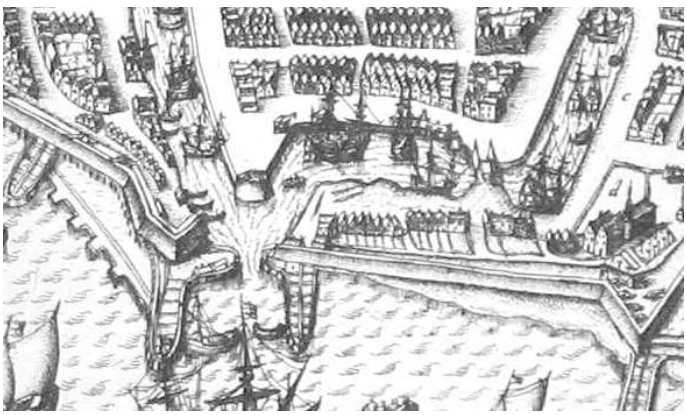


Figure 1.1: *Detail of the map of Vlissingen with a bird's eye perspective, by Willem Blaeu, 1612*

have been taken, especially from a visualisation point of view. This dissertation is intended as the next step towards 3D GIS, as it will result in a new data structure that supports 3D storage, 3D analyses and 3D validation.

Based on the research motivation, section 1.1 elaborates on the need for a conceptual data model and an accompanying data structure for 3D topography. From this motivation, the research objective and main research question are derived in section 1.2, while section 1.3 defines the research scope and limitations. Section 1.4 summarises the contribution of this research to the field of 3D GIS and section 1.5 provides the outline of this dissertation.

1.1 Motivation

The dissertation title '3D Topography' implies two important aspects of this research. First of all, it is a particularisation of 3D GIS. A geographic information system (GIS) can be defined as 'a computer-based information system that enables capture, modelling, storage, retrieval, sharing, manipulation, analysis, and presentation of geographically referenced data' (Worboys and Duckham 2004). Second of all, in this case the 'geographically referenced data' dealt with, is topographic data. A suitable definition of topography in the context of this research is 'the configuration of a surface and the relations among its man-made and natural features' (Wordnet 2007). Since a GIS is often used as a decision support system (Cowen 1988), 3D Topography is about modelling, storing and analysing more realistic 3D data to support decisions that concern our daily environment, like designing large infrastructural projects, sustainable urban planning and applications in the field of safety and security (Kwan and Lee 2005). As a result, the potential impact on society of 3D topography is large. To fulfil this potential, both an appropriate conceptual data model and an accompanying data structure are required. This research will deliver both.

1.1.1 A conceptual data model for 3D Topography

Most current topographic products are limited to representing the real world in only two dimensions. As the real world exists of three dimensional objects that are becoming more and more complex due to increasing multiple land use, accurate topographic models have to cope with the third dimension. Several true 3D applications can be recognised for these accurate models. One can think of volume computations, e.g. for real estate tax applications or excavations, line-of-sight analysis for mobile phone antenna networks and accurate modelling of noise propagation and air pollution (see figure 1.2 for an example in which the effects in between buildings are included). Applications of 3D modelling are not limited to the earth surface, as geological features or airplane and communication corridors can be modelled as well. As a last application simulation of disasters like floodings or earthquakes can be mentioned.

The number of future applications is vast. To further illustrate this, a more comprehensive description of the problem field and future applications of 3D Topography can be found in section 2.1. However, none of these applications will be supported

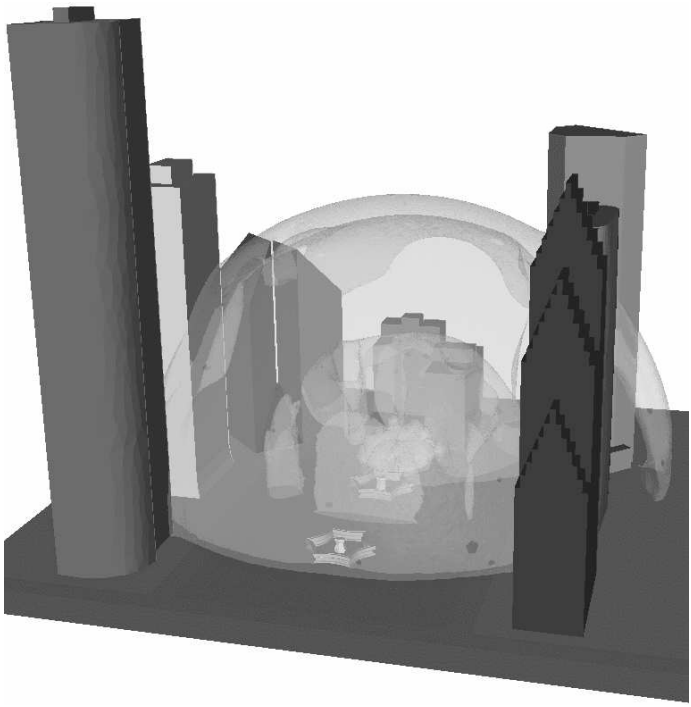


Figure 1.2: *3D blast analysis in an urban area*

without an appropriate data model for 3D topography. This research will provide such a data model in order to facilitate these desirable applications.

1.1.2 A data structure for 3D Topography

Developing a suitable data modelling approach is an important first step, but will remain useless without an appropriate data structure to support the required functionality. The data structure defines in which way the data is stored and organised in the database. Aspects like data storage requirements, query response time, data consistency and the availability of tools for editing and validation will determine the overall success of such a data structure. The data structure needs a 3D primitive (a volume) besides points, lines and faces to represent 3D objects accurately. Even though 3D coordinates can be used in some spatial databases, 3D data types were missing during the major part of the research (the new 3D data types in Oracle 11g were released in 2007 (Murray 2007)). Therefore defining a new 3D data type is part of the research.

1.2 Objective and main research question

The objective of the research is to develop a data structure that is capable of handling large data volumes and offers support for loading, updating, querying, analysing and especially validating 3D topographic data. To achieve this, a triangular (in general dimension) approach will be used, due to its advantages in maintaining consistency, its robustness and editability. A two-step approach will be adopted. First one has to decide how real-world objects should be modelled into features, secondly one needs to store these features in such a way that the requirements in terms of querying, analysis and validation are met. An obvious step in dealing with large volumes of geographically referenced data, is to use a spatial database.

This objective is expressed in the main research question:

How can a 3D topographic representation be realised in a feature-based triangular data model?

Several aspects in this question deserve further explanation:

- A *3D topographic representation* is a model of both man-made and natural features. The addition '3D' indicates that the model allows multiple 3D coordinates at a specific x,y location, for instance in vertical faces (see section 2.2 for formal definitions of model dimensions). Furthermore the model is not limited to the earth surface; subsurface features like tunnels, basements and maybe even geological layers, as well as aerial features like airplane corridors may be part of the model explicitly. Within this research '3D' will also indicate that a volume representation will be used to model the topographic features.

- The verb *realise* covers the process of specifying and developing a modelling approach. A proof-of-concept implementation in a spatial DBMS (database management system) is included as well.

- A *feature-based* data model is a data model that –independent of the actual storage structure– consists of features: 'abstractions of real world phenomena' (ISO 19101:2002 2002). These real world phenomena are the objects that the average user will recognise: the buildings, roads, hills, tunnels, viaducts, etc. Update operations will be executed by the user at feature level.

- The term *triangular* should be read as triangular in general dimension (thus including tetrahedrons as 3D triangular building block, see figure 1.3 for an example of a tetrahedron). A triangular data model is a data model that relies on triangles or tetrahedrons for its internal structure and storage. These triangles or tetrahedrons will act as the fundamental building blocks by which the model is constructed. In other words: all topographic features will be described as

Figure 1.3: *A tetrahedron shape is a rarely used in daily life. One exception is this tetrahedron-shaped milk carton from Iran (courtesy of Salomon Kroonenberg, also used in Kroonenberg (2006))*

sets of triangles or tetrahedrons and these features will be connected by triangles or tetrahedrons as well. An average user is not necessarily aware of the under-water existence of these building blocks. In this research, both Triangular Irregular Networks (TINs) and Tetrahedronised irregular Networks (TENs) will be considered as triangular data structure. These structures are selected due to computational advantages, the flatness of the faces (well defined by three points) and the presence of well-known topological relationships (Guibas and Stolfi 1985).

- a *data model* includes both a conceptual data model and a DBMS data structure, as introduced in the previous section.

As stated earlier in this section, a two-step approach will be adopted to achieve a solution to the main research question. In accordance with the two steps, two key questions can be distinguished:

- How to develop a conceptual model that describes the real world phenomena (the topographic features), regarding the general purpose-characteristic of topographic data sets? This will be the central question of Part I of this dissertation.

- How to implement this conceptual model, i.e. how to develop a suitable DBMS data structure? This will be the central question of Part II of this dissertation.

1.3 Research scope and limitations

In order to define the scope of this project as clear as possible, the research is limited in several ways. The following topics are explicitly included:

- Topographic data at scale level 1:500 – 1:25,000 with related resolution and accuracy. As a result, large and midscale topographic data, sets such as large scale base maps, can be subject of research.
- Subsurface features like tunnels, basements and parking garages.
- Both initial model creation as well as incremental updates of the model.

The following topics are explicitly excluded:

- Temporal aspects of modelling topographic features
- Vario-scale and vario-representation/generalisation of topographic features
- Dynamic models (i.e. modelling of moving objects)
- Continuous (field) representations (like for instance oceanographic and atmosphere phenomena)
- Gridded/raster approaches
- Indoor topography, despite its substantial potential in the field of disaster management
- Optimisation in the field of (realistic) visualisation and/or virtual reality. In short the main focus of the model is on enabling computations, analyses and validation (data management) and not on realistic visualisations, texture mapping, etc.
- Data collection and creation of models from this data. As a result, models of volumetric topographic features are supposed to be available.

1.4 Contribution of the work

In retrospect, this work contributes to the general research field of 3D GIS in the sense that the new 3D data modelling approach will reduce data storage of tetrahedral data structures and will eliminate the need for explicit updates of topological relationships and most parts of the triangular data structure. These results will be achieved by specifying a conceptual data model and accompanying data structure, such that operators and definitions from simplicial homology (see chapter 4), part of

the mathematical description of topology, can be applied. By doing so, the approach will tackle common drawbacks as tetrahedronised irregular network (TEN, the 3D triangular data structure) extensiveness and laboriousness of maintaining topology. Furthermore, applying operators and definitions from simplicial homology will offer full control over orientation of all TEN elements, which is a significant advantage, especially in 3D. In addition to this aspect, the mathematical theory of simplicial homology will offer a solid theoretical foundation for both the data structure and data operations. Integrating these concepts with database functionality will result in a new innovative approach to 3D data modelling.

The research described in this dissertation is one of the main components of the Bsik Space for Geo-information 3D Topography research project. The overall objective of this research project is to enforce a major break-through in the application of 3D Topography in corporate ICT environments, due to structural embedding of 3D methods and techniques (3D Topography 2006). The project consortium consists –besides Delft University of Technology– of ITC Enschede, Topografische Dienst Kadaster, Rijkswaterstaat, Oracle, NedGraphics and Steering Committee AHN, thus grouping 3D researchers, 3D data producers and 3D software developers.

1.5 Outline

In order to answer the question how a 3D topographic terrain representation could be realised in a feature-based triangular data model, a two-step approach is used. The two accompanying key questions are previously introduced. This two-step approach is also visible in the dissertation structure in figure 1.4. First the problem domain of 3D Topography is introduced in chapter 2. Based on its predicted applications, requirements for both data model and data structure can be derived. This chapter will also provide an overview of relevant 3D data structures and elaborates on triangular structures and algorithms. The model requirements act as input in the development of two conceptual data models in chapter 3 and the most appropriate one is selected.

The next part (chapters 4-6) derives an accompanying data structure. This data structure applies definitions and operators from simplicial homology, which will be introduced in chapter 4. Based on this theory, chapter 5 presents the new database structure for 3D topography. Since update capabilities are an important requirement for a feasible 3D topography data structure, chapter 6 will focus on edit operations in the simplicial complex-based data structure.

The last part evaluates and discusses the new approach. Chapter 7 shows results, based on tests with initial 3D data sets and this dissertation ends with conclusions and suggestions for future research in chapter 8. Details on the proof-of-concept implementation can be found in the appendix.

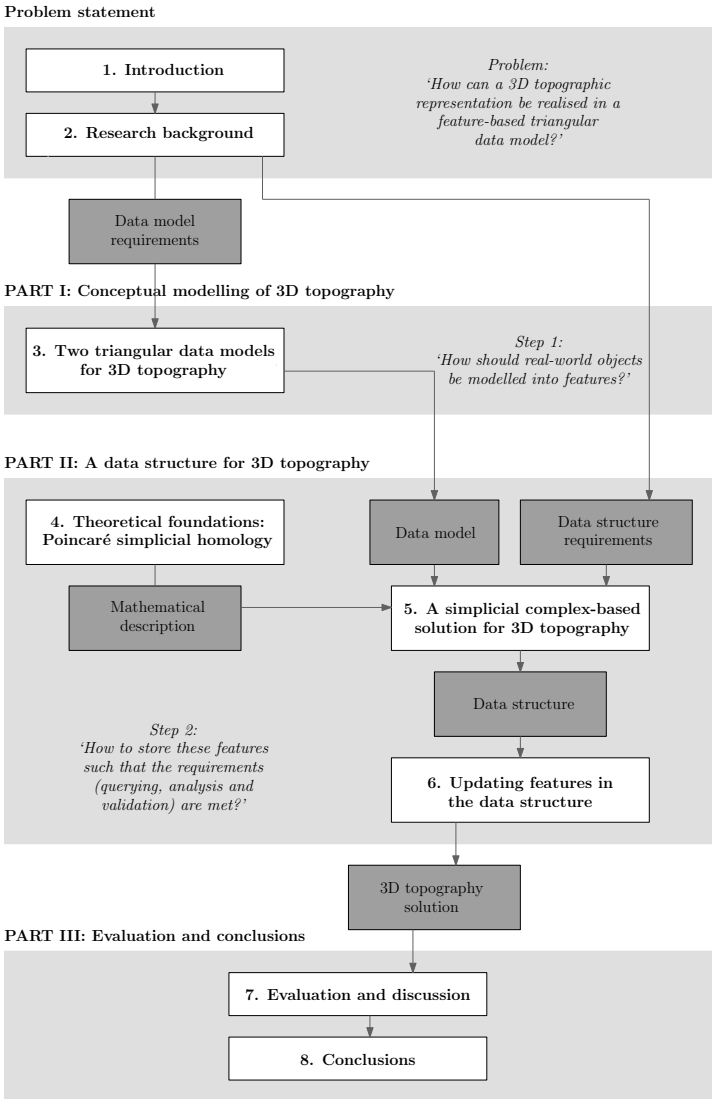


Figure 1.4: *Outline of the dissertation*

Chapter 2

Research background

As stated in the introduction, the development towards 3D topography has a significant potential impact on society, since 3D topography is about acquiring, modelling, storing and analysing data to support decisions that concern our daily environment. 3D topography might enable sustainable urban planning and disaster simulation for emergency response training. To guide this development, a clear insight in the requirements for 3D topography is essential. In order to be able to derive these requirements, one needs to understand the factors triggering the development of 3D topography. Section 2.1 will provide insight in these relevant factors. Although concepts like 2D and 3D seem rather straightforward and unambiguous, the opposite is true. Section 2.2 elaborates on dimension definitions in the range 2D-3D to provide more insight in the differences between the various types.

To achieve realisation of the intended applications of 3D topography, both an appropriate conceptual data model and an accompanying DBMS data structure are required. This problem subdivision will be applied from the requirements section (section 2.3) onwards. Especially on 3D data structures, relevant research is performed by others. Section 2.4 will provide an overview of previously proposed approaches to 3D data storage. Since this research will elaborate on a specific type of approach –the triangular data structures– the current state of research (in the closely related field of computational geometry) on algorithms to compute these triangular networks will be presented in section 2.5.

The backgrounds provided in this chapter will function as a basis for the conceptual modelling of 3D topography (part I of this dissertation), as well as for developing a data structure for 3D topography (part II of this dissertation).

2.1 Problem domain: Towards 3D topography

Current topographic products are limited to a real world representation in only two dimensions, with at best some additional point heights and contour lines. The devel-

opments towards 3D topography are both demand- and supply-driven. This section shows both limitations of current 2D data sets and expected future 3D applications, causing for the demand of 3D topography; as well as developments in the field of sensor techniques, causing increasing availability of 3D topographic data.

2.1.1 Demand-driven development: the need for 3D topography

Modelling the real world in two dimensions implies a rather drastic simplification of three dimensional real world elements. By representing these elements in two dimensions, loss of information is inevitable. Due to this simplification, accuracy of analysis results is limited and a meaningful, insightful representation of complex situations is hard to obtain. In a time with increasing attention for environmental and sustainability issues, these limitations become real problematic and trigger the need for 3D topography.

Environmental issues like high concentrations of particulates along highways in urban areas (Borst 2001), the effects of noise (Rasmussen 1998, Stoter *et al.* 2008) and odour propagation (Winther *et al.* 2006) and risk analysis of liquefied petroleum gas storage tanks (El-Harbawi *et al.* 2004) are examples of current issues in which more sophisticated analyses are required than 2D models can offer. For instance particulate matter distribution is substantially influenced by the presence of high buildings, as these buildings may act as a shield between the pollution source and urban areas behind. However, one might wonder whether these rather rare high-end applications justify the development of 3D topography, including 3D data collection, modelling and storage. Before answering, one has to question oneself whether this is a valid question. In other words, more insight in future applications is required. To provide this insight a tentative study has been performed (Nobbe *et al.* 2006) within the 3D topography project consortium (see section 1.4). The use-cases from this study show a wide range of possible applications and, compared to the previously introduced high-end analyses, most of them are relatively simple. An objective like 'gaining insight in complex situations' turned out to be one of the most important applications. Due to an increase in multiple land use, 2D models are not capable of representing vertically separated features adequately. One can think of features like viaducts, tunnels, buildings on top of highways and buildings intersected by (rail)roads. Another future application is automatic change detection, as comparing the 3D volumes turned out to be useful for real estate tax purposes and enforcement of building construction permit policy. An often required analysis is the line-of-sight operation, both for urban planners ('what does one see from a specific point; is the space perceived as confined or as open?') and telecom operators ('is there a clear line of sight between these two antennas?').

A different group of applications has to do with 3D topography as basis for a virtual model. As a low-end application, one can think of 3D models for car navigation purposes (van Essen 2008), whereas in the field of (serious) gaming, high-end applications like virtual reality applications for training and simulation are being developed (Center for Advanced Gaming and Simulation (AGS) 2007). Ongoing training is essential, especially for emergency response units, as automatisms and smooth coop-

eration might save lives. However, real life training facilities are limited due to budget or organisational limitations. Large scale exercises in public places have a significant impact on daily life. Therefore, training in virtual environments enables an increase in the number of drills, as well as useful training evaluation features like rewinding crucial events or analysing trainings from a birds eye perspective to show individual actions within the overall situation. Figure 2.1 shows an example of a fire fighting drill in a virtual forest, as described by Rossmann and Bücken (2008).

Figure 2.1: *Disaster simulation in the Virtual Forest (Rossmann and Bücken 2008, Figure 8.9)*

2.1.2 Supply-driven development: the increasing availability of 3D data

The current developments in the field of 3D Topography are not only demand-driven. The increasing availability of high density laser scan data is most certainly a trigger in this process. Due to this new technique height data becomes available with point densities that were previously unthinkable with traditional photogrammetric stereo techniques. Integrating 2D data with height data sets is an obvious objective when both data sets are available. It started in the Netherlands with the introduction of the AHN (in Dutch: Actueel Hoogtebestand Nederland), a height data set of the Netherlands obtained by laser altimetry with a density of at least one point per 16 square meters and in forests at least one point per 36 square meters (Heerd *et al.* 2000). The final processed AHN contains only earth surface points; information such

as houses, cars and vegetation has been filtered out. However, by using the unfiltered data, combining these height data with two dimensional topographic data sets became possible. Since the introduction of the AHN point density increased rapidly; datasets with multiple points per square meter are not unusual anymore. From 2008 onwards, the point density of the AHN-2 will be increased to at least 10 points per square meter (Coumans 2007). Simultaneously to the process of increasing point density, the integration process of planar with height data was further automated. Oude Elberink and Vosselman (2006) describe a fully automated integration of 2D data with height data in a topographic context (see figure 2.2 for an example of input data and the result).

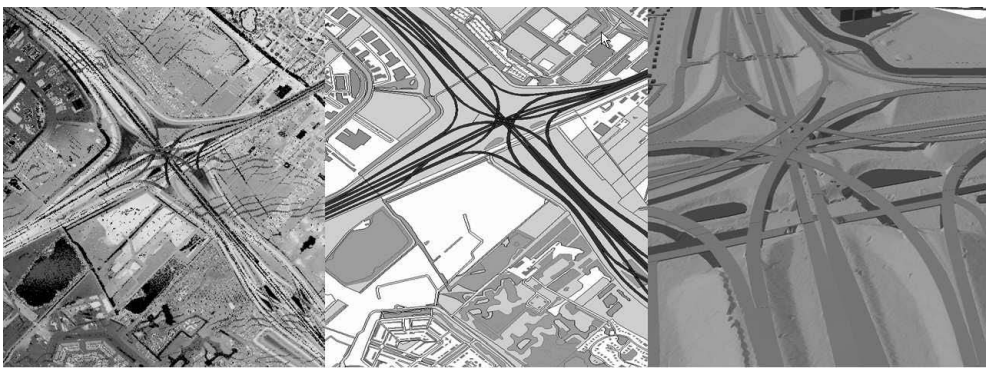


Figure 2.2: *Integrating height data (left) and topographic data (middle) results in a 3D model (right) of highway interchange 'Prins Clausplein' near The Hague, the Netherlands (Courtesy of Sander Oude Elberink, ITC Enschede)*

Besides integrating 2D data with height data (obtained by airborne laser scanning), direct 3D data acquisition by terrestrial laser scanning is emerging (see figure 2.3). As a result much more details become available, as complete facades can be measured. Depending on the required level-of-detail, terrestrial laser scanning can provide measurements varying from top and bottom of facade faces to detailed data on windows, windowsills and facade ornaments. Another important source of facade information (which cannot be acquired from traditional airborne techniques as photogrammetry and airborne laser scanning) is measuring in images or videos (Beers 1995, Verbree *et al.* 2004). Still applicability of terrestrial laserscanning is not limited to the traditional topographic features, as it also enables data acquisition of subsurface features like tunnels and even indoor topography.

A last important factor influencing the availability of 3D data, is the data acquisition for navigation purposes. Although the process in itself is more demand-driven, the resulting data and data acquisition techniques lead to the increasing availability of 3D data. To derive more accurate and recognisable maps for navigation systems, data suppliers are switching to 2.5D and 3D models. van Essen (2008) describes the data acquisition by TeleAtlas, one of the largest map data suppliers for personal nav-

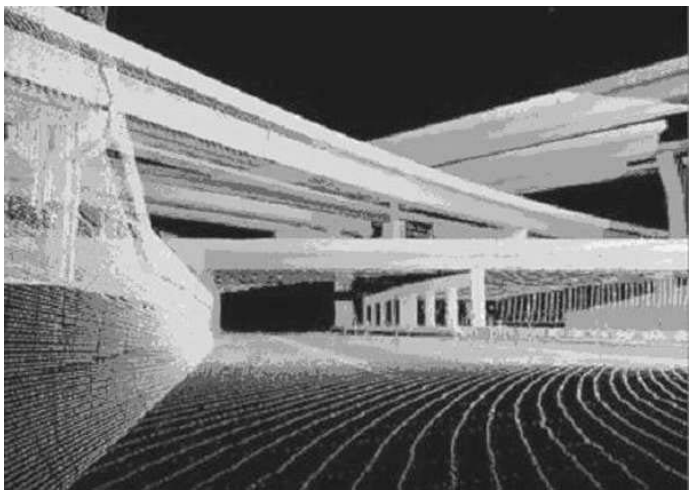


Figure 2.3: *Terrestrial laser scanning acquires 3D data of complex objects*

igation systems. Their 50 acquisition vehicles, equipped with integrated positioning and (stereo) imaging systems, collect large amounts of 3D data, which is the basis for the creation of 3D city maps.

2.2 Defining dimensions in the range 2D-3D

In the previous section terms such as '3D topography' and 'current 2D datasets' were used without proper definitions of 2D and 3D. Defining the dimension of a model is not as straightforward as one might expect. Although in day-to-day speak one discusses the dimension of a model, two different types of dimension need to be distinguished, i.e. the internal and the external dimension (Pilouk 1996). The internal dimension indicates the (highest) dimension of the primitives that are being used to describe objects. For instance, if one describes a building by a set of flat faces, the internal dimension is two, whereas the internal dimension will be three if one describes this building by a volume. The external dimension is the dimension of space in which one models. In mathematics, the term 'codimension' is often used to indicate a difference between the internal and external dimension (as used by Pilouk (1996)). If a mathematical object (in this case the primitive) is associated to another object of dimension n (in this case the space in which one models), then it is said to have codimension k if it has dimension n - k. So, the primitives in the previous three-dimensional model are said to be of codimension one.

Although concepts like internal and external dimension or dimension and codimension are available to describe a model accurately, usually one still tries to define 'the' dimension of a model. Often, this results in a classification of a model that uses

2D primitives in 3D space as a three-dimensional model. However, this classification does not acknowledge the difference with a 3D model in which 3D primitives are used in 3D space. To overcome this drawback, Pilouk (1996) uses both internal and external dimension in his definitions of model dimension:

- 2D model: modelling with 2D primitives in 2D space (in mathematical terms: dimension 2, codimension 0)
- 2.5D model: modelling with 2D primitives in 3D space (in mathematical terms: dimension 3, codimension 1)
- 3D model: modelling with 3D primitives in 3D space (in mathematical terms: dimension 3, codimension 0)

Despite his attempt, still more model dimensions can be distinguished. The following types are used in this dissertation:

- A *2D model* consists of primitives of dimension two or lower (i.e. points, lines and polygons) in 2D space. Figure 2.4 shows an example of a 2D parcel map.

Figure 2.4: *A 2D cadastral map: points, lines and polygons are used to model parcels in 2D space*

- A *2.5D model* consists of primitives of dimension two or lower in 3D space, with the requirement that at each x,y-location only a single height value can be present. This criterion applies often to terrain models. As a result vertical faces and overhangs are not allowed. Sometimes these models are classified as 'strict 2.5D', but in this dissertation 2.5D will be used. Figure 2.5 shows an example of the same parcel map as depicted in figure 2.4, but this time terrain elevation is included in the objects. Triangulated Irregular Networks (TINs, see section 2.5 for more details) are often applied in elevation models and usually meet the criterion of a single height value at a x,y-location.

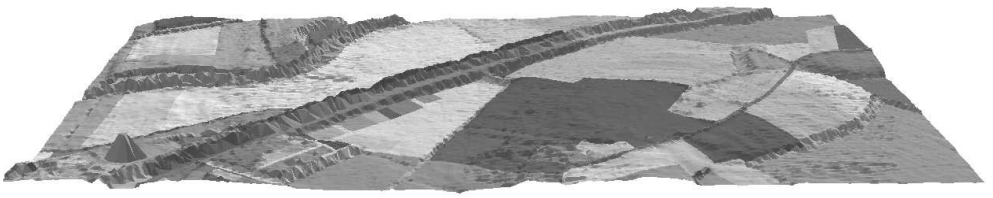


Figure 2.5: *A 2.5D cadastral map: points, lines and polygons are used to model parcels and terrain elevation in 3D space*

- A *multiple 2.5D model* uses a layer approach with several 2.5D models. Figure 2.6 shows an example. Layer 1 contains the terrain, layer 2 contains the viaduct. By combining the layers one can represent 3D situations without using 3D models. This approach is used by Simonse *et al.* (2000) in an attempt to create a '3D' topographic data set.

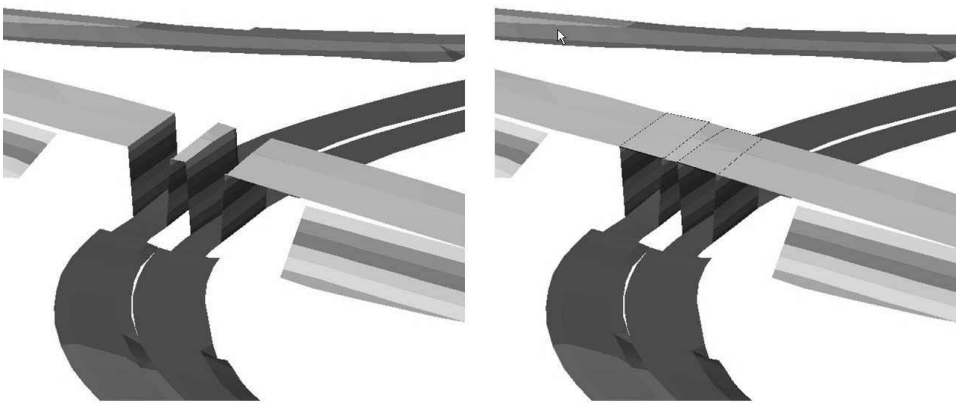


Figure 2.6: *A multiple 2.5D model: two 2.5D layers are used. Left the terrain model (layer 1), right both the terrain model and the viaduct model (layer 1 + 2)*

- A *2.5D+ model* consists of primitives of dimension two or lower in 3D space, in which vertical faces are allowed. In TINs these vertical faces are usually bounded by stop lines and computed additionally to the triangulation. A simplified example is shown in figure 2.7 in which the grey faces are vertical. As most triangulations are computed in 2D (i.e. the projection of the 2.5D situation on the x,y-plane), these vertical faces are not part of the triangulation. The GM_Tin data type as defined in the ISO 19107:2003(E) (2003) standard is an example of a 2.5D+ TIN, as it allows the inclusion of stop lines which mark local discontinuities in the triangulated surface.

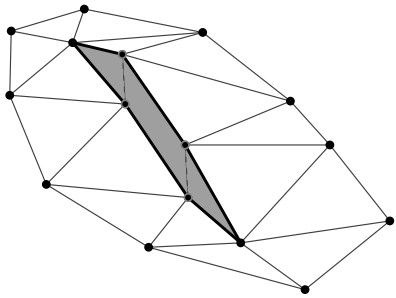


Figure 2.7: *A simplified 2.5D+ model: points, lines and polygons are used to model terrain elevation in 3D space, but in contrast to the 2.5D model this model can incorporate vertical faces*

- A *2.75D model* consists of primitives of dimension two or lower in 3D space, but without any constraints on vertical faces, overhangs or holes. Section 2.4.3 will elaborate on this approach by Tse and Gold (2004), of which figure 2.11 shows an example.

- A *3D model* consists of primitives of dimension three or lower in 3D space. A 3D model of a building and its surroundings is depicted in figure 2.8.

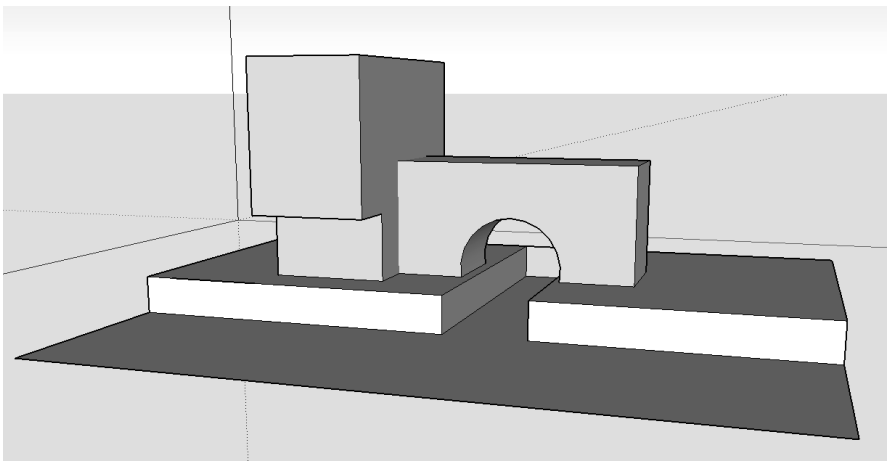


Figure 2.8: *A 3D model: points, lines, polygons and volumes are used to model a building in 3D space*

2.3 Deriving requirements for the conceptual data model and structure from the problem

2.3.1 Requirements for the conceptual data model

By combining the future 3D topography applications with current widespread usage of 2D topographic data sets, it is safe to say that topographic datasets serve a wide range of applications. Topographic data serves as a basis for calculations and analysis, but also as reference theme for many applications. Not only the type of operations varies, but there are also different domains using topographic data, each with their own interests in specific groups of objects. For instance, water boards put emphasis on other types of objects than urban planners. As a result, different opinions on the 'best' conceptual modelling approach co-exist, and a topographic survey has to take up the challenge to satisfy all domains. With respect to designing a conceptual data model for 3D topography, it is now clear that the conceptual data model should support a wide variety of tasks, thus disabling the possibility of optimising it for specific purposes.

Serving a wide variety of tasks and abandoning task-specific optimisation are somewhat difficult design criteria for a conceptual data model. More specific requirements and their relative weights are needed. Since each domain needs its own information, a topographic data set can only serve the needs of different domains when as much information as possible is available. Generally speaking the topographic data model should be as rich (in terms of information) as possible, as this will enable deriving multiple subsets for domain-specific tasks. However, with an increasing amount of content, keeping the data up-to-date will become more difficult and expensive. One has to find the delicate balance between information richness and costs in terms of acquisition, updating and storage. A possible strategy might be to include data based on the number of applications that benefit from this data, thus maximising usability while minimising the required efforts. At the same time, one should try to keep the model as close to reality as possible, as this will leave open as many options for different representations as possible. These criteria will be adopted in the next chapter, as two modelling approaches will be compared.

2.3.2 Requirements for the data structure

In order to be able to derive requirements for the data structure, two important characteristics of 3D topographic data sets need to be acknowledged:

- Switching from 2D to 3D data representation causes a substantial increase in *data volume*. Even in the simplest cases the increase is larger than what the non-expert might expect. For instance, consider a cube-shaped building. In a 2D map, this building will be represented by a single polygon, whereas the 3D representation already consists of six (four walls, a roof and a floor) polygons. In real 3D data sets this increase will be larger, as more details will be captured, like roof shapes and more complex building designs. To complicate matters

further, one will integrate terrain heights. Due to the increasing point density of laser scan data, data volumes will further increase.

- Topographic data sets *need to be updated on a regular basis*. After all, our daily environment is subject to continuous change, as old buildings are being demolished and new ones being build, new infrastructure created and nature reserves extended. Obviously, these new features have to be inserted into the model correctly. A new building, for instance, should be placed exactly on top of the existing terrain surface, even in case the new measurements would cause the building to float slightly above terrain level. Adjusting objects according to constraints (Louwsma *et al.* 2006) is therefore required. Due to the expected data volumes, updating should be possible incrementally.

Based on these two characteristics, specific requirements for the data structure can be derived. First of all, overall performance (in terms of data storage requirements and response time) should be acceptable with massive data sets being managed, i.e. in the same order of magnitude as other approaches in 2.5D or 3D. Secondly, the data structure should guarantee data consistency. More precisely, the data structure should enable validation. A third requirement is that the structure should support computational and analytical operations. As a fourth and last requirement it should be possible to update the data structure, i.e. features can be added, removed or altered. Regarding the expected data volume, it is required that the data structure allows incremental updates, as complete rebuilds will be too time-consuming.

2.4 Managing 3D data: related research on 3D data structures

Worboys and Duckham (2004, chapter 5) put the importance of data representations aptly as they state 'The manner in which spatial data is represented in an information system is key to the efficiency of the computational processes that will act upon it'. Their chapter on representations and algorithms is recommended for those interested in an introductionary exploration of different types of data structures. Research in the field of 3D GIS is performed for the last 25 years. Zlatanova *et al.* (2002) give an overview of the most relevant developments in this period and Zlatanova *et al.* (2004) elaborate especially on the topological ones. The focus of this section is limited to the relevant types of 3D data structures: constructive solid geometry, boundary representations, simplex-based approaches and regular polytopes. Voxels and other grid-based representations are outside the scope since this dissertation focuses on vector representations only.

2.4.1 Constructive Solid Geometry

Constructive Solid Geometry (CSG) is a technique to model complex objects by using Boolean operators on (usually) simple primitives, like cuboids, cylinders and spheres. Figure 2.9 illustrates this approach: an apparent complex object can be constructed

from five basic primitives by applying intersection, union and difference operators. Although this construction is a nice characteristic, the real advantage of CSG is that

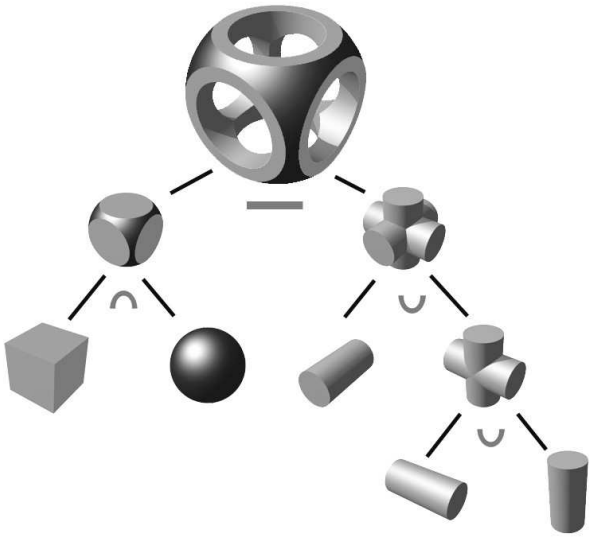


Figure 2.9: *CSG-tree: Modelling a complex shape by simple primitives (cuboid, sphere and three cylinders) and the Boolean operators intersection ∩, difference - and union ∪(source: Wikipedia, article on Constructive Solid Geometry)*

the basic primitives can be parameterised (thus scaling, translating and rotating the primitives) and therefore require little storage space. For instance, a cylinder can be parameterised by its length and diameter, located by a translation and oriented by rotations. As a result, the curved surface is not explicitly stored. Initially (Requieha and Voelcker 1980a,b, 1983), CSG was intended for design purposes. Later CSG was also applied to modelling existing polyhedral objects and, in more recent years, for instance to modelling complex industrial installations using terrestrial photogrammetry (Tangelder *et al.* 2003). CSG is very useful for this application, as object reconstruction breaks down to optimising the parameters to find a best fit through a set of measurements (often a point cloud).

2.4.2 3D boundary representation: polyhedrons

While in CSG the object boundaries are usually not described explicitly by coordinates, this is the case in most other 3D data structures. The most common approach is the polyhedron approach, in which solids are described by their boundary. These boundaries consist of polygonal faces and should form a closed, watertight volume, as illustrated in figure 2.10. Arens *et al.* (2005) show a prototype DBMS implementation

of a polyhedron, including many validation functions. With the recent (2007) launch of Oracle 11, a polyhedron data type became available within Oracle Spatial (Murray 2007). In applications polyhedrons are often used as 3D primitive (Zlatanova 2000, Stoter 2004). Polyhedrons also occur frequently in topological approaches, albeit that most approaches have implicit topology (no explicit storage of relationships). A well-known example of an implicit topological approach is the 3D Formal Data Structure (FDS) (Molenaar 1990a,b, 1992), that consists of points, lines, surfaces and bodies, whereas the Postgres-based implementation described by van Oosterom *et al.* (1994) is an example of an explicit topological approach. Topological approaches are favourable since in 3D data volumes substantially increase, so maintaining and ensuring data integrity becomes of extreme importance (Ellul and Haklay 2006). Validity checks based on these topological relationships can guarantee that the data set maintains valid during edit operations. Kazar *et al.* (2008) illustrate the large number of cases that complicate validation significantly.

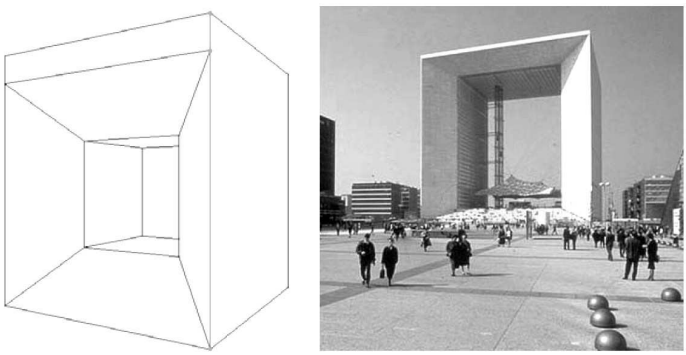


Figure 2.10: *A polyhedron is defined by its bounding faces (Arens 2003, Fig.7)*

2.4.3 Simplex-based approaches

Compared to the topological approaches with polyhedrons, simplex-based approaches take the mathematical approach one step further. A simplex can loosely be defined (see section 4.1 for a formal definition) as the simplest shape in a dimension, in which 'simplest' refers to minimising the number of points required to define such a shape. In other words, simplex-based approaches describe the world with points, line segments, triangles and tetrahedrons. Such a network of simplexes is an example of an irregular tessellation, in which the model of the real world will be decomposed into smaller building blocks. Simplex-based approaches are popular due to computational advantages, the flatness of the faces (well defined by three points) and the presence of well-known topological relationships (Guibas and Stolfi 1985). Section 2.5 will elaborate on both 2D (triangulations) and 3D (tetrahedronisations) simplex-based models, while chapter 4 will introduce a proper mathematical description of simplex-based

models. With respect to simplex-based modelling, Carlson (1987) can be seen as the starting point as he applied it to 3D subsurface structures. However, he limited himself for reasons of simplicity to the use of 0-, 1- and 2-simplexes in 3D space. Nevertheless, he acknowledged the possibility of extending the simplex approach into n dimensions (as indicated by Frank and Kuhn (1986)). The possibility of including 3D simplexes is explored by Pigot (1992, 1995), who focussed mainly on the underlying topological model. Pilouk (1996) introduces the TEtrahedral irregular Network (TEN), in which also the 3-simplex is used as 3D building block.

The concept of simplicial complexes (a collection of simplexes connected through certain rules) and its mathematical description (part of the field of algebraic topology (Hatcher 2002)) is described by Giblin (1977). It is mentioned by Frank and Kuhn (1986) as one of the possible cell graph approaches. A topological data model based on 2D simplicial complexes in 2D space is introduced in Egenhofer *et al.* (1989) and implemented in the PANDA system (Egenhofer and Frank 1989), an early object-oriented database. The mathematical approach of simplexes is also used by Pigot (1992) and Paoluzzi *et al.* (1993), but full applications of simplicial homology in three dimensions in a GIS context are not known to the author.

However, not everybody is convinced that 3D models require the usage of a 3-simplex. Tse and Gold (2004) describe a simplicial approach that modifies a TIN (Triangulated Irregular Network) in such a way that it can contain vertical faces, holes and bridges. Their approach applies Euler operations using Quad-Edge structures, which is an elegant solution for storing both Delaunay and Voronoi cell complexes (Gold *et al.* 2005), a concept originally introduced by Guibas and Stolfi (1985). Since Tse and Gold (2004) extend the characteristics of a TIN (a 2.5D or 2.5D+ approach), they refer to their approach as 2.75D (Tse and Gold 2004). Figure 2.11 (Gold 2006, figure 3) illustrates the possibilities of their approach, as a bridge, a hole and several buildings are being integrated within a terrain model.

2.4.4 Regular polytopes

Another topological approach to 3D data modelling is the regular polytope (Thompson 2007). This concept represents geometric objects in a rigorous representation without assuming infinite precision arithmetic, as this is not feasible in a finite digital machine. It uses convex regular polytopes, which are defined as the intersection of a finite set of half spaces (Thompson 2006). By doing so, one can define the object boundaries exactly, even when some of the actual coordinates of the boundary can't be represented in a finite digital computer. By combining several convex regular polytopes, one can represent more complex objects. These unions of convex regular polytopes are the so-called regular polytopes. An example of such a regular polytope can be found in figure 2.12. It shows a regular polytope consisting of three convex regular polytopes, which are, in turn, defined by intersection of the half spaces. Thompson and van Oosterom (2006) present a Java-based implementation of regular polytopes in 2D and 3D.

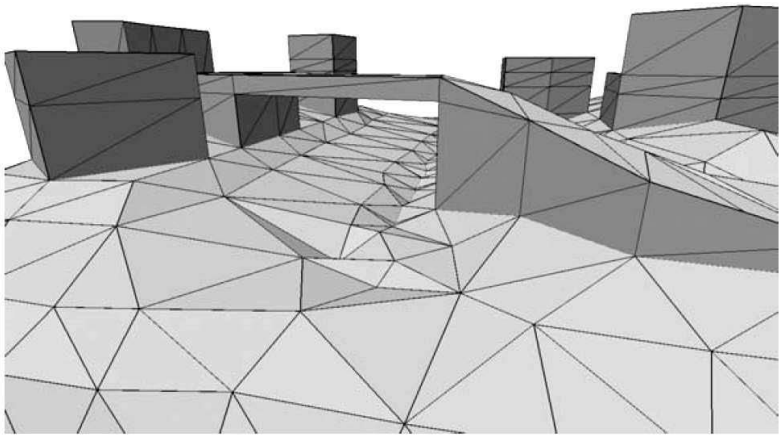


Figure 2.11: *Integration of bridges, holes and buildings within a terrain model (Gold 2006, fig.3)*

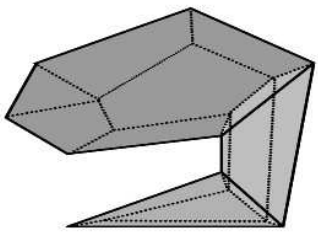


Figure 2.12: *A regular polytope as union of three convex regular polytopes. (Thompson* and *van Oosterom 2006, Fig.2)*

2.5 Triangular data structures and algorithms

The previous section presented several approaches to 3D data representation within an information system. Within this research, one of the basic assumptions is the use of a triangulation approach. A triangulation can be categorised as an irregular tessellation (Worboys and Duckham 2004): a partition of the plane (2D) or volume (3D) as the union of a set of disjoint areal (2D) or volumetric (3D) elements of varying size. In other words, the space is decomposed in smaller elements without intermediate holes or overlaps. Such tessellations are also known under the more general term meshes. Meshing can be defined as the process of breaking up a physical space into smaller subdomains (elements) in order to facilitate the numerical solution of a partial differential equation (Meshing Research Corner 2008). This section limits itself to simplicial meshes, discretisations consisting of triangles or tetrahedrons. These elements are

connected such that a node of an element is always also a node of a neighbouring element, thus excluding configurations in which a node of an element is located on an edge of the neighbouring element.

This section provides an overview of triangulations and tetrahedronisations with different properties and describes the state-of-the-art algorithms. In this research these algorithms are used to create a simplicial mesh of the topographic features. Up till now the terms 'triangular' and 'triangulated' were used as generic terms in general dimension. From now on onwards, a distinction will be made between triangulation, resulting in a mesh constructed of triangles, and tetrahedronisation, resulting in a mesh of tetrahedrons. Triangulations and accompanying algorithms will be introduced respectively in sections 2.5.1 and 2.5.2, whereas tetrahedronisations and accompanying algorithms will be described in sections 2.5.3 and 2.5.4. The description of algorithms will limit itself to incremental algorithms, since section 2.3 acknowledges that edit functionality (without the need for full rebuilds) is crucial for topographic data sets.

2.5.1 2D data triangular structures: triangulations

Before one can describe specific algorithms, it is necessary to determine the actual definition of the required triangulation. A triangulation of a set vertices V is a set of triangles T, whose interiors do not intersect and whose union forms the convex hull of V. Based on this definition one can see that the criteria are met by multiple triangulations, i.e. the triangulation is not unique (see also figure 2.13). As a result,

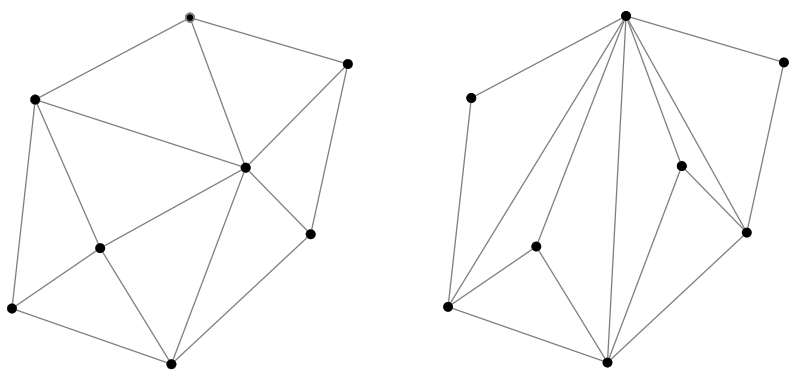


Figure 2.13: *Two different triangulations, based on the same input set of vertices*

additional requirements are used and based on these requirements different types of triangulations exist:

- The *Delaunay triangulation* is the most commonly used triangulation. Delaunay (1934) introduced the empty circumcircle criterion, stating that the circumcircle (circle passing through all three triangle vertices) of any triangle t in T is empty,

i.e. the interior of the circle does not contain any vertex from V. Figure 2.14 illustrates this: the triangulation at the left hand side is a Delaunay triangulation as every circumcircle is empty, while the triangulation at the right hand side does not meet this criterion. Delaunay triangulations owe their popularity to the fact that the resulting triangulation is as equilateral as possible, or, in other words, the minimal angle in the triangulation is maximised (Lawson 1977). This is a desirable property for numerical approximation using triangulations. Unfortunately, Delaunay triangulations are not unique if the points are not in general position, thus enabling multiple triangulations that meet the empty circumcircle criterion. If a triangle meets the Delaunay criterion, its edges meet

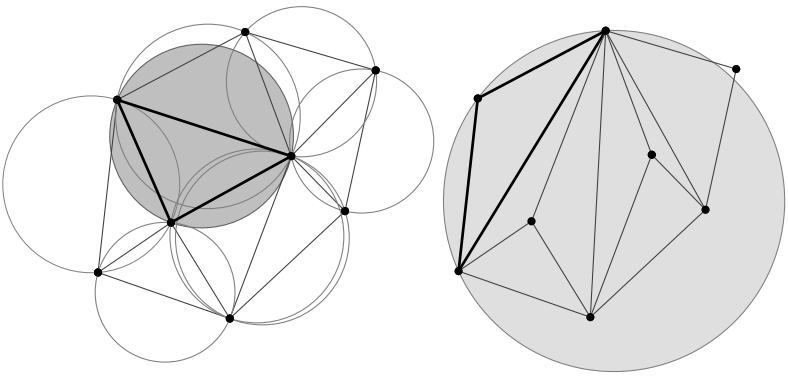


Figure 2.14: *Left: a Delaunay triangulation, as every circumcircle is empty. Right: a non-Delaunay triangulation, as the circumcircle contains multiple other vertices*

the Delaunay criterion (an empty circumcircle of the edge exists) and vice versa (Shewchuk 1999). This characteristic is important since the flipping operation (see section 2.5.2) selects edges for flipping depending on whether they are locally Delaunay or not. The fact that a triangle's edges is locally Delaunay if the triangle meets the Delaunay criterion, is rather obvious. After all, if the triangle has an empty circumcircle, its edges lie in an empty circumcircle. To illustrate this point, the contradictionary situation is illustrated in figure 2.15: triangle t is not Delaunay, so a vertex v lies inside its circumcircle but outside t itself (otherwise it wouldn't be a triangle). Now consider edge e, the edge that separates vertex v from the inside of triangle t and vertex w, the vertex opposite this edge. Since it is not possible to find a containing circle of e containing neither v nor w, the edge is not locally Delaunay either.

- So far, the triangulation is based on a vertex set V. However, if one wants to represent planar features in a triangulation, their outlines should serve as input as well. This input set is often referred to as a planar straight line graph (PSLG): a graph embedding of a planar graph (i.e. a graph without graph edge crossings) in which only straight line segments are used to connect the graph

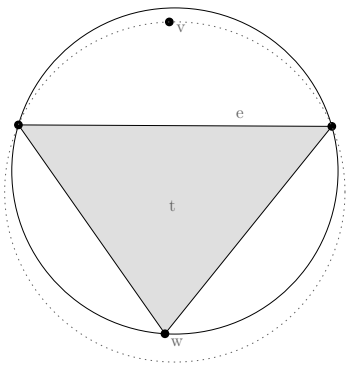


Figure 2.15: *If a triangle is not Delaunay, at least one of its edges is not locally Delaunay*

vertices (MathWorld 2007). The *constrained triangulation* of a PSLG is a triangulation in which every segment of the PSLG appears as an edge. As one can imagine that the guaranteed presence of specific edges can interfere with fulfilment of the empty circumcircle condition, the Delaunay criterion definition is slightly altered. Lee and Lin (1986) and Chew (1989) independently introduce the constrained Delaunay criterion. A triangle is *constrained Delaunay* when two conditions are met. First of all, its vertices are mutually visible, where visibility is supposed to be blocked by a constrained edge (a segment of the PSLG). Secondly, the circumcircle contains no points that are visible from the triangles' interior. This can also be seen in figure 2.16. The thick black edge is a constrained edge and therefore its presence in the triangulation is guaranteed. Due to this presence, the circumcircle of the grey triangle is not empty. Nevertheless, the vertex in the circumcircle is not visible from the triangles interior, since visibility is blocked by the constrained edge. Since the triangulation meets the two requirements, it is a constrained Delaunay triangulation.

- Since a constrained Delaunay triangulation is not a Delaunay triangulation (which can be observed in figure 2.16 when looking at the four points in or on the circle: if the thick black edge in the circle is replaced by an edge, connecting the other two vertices, the minimal angle would be larger), constrained Delaunay triangulations offer less guarantees on numerical stability than regular Delaunay triangulations. This drawback is tackled by the conforming constrained Delaunay triangulation (usually referred to as *conforming Delaunay triangulation*). A *conforming Delaunay triangulation* contains all segments of the PSLG as well, but this time it is allowed to split PSLG segments into multiple edges by the insertion of additional nodes, the so-called Steiner points. These Steiner points are inserted in such a way that the original empty circumcircle criterion always holds, regardless of visibility of vertices. Figure 2.17 shows such a conforming

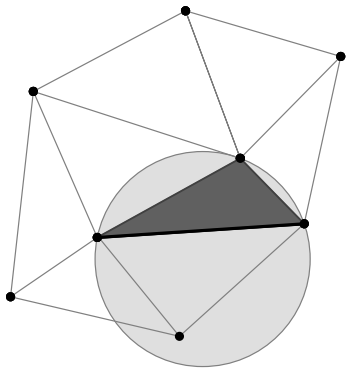


Figure 2.16: *A constrained Delaunay triangulation (with the thick black edge indicating the constrained edge): the vertex within the circumcircle is not visible from the triangles interior, as visibility is blocked by the constrained edge*

Delaunay triangulation. Compared to the constrained Delaunay triangulation from the previous figure, the constrained edge is split into two parts by the insertion of a Steiner point. As one can check in the figure, the circumcircles of the four newly created triangles are all empty again. As a result, the minimal angle is maximised again, but this advantage comes at the expense of an increase in data volume, as additional points are required. Unfortunately, in some cases the number of additional points can grow virtually unlimited, see figure 2.18 (Stoter *et al.* 2005). Two intersecting near-parallel constrained edges will generate more and more small triangles towards the intersection point.

- With the drawbacks of the conforming Delaunay triangulation in mind, a lot of research effort (Chew 1993, Ruppert 1995, Shewchuk 1997b, 2002, Si 2006a) is put into the *refined constrained Delaunay triangulation*. The idea is still to insert Steiner points to obtain better shaped triangles, but in this triangulation satisfying the empty circumcircle criterion (and thus maximising the minimum angle) is no longer the goal. Other quality indicators, like the circumradius-to-shortest edge ratio, are used to guarantee a quality triangulation (i.e. a triangulation with 'nice' triangle shapes, such that numerical instability is avoided) without the need to add as many Steiner points as required to fulfil the empty circumcircle criterion. As a result, one can consider the refined constrained Delaunay triangulation as a very suitable compromise between quality and data volume on the one hand and between constrained and conforming Delaunay triangulations on the other hand.

Delaunay triangulations are very popular methods for digital elevation models. Each x,y-point has a height value attribute in such models, thus creating a 2.5D elevation representation. Despite the 2.5D nature of such a model, the triangulation itself is still computed and optimised in 2D. The problem caused by this is that 2.5D triangles,

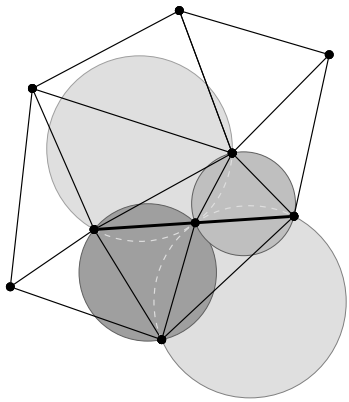


Figure 2.17: *A conforming Delaunay triangulation (with the thick black edges indicating the constrained edges): due to the inserted Steiner point the constrained edge is spit into two segments, and all related triangles have empty circumcircles again*

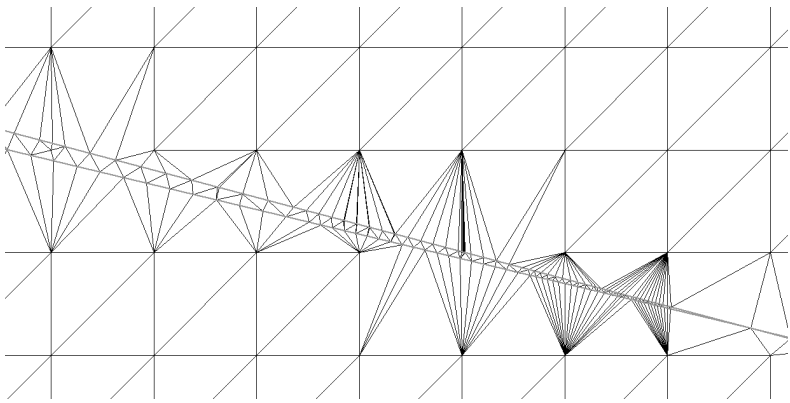


Figure 2.18: *Two near-parallel lines cause addition of large numbers of Steiner points in a conforming Delaunay triangulation (Stoter et al. 2005)*

despite the optimised shape of their 2D projections, still might have all unwanted characteristics like small, sharp angles in 3D. Figure 2.19 illustrates this threat, as it shows a Delaunay triangulation in 2D (with maximised minimum angles) at the left and its 2.5D counterpart at the right. The two middle points are shifted vertically compared to the 2D triangulation, thus introducing sharp angles in 2.5D that are not that sharp in the 2D projection. Verbree and van Oosterom (2003b,a) address this problem and introduce a new method that optimises the actual triangles and not their 2D projections.

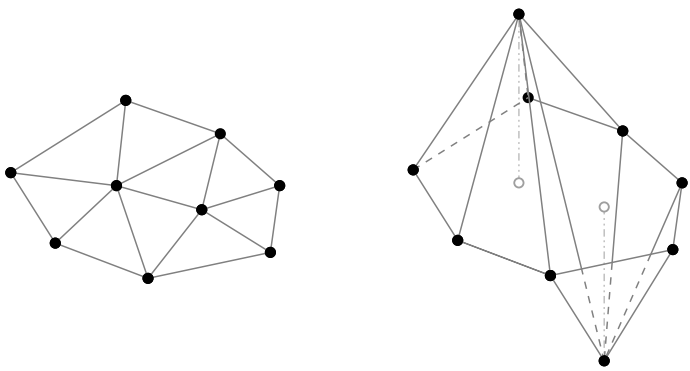


Figure 2.19: *Fulfilling the Delaunay criterion in 2D (left) does not guarantee well-shaped triangles in 2.5D (right). The two middle points in the 2.5D triangulation are shifted vertically compared to the 2D situation*

2.5.2 2D triangulation algorithms

Basically, two major types of incremental Delaunay triangulation algorithms exist: the Lawson algorithm and the Bowyer/Watson algorithm. The first type, introduced by Lawson (1977), is based on edge flipping. Figure 2.20 illustrates Lawson's incremental insertion algorithm: first a new vertex is inserted. Then the triangle in which the new vertex is inserted, is detected. The new vertex is then connected with all three vertices of this triangle. As a next step all edges that are not Delaunay, i.e. all edges with vertices in their minimum circumcircle, are flipped. Flipping is the process of taking the two triangles that share a specific edge e, and replace this edge e by edge e' that connects the two vertices that not span edge e. So for instance, triangle ABC and triangle BCD share the non-Delaunay edge BC. This edge BC is than replaced with edge AD, i.e. the internal edge 'flips'. Shewchuk (1997b, chapter 2, lemma 4) shows that for a triangulation with n vertices, the flip algorithm will terminate after $O(n^2)$ edge flips and result in a Delaunay triangulation.

The second type, the Bowyer/Watson algorithm, is presented indepently by both Bowyer (1981) and Watson (1981), at the same time and in the same journal. Figure 2.21 explains the concept of their approach. It starts with insertion of a new vertex. As a second step, the triangles whose circumcircles contain the vertex, are removed from the triangulation. This results in a polygon-shaped gap, the insertion polygon. As a last step the new vertex is connected with all vertices of the insertion polygon.

The resulting new triangles will always meet the Delaunay criterion. Figure 2.22 illustrates this. Vertex v is the only vertex in the circumcircles of the deleted triangles. Considering a random vertex w of the deleted triangle t, the containing circle C of the new edge vw will lie inside the circumcircle of t and thus be empty. Since this holds for all newly inserted edges, these edges are all Delaunay and thus all new triangles are Delaunay.

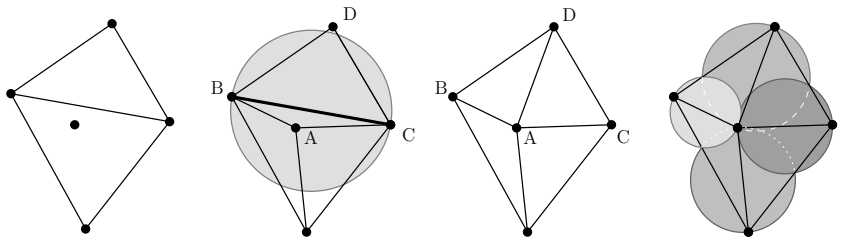


Figure 2.20: *Lawson's algorithm: a new vertex is inserted and connected to the three vertices of the triangle in which it is inserted. As a next step non-Delaunay edges (i.e. edges with non-empty circumcircles) are flipped, so that all edges (and thus triangles) are Delaunay again*

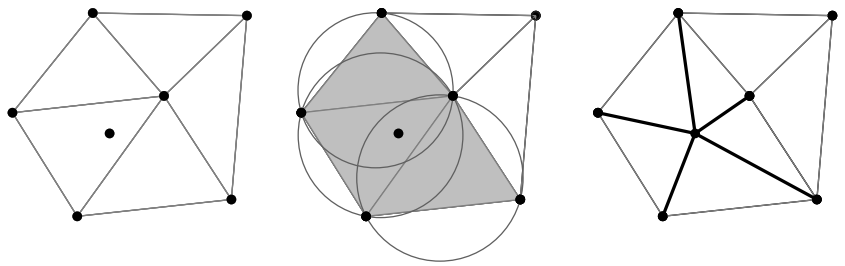


Figure 2.21: *Bowyer/Watson algorithm: insert a new vertex, remove triangles with the new vertex within their circumcircle and connect the new vertex to all vertices on the resulting gap's boundary*

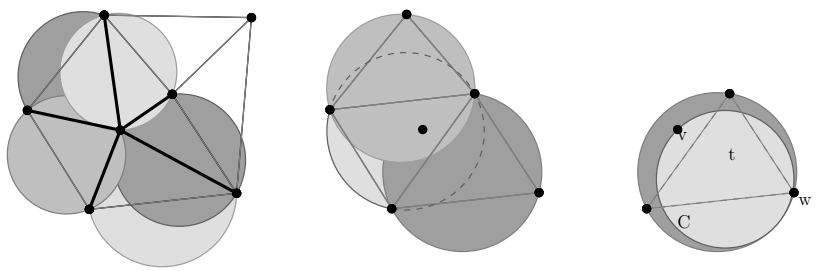


Figure 2.22: *Resulting triangles of Bowyer/Watson algorithm are Delaunay (left). The circumcircles of deleted triangles contain only the new vertex (middle). If v is a new vertex and w is a vertex of a deleted triangle, edge vw is Delaunay. Since this holds for all newly inserted edges, all resulting new triangles are Delaunay*

Flip-based algorithms are usually preferred because they are slightly easier to implement (Shewchuk 1997b). Partially this is caused by the fact that Lawson's algorithm maintains the topological structure, since it remains a triangulation, while in Bowyer/Watson algorithms the insertion polygons are non-triangular. For more details on specific algorithm implementations and their performance, the PhD thesis of Shewchuk (1997b) is strongly recommended.

Algorithms for constrained Delaunay triangulations, conformal Delaunay triangulations and refined constrained Delaunay triangulations are usually based on Lawson's algorithm, with the modification that constrained edges can not be removed by flips. In case of a constrained Delaunay triangulation, the empty circumcircle criterion for non-Delaunay (and non-constrained) edges is adjusted by taking visibility into account (Shewchuk 2003). Conformal Delaunay triangulation algorithms do not flip constrained edges as well, but split them in two equal segments in case the empty circumcircle criterion is not met. Refined constrained Delaunay triangulation algorithms differ from regular constrained Delaunay triangulation algorithms in the sense that they also detect ill-shaped triangles and refine the triangulations by adding additional points, often at the centrepoint of the circumcircle of the ill-shaped triangle (Chew 1993, Shewchuk 2002). Implementations of Delaunay triangulation algorithms are available in a wide range of commercial (GIS) packages. A popular implementation of a refined constrained Delaunay triangulation algorithm is Triangle (Shewchuk 1996).

2.5.3 3D triangular data structures: tetrahedronisations

Although the term triangulation is sometimes used in general dimension, tetrahedronisation is the more often used term specific for triangulation in 3D. Parallel to the 2D case, several types of 3D tetrahedronisations can be distinguished:

- The *Delaunay tetrahedronisation* of a set vertices V is a straightforward generalisation of the previously introduced Delaunay triangulation. A tetrahedronisation is said to be Delaunay if all elements fulfil the empty circumsphere criterion: the sphere that passes through all tetrahedron vertices should be empty. Delaunay tetrahedronisation algorithms are the most popular tetrahedronisation algorithms, albeit that the Delaunay criterion in 3D does not lead to maximisation of the minimal dihedral angle, as can be observed in figure 2.23.

- Besides a vertex set V one can also use a piecewise linear complex (PLC) as input for tetrahedronisation. A PLC (Miller *et al.* 1996) is a set of vertices, segments and facets and each facet can be represented by a planar straight line graph (PSLG, as defined in section 2.5.1), embedded in 3D. Such an input data set can be used to represent for instance object boundaries within a tetrahedronisation. The *constrained Delaunay tetrahedronisation* of a PLC is a tetrahedronisation in which all the PLC components are represented. Parallel to the constrained Delaunay triangulation, the empty circumsphere criterion is

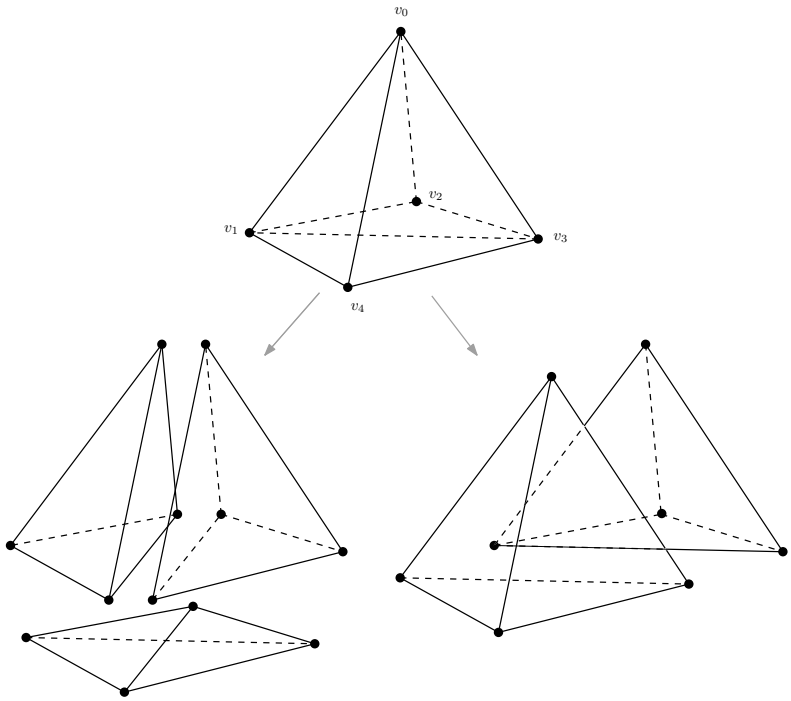


Figure 2.23: *A hexahedron (note that v_2 and v_4 lie above v_1 and v_3) and its Delaunay tetrahedronisation (left) and another tetrahedronisation. The lower tetrahedron in the Delaunay tetrahedronisation has very small dihedral angles, while the non-Delaunay tetrahedronisation results in two well-shaped tetrahedrons (Shewchuk 1997b).*

altered to the constrained Delaunay criterion that the circumsphere of a tetrahedron encloses no vertex of V that is visible from the tetrahedrons interior, where visibility is supposed to be blocked by PLC segments and facets.

A notable difficulty in extending triangulation into 3D is that, whereas in 2D each polygon can be triangulated, polyhedra exist in 3D that can not be tetrahedronised without the insertion of additional vertices. The smallest example is shown by Schönhardt (1928) and this example, currently known as the Schönhardt polyhedron, can be seen in figure 2.24. This polyhedron can be seen as a prism, of which the top triangle is then rotated over 120 degrees. The resulting polyhedron can not be tetrahedronised without additional Steiner points.

As a result different definitions of a constrained (Delaunay) tetrahedronisation can be found in literature. Shewchuk (1997b) does not allow the addition of Steiner points in a constrained tetrahedronisation and thus states that a constrained tetrahedronisation of a PLC does not always exist. However, others

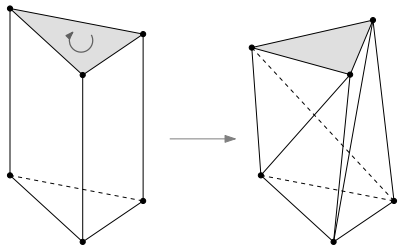


Figure 2.24: *The Schönhardt polyhedron (Schönhardt 1928) can not be tetrahedronised without additional Steiner points*

(Liu and Baida 2000, Si 2006b) use a less strict definition and include Steiner points in a constrained tetrahedronisation, although no consensus exists about the location of these Steiner points. Liu and Baida (2000) allow Steiner points to be included in a constrained tetrahedronisation, except on constraints. Tetrahedronisations in which Steiner points are inserted at constraints are called conforming tetrahedronisations, whereas Si (2006b) does this exactly the other way around. In this dissertation Steiner points are allowed in a constrained tetrahedronisation. The issue of the location of these Steiner points is not fixed and will be addressed explicitly wherever necessary.

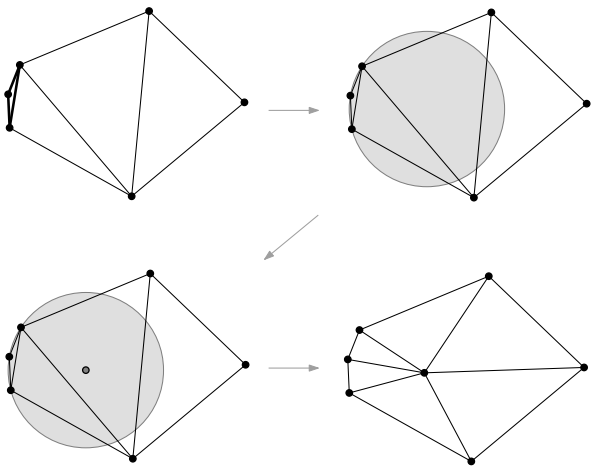


Figure 2.25: *Refining a triangulation: detect ill-shaped triangle by a poor radius-edge ratio, calculate circumcircle, add centrepoint as Steiner point and retriangulate. Note that this method works also in 3D, but for reasons of clarity the 2D situation is illustrated.*

- Since the Delaunay criterion offers less guarantees on tetrahedronisation quality, additional requirements are developed. The resulting *refined Delaunay tetrahedronisations* provide more guarantees in terms of for instance a minimum radius-edge ratio of minimum angles. A lot of research is performed on refinement strategies (Joe 1995, Cavalcanti and Mello 1999, Liu and Baida 2000, Shewchuk 2004, Si 2006a). Figure 2.25 shows an example of such a refinement algorithm. Note that for reasons of clarity this example is drawn in 2D, while it can be performed in 3D as well. It shows the detection of a badly-shaped triangle (3D: tetrahedron). Its circumcircle (3D: circumsphere) is computed and its centrepoint is added as new vertex. The resulting triangles (3D: tetrahedrons) have better shapes.

2.5.4 3D tetrahedronisation algorithms

As in 2D, two different types of incremental algorithms can be distinguished in 3D as well, i.e. the ones based on the Lawson algorithm and the ones based on the Bowyer/Watson algorithm. The idea of the Lawson algorithm is extended first into 3D and later into general dimension by Joe (1991, 1993). It follows a two-step approach: first a vertex is inserted and connected to the vertices of the tetrahedron in which it was inserted, after that the tetrahedronisation is altered by flips to obtain tetrahedrons that meet the empty circumsphere criterion. These two steps are basically the same as in 2D, albeit that in 2D one flips edges to obtain better-shaped triangles, whereas in 3D one flips triangles to obtain better-shaped tetrahedrons. Edge flipping can be performed in one specific way, whereas four types of triangle flipping can be distinguished. This is illustrated in figure 2.26. These different flip types are denoted by a T and the numbers of tetrahedrons before and after the flip:

T_{23}: One of the two most often used flip types in 3D is T_{23}. As illustrated in figure 2.26, T_{23} flips tetrahedrons $< v_0, v_1, v_2, v_3 >$ and $< v_0, v_2, v_3, v_4 >$ into tetrahedrons $< v_0, v_1, v_3, v_4 >$, $< v_0, v_1, v_2, v_4 >$ and $< v_1, v_2, v_3, v_4 >$. Note that T_{23} can only be performed if edge $< v_1, v_4 >$ intersects the interior of triangle $< v_0, v_2, v_3 >$.

T_{32}: One of the two most often used flip types in 3D is T_{32}, the reverse operation from flip T_{23}. T_{32} flips tetrahedrons $< v_0, v_1, v_3, v_4 >$, $< v_0, v_1, v_2, v_4 >$ and $< v_1, v_2, v_3, v_4 >$ into tetrahedrons $< v_0, v_1, v_2, v_3 >$ and $< v_0, v_2, v_3, v_4 >$. Flip T_{32} can only be performed if edge $< v_1, v_4 >$ intersects the interior of triangle $< v_0, v_2, v_3 >$.

T_{22}: This flip is less frequently used, since this operation requires that triangles $< v_1, v_2, v_3 >$ and $< v_1, v_2, v_4 >$ are coplanar, or to define the criterion in line with the one for T_{23} and T_{32}, it requires that edge $< v_1, v_4 >$ intersects the boundary of triangle $< v_0, v_2, v_3 >$. T_{22} flips tetrahedrons $< v_0, v_1, v_2, v_4 >$ and $< v_0, v_1, v_3, v_4 >$ into tetrahedrons $< v_0, v_1, v_2, v_3 >$ and $< v_0, v_2, v_3, v_4 >$.

T_{44}: T_{44} is basically a combination of two T_{22} flips. T_{44} is the more common operation of the two, since a single T_{22} operation can only be applied if the two

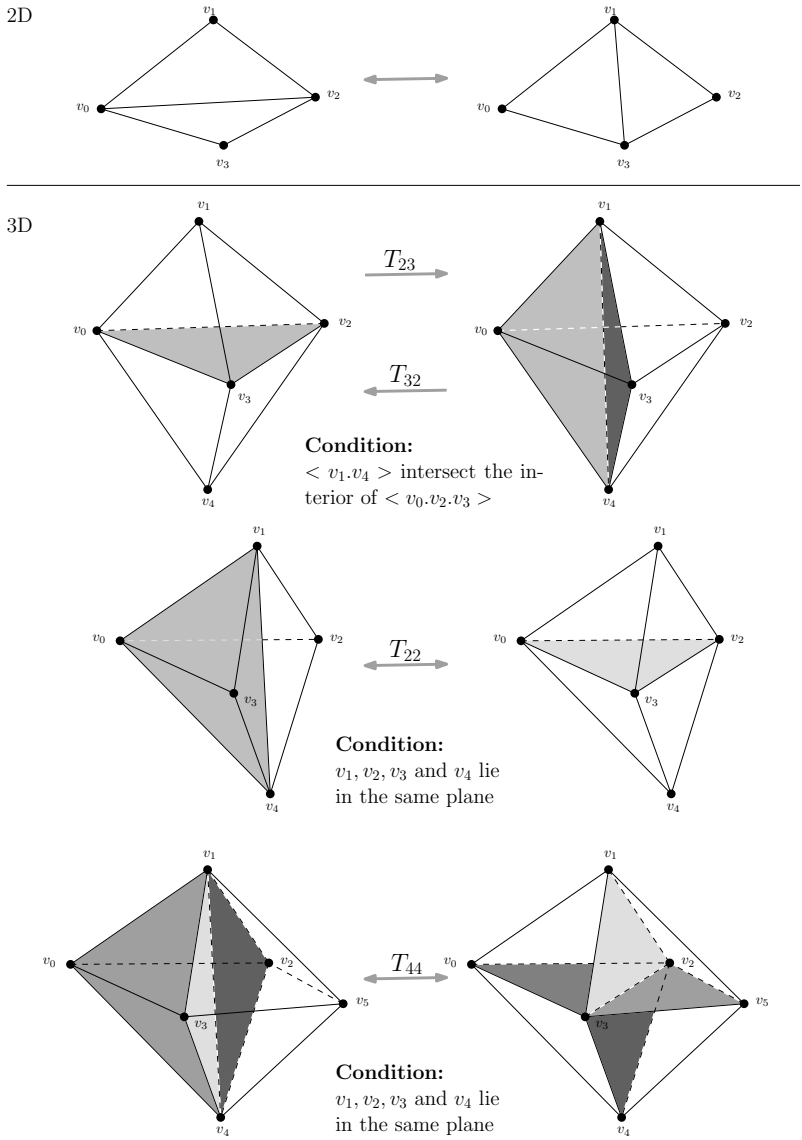


Figure 2.26: *In 2D, only one type of edge flip exists. In 3D, four types of triangle flips exist:* T_{23} *flips* $< v_0, v_1, v_2, v_3 >$ *and* $< v_0, v_2, v_3, v_4 >$ *into* $< v_0, v_1, v_3, v_4 >$, $< v_0, v_1, v_2, v_4 >$ *and* $< v_1, v_2, v_3, v_4 >$. T_{32} *is its reverse.* T_{22} *flips* $< v_0, v_1, v_2, v_4 >$ *and* $< v_0, v_1, v_3, v_4 >$ *into* $< v_0, v_1, v_2, v_3 >$ *and* $< v_0, v_2, v_3, v_4 >$. T_{44} *is a combination of two* T_{22} *flips.*

coplanar triangles lie on the outer boundary (i.e. the convex hull) of the tetrahedronisation, in all other cases the T_{22} flips come in pairs. T_{22} flips tetrahedrons $< v_0, v_1, v_2, v_4 >$, $< v_0, v_1, v_3, v_4 >$, $< v_1, v_2, v_4, v_5 >$ and $< v_1, v_3, v_4, v_5 >$ into tetrahedrons $< v_0, v_1, v_2, v_3 >$, $< v_0, v_2, v_3, v_4 >$, $< v_1, v_2, v_3, v_5 >$ and $< v_2, v_3, v_4, v_5 >$.

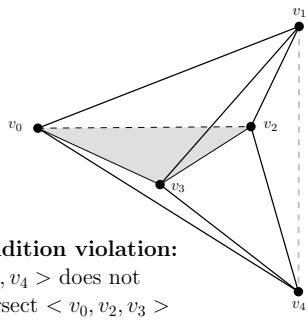


Figure 2.27: *In this case, no T_{23} flip can be performed, since $< v_1, v_4 >$ does not intersect and $< v_0, v_2, v_3 >$*

So far the applicable flip type is determined by the intersection between edge $< v_1, v_4 >$ and triangle $< v_0, v_2, v_3 >$. If this edge intersects the interior of the triangle, T_{23} and T_{32} flips can be performed, and if the edge intersects the boundary of the triangle, T_{22} and T_{44} flips are applicable. Obviously a third category can be distinguished: if the edge intersects the exterior of the triangle, or in other words, if the input tetrahedrons form a concave polyhedron. For sets of tetrahedrons in this third and last category (shown in figure 2.27) no flip can be performed. The concept of the Bowyer/Watson algorithm can be extended into 3D quite straightforward. After inserting a new vertex, all tetrahedrons with non-empty circumspheres are deleted and the resulting polyhedral hole will be tetrahedronised by adding edges from the new vertex to all polyhedron vertices. This concept is illustrated in figure 2.28: a new vertex is inserted and the two tetrahedrons in which circumspheres (not drawn) the new vertex lies, are deleted (the surrounding tetrahedrons with empty circumspheres are not drawn). The resulting hole is then tetrahedronised, resulting in this case in five new edges, nine new triangles and six new tetrahedrons.
Although numerous publications on (refined) tetrahedronisation are available, implementations are still scarce. Shewchuk announced (Shewchuk 2008) Pyramid as 3D successor of Triangle, but Pyramid is still not available. Another implementation, also partially based on work of Shewchuk, is TetGen (TetGen 2007, Si 2006c) by Si. For those interested in more research on algorithms and implementations, the Meshing Research Corner (2008) by Owen is highly recommended.

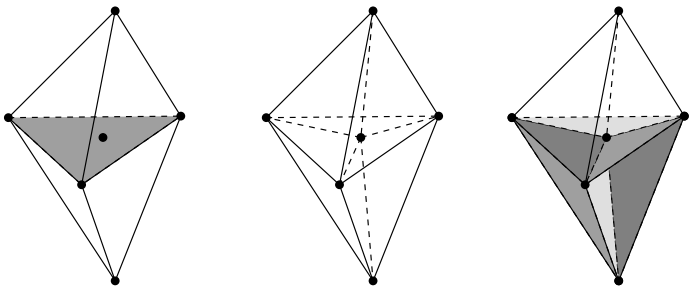


Figure 2.28: *Bowson/Watson algorithm in 3D: insert a new vertex, delete tetrahedrons with non-empty circumcircles and tetrahedronise resulting hole*

2.6 Relevant database concepts

As explained in section 1.2, one of the assumptions within this research is the use of a (spatial) database. A database is a system whose overall purpose is to maintain data and to make that data available on demand (Date 1986). Creating and maintaining such a database is controlled by software known as the database management system (DBMS). Data are stored in tables, consisting of a row of column headings (specifying amongst others the data type for each column) and zero or more rows of data values (Date 1986). The data values in a table are physically stored on disk. However, data are not only accessible in tables, but also in views. Views may be regarded as virtual tables, where 'virtual' indicates that a view contains no physically stored data, but only queries that dynamically retrieve data when needed (Forta 2004). One exception is the materialised view, in which the query result is cached as a concrete table. A view requires no storage space, but may deteriorate performance since the queries are executed dynamically. For a materialised view the opposite holds: it does require storage space, but it will increase performance.

A related important concept is indexing. An index is 'an auxiliary structure that is specifically designed to speed the retrieval of records' (Worboys and Duckham 2004). Indexes in databases are similar to an index in a book: instead of searching a book from cover to cover in order to find a part on a specific subject, one can browse the index to find the page number related to the keyword. Such an index (consisting of an index and a pointer) is an example of a single-level index, whereas more complex data structures may require an index that is recursively indexed itself, a multi-level index (Worboys and Duckham 2004). Well-known examples of multi-level indexes are tree structures, such as the B-tree (Comer 1979) and the R-tree (Guttman 1984). The obvious advantage of indexing is the increased performance, but its disadvantage are the increased storage requirements. Especially in case of multi-dimensional indexes, these storage requirements can become very large (i.e. the same order of magnitude as the indexed table itself). One way of dealing with this is the use of ordering, which basically 'translates' data into a form of lower dimension in order to be able

to use more compact indexes. Figure 2.29 shows two well-known examples of two-dimensional orderings: the Peano-Hilbert (Peano 1890, Hilbert 1891) and Morton (Morton 1966) ordering. These orderings try to ensure that data of locations that are close by in the real world will also be close by on the disk.

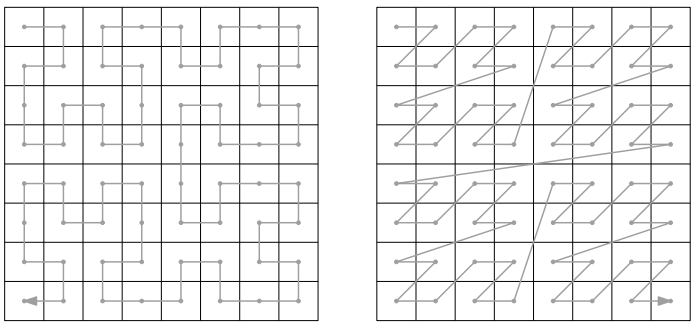


Figure 2.29: *Two well-known examples of 2D ordering: Peano-Hilbert and Morton*

Part I

Conceptual modelling of 3D Topography

Chapter 3

Two triangular data models for 3D topography*

Developing a conceptual model for 3D topography means that one has to decide how to represent reality in a model. Modelling topography in 2D is so common that most people are not aware anymore of the underlying modelling decisions, such as 'a building is represented as a polygon, equivalent to its shape as seen from above' or 'roads and rivers are represented by curved lines'. To extrapolate these rules into 3D is a non-trivial task, especially since one of the requirements from the previous chapter is to deal with complex objects (for instance in case of multiple land use) and the model should serve many applications.

The first triangular approach explored within this research, is based on a fit-for-purpose principle: model in 2.5D (as defined in section 2.2) wherever possible and switch to 3D modelling only in complex (exceptional) cases. Basic idea behind this approach was to keep the model as simple as possible. Section 3.1 introduces this approach and discusses both advantages and disadvantages.

Due to limitations of the 2.5D/3D approach, a new full 3D approach was developed. Although one might expect that this will lead to a more complex modelling approach, section 3.2 will show the opposite. Since the full 3D model has some favourable characteristics, it is selected in section 3.3 as modelling approach for 3D topography. The data structure that will be presented in the next three chapters is designed with this full 3D modelling approach in mind.

3.1 Approach 1: an integrated 2.5D/3D model

Initially (Verbree 2002, Penninga 2005b) the research presented in this dissertation aimed at providing a pragmatic solution to the field of 3D topography. This pragmatic solution was mainly based on the assumption that the need for volumetric objects

*This chapter is largely based on Penninga (2005a), *3D Topographic Data Modelling: Why Rigidity Is Preferable to Pragmatism*. In: Spatial Information Theory, Cosit'05, A.G. Cohn and D.M. Mark (Eds), Vol.3693 of Lecture Notes on Computer Science (Springer), pp. 409-425

remains relatively limited to some specific object categories, of which 'buildings' is the most obvious one. Other relevant topographic features like roads, the earth surface and land use can be sufficiently represented as (curved) surfaces. In other words: the initial idea was that only in some specific cases true 3D modelling would be necessary, whereas in the majority of cases modelling in 2.5D would be sufficient. It assumes that the earth's surface can be modelled in 2.5D and that more complex situations like buildings, viaducts or tunnels can be 'glued' on top or below this surface.

3.1.1 Concepts of the integrated TIN/TEN approach

Combining the preference for a triangular data structure (as explained in the previous chapter) with the pragmatic modelling ideas, leads to the concept of a topographic terrain representation in an integrated TIN/TEN model. Four types of topographic features can be determined: point features (0D), line features (1D), area features (2D) and volume features (3D). For each feature type simplexes of corresponding dimension are available to represent these features within the TIN/TEN model, i.e. nodes, (straight) edges, triangles and tetrahedrons. The basic idea of the integrated TIN/TEN model is to represent 0D-2D objects in a TIN and 3D objects as separate TENs, that will be placed on top or below the TIN. Note that it is presumed that all 3D features are connected to the earth surface. This principle is illustrated in figure 3.1. Note that the TIN is shown as a 2D TIN for simplicity reasons, but that the model uses a 2.5D TIN. As both TINs and TENs are using triangles they can be 'put together' by making sure that they both contain the corresponding triangles.

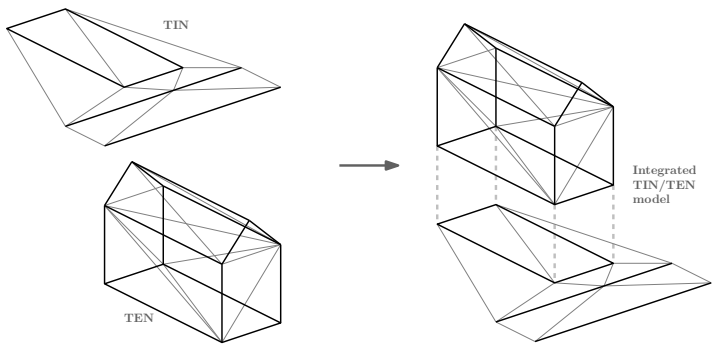


Figure 3.1: *Principle of modelling in an integrated TIN/TEN model*

Similar to the Formal Data Structure (Molenaar 1990a,b) the initial modelling approach is feature-oriented. The model is able to represent point, line, area and volume features. In order to integrate TIN and TENs (and thus the 2.5D world with the 3D worlds) at semantical level as well, the footprints of volume features will be integrated in the TIN. These footprints are the intersection between the terrain surface and 3D features such as buildings, tunnels and bridges. As a result, the TIN can be considered as a topographic representation at terrain level.

Figure 3.2 shows the integration of TIN, TENs and features in a UML class diagram. In this class diagram, one can see the different types of features (Volume feature, Area Feature, Line Feature, Point feature) and the different building blocks of the TIN and TEN (Tetrahedron, Triangle, Edge, Node). While comparing for instance the TIN Edge and the TEN Edge, one can see that these edges are not identical, since a TIN and a TEN Edge have different associations, especially in the multiplicity. A TIN Edge will bound two TIN Triangles (unless it lies on the outer boundary of the TIN), while a TEN Edge will bound two or more TEN Triangles. As a result, a directed TIN Edge will have a TIN Triangle on its left and one on its right, whereas this specific association does not exist for a TEN Edge (but a directed TEN Triangle will have a tetrahedron on its left and one on its right). In the depicted UML diagram, only linking at node level between TIN and TEN is shown (the 'isIdentical' association). Obviously, the TIN and TEN do not only need to be linked at data structure level, but also at feature level. Therefore each Volume feature has a Volume Footprint (an orthogonal projection on the terrain surface), that will be part of the Terrain level representation. As a result, Volume Footprints and Area features are represented by TIN Triangles, while Line features are represented by TIN Edges and Point features by TIN Nodes. The resulting Terrain level representation forms a TIN. Only Volume features are represented by TEN Tetrahedrons, resulting in separate TENs.

One might have expected the underlying data structure to be designed quite straightforwardly, consisting only of nodes, edges, triangles and tetrahedrons (and thus not of separate TIN and TENs). However, although for instance edges appear both in TIN and TEN, Penninga (2005a) argues that a TIN edge is not conceptually the same as a TEN edge. The same holds for TIN triangles and TEN triangles. The differences are in their mutual topological relationships: within a TIN an edge has two neighbouring faces (left/right), while the number of associated faces is unbounded within a TEN. Therefore it is necessary to model the TIN and TEN separately and, when appropriate, link or even merge its components. Nevertheless, a close relationship exists between for instance a TIN edge and a TEN edge, as they are both 1-simplexes and their geometries might be identical. Although the connection between TIN and TEN is only available at node level –and not at edge or triangle level– in the UML diagram (figure 3.2), one can distinguish in general three types of linking between TIN and TEN (see also figure 3.3):

- link at triangle level (thus 'glueing' the TENs on top of the TIN)
- link at edge level (thus 'stitching' the TENs on top of the TIN)
- link at node level (thus 'nailing' the TENs on top of the TIN)

Linking at node level is the most fundamental one of this three, as edges and (indirect) triangles are defined by their nodes. However in order to optimise analytical capabilities one could prefer to model the link at triangle level (which implies also the relationship on edge and node level). If one would only link at node level, one can not be sure that the TIN and TEN will contain identical triangles. In the illustrated

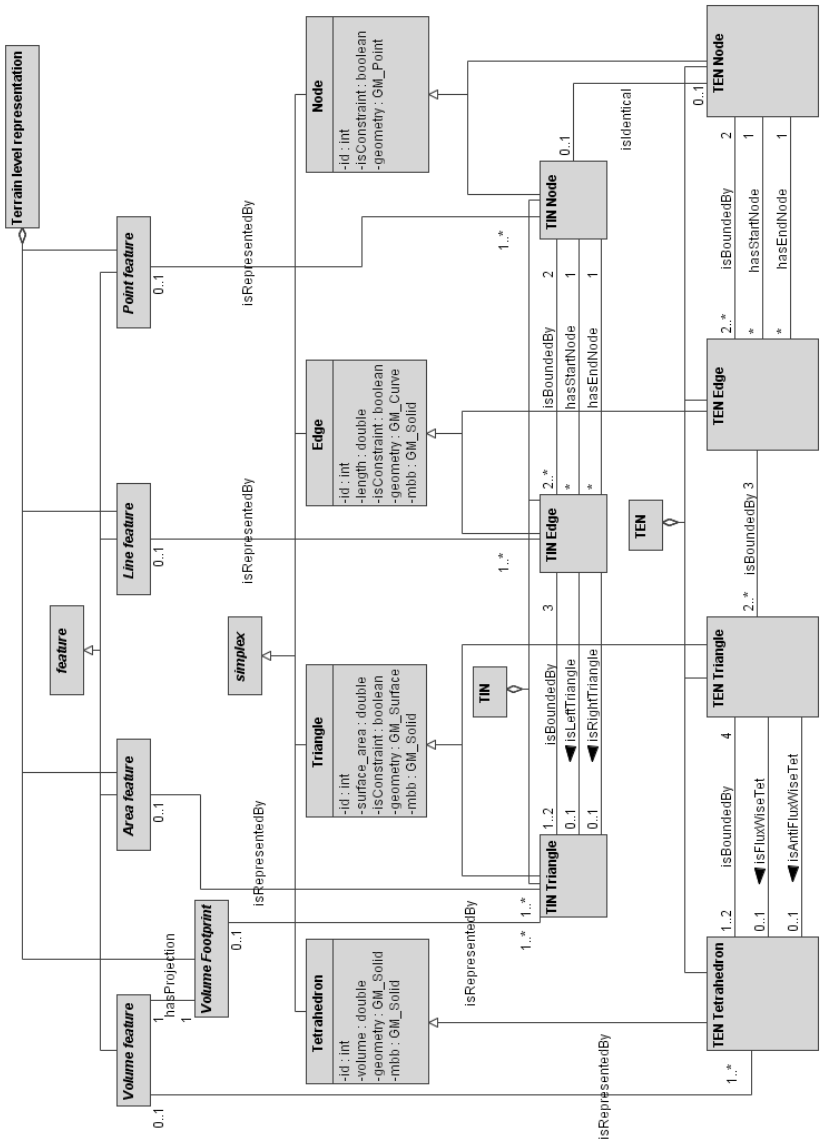


Figure 3.2: *UML class diagram of the conceptual model of the feature-based integrated TIN/TEN approach (note that current attributes and associations will not necessarily be stored explicitly in an actual implementation, but are included here for clarification purposes only)*

example, the edge dividing the two triangles might be flipped in either the TIN or TEN, while this would not be the case if one uses edge or triangle linking.

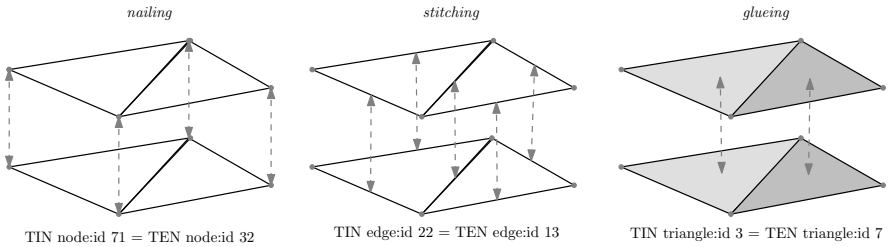


Figure 3.3: *Three options for linking TIN and TENs: nailing, stitching and glueing*

3.1.2 Advantages of the TIN/TEN approach

In a way the integrated TIN/TEN approach is the ultimate example of applying the fit-for-purpose principle. First of all a triangular approach is selected, as its irregularity enables one to use a lot of points in complex areas, whereas fewer points will be sufficient in less complex or less important regions. This flexibility provides the required degree of detail at each location, without causing a substantial increase in data storage, as a raster-like regular approach would. Applying the integrated 2.5D/3D approach further extends this concept, as it provides a 2.5D TIN model wherever sufficient and only switches to the more complex and expensive (in terms of data storage) 3D TEN models when necessary. At the same time this approach fits with the intended purpose. Within a topographic data set the earth surface is a unique feature, as most topographic objects are part of or located directly on top of this surface. This uniqueness can be observed in figure 3.4, in which a 2D topographic map is draped on top of a height data set, thus approaching the actual look of a 2.5D/3D topographic data model. This illustration endorses the use of the 2.5D earth surface as a model base.

A second fit-for-purpose aspect is that topographic data sets are very large, as they usually offer nationwide coverage. Extending these data sets into the third dimension will increase their size dramatically. Minimising modelling in 3D is certainly a valid strategy in dealing with this potential increase in data volume.

The third fit-for-purpose aspect is that a lot of analyses do not require 3D models. Operations like for instance slope analysis and line-of-sight computations in natural terrain, can be performed on 2.5D surfaces. With the integrated model, in which the footprints of 3D features are present in the 2.5D TIN, the user still has the option to ignore the TENs completely and confine oneself to using only the TIN surface for his application.

Figure 3.4: *Draping a 2D topographic map onto a height data set shows the plausibility of the assumption that the earth surface is a very important topographic feature (source: Actueel Hoogtebestand Nederland (2006))*

3.1.3 Disadvantages of the TIN/TEN approach

Although the initial 2.5D/3D TIN/TEN modelling approach seems to make sense from a practical topographical point of view, it has some serious hidden problems. The problems lie within the integration of the TIN with the TENs. At a conceptual level the 'isIdenticalTo' relationship between a TIN Node and a TEN Node (as used in the UML class diagram in figure 3.2) is an appropriate way of linking both models. However, in order to optimise analytical capabilities one would prefer this link to exist on node as well as on edge and triangle level. In order to do so, one needs an implementation that takes care of ensuring the 1:1 relationship between TIN surface and TEN bottom.

If one considers the example of a building placed on top of the terrain (as illustrated in figure 3.1), not only the footprint of the building should match in TIN and TEN, but also the internal edges in the shared face. The building will be represented as a rectangle in the TIN and this rectangle is identical to the floor boundaries in the TEN. Furthermore one needs the guarantee that the internal edge, that triangulates this rectangle, is the same diagonal in both data structures. Unfortunately it is not possible to ensure such a match between the TIN and TEN triangles, as constrained triangulations and tetrahedronisations are only capable of handling constrained edges (Shewchuk 2004). This implies that not only the outer boundaries

should be handled as constraints, but also this internal edge. As a result the outcome of the TIN triangulation can be transferred as input into the TEN triangulation or vice versa. This dependency wouldn't be a problem if one could ensure that these constraints can be triangulated without problems. However additional Steiner points are often required in order to enable tetrahedronisation (for instance Schönhardts polyhedron, introduced in section 2.5.3) or to improve the quality of a constrained Delaunay triangulation (and thus the numerical stability). As Steiner points are inserted for instance in the TIN, they need to be transferred into the TEN, accompanied with the additional edges created by the Steiner points. At the same time the TEN algorithm might insert additional Steiner points, which should be transferred back to the TIN, thus resulting in the threat of a significant (but unnecessary) increase in the number of Steiner points. An alternative approach is described by Verbree (2006). His approach consists of the addition of adding enough Steiner points such that the tetrahedronisation becomes a conforming Delaunay tetrahedronisation. In this case (except for degenerate cases with four points on a circle/sphere) both triangulation and tetrahedronisation of the extended node set will result in the same triangles.

This problem can partially be solved by handling Steiner points in a different way. Within algorithms used in GIS, Steiner points are almost always used to split long edges into smaller ones. In the more general research field of meshing, where amongst others triangulation and tetrahedronisation are used in order to simplify complex objects to enable appliance of partial differential equations, Steiner points are also added in the interior of a triangle or tetrahedron (Shewchuk 1997b). In particular most refinement algorithms select skinny triangles and add the centrepoint of the circumcircle as a Steiner point. In TENs the centrepoint of the circumsphere can be used. Within GIS, adding internal nodes is rather unusual, probably due to the fact that this data is collected by surveying techniques as GPS or photogrammetry, which are point measurement techniques. As a result, a node usually represents a measurement, which is not the case for Steiner points in the interior of an object. Nevertheless, the question whether every polyhedron can be triangulated or tetrahedronised without adding Steiner points at the boundary of the object is theoretically not answered yet.

Although the modelling approach is quite straightforward ('model in 2.5D, switch to 3D only in exceptional cases'), the questions how to link both models and when to switch between the two representations are not easy to answer (and thus to implement in practice). To this important design question two possible solutions exist:

- The first option is that both TIN and TEN should exist in case of more complex situations, resulting in a *TIN+TEN* approach. One can imagine a situation in which a single building is represented in a TEN. At this location it will be quite easy to include the footprint in the TIN, thus using a TIN+TEN approach. This approach is illustrated in the UML class diagram in figure 3.2, although it would require that the 'isIdentical' association would exist between TIN Triangle and TEN Triangle (and as a result, also between TIN Edge and TEN Edge and between TIN Node and TEN Node).

- The second option is that only the TEN should be available in these situations (implying a 'hole' in the TIN), resulting in a *TIN-TEN* approach. As a consequence, a 2.5D representation is created that is not a surface partition. Kolbe *et al.* (2005) suggest the use of virtual closure surfaces to obtain a closed surface, although these surfaces have no relationships with actual features. The need for the TIN-TEN approach can be illustrated best with the complex situation in figure 3.5, in which a highway and railroad tracks are planned in tunnels, with a station and offices build on top. The question arises how to give a meaningful definition of the 'earth surface' and thus what to include in the 2.5D representation. In such a situation it would make more sense to model the entire complex situation in one TEN, which shares its borders with the surrounding TIN, thus using a TIN-TEN approach. Such an approach would require changes in the UML class diagram in figure 3.2. The Volume Footprint would no longer be represented by one or more TIN Triangles, but result in a hole. An additional type 'Virtual closure surface' need to be added to obtain a closed surface again. The TIN-TEN approach also affects the 'IsIdentical' relationship between TIN and TEN elements. Obviously, this association will not exist on triangle level (as it will in the TIN+TEN approach), but on edge level.

As a result from both options and examples, one cannot select either the TIN+TEN or the TIN-TEN approach alone. It makes sense to use both approaches, but how to satisfactory define a general rule when to apply TIN+TEN and when TIN-TEN? This criterion adds more complexity to the initial simple modelling concept. In order to contribute to further confusion, let us concentrate at the question which feature types are modelled in the TIN and which in a TEN. If one considers the simplified viaduct in figure 3.6, applying the initial modelling approach would imply that only the viaduct itself will be modelled in 3D and the ascent and descent in 2.5D. Suppose that both on and under the viaduct a highway exists. In this case the bottom highway is represented in a TIN and the upper highway in a TEN. As a result the upper highway will have a thickness, while the bottom highway has not. One can also decide to only label the top triangles of the TEN as highway, but this will result in a meaningless volume, acting as a 'carrier object' for the highway object. The thickness of the TEN (the viaduct) can be surveyed. Still, why should one include this thickness or a 'carrier object' for highways on a viaduct and not include such a thickness for the foundation of the bottom highway? As a result, a highway will sometimes have a thickness, and sometimes not. This inconsistency is hard to accept.

The earlier observation that it will be difficult to define a 2.5D terrain surface everywhere, implies that it is apparently not possible to extend all characteristics of a 2D representation into 2.5D, especially when this 2.5D representation has to fit with 3D TENs. In 2D, a topographic representation can be considered as a topologically closed surface. This plays an important role in consistency checks of the data. However, in 2.5D this rule no longer applies, for instance if one thinks of a tunnel entrance. In 2D the road stops at the tunnel entrance, which is also the border of the terrain feature lying above the tunnel. In 2.5D there will be a vertical gap between these two features, resulting in a non-watertight surface. Additional closure surfaces might

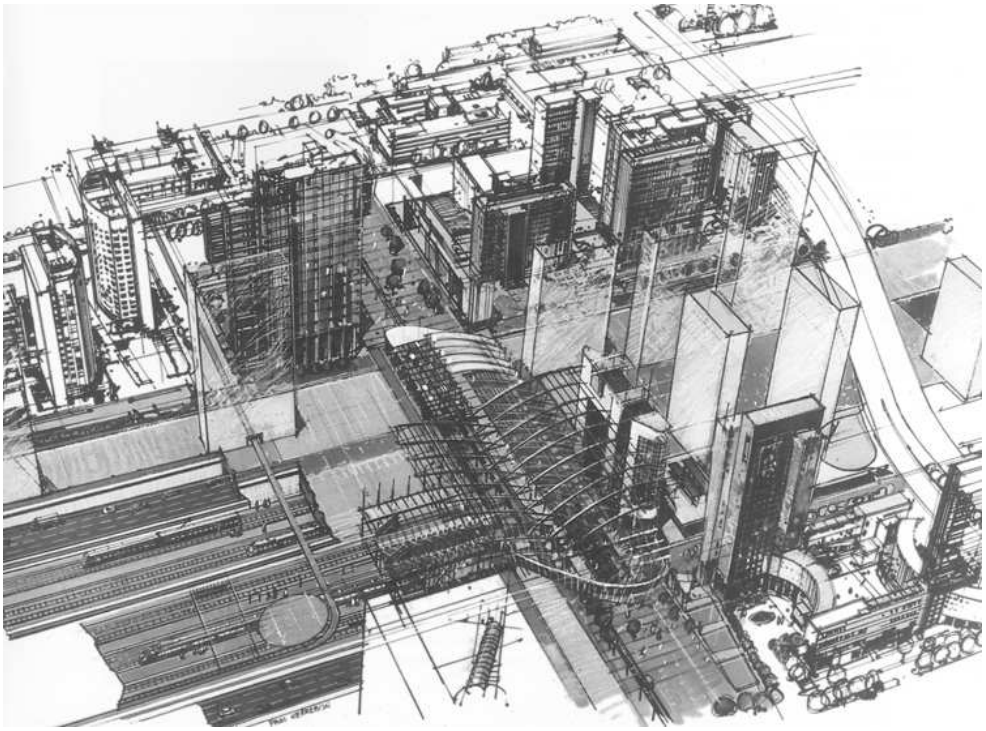


Figure 3.5: *Impression of the plans for the renewal of Amsterdam WTC Station, with several tunnels, offices and a station build on top of each other*

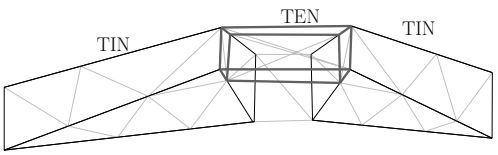


Figure 3.6: *Simplified viaduct, partially modelled as TIN and partially as TEN*

solve this, thus resulting in a 2.5D+ model, but standard triangulation algorithms will not create such 2.5D+ models.

So far it is clear that - after solving a number of practical problems as described above - it will be possible to create a model in which TIN and TENs are combined, thus enabling a representation in which both surface and volume objects are present. Now the question arises whether this representation also offers the required functionality. For instance, figure 3.6 illustrated the possibility to model a viaduct as a combination of a TIN and a TEN, but does this model enable the user to analyse the clearance

under the viaduct? In a geometrical sense a relationship can be discovered between the TIN triangle of the bottom highway and the TEN lying above, but in a topological sense these objects are not related. Wouldn't it be better if the air between the viaduct and the underlying terrain was also modelled, thus enabling the user to calculate the height of these 'air' tetrahedrons in order to solve his query? The same holds for a tunnel. In the TIN+TEN case the tunnel TEN is only attached to the TIN at both tunnel entrances, but would this enable a quick analysis of which buildings are located on top of the tunnel? If the earth between the tunnel and the buildings is also modelled, this would simplify the query. Again this raises the question whether it is possible to satisfactory define a general rule which 'air' or 'earth' tetrahedrons should be included and which not.

3.2 Approach 2: a full 3D data model

3.2.1 Concepts of the TEN approach

With respect to modelling 3D topographic data, two fundamental observations are of great importance (Penninga 2005a):

- Physical objects have by definition a volume. In reality, there are no point, curve or surface objects, only point, curve or surface representations exist (at a certain level of abstraction/simplification). The ISO 19101 Geographic information - Reference model (ISO 19101:2002 2002) defines features as 'abstractions of real world phenomena'. In most current modelling approaches the abstraction (read 'simplification') is in the choice for a representation of lower dimension. However, as the proposed method uses a tetrahedral network (or mesh), the simplification is already in the subdivision into easy-to-handle parts (i.e. it is a finite element method!).

- The physical real world can be considered a volume partition: a set of nonoverlapping volumes that form a closed (i.e. no gaps within the domain) modelled space. As a consequence, objects like 'earth' or 'air' are explicitly part of the real world and thus have to be modelled.

Figure 3.7 shows an UML class diagram of this model. Although the Physical world consists of VolumeFeatures, some AreaFeatures might still be very useful, as they mark the boundary (or transition) between two VolumeFeatures. In our modelling approach AreaFeatures can exist, but only as 'derived features'. This also holds for LineFeatures and PointFeatures. In UML terms, AreaFeatures are modelled as association classes. For instance, a 'wall' might be the result of the association between a 'building' and the 'air'. It is important to realise that planar features that mark borders between volumes might be labelled (for instance as 'roof' or 'wall' with additional attributes), but that they do not represent or describe the building. In this example the building in itself is represented by a volume, with neighbouring volumes that represent air, earth or perhaps another adjacent building. Labelled features like

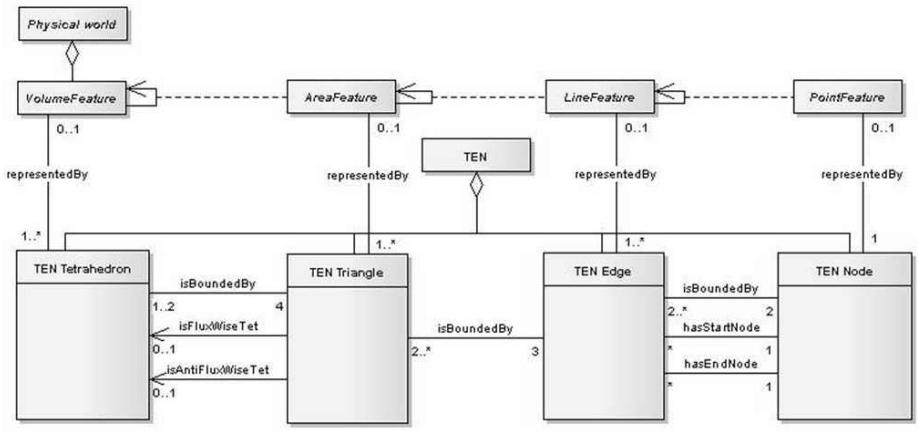


Figure 3.7: *UML class diagram of the conceptual model of the full 3D data model*

'roof' and 'wall' are lifetime dependent from the association between the building and its exterior.

As a result the actual 3D model will show more resemblance with the real world. Deriving visualisations from this model might result in more simplified models, as there is no one standard 'best' visualisation for all purposes. However, the choice which representation to use, should be made in the Digital Cartographic Model (DCM, a set of cartographic rules) and not in the Digital Landscape Model that contains the 3D physical objects (for DCM and DLM, see Kraak and Ormeling (1996)).

3.2.2 Advantages of the TEN approach

The explicit inclusion of earth and air features is not very common, as these features are often considered as empty space in between topographic features. However, this inclusion is not only serving the abstract goal of 'clean' modelling, but has actually some useful applications. Firstly, air and earth features do not just fill up the space between features of the other types, but are often also subject of analyses. One can think of applications like modelling noise propagation or air pollution. Secondly, by introducing earth and air features, future extensions of the model will be enabled. Space that is currently labelled as air can be subdivided in for instance air traffic or telecommunication corridors, while earth might be subclassified in geological layers or polluted regions. Figure 3.8 shows some examples of future extensions.

Another advantage of modelling these 'empty' spaces is that it enables very pragmatic solutions for short term problems. If one thinks again of the viaduct in figure 3.6, the problem was that feature instances of the same type were sometimes represented in 2.5D and sometimes in 3D, even though the thickness was not known. As long as no real data is available for the viaduct, one might model both crossing roads as faces. The upper highway will be represented as a set of flat faces on the border of

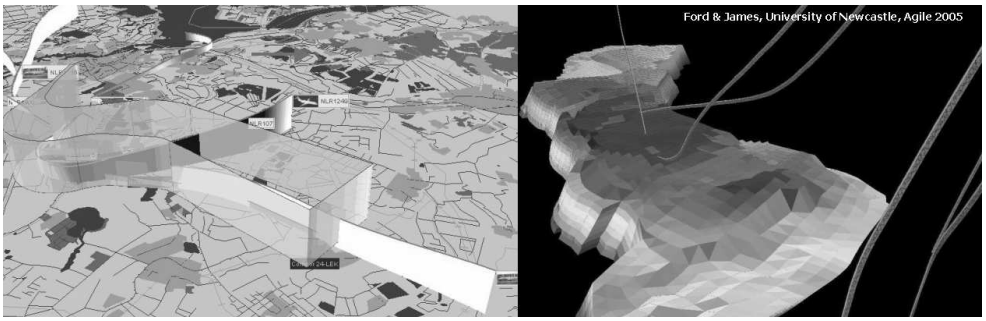


Figure 3.8: *Air traffic corridors towards Schiphol Airport, the Netherlands (left) and an oil reservoir (right): examples of future extensions (Illustration oil reservoir from Ford* and *James (2005))*

two volumes, this time both 'air' volumes. This might not be a desirable permanent solution, but until real 3D data of viaducts becomes available, this is a pragmatic solution.

As a last advantage, the simplicity of the concept in itself can be mentioned: modelling 'everything' in a TEN is easier than modelling sometimes in a TIN, sometimes in a TEN. Immediately it should be mentioned that modelling in a TEN is more complex, both in terms of insightfulness and of the required algorithms, than modelling in a TIN. Thus the question remains whether modelling everything in a TEN will be also easier at implementation level.

Another interesting concept that is loosely related to the new modelling approach, is an idea that still enables a user to work with a 2.5D TIN, even though the topographic model is stored as an explicit TEN. Up till now, no distinction has been made between the way in which features are presented to the user and the way these representations are stored. One of the goals of the 3D Topography research is to implement the model in a spatial database, as this will improve manageability of the data. Databases have a rather nice feature and that is that one can work with views instead of tables. Whereas a table is physically stored in the database, a view can be seen as some kind of a virtual table. The user has all functionality as if he is working with a table, but the view is only a certain filter on top of one or more tables. Within the topography model one can think of node, edge, triangle and tetrahedron tables as basic storage structure of the 3D TEN. As stated at the beginning of this chapter, the observation is that large parts of a country can be considered to be 2.5D, thus it still would simplify some applications if a TIN surface would be available. The idea is to define a view on top of the triangle table of the TEN, such that this view consists of TIN triangles representing the earth surface. As seen in the previous section, linking the TIN with the TENs was very difficult due to several problems, amongst others with Steiner points. The concept of using a view on a TEN does not try to solve the integration problem of the TIN and TEN, as it avoids the problem completely.

3.2.3 Disadvantages of the TEN approach

The drawback of TENs, most often mentioned in literature, is the required amount of storage space. Amongst others, Zlatanova *et al.* (2004) state that TENs have a much larger database size in comparison with other 3D approaches. This line of reasoning is based on a comparison for a single building as shown in table 3.1, in which the number of tetrahedrons, triangles, edges and nodes is compared to the number of volumes, faces, edges and points in a polyhedron approach.

Building as polyhedron	**Building as explicit TEN**
(1 volume)	8 tetrahedrons
7 faces	24 triangles
(15 edges)	25 edges
10 points	10 nodes

Table 3.1: *Comparing storage requirements of a polyhedron and a TEN approach for the building in figure 4.7. The brackets indicate implicit presence (as opposite of explicit storage). In an explicit TEN, all elements are stored explicitly, although more elements might be modelled implicitly in an actual TEN implementation.*

In order to reach acceptable performance, it has to be decided which primitives and which relationships between TEN elements will be stored explicitly. The performance requirements do not tolerate full storage of all possible relationships. Several approaches exist in 2D to reduce storage requirements of TINs by either working with an edge or a triangle based approach, in which not both triangles, edges and nodes are stored explicitly. Unfortunately, in the 3D situation and especially in case of a constrained TEN this might become very difficult. Nevertheless this drawback should be tackled at design level.

In addition to the (supposed) drawback of TEN storage requirements, one should realise that the described TEN approach includes a full tetrahedronisation of 3D space, thus adding air and earth features to the model. The addition of these features will influence storage requirements significantly.

Although both factors relate to storage requirements, a clear distinction has to be made between increasing storage requirements, triggered by the selection of a triangular data structure and increasing storage requirements, triggered by the full volumetric modelling approach. Especially the increases caused by the triangular data structure might be limited by a proper implementation of a TEN structure in a database. The upcoming chapters will present such an implementation.

3.3 The choice for the full 3D approach

This chapter first introduced a very pragmatic approach to 3D modelling, as it aims at modelling as much as possible in (less complicated) 2.5D, whereas full 3D modelling will be applied only in exceptional cases. Triangulations were selected (in the

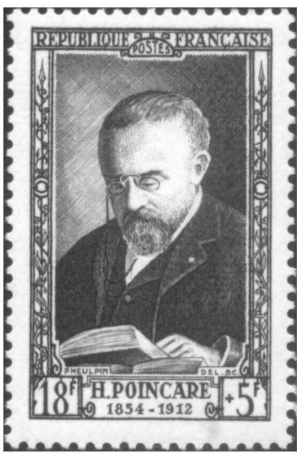
Part II

A Data structure for 3D Topography

Chapter 4

Theoretical foundations: Poincaré simplicial homology*

The previous chapter answered the question how to develop a conceptual model describing topographic features: a full 3D approach based on tetrahedrons is the most appropriate. As shown in the chapter 3, this network of tetrahedrons consists of tetrahedrons, triangles, edges and nodes. These simplexes are constructed recursively: a tetrahedron is defined by four triangles, a triangle by three edges and an edge by two nodes. Intuitively, these relationships offer computational advantages in operations since they are known in advance.



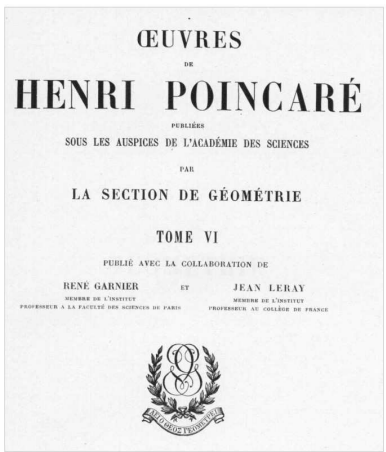


Figure 4.1: *Henri Poincaré, founder of simplicial homology research*

*This chapter is partially based on Penninga and van Oosterom (2008a), *A Simplicial Complex-based DBMS Approach To 3D Topographic Data Modelling*. Accepted for International Journal of Geographical Information Science: August 18, 2007. Publication scheduled for 2008.

Although these favourable characteristics are recognised by others (Guibas and Stolfi 1985, Pilouk 1996, Wei *et al.* 1998), they are rarely explored in a structured fashion, except for Frank and Kuhn (1986), Carlson (1987), Egenhofer *et al.* (1989) and Paoluzzi *et al.* (1993) (see section 2.4.3). Nevertheless, in this research the solid mathematical foundations of simplexes will be applied. Simplexes, their characteristics and their mutual relationships are extensively studied in the late 19th century by Henri Poincaré (see figure 4.1). His results still form the basis of the field of simplicial homology.

Applying simplicial homology offers significant advantages. Before introducing the simplicial homology-based data structure in the next chapter, this chapter will provide all relevant backgrounds of simplicial homology. Simplicial homology offers a mathematical description of simplexes and of their mutual relationships (section 4.1). A very important characteristic in handling geometries is orientation, as it is the basis for determining crucial relationships as left/right and inside/outside. The new data structure greatly benefits from the fact that simplicial homology enables full control over simplex orientation in each dimension. The underlying theory will be introduced in section 4.2. Another important aspect of the new approach is the representation of topographic features. These features are composed of several tetrahedrons. In mathematical terms these combined simplexes are known as simplicial complexes (section 4.3). The last section (section 4.4) will show how the use of tetrahedrons as basic building blocks for features will simplify operations on these features.

In this chapter the following annotations will be used: S_n for a simplex of dimension n, ∂ for the boundary and v_i for a node. In mathematical terms vertex is the appropriate term, but in a GIS context node is more often used (especially in TINs and TENs). As a compromise this chapter will use the term node and the annotation v_i.

4.1 Mathematical description of simplexes

Simplexes and the relationships between simplexes of different dimensions were studied by mathematicians in the late 19th and early 20th century. This field of mathematics was known as simplicial homology and is today considered part of the field of Algebraic Topology (Hatcher 2002). The foundations of simplicial homology are described by Jules Henri Poincaré (1854-1912) in (Poincaré 1895). Some relevant corrections and additions can be found in (Poincaré 1899). Simplicial homology is the part of mathematics that deals with topological constructions of simplexes. Intuitively a n-simplex can be described as the simplest geometry of dimension n, where simplest refers to minimising the number of points required to define such a simplex. For instance, one needs at least three points to define a 2D shape (a triangle) and these three points should not lie on the same line (since that would result in a 1D edge). A simplex can be seen as an elementary building block of its dimension; they are used to construct simplexes of higher dimension.

The previously introduced volumetric approach uses tetrahedrons to model the real world. These tetrahedrons in the TEN structure consist of nodes, edges and

triangles. All four data types are simplexes. A formal definition (Hatcher 2002) of a n-simplex S_n is given below:

Definition 1 *A n-simplex S_n is the smallest convex set in Euclidian space (denoted $\mathbb{R}^m$) with $n \leq m$, containing $n+1$ points $v_0, \ldots, v_n$ that do not lie in a hyperplane of dimension less than n. As the n-dimensional simplex is defined by $n+1$ nodes, it has the following notation: $S_n =< v_0, \ldots, v_n >$.*

Equivalent conditions to the hyperplane condition would be that the difference vectors $v_1 - v_0, \ldots, v_n - v_0$ are linearly independent or, if one considers $v_0, \ldots, v_n$ as set of vectors, that these vectors are affinely independent. Figure 4.2 shows simplexes with dimension $n = 0 \ldots 3$: respectively a node, an edge, a triangle and a tetrahedron.

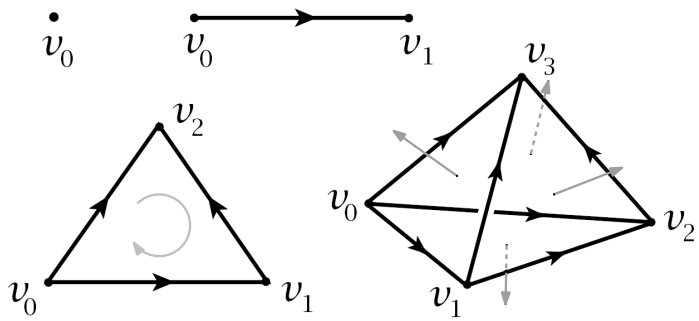


Figure 4.2: *Simplexes with dimension $n = 0 \ldots 3$: node, edge, triangle, tetrahedron and their 'orientation' by (GIS) convention*

Some observations on simplexes:

1. It is assumed that all simplexes are ordered. With any n-simplex, $(n+1)!$ distinct ordered simplexes are associated. Within GIScience, this ordering can be used to orient these simplexes according to specific conventions. Orientation is not a real mathematical characteristic of simplexes, since the concept of orientation is based on arbitrary conventions; nevertheless it enables useful operations within a GIS context, for instance in defining inside and outside of geometries. As a result, one should interpret the term 'orientation' within the scope of this dissertation according to its common meaning within GIScience. Within this framework, one can say that all even numbers of permutations of an arbitrary ordered simplex $S_n =< v_0, \ldots, v_n >$ have similar orientation and all odd numbers of permutations an opposite one. So for instance the following four statements are true:

previous chapter) as data structure due to their strong computational capabilities. The triangulation and tetrahedronisations can be integrated at a conceptual level, as both TIN and TEN use nodes, edges and triangles. However, the actual connection at design and implementation level appeared to be very difficult. Another consequence of selecting a hybrid 2.5D/3D approach is inconsistent feature modelling: the viaduct example illustrated that a highway might be modelled sometimes as a volume and sometimes as a face. This inconsistency is hard to accept.

To overcome these drawbacks, a more rigid approach was designed. The full volumetric approach models everything in a TEN, including air and earth features, thus explicitly including space that is usually perceived as 'empty'. Despite its potential substantial data storage requirements, the full 3D approach is selected as starting point for further research. The next part of this thesis (part II) will focus on a data structure, based on this conceptual model. As data storage is often considered to be a weakness of a TEN data structure (Zlatanova *et al.* 2004), this part will also focus on methods to reduce explicit data storage.

$$
\begin{aligned}
S_0 = \; & < v_0 > \\
S_1 = \; & < v_0, v_1 > = - < v_1, v_0 > \\
S_2 = \; & < v_0, v_1, v_2 > = - < v_0, v_2, v_1 > = < v_1, v_2, v_0 > = \\
& - < v_1, v_0, v_2 > = < v_2, v_0, v_1 > = - < v_2, v_1, v_0 > \\
S_3 = \; & < v_0, v_1, v_2, v_3 > = - < v_0, v_1, v_3, v_2 > = < v_0, v_3, v_1, v_2 > = \\
& - < v_0, v_3, v_2, v_1 > = < v_0, v_2, v_3, v_1 > = - < v_0, v_2, v_1, v_3 > = \\
& < v_2, v_0, v_1, v_3 > = - < v_2, v_0, v_3, v_1 > = < v_2, v_3, v_0, v_1 > = \\
& - < v_2, v_3, v_1, v_0 > = < v_2, v_1, v_3, v_0 > = - < v_2, v_1, v_0, v_3 > = \\
& < v_1, v_2, v_0, v_3 > = - < v_1, v_2, v_3, v_0 > = < v_1, v_3, v_2, v_0 > = \\
& - < v_1, v_3, v_0, v_2 > = < v_1, v_0, v_3, v_2 > = - < v_1, v_0, v_2, v_3 > = \\
& < v_3, v_0, v_2, v_1 > = - < v_3, v_0, v_1, v_2 > = < v_3, v_1, v_0, v_2 > = \\
& - < v_3, v_1, v_2, v_0 > = < v_3, v_2, v_1, v_0 > = - < v_3, v_2, v_0, v_1 >
\end{aligned}
$$

The second line can be read as 'an edge directed from point v_0 to point v_1 has opposite orientation to the edge directed from point v_1 to point v_0. This characteristic can be used to change simplex orientation by performing a transposition (a single permutation), thus eliminating the need of using signed simplex descriptions. Section 4.2 will describe simplex orientation in more detail.

2. A face of S_n is a simplex of lower dimension whose nodes form a non-empty subset of $\{v_0, \ldots, v_n\}$. In other words, a simplex consists of simplexes of lower dimension and these simplexes are defined by some of the points that define the original simplex. For instance, a tetrahedron $S_3 = < v_0, v_1, v_2, v_3 >$ consists of four triangles $< v_1, v_2, v_3 >$, $< v_0, v_2, v_3 >$, $< v_0, v_1, v_3 >$ and $< v_0, v_1, v_2 >$. The formula to derive these less dimensional boundaries will be given in definition 2 below.

3. If the subset is proper (i.e. not the whole of $\{v_0, \ldots, v_n\}$), than the face is called a proper face (Giblin 1977).

4. A n-simplex has in total $2^{(n+1)} - 2$ proper faces. For instance a triangle has six proper faces (three edges and three nodes), while a tetrahedron has 14 proper faces (four triangles, six edges and four nodes).

5. For the number of faces of a specific dimension the following holds: a n-simplex has $\begin{pmatrix} n+1 \\ p+1 \end{pmatrix}$ faces of dimension p with $(0 \leq p < n)$. For instance: a tetrahedron consists of four triangles, six edges and four points.

6. The 0- and 1-dimensional faces (i.e. nodes and edges) of a n-simplex form a complete graph on $n + 1$ nodes.

7. The boundary of a n-simplex is defined by the following sum of $n-1$ dimensional simplexes (Poincaré 1899) (the *hat* indicates omitting the specific node):

Definition 2

$$\partial S_n = \sum_{i=0}^{n} (-1)^i < v_0, \ldots, \hat{v}_i, \ldots, v_n >$$

This results in (see Figure 4.3) the following boundaries (note that only one transposition of S_3 is shown for compactness reasons):

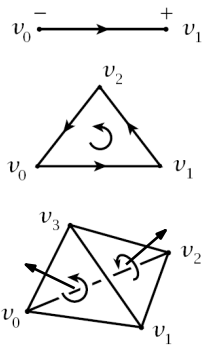
$$\begin{array}{ll}
S_0 =< v_0 > & \partial S_0 = \emptyset \\
 & \\
S_1 =< v_0, v_1 > & \partial S_1 =< v_1 > - < v_0 > \\
S_1 =< v_1, v_0 > & \partial S_1 =< v_0 > - < v_1 > \\
 & \\
S_2 =< v_0, v_1, v_2 > & \partial S_2 =< v_1, v_2 > - < v_0, v_2 > + < v_0, v_1 > \\
S_2 =< v_0, v_2, v_1 > & \partial S_2 =< v_2, v_1 > - < v_0, v_1 > + < v_0, v_2 > \\
S_2 =< v_1, v_0, v_2 > & \partial S_2 =< v_0, v_2 > - < v_1, v_2 > + < v_1, v_0 > \\
S_2 =< v_1, v_2, v_0 > & \partial S_2 =< v_2, v_0 > - < v_1, v_0 > + < v_1, v_2 > \\
S_2 =< v_2, v_0, v_1 > & \partial S_2 =< v_0, v_1 > - < v_2, v_1 > + < v_2, v_0 > \\
S_2 =< v_2, v_1, v_0 > & \partial S_2 =< v_1, v_0 > - < v_2, v_0 > + < v_2, v_1 > \\
 & \\
S_3 =< v_0, v_1, v_2, v_3 > & \partial S_3 =< v_1, v_2, v_3 > - < v_0, v_2, v_3 > \\
 & \quad + < v_0, v_1, v_3 > - < v_0, v_1, v_2 >
\end{array}$$

An interesting case for the boundary operator is *i=0*. The boundary operator will result in an empty collection of simplexes. According to Giblin (1977), such an empty collection is a simplex whose underlying space (the underlying space (sometimes called the carrier) of a simplicial complex (see section 4.3) is the union of its simplexes) is empty. This simplex does not have a dimension, although assigning dimension -1 by convention is popular. Within the scope of this research, definition 2 will be implemented with the additional condition $n > 0$.

It is important to realise that all previous observations are true in any dimension. As a result, simplicial homology definitions and operations are not only applicable to 2D and 3D modelling, but a simplicial homology-based modelling approach can also easily be extended into 4D, thus offering a potential foundation for spatio-temporal modelling. However, this dissertation will focus only on the 3D modelling approach.

4.2 Orientation of simplexes

Observation 1 introduced the concept of ordering of simplexes. Within GIScience, the concept of orientation is based on conventions about this ordering. For a 3D simplex (a tetrahedron) orientation is specified by the direction of the normal vectors of the boundary faces. Normal vectors pointing outwards are denoted positive and normal vectors pointing inwards negative. The direction of these normal vectors is a direct result from the orientation of the triangles, as orientation of 2D simplexes (triangles)



$$\partial[v_0, v_1] = [v_1] - [v_0]$$

$$\partial[v_0, v_1, v_2] = [v_1, v_2] - [v_0, v_2] + [v_0, v_1]$$

$$\partial[v_0, v_1, v_2, v_3] = [v_1, v_2, v_3] - [v_0, v_2, v_3] + [v_0, v_1, v_3] - [v_0, v_1, v_2]$$

Figure 4.3: *Simplexes and their boundaries (Hatcher 2002)*

is specified by the order, in which edges are travelled (observed from outside the tetrahedron). By convention counter clockwise orientation is denoted positive (+) and clockwise orientation negative (−). The normal vector of a triangle is defined as the dot product of two directed edges of the triangle. For edges, orientation is specified in terms of direction: an edge from A to B has opposite orientation to an edge from B to A.

As a simplex S_n is defined by $n+1$ nodes, $(n+1)!$ permutations exist. So, for an edge two permutations exist, for a triangle six and for a tetrahedron 24. All even numbers of permutations of an ordered simplex $S_n =< v_0, \ldots, v_n >$ have the same orientation, all odd numbers of permutations have opposite orientation. So edge $S_1 =< v_0, v_1 >$ has boundary $\partial S_1 =< v_1 > - < v_0 >$. The other permutation $S_1 = - < v_0, v_1 >=< v_1, v_0 >$ has boundary $\partial S_1 =< v_0 > - < v_1 >$, which is the opposite direction. In a similar way the boundaries of the other five permutations of S_2 and the other 23 permutations of S_3 can be given. The six permutations of S_2 are illustrated in figure 4.4. It shows that two groups of three permutations are equivalent to each other, i.e. the three $(1^{st}, 3^{rd}, 5^{th})$ with positive and the three $(2^{nd}, 4^{th}, 6^{th})$ with negative orientation. As a consequence operators like the opposite of a simplex (the opposite of a simplex is the simplex with identical geometry and opposite orientation) become very simple: it only requires a transposition.

This flexibility in handling orientations is a convenient characteristic of simplicial homology, but it brings another favourable characteristic in 3D. S_3 has 24 permutations, 12 with positive and 12 with negative orientation. It holds for all 24 permutations that the four bounding triangles have consistent orientation; either all normal vectors of the triangles point inwards or all normal vectors of the triangles point outwards. This is of course a desired characteristic and it requires no effort at all; as it is a direct result from definition 2. Lemma 3 will prove this consistent orientation by applying the boundary operator twice:

Lemma 3 *Since $\partial^2 S_n = 0$ ('the boundary of the boundary equals zero'), all boundary*

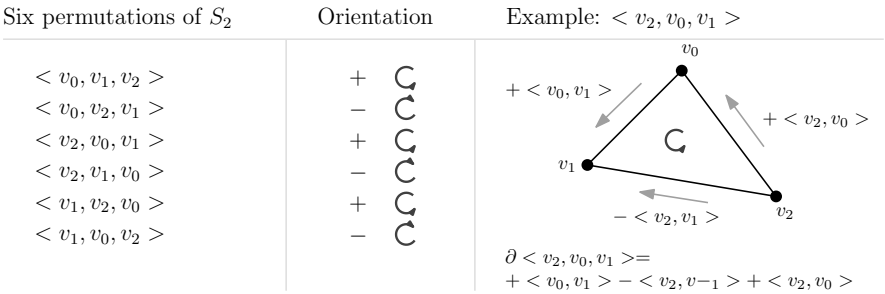


Figure 4.4: *The six permutations of simplex S_2 and their orientation. Permutation $< v_2, v_0, v_1 >$ is illustrated in more detail*

triangles of S_3 have the same orientation.

Proof: First the so-called zero homomorphism (∂^2=0) needs to be proven when applied to any oriented n-simplex. Now:

$$\begin{aligned}\partial^2 S_n = \quad & \partial \textstyle\sum_{i=0}^{n}(-1)^i < v_0, \ldots, \hat{v}_i, \ldots, v_n >= \\ & \textstyle\sum_{i=0}^{n}(-1)^i \sum_{j=i+1}^{n}(-1)^{j-1} < v_0, \ldots, \hat{v}_i, \ldots, \hat{v}_j, \ldots, v_n > + \\ & \textstyle\sum_{i=0}^{n}(-1)^i \sum_{j=0}^{i-1}(-1)^{j} < v_0, \ldots, \hat{v}_j, \ldots, \hat{v}_i, \ldots, v_n >\end{aligned}$$

All terms in this expression cancel in pairs, since each oriented $(n-1)$-simplex $< v_0, \ldots, \hat{v}_i, \ldots, \hat{v}_j, \ldots, v_n >$ appears two times, the first time with sign $(-1)^{i+j-1}$ and the second time with the opposite sign $(-1)^{i+j}$.

Now consider $\partial^2 S_3$. The boundary of a tetrahedron consist of four triangles, and the boundaries of these triangles consist of edges. Each of the six edges of S_3 appears two times, as each edge bounds two triangles. The zero homomorphism states that the sum of these edges equals zero. This is the case if and only if the edges in these six pairs have opposite signs. The edges of two neighbouring triangles have opposite signs if and only if the triangles have the same orientation, i.e. either both are oriented inwards or both are oriented outwards. As this is true for each random combination of two neighbouring triangles, all triangles have consistent orientation. □

4.3 Combining simplexes: simplicial complexes

As most volume features will be represented by more than one tetrahedron (as can be observed in figure 4.5), operations on sets of simplexes will be useful. A simplicial complex is such a combinatorial object of a number of simplexes. A formal definition (Giblin 1977) is given below:

Definition 4 *A simplicial complex C is a finite set of connected simplexes that satisfies the following two conditions:*

- *Any face of a simplex from C is also in C*
- *The intersection of any two simplexes from C is either empty or is a face for both of them (note that 'face' refers to a face in general dimension, as introduced in observation 2 in section 4.1)*

The dimension of C is the largest dimension of any simplex in C (Giblin 1977). A simplicial complex is said to be of homogeneous dimension n if all simplexes of lower dimension than n in C are proper faces (refer to observation 3 in section 4.1) of n-simplexes in C.

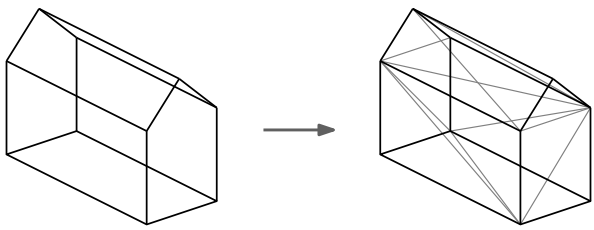


Figure 4.5: *Topographic features will be represented by multiple simplexes: simplicial complexes. In this example a building is represented by eight tetrahedrons*

Up to this point only simplexes and simplicial complexes are discussed. A special case of simplicial complexes in 3D is the Tetrahedronised Irregular Network (TEN).

Definition 5 *A Tetrahedronised Irregular Network (TEN) is a simplicial complex of homogeneous dimension of three that consists of face-connected 3-simplexes, where face-connected indicates that two 3-simplexes are connected through a shared 2-simplex.*

Although this definition is short it implies several important characteristics:

- There are no self-intersections in the TEN (due to the second condition for simplicial complexes in definition 4)
- There are no dangling edges or faces in the TEN (as it is of homogeneous dimension)
- There are no dangling tetrahedrons in the TEN (since these tetrahedrons would not be face-connected)

In addition to the requirements following from this definition, a TEN is also supposed to consist of positively oriented *3*-simplexes within the scope of this dissertation. As

a result, two neighbouring tetrahedrons share a triangle from a geometrical point of view, but due to the positive orientation of both *3*-simplexes, the boundary triangles will have the same geometry, but opposite orientation. Within this dissertation, a triangle with the same geometry but opposite orientation will be referred to as an opposite triangle.

Such a structure contains several topological relationships, thus enabling both topological querying and, perhaps even more important, validation tools in order to maintain data integrity. These operations will be discussed in section 4.4. Note that usually many features (each represented by a set of tetrahedrons, i.e. a simplicial complex) exist within a single TEN. In other words: the complete TEN can be seen as a simplicial complex of homogeneous dimension, but smaller subsets of the TEN as well. An interesting application of the boundary operator, as introduced in definition 2, is joining or merging two simplexes of equal dimension into a simplicial complex. The boundary of this simplicial complex can be derived by adding the boundaries of the separate simplexes. Since volume features will be represented by 3D simplicial complexes, this operation will result in the volume feature boundary when applied to the topographic data model:

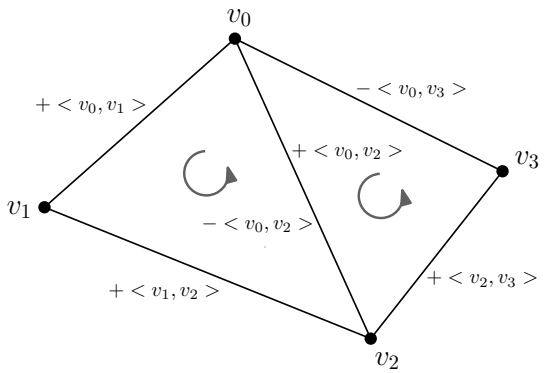
Definition 6 *The boundary of simplicial complex C_n, consisting of m+1 simplexes of dimension n (with m>0), is defined as:*

$$\textit{Simplicial complex } C_n =< S_{n0}, \ldots, S_{nm} > \textit{ has boundary } \partial C_n = \sum_{i=0}^{m} \partial S_{nm}$$

For example, if we join the two neighbouring triangles $S_{20} =< v_0, v_1, v_2 >$ and $S_{21} =< v_0, v_2, v_3 >$ into a 2D complex C_2, adding the boundaries result in (see also figure 4.6) $< v_1, v_2 > + < v_0, v_1 > + < v_2, v_3 > - < v_0, v_3 >$.
Note that the shared boundary $< v_0, v_2 >$ is removed from the boundary description as it appeared once with positive and once with negative sign. This appearance with opposite signs relies on the assumption of consequent orientation of simplexes in the simplicial complexes. As long as this consequent orientation is ensured, the zero homomorphism (see lemma 3) will also apply to simplicial complexes: $\partial^2 C_n = \sum_{i=0}^{m} \partial^2 S_{nm} = 0$.

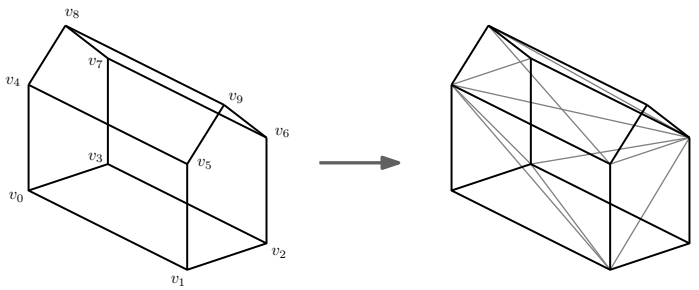
As stated earlier, joining simplexes into simplicial complexes and deriving its outer boundary can be very useful in our modelling approach. If for instance a building is modelled as a set of eight tetrahedrons (see figure 4.7), the building's boundary representation can be obtained by merging the boundaries of all eight tetrahedrons. The 16 triangles of C_3 are the boundary triangulation of this building. This boundary triangulation might be used in the visualisation process. It is already a polyhedron, but if one is interested in a polyhedron with a minimal number of faces, merging boundary triangles with identical (within a tolerance) normal vector direction into flat polygons will result in seven flat boundary faces for this building.

A related concept with respect to topological relationships in a TEN structure is the coboundary, although its definition is not limited to TENs. Intuitively a coboundary is the opposite of a boundary:



$$\begin{aligned}\partial C_2 = \partial S_{21} + \partial S_{22} &= (< v_1, v_2 > - < v_0, v_2 > + < v_0, v_1 >) \\ &+ (< v_2, v_3 > - < v_0, v_3 > + < v_0, v_2 >) \\ &= < v_1, v_2 > + < v_0, v_1 > + < v_2, v_3 > - < v_0, v_3 >\end{aligned}$$

Figure 4.6: *Merging two simplexes into one simplicial complex*



$$\begin{array}{ll}
S_{31} = < v_0, v_1, v_3, v_4 > & \partial S_{31} = < v_1, v_3, v_4 > - < v_0, v_3, v_4 > + < v_0, v_1, v_4 > - < v_0, v_1, v_3 > \\
S_{32} = < v_1, v_2, v_3, v_6 > & \partial S_{32} = < v_2, v_3, v_6 > - < v_1, v_3, v_6 > + < v_1, v_2, v_6 > - < v_1, v_2, v_3 > \\
S_{33} = < v_1, v_3, v_4, v_6 > & \partial S_{33} = < v_3, v_4, v_6 > - < v_1, v_4, v_6 > + < v_1, v_3, v_6 > - < v_1, v_3, v_4 > \\
S_{34} = < v_1, v_4, v_5, v_6 > & \partial S_{34} = < v_4, v_5, v_6 > - < v_1, v_5, v_6 > + < v_1, v_4, v_6 > - < v_1, v_4, v_5 > \\
S_{35} = < v_3, v_4, v_6, v_7 > & \partial S_{35} = < v_4, v_6, v_7 > - < v_3, v_6, v_7 > + < v_3, v_4, v_7 > - < v_3, v_4, v_6 > \\
S_{36} = < v_4, v_6, v_7, v_8 > & \partial S_{36} = < v_6, v_7, v_8 > - < v_4, v_7, v_8 > + < v_4, v_6, v_8 > - < v_4, v_6, v_7 > \\
S_{37} = < v_4, v_5, v_6, v_8 > & \partial S_{37} = < v_5, v_6, v_8 > - < v_4, v_6, v_8 > + < v_4, v_5, v_8 > - < v_4, v_5, v_6 > \\
S_{38} = < v_5, v_6, v_8, v_9 > & \partial S_{38} = < v_6, v_8, v_9 > - < v_5, v_8, v_9 > + < v_5, v_6, v_9 > - < v_5, v_6, v_8 > \quad +
\end{array}$$

$$\begin{aligned}C_3 \qquad & \partial C_3 = - < v_0, v_3, v_4 > + < v_0, v_1, v_4 > - < v_0, v_1, v_3 > + < v_2, v_3, v_6 > \\ & + < v_1, v_2, v_6 > - < v_1, v_2, v_3 > - < v_1, v_5, v_6 > - < v_1, v_4, v_5 > \\ & - < v_3, v_6, v_7 > + < v_3, v_4, v_7 > + < v_6, v_7, v_8 > - < v_4, v_7, v_8 > \\ & + < v_4, v_5, v_8 > + < v_6, v_8, v_9 > - < v_5, v_8, v_9 > + < v_5, v_6, v_9 >\end{aligned}$$

Figure 4.7: *Deriving the boundary triangulation from a simplicial complex*

Definition 7 *The coboundary of a n-dimensional simplex S_n is the set of all $(n+1)$-dimensional simplexes S_{n+1} that are (partially) bounded by S_n, i.e. it is the set of which the simplex S_n is part of their boundaries ∂S_{n+1}.*

With respect to orientation of simplexes, this definition can be interpreted in a strict and in a less strict way. For example, if one ignores orientation, a triangle has three boundary segments (its edges) and two coboundary segments (the adjacent tetrahedrons). But if one applies the concept of orientation strict, a triangle has one coboundary (a tetrahedron), while the second tetrahedron from the previous example will be the coboundary of this triangles opposite (the triangle with identical geometry, but opposite orientation). Note that this is only the case if all tetrahedrons have similar orientation, i.e. all positive or all negative. If this is not the case, a triangle might have one (in case of consistent orientation of two neighbouring tetrahedrons) or two (in case of opposite orientation of two neighbouring tetrahedrons) coboundaries. Although these examples all deal with triangles, coboundary relationships exist in all dimensions, so for instance an edge has two boundary segments (its nodes) and (in a TEN) an unknown number of coboundary segments (adjacent triangles).

4.4 Operations on simplexes and simplicial complexes

In a simplicial complex-based approach, features will be represented by a set of simplexes. As a result, certain operations on features will translate into operations on simplexes. For instance, a point-in-polyhedron test at feature level will be performed in the simplicial complex, as all simplexes are convex. Guaranteed convexity will enable the use of more efficient point-in-polyhedron algorithms.

Another example is the operation to obtain the volume of a building, which will be performed as the sum of the volume calculations of the individual tetrahedrons. The Cayley-Menger determinant (Colins, K.D. 2003) is a determinant that gives the volume of a n-simplex in m-dimensional space. With simplex $S_n =< v_0, \ldots, v_n >$ the $(n+1) \times (n+1)$ matrix $B = (\beta_{ij})$ is given by $\beta_{ij} = |v_i - v_j|^2$, i.e. a symmetrical matrix with the squares of the lengths of the tetrahedron's edges. Matrix $\hat{B}$ is the $(n+2) \times (n+2)$ matrix obtained by bordering matrix B with a top row $(0, 1, \ldots, 1)$ and a left column $(0, 1, \ldots, 1)^T$.

Definition 8 *The volume V of a simplex S_n is given by:*

$$V^2(S_n) = \frac{(-1)^{n+1}}{2^n (n!)^2} det(\hat{B})$$

Since matrix $\hat{B}$ consists of distances between nodes instead of node coordinates, the formula is dimension independent, meaning that it will produce the volume of a n-simplex, irrespective of the dimension of the space in which S_n is located. Note that

matrix $\hat{B}$ is symmetrical! For $n = 2$ (a triangle) and $n = 3$ (a tetrahedron) this results in (with d_{ij} as length of edge $< v_i, v_j >$):

$$n = 2: \ -16V^2 = \begin{vmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & d_{01}^2 & d_{02}^2 \\ 1 & d_{10}^2 & 0 & d_{12}^2 \\ 1 & d_{20}^2 & d_{21}^2 & 0 \end{vmatrix}$$

$$n = 3: \ 288V^2 = \begin{vmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & d_{01}^2 & d_{02}^2 & d_{03}^2 \\ 1 & d_{10}^2 & 0 & d_{12}^2 & d_{13}^2 \\ 1 & d_{20}^2 & d_{21}^2 & 0 & d_{23}^2 \\ 1 & d_{30}^2 & d_{31}^2 & d_{32}^2 & 0 \end{vmatrix}$$

Simplexes offer not only support for calculations, but also for more complex spatial operations like buffer (de Vries 2001) and overlay (van der Most 2004). Verbree *et al.* (2005) describe the possibilities of executing these classic GIS operators on tetrahedrons. Validation is another operation that can be performed on both simplexes and features. At simplicial complex level the validation of a TEN can performed by applying the Euler-Poincaré formula: $N - E + F - V = 0$ with N the number of nodes, E the number of edges, F the number of faces and V the number of volumes (including the exterior). This formula holds not only in three dimensions, but can be generalised into general dimension:

$$\sum_{i=0}^{n} (-1)^{i-1} N_{i-1} = 1 - (-1)^i$$

with N_i the number of i-dimensional simplexes and n the dimension of the simplicial complex plus one. Note that N_{n-1} includes the exterior space! In this formula, n equals the number of different simplex types, as for instance in a 3D simplicial complex four simplex types occur: 0-simplexes, 1-simplexes, 2-simplexes and 3-simplexes. This formula leads to the following dimension-specific formulas for the 0D-3D cases:

$$\begin{aligned} i = 1 \quad & N_0 = 2 \\ i = 2 \quad & N_0 - N_1 = 0 \\ i = 3 \quad & N_0 - N_1 + N_2 = 2 \\ i = 4 \quad & N_0 - N_1 + N_2 - N_3 = 0 \end{aligned}$$

As illustrated in figure 4.8 with a 2D and a 3D example, the Euler-Poincaré formula holds for simplicial complexes in general and not only for simplicial complexes of homogeneous dimension. As a result, dangling edges and faces cannot be detected with this formula. Combining simplex validation results leads to validation on feature level. If one is interested in validating a volume feature, three checks need to be performed:

1. A valid TEN is required, so all criteria implied by definition 5 need to be met and the tetrahedrons should have consequent positive orientation.

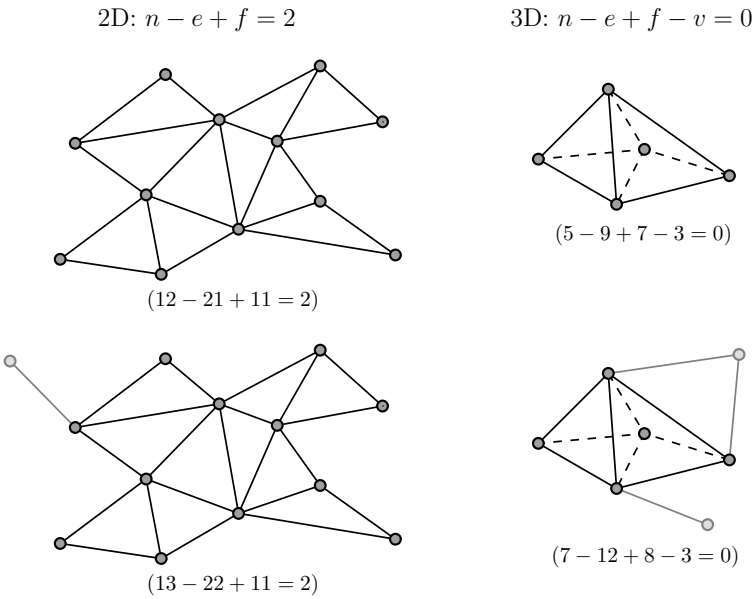


Figure 4.8: *The Euler-Poincaré formula holds for all simplicial complexes; it does not require homogeneous dimension. As a result, dangling edges and faces cannot be detected with this formula*

2. The boundary of the volume feature (represented by a set of constraints) should be watertight.

3. The interior of the volume feature is face connected, thus preventing the creation of two separate volumes that are either disjoint or touch only on edge or node level.

This support for validation of 3D features is an important characteristic of a simplicial complex-based approach, since it simplifies validation significantly, compared to the validation of polyhedrons as described by Kazar *et al.* (2008).

Chapter 5

A Simplicial Complex-based Solution for 3D topography

With the introduction of the simplicial complex-based data structure, this chapter gets down to the very core of the research. Previously the problem field of 3D topography was introduced and requirements for both data model and data structure were derived. At a conceptual level it was decided to model the real world with a full 3D approach. A solid foundation for working with simplexes was found in the mathematical field of simplicial homology. This chapter will present a data structure that takes maximal advantage of the possibilities of simplicial homology, in order to represent the 3D data model such that it meets the requirements derived earlier.

Although the actual insertion of topographic features in the data structure will be the subject of the next chapter, section 5.1 will present the representation of topographic features in the TEN, since this will provide more insight in the presented data structure. In contrast to what one might expect, TEN structures can be applied and implemented in multiple ways. Section 5.2 will present three different conceptual models of a TEN data structure. The third version is based on simplicial homology as introduced in the previous chapter and will be discussed in more detail in section 5.3. This chapter ends in section 5.4 with the actual proof-of-concept implementation as this will demonstrate the synergy between the simplicial homology-based approach and a DBMS implementation.

5.1 Representing topographic features in a TEN

The most important question is how topographic features are represented within the TEN. The UML class diagrams presented in chapter 3 and the updated versions that will be presented in the next section, illustrate the use of constrained edges and triangles to ensure the presence of the feature boundaries in a constrained tetrahedronised irregular network. Section 2.5.1 showed that in the 2D case (a TIN) constrained edges are used to guarantee that polygon boundaries remain present, regardless of other point insertion or edge update operations. In the current 3D case one would like to

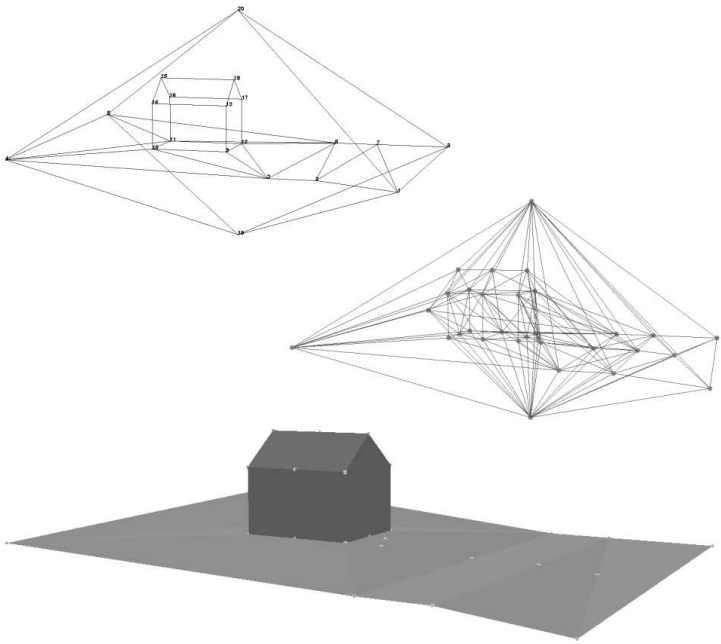


Figure 5.1: *Input data (top), the resulting tetrahedronisation (mid) and as output the constrained triangles (i.e. the feature boundaries)(bottom)*

ensure the presence of boundary faces (section 2.5.3), as they bound the volume features, but unfortunately 3D algorithms still work with constrained edges (Shewchuk 2004). As a result using constraints requires a two-step approach: first one needs to ensure the presence of all constrained edges, second one needs to check whether the required constrained faces are present (i.e. whether they are intersected by other edges) and if necessary, to alter the TEN configuration such that the required faces are present. To further illustrate the concept of representing features in a TEN, the following four steps are required to insert a volume feature in a TEN (Penninga and van Oosterom 2006b):

1. Its outer boundary needs to be triangulated and all resulting edges (and faces) should be treated as constraints

2. The interior needs to be tetrahedronised. This tetrahedronisation can be performed either directly in the complete TEN or separately, after which all resulting edges can be inserted into the TEN. Input in both cases is the set of constraint edges of the outer boundary

3. Regardless which of the two previous options is used in the previous step, local re-tetrahedronisation might be necessary in order to optimise the structure

by creating better-shaped tetrahedrons. Note that constrained edges and constrained triangles will be preserved in this process

4. Updating the relevant feature table(s)

The next chapter will elaborate on update operations in the newly proposed data structure. To illustrate the different steps in the described insertion operation, figure 5.1 shows the input data set, which contains all feature boundaries, the constrained tetrahedronisation and the resulting model (where only feature boundaries are drawn).

5.2 Early ideas: three TEN-based data structures for the full 3D approach

Chapter 3 ended with the selection of a full 3D conceptual data model: all topographic features are considered to be volume features. In this chapter an accompanying data structure will be presented. Three different conceptual models (UML class diagrams) have been created and previously published in (Penninga *et al.* 2006). All three conceptual models are somewhat similar as they all capture the TEN structure and the embedded features. However, there are already some remarkable differences in these conceptual models. At implementation level the impact of these differences can result in substantial differences in for instance storage requirements and performance. These three models can be seen as successors to the original conceptual model of the feature-based TIN/TEN approach as depicted in figure 3.2:

- The first one is given in figure 5.2. VolumeFeatures are represented by a set of Tetrahedrons, AreaFeatures by a set of Triangles and so on. A Tetrahedron is specified by (through the SpecifiedBy association) its four bounding Triangles, a Triangle by its three bounding Edges and an Edge by its two bounding nodes. These associations are bidirectional, thus specifying both the boundaries and the coboundaries of a primitive. All primitives are now directed (positive by convention) and the boundary/coboundary associations are signed. In the UML class diagram this is indicated by the association class Orientation. An efficient implementation of this model will not explicitly include the association class Orientation, but will use signed (+/-) references to encode the orientation. The association between Tetrahedron (or Triangle) and Node can be derived (and gives a correct ordering of the nodes within the primitive). Deriving these nodes is necessary to construct the geometry of a primitive, since geometry is only stored at node level to avoid redundant storage. Redundancy is avoided both to save storage space and to eliminate possible data inconsistencies.
- The second model (figure 5.3) is derived from the first model, but this model differentiates between undirected primitives (useful for reconstructing geometries) and directed primitives (useful for using topology). An undirectedTetrahedron is bounded by four directed Triangles, an undirectedTriangle by three directed Edges and so on. An undirected primitive is composed of a primitive with positive and one with negative orientation. Although this conceptual model may

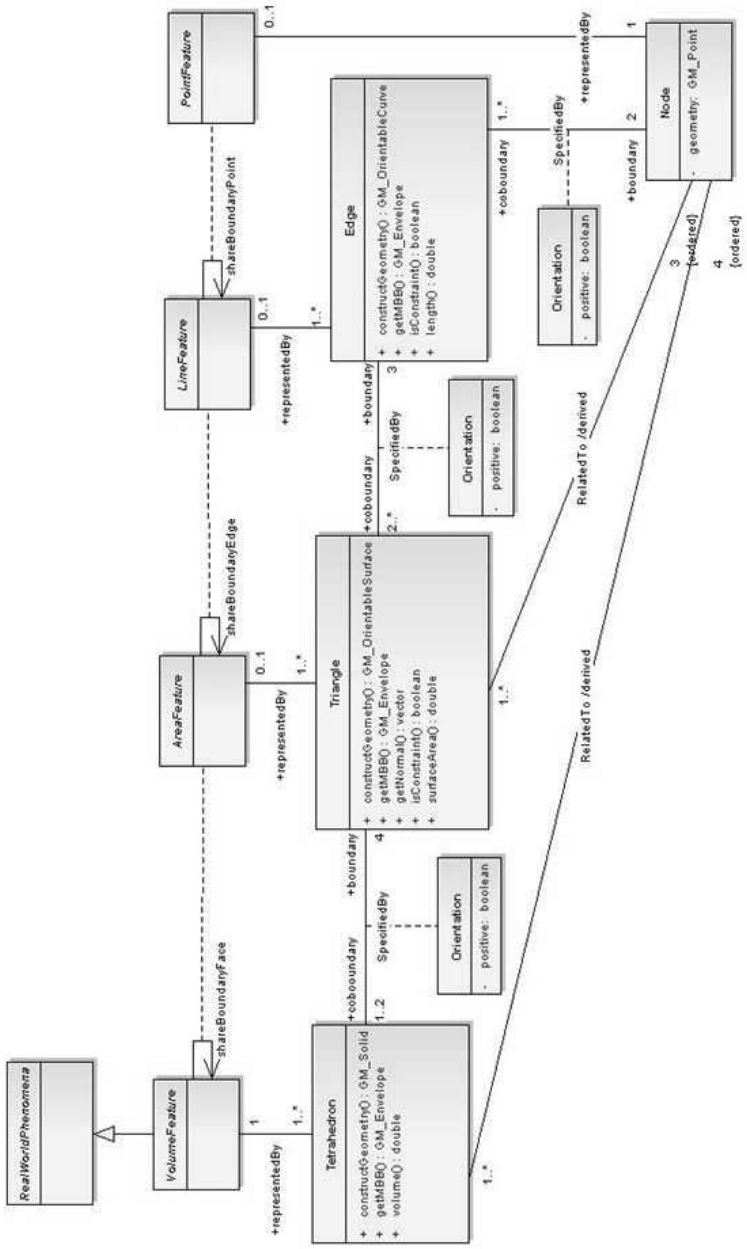


Figure 5.2: *First conceptual model of the feature-based TEN approach*

suggest that both positive and negative versions of the directed primitives are stored, this is not the case in the envisaged implementation: only positively oriented primitives are stored. Similar to the first model (but not depicted in figure 5.3 for clarity reasons), the association between Tetrahedron (or Triangle) and Node can be derived in order to reconstruct geometry and is also ordered.

- Finally, the third model (figure 5.4) is based on definitions and operators from Poincaré simplicial homology, as described in detail in the previous chapter. In simplicial homology simplexes are defined by their vertices. Associations between other simplexes (the boundary/coboundary associations), for instance between Tetrahedrons and Triangles, can be derived by applying the boundary operator and as a result, even Triangles and Edges may be derived. Therefore, triangle and edge data might not be explicitly stored at all, but only derived when needed. The ordering of the nodes to define the primitives is very important as it determines the orientation.

Penninga *et al.* (2006) observe that this last model will be the least redundant with respect to references. Nevertheless, the preliminary implementation as described in their paper is not yet based at the third model, but describes a preliminary version with all elements explicitly stored. In contrast to this, the remaining part of this dissertation will be based on the third model. Therefore the next section will elaborate on this model.

5.3 Preferred solution: applying simplicial homology to the TEN

As shown in the previous chapter, simplicial homology offers a solid theoretical foundation for working with simplexes. Applying this mathematical theory to the TEN approach resulted in a new conceptual model (figure 5.4). Benefits of applying simplicial homology are not limited to providing a sound mathematical base for a data model. It will enable data storage reduction, especially in a DBMS context, as it eliminates explicit storage of both boundary/coboundary associations and some of the simplex types, as these can be derived as well.

5.3.1 A DBMS-based approach for 3D Topography

Due to the utilisation of Poincaré simplicial homology in the TEN structure, substantial parts of the structure can be derived. Explicit storage of tetrahedrons is required, while other data like simplexes of lower dimension and topological relationships can be derived. In a DBMS environment this would translate into storing tetrahedrons in a table, while functions like the boundary operator are used to define views with triangles, edges and nodes. By doing so, the data structure still appears to be a TEN with all 0-, 1-, 2- and 3-simplexes at *conceptual level*, although only tetrahedrons are stored explicitly.

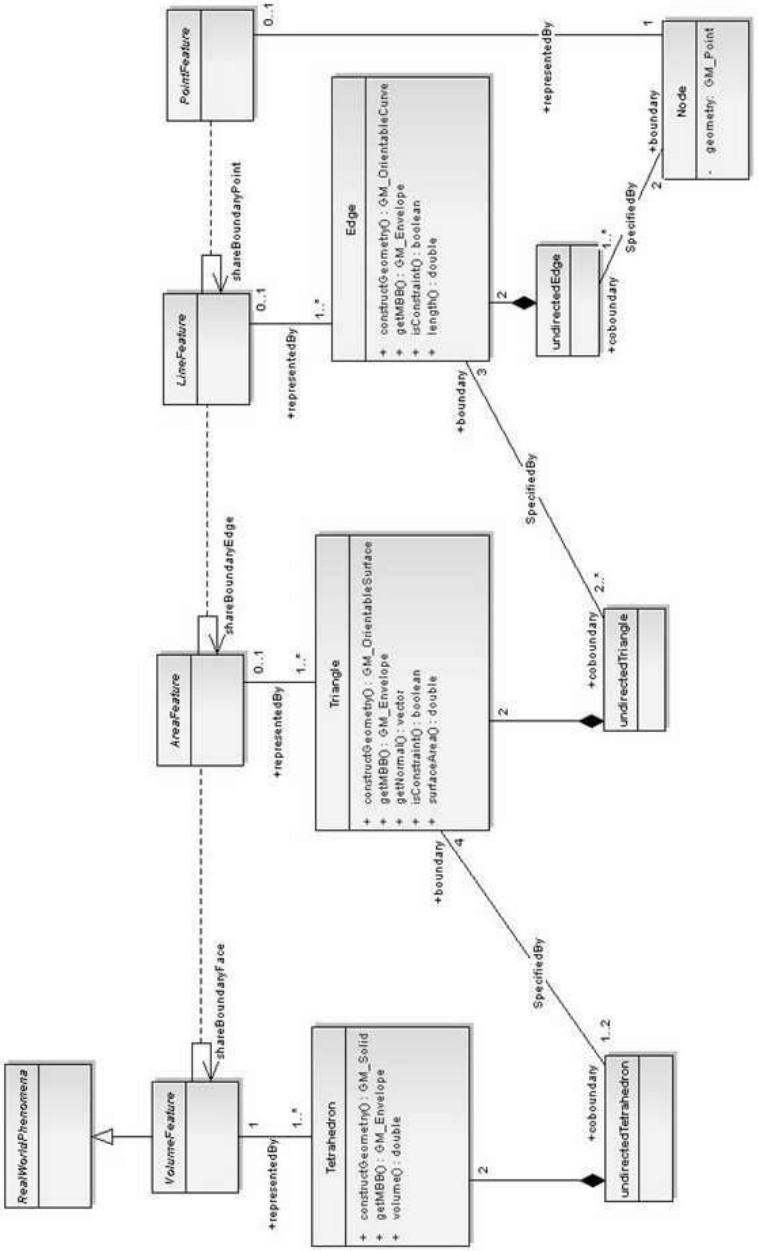


Figure 5.3: *Second conceptual model of the feature-based TEN approach*

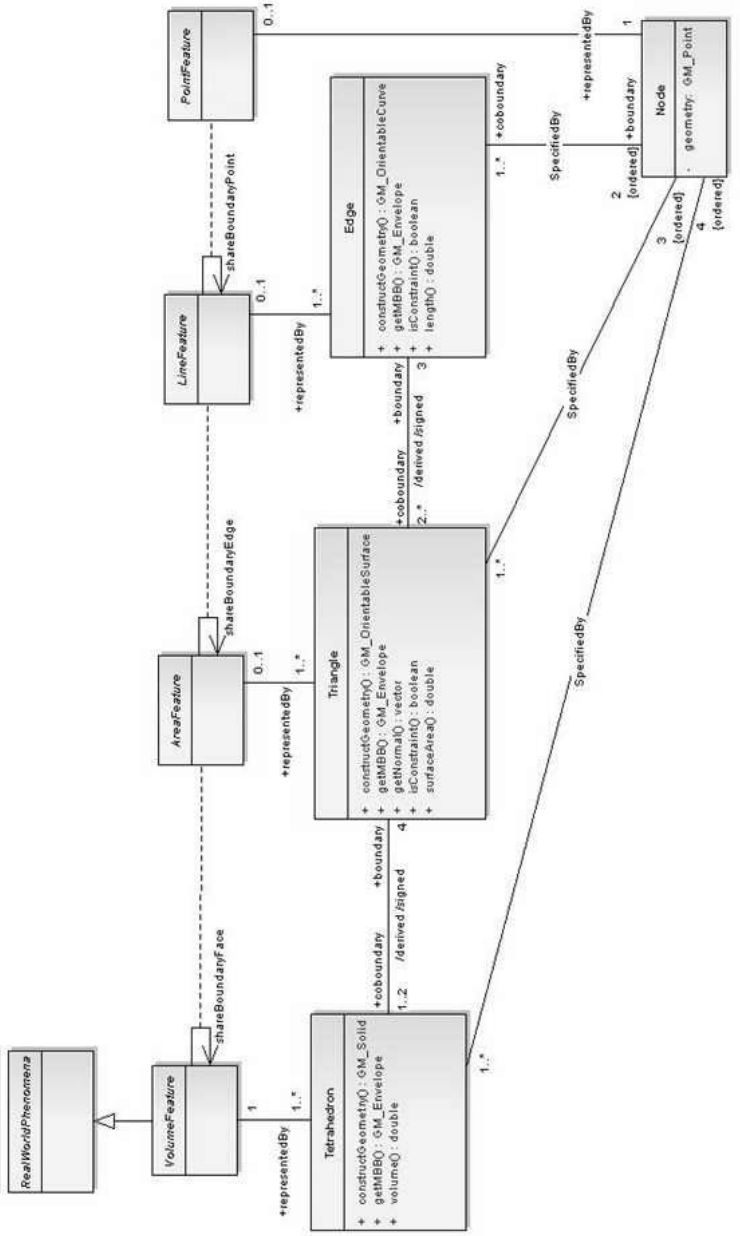


Figure 5.4: *Third conceptual model of the feature-based TEN approach, based on simplicial homology*

As stated earlier in chapter 2, features can be stored in TENs by insertion of constraints. These constraints on edges and triangles ensure the presence of feature boundaries in the TEN structure, i.e. these elements can not be deleted during retriangulation or update processes. Obviously explicit storage of these constraints (the triangulated feature boundaries) needs to be avoided, as it would introduce a separate polyhedron-like data storage besides the TEN structure. Section 5.4 will introduce a technique which enables deriving the constrained triangles and edges. As a result the DBMS data structure consists of a single (tetrahedron) table and several views (see figure 5.5). In terms of explicit storage this means that the presented TEN approach

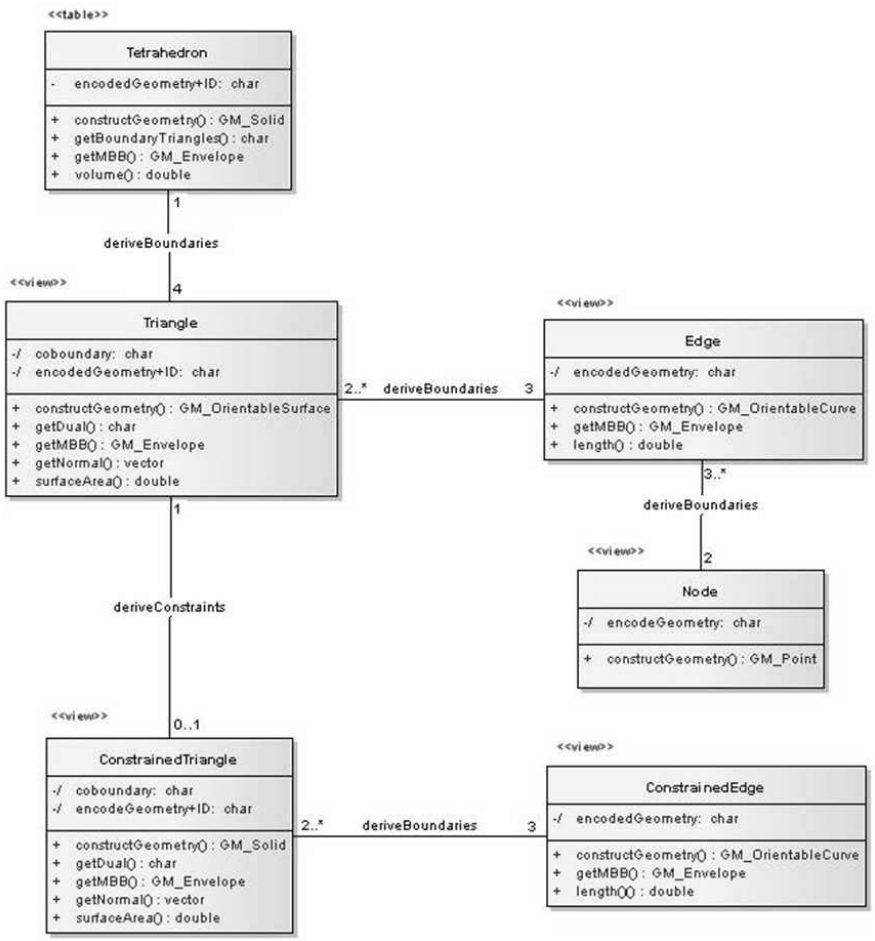


Figure 5.5: *Logical model fitting with the conceptual model from figure 5.4*

is relatively compact. Consider the case of the building in figure 5.6. In a polyhedron approach it would be described by its seven faces. Implicitly these seven faces also

define the enclosed volume and the edges and nodes. In a classic TEN approach the same building would require a lot more components, as both tetrahedrons, triangles, edges and nodes would be stored. However, in the TEN approach presented in this dissertation, only eight tetrahedrons are stored explicitly. As a result, a comparison like the one in section 3.2.3 can be made. Based on such a comparison (see table 5.3.1), one might expect storage requirements to be in the same order of magnitude as a polyhedron approach since the amount of stored primitives is in the same order of magnitude.

Building as polyhedron	**Building as explicit TEN**	**Building as new TEN**
(1 volume)	8 tetrahedrons	8 tetrahedrons
7 faces	24 triangles	(24 triangles)
(15 edges)	25 edges	(25 edges)
10 points	10 nodes	(10 nodes)

Table 5.1: *Extending table 3.1: Intuitive comparison of storage requirements of the polyhedron, explicit TEN and new TEN approach for the building in figure 4.7. The brackets indicate implicit presence (as opposite of explicit storage). It appears that the polyhedron and new TEN approach do not differ much.*

Notwithstanding the fact that compactness is one of our goals, our approach still contains some redundancy as the coordinates of a vertex will be encoded in several tetrahedron codes (the implementation of the simplex notation from simplicial homology: $S_3 =< v_0, v_1, v_2, v_3 >$. However, the delicate balance between compactness and manageability needs fine-tuning. Compactness is not the only advantage of deriving substantial parts of the structure. Another favourable characteristic is that updates are relatively easy, as only the tetrahedron table needs to be updated. Any changes in less dimensional simplexes or topological relationships are derived automatically. Especially these automatic updates in the topological relationships are a significant advantage, as it implies that one can benefit from the presence of topology without the need to maintain it. The availability (in views) of for instance neighbour relationships between tetrahedrons or the relationship between triangles and the tetrahedrons they are part of, enables the user to use these topological relationships as if they were stored explicitly.

5.3.2 Two variants in simplex encoding

In the Poincaré-TEN approach to 3D topographic data modelling, simplexes are defined by their vertices. Identical to the simplex notation from simplicial homology, where for instance a tetrahedron is denoted as $S_3 =< v_0, v_1, v_2, v_3 >$, simplex identifiers are constructed by concatenating the vertex identifiers. By doing so, unique identifiers exist that contain orientation information as well, since the order of vertices determines the orientation. Two different approaches can be distinguished:

- The *coordinate concatenation* (Penninga and van Oosterom 2007) uses the concatenation of the x, y and z coordinate as node ID. Since geometry is the only attribute of a vertex, adding a unique identifier to each point and building an index on top of this table will cause a substantial increase in data storage. The geometry in itself will be a unique identifier. Concatenating the coordinate triplets into one long identifier code (by bitwise interleaving, a Morton code-like approach, see section 7.6.1 for more details) and sorting the resulting list, will result in a basic clustering and indexing. These promising ideas will be described in more detail in section 7.6.1. In a way, this approach can be seen as building and storing an index, while the original table is deleted. As a result this approach minimises the number of references in the data structure.

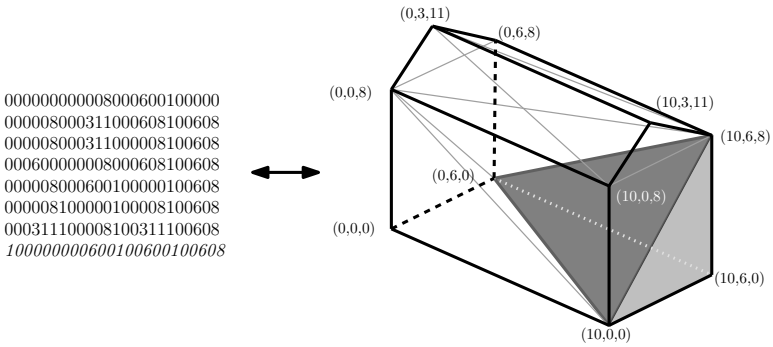


Figure 5.6: *Describing tetrahedrons by their encoded vertices. The shaded tetrahedron is described by the last row*

Figure 5.6 illustrates the coordinate concatenation. A house is tetrahedronised and the resulting tetrahedrons are coded as concatenation of their four vertices' coordinates. Each row in the tetrahedron encoding can be interpreted as $x_1y_1z_1x_2y_2z_2x_3y_3z_3x_4y_4z_4$. For reasons of simplicity, only two positions are used for each coordinate element. Therefore the last row should be interpret as the tetrahedron defined by the vertices $(10, 00, 00)$, $(00, 06, 00)$, $(10, 06, 00)$ and $(10, 06, 08)$, which is the grey coloured tetrahedron at the bottom right of the house.

- The *identifier concatenation* (Penninga and van Oosterom 2008b) is an alternative approach, developed after questioning whether the coordinate concatenation approach is really reducing storage requirements. Since each node is part of multiple tetrahedrons (the Rotterdam test data set described in (Penninga and van Oosterom 2008b) and in chapter 7 shows an average of about fifteen tetrahedrons per node), the concatenated coordinate triplet is used multiple times. As long as a node identifier requires considerably less storage space compared to this concatenated geometry, switching to a identifier concatenation approach might be feasible. An additional node table containing a node identifier is re-

quired to create shorter simplex codes like $nid_1 nid_2 nid_3 nid_4$.

However, reducing data storage might deteriorate performance, as additional operations are necessary to perform geometrical operations on top of simplexes. If one thinks for instance of the operation that checks whether a tetrahedrons is oriented positively or negatively, one needs the node coordinates to calculate a normal vector on one of the triangles and calculate the angle between this normal vector and a vector from the triangle to the opposite, fourth node to determine whether the normal points inwards or outwards. To perform this operation in the tetrahedron-node implementation, one has to search the node table first to obtain the node geometries. Another drawback is that applying spatial clustering and indexing based on these concatenations is not that easy anymore and as a result, an explicit R-tree might be needed. Storing this additional index might even undo the intended data storage reduction.

5.4 Implementing the data structure in a DBMS environment

5.4.1 Creating the data structure

The described implementation here uses coordinate concatenation and, as a result, only a single table is used (see figure 5.5). The data structure is implemented within the Oracle DBMS and this section will show SQL and PL/SQL code examples. The appendices I and II provide more details on respectively the functions and procedures and the creation of the data structure. Although this implementation uses the Oracle DBMS, all ideas can be implemented in other DBMSs as well.

In the coordinate concatenation implementation, the tetrahedron table is the base table. It consists of a single column in which the encoded tetrahedrons are described in the form $x_1 y_1 z_1 x_2 y_2 z_2 x_3 y_3 z_3 x_4 y_4 z_4 oid$. Note that –besides the concatenation of the four encoded vertices– also an unique object identifier is added, which describes which volume object is (partly) represented by the tetrahedron: $S_3 =< v_0, v_1, v_2, v_3, oid >$.

```
create table tetrahedron(tetcode NVARCHAR2(300));
```

The previous chapter showed that internal boundaries in a simplicial complex cancel out in pairs due to their appearance with opposite signs. However, since a tetrahedron is described by its four vertices, there are 4! (equals 24) permutations of this single tetrahedron. To gain control over which permutation is used, the procedure `sortalltet` rewrites each tetrahedron $< a, b, c, d, oid >$ such that $a < b < c < d$ holds, which is an arbitrary criterion. For two nodes a and b $a < b$ holds if $x_a < x_b$. In case of equal x-coordinates the additional criterion $y_a < y_b$ will be used. In case both x- and y-coordinates are equal, $z_a < z_b$ will be the decisive criterion. As each node in a TEN is unique, these z-coordinates will differ.

Another convention is that each tetrahedron in the tetrahedron table has positive orientation, i.e. all normal vectors of the bounding triangles are oriented outwards.

The procedure `outwardsalltet` ensures that the tetrahedrons meet this criterion. In the `checkorientation` procedure each tetrahedron's orientation is checked, as it compares the direction of the normal vector of one of the boundary triangles with a vector from this triangle to the fourth (opposite) point of the tetrahedron. In case of an inward orientation a transposition is carried out by the procedure `permutation34`, which permutes the third and fourth vertex: `permutation34`$(- < v_0, v_1, v_2, v_3, oid >)$ results in $< v_0, v_1, v_3, v_2, oid >$. The ordering now meets the criterion $a < b < d < c$.

```
create or replace procedure outwardsalltet
   (...)
   checkorientation(tetcode,bool);
   if (bool = 0) then
      permutation34(tetcode,newtetcode);
      update tetrahedron set tetcode=newtetcode
      where current of tetcur;
   (...)
```

The overall result of sorting and orienting all tetrahedrons as described, is that each tetrahedron will be described with its smallest positive tetrahedron code. This is a unique identifier indicating which of the 24 permutations will be used. Storing tetrahedrons according to these rules, ensures that benefits from simplicial homology are maximised.

Based on the encoded tetrahedrons, the boundary triangles can be derived by applying the boundary operator from definition 2 in section 4.1. The procedure to derive the four boundary triangles of a tetrahedron looks like (for reasons of clarity, this code is simplified, but the full code is in appendix I):

```
create or replace procedure deriveboundarytriangles(
   (...)
is
   node_array narray := narray();
begin
   Split(tetcode,'x',node_array);
   a := node_array(1)||node_array(2)||node_array(3);
   b := node_array(4)||node_array(5)||node_array(6);
   c := node_array(7)||node_array(8)||node_array(9);
   d := node_array(10)||node_array(11)||node_array(12);
   oid := node_array(13);
   ordertriangle(codelength,'+'||b||c||d||oid, tricode1);
   ordertriangle(codelength,'-'||a||c||d||oid, tricode2);
   ordertriangle(codelength,'+'||a||b||d||oid, tricode3);
   ordertriangle(codelength,'-'||a||b||c||oid, tricode4);
   (...)
```

Note that the triangles inherit the object id from the tetrahedron, i.e. each triangle has a reference to the object which is represented by the tetrahedron of which the triangle

is a boundary. In order words, the boundary operator (see definition 2 in section 4.1) is slightly altered to derive the boundary triangles from $S_3 =< v_0, v_1, v_2, v_3, oid >$:

$$\partial S_3 = \sum_{i=0}^{3} (-1)^i < v_0, \ldots, \hat{v_i}, \ldots, v_3, oid >$$

It can also be seen that each boundary triangle is ordered by the `ordertriangle` procedure. The objective of this procedure is to gain control over which permutation is used. A triangle has six (3!) permutations, but it is important that both in positive and negative orientation the same permutation is used, as they will not cancel out in pairs otherwise (as described in section 4.3 on simplicial complexes). The procedure `ordertriangle` rewrites a triangle $< a, b, c, oid >$ such that $a < b < c$ holds (similar to the `ordertetrahedron` procedure described before), which is again an arbitrary criterion. For example: `ordertriangle`(+014035012022035012014035018003) results in -014035012014035018022035012003, as its input (+ 014035012 022035012 014035018 003) does not satisfy the criterion $a < b < c$, so the second and third term are permutated and the odd permutation causes a sign change. One might expect that this ordering of triangles is not necessary due to the tetrahedron ordering, as performed earlier. However, this is not the case, as the need for consequent orientation (all tetrahedrons have positive orientation) causes tetrahedron codes that are not strictly ordered anymore due to the permutation.

Slightly altered versions of the `deriveboundarytriangles` procedure are used to create the triangle view. The modified procedures separately derive respectively the first, second, third and fourth boundary triangle of a tetrahedron. The resulting view contains all triangles and their coboundaries (see definition 7). In this case, the coboundary is the tetrahedron of which the triangle is part of the boundary. This coboundary will prove useful in deriving topological relationships later in this section. The view is defined as:

```
create or replace view triangle as
   select deriveboundarytriangle1(tetcode) tricode,
   tetcode fromtetcode from tetrahedron
   UNION ALL
   select deriveboundarytriangle2(tetcode) tricode,
   tetcode fromtetcode from tetrahedron
   UNION ALL
   select deriveboundarytriangle3(tetcode) tricode,
   tetcode fromtetcode from tetrahedron
   UNION ALL
   select deriveboundarytriangle4(tetcode) tricode,
   tetcode fromtetcode from tetrahedron;
```

The resulting view will contain four times the number of tetrahedrons, and every triangle (except for triangles on the outer boundary of the tetrahedronisation) appears two times: once with positive and once with sign negative sign (and not in a permutated form, due to the `ordertriangle` procedure). Although each signed

triangle should be unique, the `union all` statement is used to enable detection of non-uniqueness.

In a similar way the views with edges and nodes can be constructed. In the current implementation edges are undirected and do not inherit object id's, as no application for this is identified at the moment. However, strict application of the boundary operator would result in directed triangles. The edge and node views are defined as:

```
create or replace view edge as
   select distinct deriveabsboundaryedge1(tricode) edcode
   from triangle
   UNION
   select distinct deriveabsboundaryedge2(tricode) edcode
   from triangle
   UNION
   select distinct deriveabsboundaryedge3(tricode) edcode
   from triangle;

create or replace view node as
   select distinct deriveboundarynode1(edcode) nodecode
   from edge
   UNION
   select distinct deriveboundarynode2(edcode) nodecode
   from edge;
```

In contrast with the triangle view creation, these `create` statements use `select distinct` and `union` statements, whereas the triangle view was created by a combination of a `select` and `union all` statements. This difference results from the fact that each signed triangle should be unique, whereas an unsigned edge will bound several triangles. Since one cannot predict whether these multiple occurrences happen within the select statement or within the union statement, both operations need to filter out these multiple occurrences.

With the tetrahedron table and triangle, edge and node views the data structure is accessible at different levels. Another characteristic of this approach is that both geometry and topology are present at every level, so one can determine for each operation whether a geometrical (i.e. using the simplex coordinates) or a topological (i.e. using references to boundaries, see section 5.4.3 for more details) approach is the most appropriate.

5.4.2 Deriving constraints (i.e. feature boundaries)

As mentioned at the end of section 5.1, features in the model are represented by a set of tetrahedrons. To ensure that these tetrahedrons represent the correct geometry, the outer boundary is triangulated and these triangles are used as constraints. This implies that these triangles will remain present as long as the feature is part of the model (i.e. they are not deleted in an update or quality improvement process). To achieve this, the incremental tetrahedronisation algorithm needs to keep track of these

constrained triangles. Obviously, explicit storage of constraints has to be avoided, as this would lead to double storage of features. Due to the inherited object identifiers, these constraints can be derived as well:

```
create or replace view constrainedtriangle as
   select t1.tricode tricode from triangle t1
   where not exists (select t2.tricode from triangle t2
                     where t1.tricode = t2.tricode*-1);
```

This statement uses the characteristic that although every triangle (in a geometric sense) appears twice (with opposite orientation) in the triangle view, not every triangle code appears twice. Boundary triangles are unpaired, since in this case the triangle code will differ due to the different inherited object ids. In case of internal triangles (i.e. within an object) the triangle and its opposite (as introduced in section 4.3) will have (apart form the sign) the exact same triangle code (geometry + object id). Deriving constrained edges from constrained triangles is easy, as all boundary edges from constrained triangles are constrained edges.

5.4.3 Deriving topological relationships

In a TEN the number of possible topological relationships between simplexes is limited. As the TEN can be considered as a decomposition of space, relationships like overlap, cover or inside do not occur. Only relationships based on the interaction between tetrahedron boundaries occur. Tetrahedrons (and their boundaries) are either disjoint or touch. Three different types of the topological relationship touch can be distinguished:

- neighbouring tetrahedrons touch in only one node
- neighbouring tetrahedrons touch in only one edge
- neighbouring tetrahedrons touch in one triangle

The third case is the definition of a true neighbour relation between two tetrahedrons. The other two cases are less important. As the neighbour relation is very important in certain operations and algorithms, two related relationships are derived in views in the implementation. The first is the relationship between a triangle and its opposite. This relationship is important in the process of finding neighbours from tetrahedrons. In this context, one can consider a triangle and its opposite as halftriangles, analogous to halfedges (Mäntylä 1988). Obviously triangles at the outer boundary of the tetrahedronisation will not have an opposite triangle. The view is created by a select statement that uses the identical geometric part of the triangle codes:

```
create or replace view oppositetriangle as
   select t1.tricode tricode, t2.tricode oppositetricode
   from triangle t1, triangle t2
   where removeobjectid(t2.tricode) =
    -1 *removeobjectid(t1.tricode);
```

By combining the triangle view and the oppositetriangle view, neighbouring tetrahedrons can be found:

```
create or replace function getneighbourtet1(
(...)
   select fromtetcode into neighbourtet from triangle
   where removeobjectid(tricode) = -1 *removeobjectid(tricode);
(...)
```

and based on functions like this one the view with tetrahedrons and their neighbours can be created. Analogue to this approach topological relationships at feature level can be derived.

5.5 Summary

This chapter introduced the simplicial complex-based solution for 3D topography. Simplicial homology, as introduced in chapter 4, was applied to the full 3D data model, as selected in chapter 3. The resulting simplicial complex-based data structure requires only explicit storage of tetrahedrons, while simplexes of lower dimensions (triangles, edges, nodes), constraints and topological relationships can be derived in views. The next chapter demonstrates the capabilities for updating of the simplicial complex-based data structure, which is an important prerequisite for the data structure to be feasible for 3D topography.

Chapter 6

Updating features in the Data Structure

The previous chapter focussed on the simplicial complex-based data model and its implementation within a database management system. Applying definitions and operators from the mathematical field of simplicial homology helped creating a compact data structure with a minimal amount of redundancy. This result is the desired solution for the first important criterion of 3D topographic data (as introduced in section 2.3.2): switching from 2D to 3D representations of topographic features will lead to a substantial increase in data volume and thus, the new data structure has to reduce this volume as much as possible, without compromising the benefits of using a TEN structure. However, section 2.3.2 also introduced a second important characteristic: topographic data sets will be updated on a regular basis. Topographic features like buildings and roads can be build, extended, demolished and sometimes even moved, thus resulting in the need for updates in the data set. This chapter focuses on the operators that are required to perform these updates. Two basic operators can be distinguished: the insertion of features and the deletion of features (as moving features can be seen as a combination of deletion and insertion). Since the modelling approach assumes a full space partition, inserting a feature automatically includes either feature deletion or feature resizing to make room for the new feature. The same holds for feature deletion: the space previously occupied by the deleted feature has to be assigned to another feature.

The emphasis of this chapter is on incremental updates, since repeated rebuilds of the TEN structure will be time-consuming due to the expected data volume. Besides that, topographic data sets cover a large area and since local changes will (most likely) cause only local alterations, full rebuilds will be superfluous. Section 6.1 focuses on the different steps in the process of feature insertion. Its reverse operator, feature deletion, is described afterwards in section 6.2. Since the described insertion and deletion operators are designed to act as locally as possible, one cannot always guarantee optimal TEN quality (which would have been the case if refined Delaunay tetrahedronisations were used, section 2.5.3). Therefore a set of quality update operations is suggested in section 6.3, which can be used on a regular basis to ensure

well-shaped (i.e. no small angles, thus avoiding numerical instability) tetrahedrons and acceptable data volume. Although most updates will be handled incrementally, bulk loading of data is still useful, both for initial creation of a topographic data set and for quality improvement by TEN optimisation during a periodical rebuild. Therefore section 6.4 will elaborate on bulk loading of features.

6.1 Incremental update: feature insertion*

The previous chapter started (in section 5.1) with describing feature representation in the TEN data structure. The presence of feature boundaries in the TEN is ensured by inserting them as constraints. The overall incremental feature insertion procedure can be subdivided into four steps:

1. Triangulating feature boundaries
2. Inserting constrained edges
3. Ensuring presence of constrained faces
4. Modelling feature interiors

This section describes each step into more detail.

6.1.1 Motivation

This section (more specifically, section 6.1.3) will introduce the feature insertion process, including a new approach for the insertion of constrained edges in a constrained TEN. Alternative approaches do not directly insert the constrained edge, but insert only the nodes and try to perform edge recovery. The process of recovering missing edges is described by Cavalcanti and Mello (1999) and covers stitching, a process that is based on adding nodes on midpoints of missing constrained edges. This idea uses the property of the Delaunay triangulation that each node is connected to the closest node and by adding nodes on midpoints one tries to make these additional nodes the closest nodes. Shewchuk (1997b) also uses this approach. However, Cavalcanti and Mello (1999) state that this process does not always converge, neither in 2D nor in 3D. As a result, no proofs are known to the author that edge recovery will always be successful.

Another approach to edge recovery is given by Liu and Baida (2000), whose approach is based on flipping. They use four types of flipping, namely T_{23}, T_{32}, T_{22} and T_{44}, following the notation of Joe (Joe (1995)). These flips were previously discussed in chapter 2 and illustrated in figure 2.26. Although these operators are relatively simple, the overall process might require a lot of time before all edges will be present and no guarantees that the process will be successful, are known to the author.

*This section is partially based on Penninga and van Oosterom (2006b) and the accompanying Technical Report (Penninga and van Oosterom 2006a)

In order to demonstrate the usability of the simplicial complex-based approach for 3D topographic purposes, one need to guarantee successful update operations, since the frequent updates are a key characteristic of topography. Therefore the described approach might not excel in terms of a minimised increase in TEN elements, optimal shapes of the resulting tetrahedrons or speed of the update, but this approach will provide an alternative method in cases in which edge recovery fails. As a result, it guarantees the success of feature updates.

6.1.2 Step 1. Feature boundary triangulation: calculating constraints

If one wants to insert a volume feature, for instance a building, one needs to ensure correct geometrical representation in the TEN by enforcing the boundary triangles to be present in the structure. This is not trivial, as the basic input of either triangulation or tetrahedronisation algorithms is a set of nodes. Usually the algorithm determines –based on certain rules (for instance the well-known Delaunay criterion)– which nodes will be connected by edges and therefore one cannot be sure that two specific nodes will be connected by an edge or that three specific nodes will form a triangle. As a result one cannot represent a specific shape into either a triangulation or a tetrahedronisation by inserting only its nodes. To overcome this drawback constrained edges were introduced. The constraints on these edges safeguard these edges from being removed by the algorithm in order to meet certain criteria, like the Delaunay criterion or other quality parameters. To insert a feature into the TEN structure, first its boundary has to be triangulated. The resulting triangulation delivers two input sets for the subsequent steps of the feature insertion procedure. First the resulting edges will serve as input set of constrained edges in the TEN structure. Since insertion of these edges will prove insufficient to guarantee presence of the feature boundary in the TEN, the triangles from the boundary triangulation are important input for step 3 of the procedure.

6.1.3 Step 2. Inserting constrained edges: nine unique cases

As seen in the previous section, the complete boundary of the volume feature needs to be inserted into the TEN structure. This boundary is translated into a set of constrained edges and these edges need to be inserted. Edge insertion operators are more complex than just repeating a node insertion operator twice, as one also needs to ensure that these start and end node are connected by an edge. First the node insertion operators will be introduced, followed by the more complex edge insertion operators.

The following annotations will be used to distinguish the operators: I_n for node insertion, with $n = 0 \ldots 3$, indicating whether the newly inserted node lies on an existing node ($n = 0$) or in the interior of an edge ($n = 1$), triangle ($n = 2$) or tetrahedron ($n = 3$). In other words, n equals the dimension of the simplex on which (in case $n = 0$) or in which interior (in case $n > 0$) the new node lies. A similar annotation is used for the operators to insert an edge in a tetrahedron: I_{ij}, where i

and j represent the location of the first and the second node. For instance I_{23} inserts an edge, of which the first node lies in the interior of a triangle and the second one in the interior of a tetrahedron.

Node insertion operators

The basic node insertion operators are introduced by Penninga *et al.* (2006). Four types of node insertion can be distinguished, depending on the exact location of the inserted node:

I_0: The new node lies on a node: in this case the node is already present in the TEN structure, thus no updates are required.

I_1: The new node lies in the interior of an edge: n tetrahedrons are involved and the TEN grows with +1 node, +(n+1) edges, +2n triangles and +n tetrahedrons, as illustrated in figure 6.1 for the case n=4.

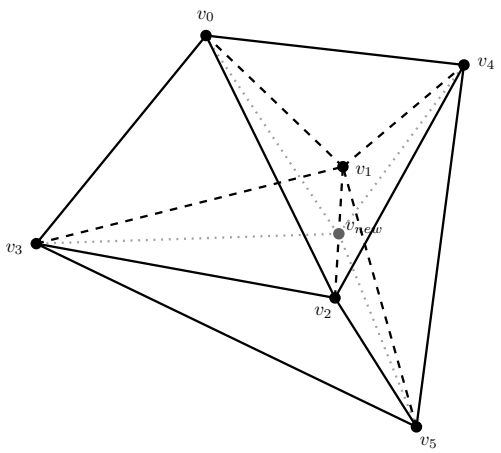


Figure 6.1: *I_1: the new node v_{new} is inserted in the interior of edge $< v_1, v_2 >$ ($n = 4$). Each tetrahedron is split into two new tetrahedrons*

I_2: The new node lies in the interior of a triangle: two (adjacent) tetrahedrons are involved and the TEN grows with +1 node, +5 edges, +8 triangles and +4 tetrahedrons, as illustrated in figure 6.2.

I_3: The new node lies in the interior of the tetrahedron: one tetrahedron is involved and the TEN grows with +1 node, +4 edges, +6 triangles and +3 tetrahedrons, as illustrated in figure 6.3.

Table 6.1 summarises the results of the node insertion operators. Note that the increased numbers of TEN elements respect the Euler-Poincaré formula $N-E+F-V = 0$, as described in section 4.4. $I_0$is not a real case, since this node is already present

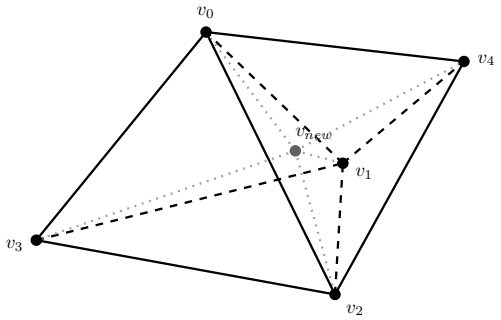


Figure 6.2: *I_2: the new node v_{new} is inserted in the interior of triangle $< v_0, v_1, v_2 >$. The two adjacent tetrahedrons are each replaced by three new tetrahedrons. Triangle $< v_0, v_1, v_2 >$ is thus replaced by three new triangles.*

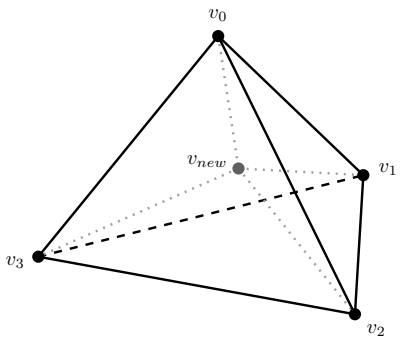


Figure 6.3: *I_3: the new node v_{new} is inserted in the interior of tetrahedron $< v_0, v_1, v_2, v_3 >$. This tetrahedron is replaced by four new tetrahedrons.*

	Node	**Edge**	**Triangle**	**Tetrahedron**
(I_0)	(+0)	(+0)	(+0)	(+0)
I_1	+1	$+n$+1	$+2n$	$+n$
I_2	+1	+5	+8	+4
I_3	+1	+4	+6	+3

Table 6.1: *Overview of all node insert operations with the increases in TEN size*

in the structure. Each of the three resulting true cases can be described nicely by the use of Poincaré simplicial homology, as introduced in chapter 4 and applied in the previous one. Since only tetrahedrons (or tetrahedrons and nodes, depending on which implementation one prefers) are explicitly stored, updates in edges and triangles are implicit. For instance in the case of inserting node v_{new} in the interior of tetrahedron

Node lies on	**Node**	**Edge**	**Triangle**	**Tetrahedron**
Node	I_{00}	I_{01}	I_{02}	I_{03}
Edge	(I_{01})	I_{11}	I_{12}	I_{13}
Triangle	(I_{02})	(I_{12})	I_{22}	I_{23}
Tetrahedron	(I_{03})	(I_{13})	(I_{23})	I_{33}

Table 6.2: *4 x 4 = 16 theoretical cases for the insertion of an edge in a tetrahedron, of which 9 are unique (and I_{00} is no insertion, since this edge would already exist)*

$< v_0, v_1, v_2, v_3 >$ this tetrahedron needs to be deleted from the tetrahedron table and the four newly created tetrahedrons ($< v_0, v_1, v_2, v_{new} >, < v_0, v_2, v_3, v_{new} >, < v_0, v_1, v_3, v_{new} >$ and $< v_1, v_2, v_3, v_{new} >$ need to be added to the table. Each of these four new tetrahedrons can be calculated by replacing one of the original nodes with v_{new}. Note that the actual number of new triangles might differ from the numbers given in table 6.1, since the implementation as described in the previous chapter uses the concept of triangles and opposite triangles (or 'halftriangles', see section 4.3). However, since these triangles are never explicitly stored, this difference is not really relevant at this point.

Edge insertion operators

Since four different cases can be distinguished for node insertion, $4 \times 4 = 16$ cases can be distinguished when inserting two (connected) nodes into a tetrahedron. Table 6.2 shows that nine unique cases can be distinguished within these 16 theoretical options. Since observation 6 in section 4.1 stated that the nodes and edges of a n-simplex form a complete graph, the edge from I_{00} is already present in the structure.

Note that the operators in table 6.2 work on tetrahedrons and not on the complete TEN, since that would result in an infinite number of possible combinations of these tetrahedron cases. As a result, a constrained edge that crosses several tetrahedrons, will be split first into parts. For instance, an edge crossing three tetrahedrons will result in a combination of I_{23}, I_{22} and I_{23}. This example is illustrated in figure 6.4: in tetrahedron $< v_0, v_1, v_2, v_3 >$ an edge segment from node v_{start} inside a tetrahedron to an additional node on triangle $< v_0, v_1, v_3 >$, in tetrahedron $< v_0, v_1, v_3, v_5 >$ from this node to another additional node in triangle $< v_0, v_3, v_5 >$ and in tetrahedron $< v_0, v_3, v_4, v_5 >$ from this node to the end node v_{end} (in the tetrahedron's interior). The concept of this step-by-step approach is illustrated in figure 6.5.

To deal with these cases, nine exhaustive and mutually exclusive operators will be described (as summarised in table 6.2):

I_{01}: One node is an existing node, the other node lies on an edge. This involves the n tetrahedrons of which the edge is part of the boundary. In figure 6.6 the case n=4 is illustrated: v_{start} is an existing node, while v_{end} lies on edge $< v_3, v_4 >$. Each involved tetrahedron is split into two new tetrahedrons, resulting in an increase of TEN elements: +1 node, $+n$+1 edges, $+2n$ triangles and $+n$ tetrahedrons.

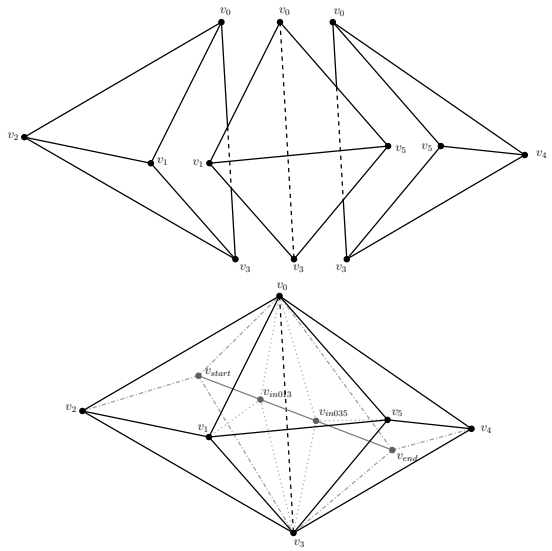


Figure 6.4: *Insertion of a constrained edge in three tetrahedrons: edge will be split into three separate parts (for clarity reasons the three tetrahedrons are emphasised in the exploded view).* v_{start} *and* v_{end} *lie in the interior of respectively* $< v_0, v_1, v_2, v_3 >$ *and* $< v_0, v_3, v_4, v_5 >$

Note that this case resembles node insertion operation I_1 with the only difference that one of the new edges is now the constrained edge.

I_{02}: One node is an existing node, the other node lies on a triangle. This involves the two tetrahedrons that share the triangle. Figure 6.7 illustrates this: v_{start} is an existing node, while v_{end} lies on triangle $< v_0, v_2, v_3 >$. Both tetrahedrons are split into three new tetrahedrons, resulting in an increase of TEN elements: +1 node, +5 edges, +8 triangles and +4 tetrahedrons. Note that this case resembles node insertion operation I_2 with the only difference that one of the new edges is now the constrained edge.

I_{03}: One node is an existing node, the other node lies inside the tetrahedron. This case is similar to the first operator for a single node insertion. As shown in figure 6.8, the tetrahedron is split into four new tetrahedrons, resulting in an increase of TEN elements: +1 node, +4 edges, +6 triangles and +3 tetrahedrons. Note that this case resembles node insertion operation I_3 with the only difference that one of the new edges is now the constrained edge.

I_{11}: Both nodes lie on edges. Depending on which edges the two nodes lie, three sub-cases can be distinguished:

a. The nodes lie on the same edge. This involves the n tetrahedrons of which

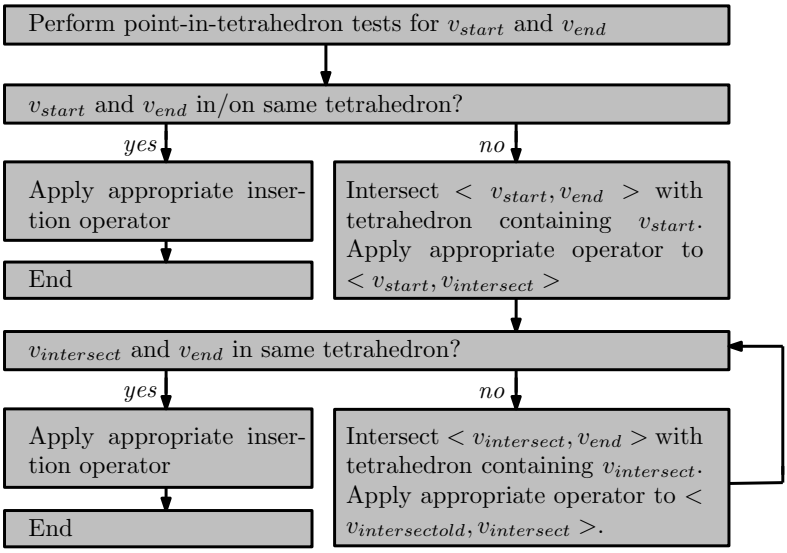


Figure 6.5: *Step-by-step insertion of constrained edges: incrementally finding and applying the appropriate operator for each edge segment*

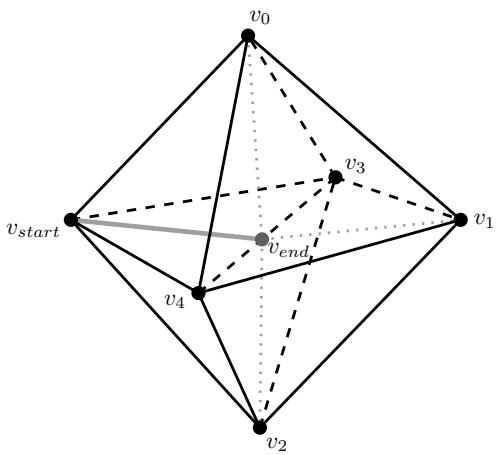


Figure 6.6: I_{01}*: v_{start} is an existing node, while v_{end} lies on edge $< v_3, v_4 >$. Each tetrahedron is split into two new tetrahedrons (note: case* n=*4 is illustrated, with* n *the number of incident tetrahedrons to the edge)*

the edge is part of the boundary. Figure 6.9 illustrates the case n=4: both v_{start} and v_{end} lie on edge $< v_3, v_4 >$. Each involved tetrahedron is split

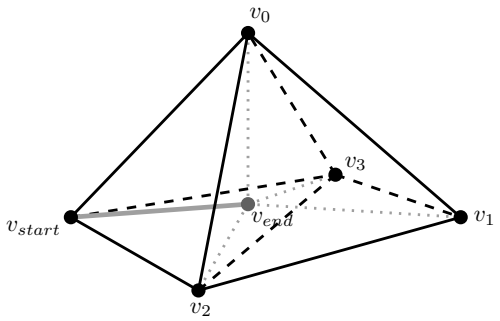


Figure 6.7: *I_{02}: v_{start} is an existing node, while v_{end} lies on triangle $< v_0, < v_2, v_3 >$. Both tetrahedrons are split into three new tetrahedrons)*

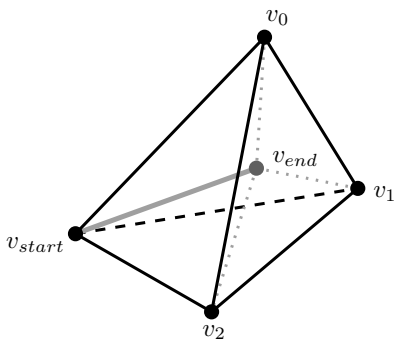


Figure 6.8: *I_{03}: v_{start} is an existing node, while v_{end} lies in the interior of the tetrahedron. The tetrahedron is split into four new tetrahedrons*

into three new tetrahedrons, resulting in an increase of TEN elements: +2 nodes, +2n+2 edges, +4n triangles, +2n tetrahedrons

b. The nodes lie on different, adjacent edges (two edges in a tetrahedron are said to be adjacent if they share a node (see figure 6.10: edges $< v_3, v_4 >$ and $< v_4, v_5 >$ share node v_4). This involves *n+m-2* tetrahedrons, with n the number of tetrahedrons of which the first edge is part of the boundary and m the number of tetrahedrons of which the second edge is part of the boundary. Although in the illustrated case in figure 6.10 both n and m are 4, these numbers are usually not equal. In this example v_{start} lies on edge $< v_4, v_5 >$ and v_{end} lies on edge $< v_3, v_4 >$. As a result the constrained edge $< v_{start}, v_{end} >$ lies in the plane of triangle $< v_3, v_4, v_5 >$. Now each tetrahedron that does not contain this triangle is split into two new tetrahedrons, while the two tetrahedrons that contain the triangle are split into three new triangles, resulting in an increase of TEN elements: +2

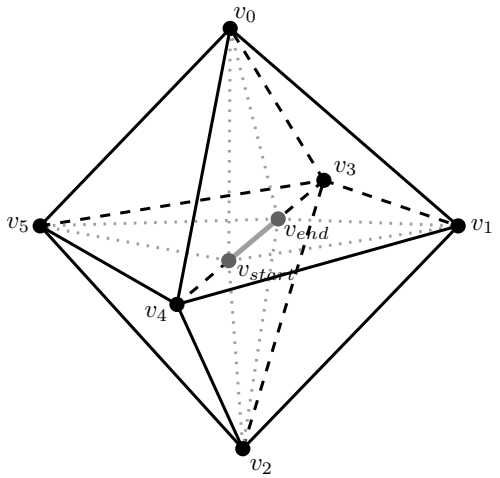


Figure 6.9: I_{11} *a: both* v_{start} *and* v_{end} *lie on edge* $< v_3, v_4 >$*. Each involved tetrahedron is split into three new tetrahedrons*

nodes, $+n+m+2$ edges, $+2(n+m)$ triangles and $+n+m$ tetrahedrons.

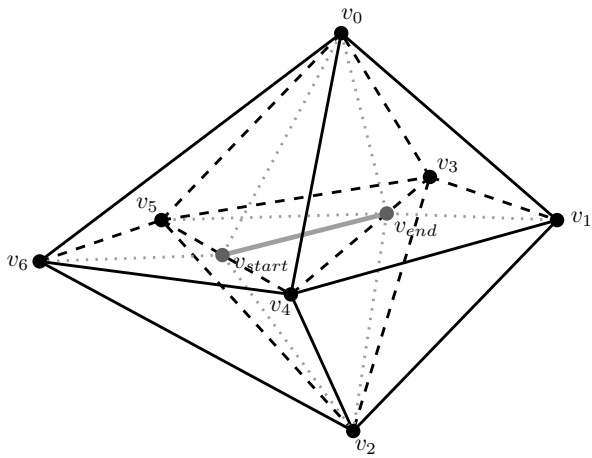


Figure 6.10: I_{11} *b:* v_{start} *lies on edge* $< v_4, v_5 >$ *and* v_{end} *lies on adjacent edge* $< v_3, v_4 >$*. Each tetrahedron without triangle* $< v_3, v_4, v_5 >$ *is split into two new tetrahedrons, while the two tetrahedrons that include this triangle are split into three new tetrahedrons*

c. The nodes lie on different, opposite edges (two edges in a tetrahedron are said to be opposite if they do not share a node (see figure 6.11: the edges $< v_0, v_3 >$ and $< v_4, v_5 >$ share no node). This involves *n+m-1*

tetrahedrons, with n the number of tetrahedrons of which the first edge is part of the boundary and m the number of tetrahedrons of which the second edge is part of the boundary. Figure 6.11 shows an example in which both n and m are 4. As a result of this insertion, each tetrahedron that does not contain $< v_{start}, v_{end} >$ is split into two new tetrahedrons, whereas the one tetrahedron that contains $< v_{start}, v_{end} >$, will be split into four new tetrahedrons. This results in the following increase of TEN elements: +2 nodes, +n+m+3 edges, +2(n+m)+2 triangles and +n+m+1 tetrahedrons.

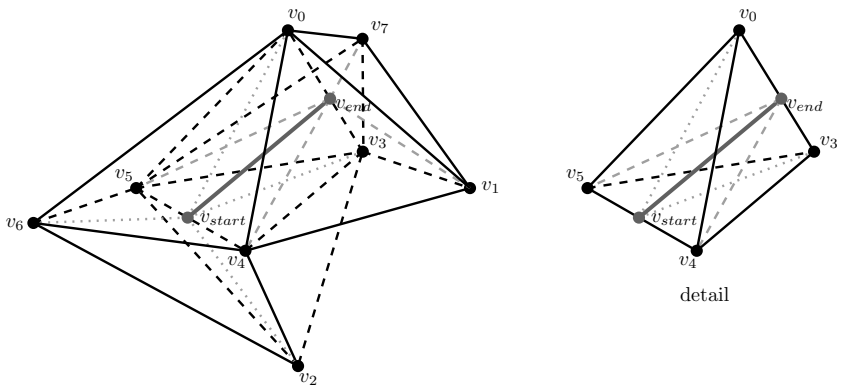


Figure 6.11: *I_{11} c: v_{start} lies on edge $< v_4, v_5 >$ and v_{end} lies on opposite edge $< v_0, v_3 >$. Each tetrahedron without edge $< v_{start}, v_{end} >$ is split into two new tetrahedrons, while the tetrahedron that contains $< v_{start}, v_{end} >$ (see detail at the right) is split into four new tetrahedrons*

I_{12}: One node lies on a triangle, the other node lies on an edge. Depending on whether this edge is part of the triangle or adjacent to the triangle, two subcases can be distinguished:

a. One node lies on a triangle, the other node lies on an adjacent edge. This involves *n+1* tetrahedrons, with n the number of tetrahedrons of which the edge is part of the boundary. The n=4 case is illustrated in figure 6.12. v_{start} lies on edge $< v_4, v_5 >$ while v_{end} lies on triangle $< v_0, v_3, v_4 >$. By first inserting v_{start} the TEN will grow (see case 1) with +1 node, +n+1 edges, +2n triangles and +n tetrahedrons. The tetrahedron in which $< v_{start}, v_{end} >$ is also split into two new tetrahedrons. On one of these new tetrahedrons, $< v_0, v_3, v_4, v_{start} >$ and tetrahedron $< v_0, v_1, v_3, v_4 >$ I_{02} applies, thus resulting in a further increase with +1 node, +5 edges, +8 triangles and +4 tetrahedrons. In total the TEN grows with +2 nodes, +n+6 edges, +2n+8 triangles and +n+4 tetrahedrons.

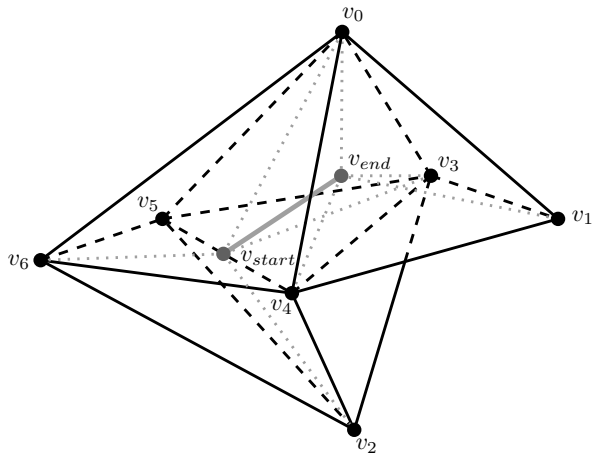


Figure 6.12: *I_{12}a.: v_{start} lies on edge $< v_4, v_5 >$ while v_{end} lies on triangle $< v_0, v_3, v_4 >$. The - 1 tetrahedrons that are influenced by v_{start} only are each split into two tetrahedrons. The tetrahedron that is influenced only by v_{end} is split into three new tetrahedrons, while the tetrahedron with v_{start} and v_{end} is split into four tetrahedrons.*

b. One node lies on an edge of the boundary triangle, in which the other node is inserted. Figure 6.13 shows this case. Applying I_{12} to this situation results in identical increases in the number of TEN elements: +2 nodes, $+n$+6 edges, $+2n$+8 triangles and $+n$+4 tetrahedrons.

I_{13}: One node lies on an edge, the other node lies in the interior of one of the n involved tetrahedrons. First the node on the edge is inserted (v_{end} in figure 6.14). The TEN is extended with +1 node, $+n$+1 edges, $+2n$ triangles and $+n$ tetrahedrons. In figure 6.14 the case n=4 is illustrated. As one can see, each original tetrahedron in split into two new tetrahedrons. Since one knows in which original tetrahedron the second node would be inserted, two sub-cases can be distinguished, related to these two new tetrahedrons:

a. The second node lies in the interior of one of the two tetrahedrons, changing the totals to: +2 nodes, $+n$+5 edges, $+2n$+6 triangles and $+n$+3 tetrahedrons.

b. The second node lies in the triangle between the two tetrahedrons, changing the totals to: +2 nodes, $+n$+6 edges, $+2n$+8 triangles and $+n$+4 tetrahedrons.

As the second node v_{start} lies in the interior of the split tetrahedrons and not on its boundary, it is impossible for this second node to lie on an edge, as this edge lies in the triangle (on the boundary) of the original tetrahedron.

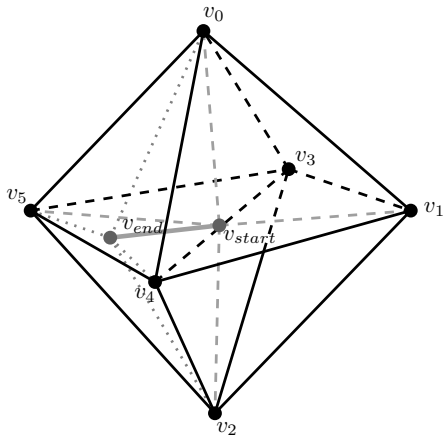


Figure 6.13: $I_{12}b$.: v_{start} *lies on edge* $< v_3, v_4 >$ *while* v_{end} *lies on triangle* $< v_3, v_4, v_5 >$. *This sub-case has the same results as* $I_{12}a$. *in figure 6.12. The* n*-2 tetrahedrons that are only influenced by* v_{start} *are split into two tetrahedrons, whereas the two tetrahedrons that are influenced both by* v_{start} *and* v_{end} *are each split into four tetrahedrons.*

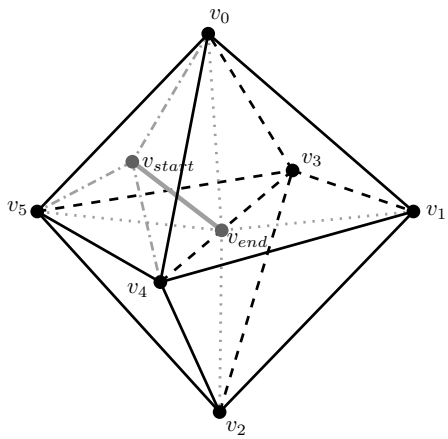


Figure 6.14: $I_{13}a$: v_{end} *lies on edge* $< v_3, v_4 >$ *and* v_{start} *in the interior of one of the newly formed tetrahedrons. Each tetrahedron is split into two new tetrahedrons, of which the new one with* v_{start} *is further split into four new tetrahedrons (note that* $I_{13}b$ *is not illustrated)*

I_{22}: Both nodes lie on boundary triangles. Again two sub-cases can be distinguished, depending whether the two nodes lie on the same triangle or not:

a. In case both nodes lie on the same triangle, two tetrahedrons will be involved. Both tetrahedrons are split into five new tetrahedrons, resulting in an increase of TEN elements: +2 nodes, +10 edges, +16 triangles and +8 tetrahedrons. Figure 6.15 illustrates this case.

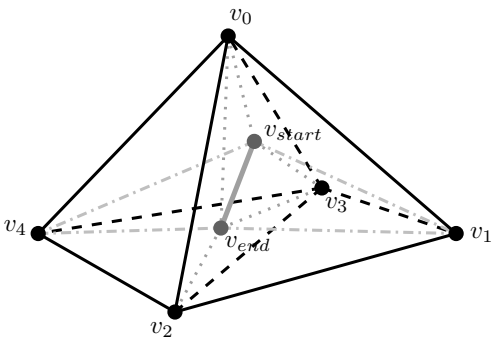


Figure 6.15: *I_{22}a.: v_{start} and v_{end} lie in the same triangle. Both tetrahedrons are split into five new tetrahedrons*

b. In case the nodes lie on different triangles, three tetrahedrons will be involved. The tetrahedron in which edge $< v_{start}, v_{end} >$ lies is split into five new tetrahedrons, while the other two tetrahedrons are split into three new tetrahedrons. This results in the same increase of TEN elements as case a.: +2 nodes, +10 edges, +16 triangles and +8 tetrahedrons. Figure 6.16 shows an example of this sub-case.

I_{23}: One node lies on a triangle, the other node lies in the interior of the tetrahedron. This involves two tetrahedrons, as illustrated in figure 6.17. First v_{start}, the node in the interior, is inserted into the tetrahedron, thus splitting this tetrahedron into four new tetrahedrons. Now the second node is inserted in the triangle. As a result, both the other tetrahedron and one of the newly formed tetrahedrons are split into three new tetrahedrons, changing the totals to: +2 nodes, +9 edges, +14 triangles and +7 tetrahedrons.

I_{33}: Both nodes lie in the interior of the tetrahedron, so only one tetrahedron is involved. First the start node v_{start} is inserted, by which the tetrahedron is split in four new tetrahedrons. Depending on the location of the end node v_{end}, three sub-cases can be distinguished:

a. v_{end} lies within newly formed tetrahedron $< v_1, v_2, v_3, v_{start} >$, see figure 6.18. This tetrahedron is again split into four new tetrahedrons, resulting in a total growth of +2 nodes, +8 edges, +12 triangles and +6 tetrahedrons

b. v_{end} lies on one of the six newly formed triangles. In the example in figure 6.19 v_{end} lies in triangle $< v_0, v_3, v_{start} >$, thus splitting both adjacent

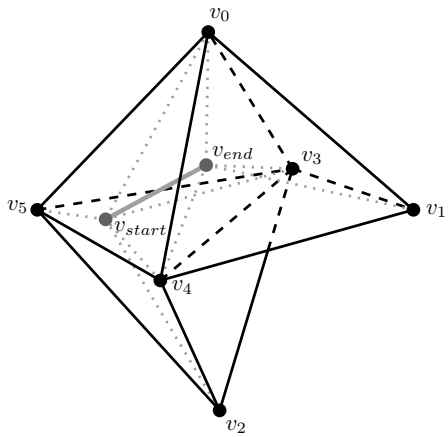


Figure 6.16: *I_{22}b.: v_{start} and v_{end} lie in different triangles. The tetrahedron in which the constrained edge is inserted is split into five new tetrahedrons, while the other two tetrahedrons are split into three new tetrahedrons*

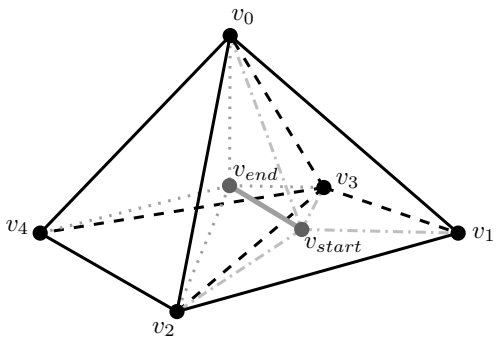


Figure 6.17: *I_{23}: v_{start} lies in the interior of newly formed tetrahedron $< v_0, v_1, v_2, v_3 >$, v_{end} lies on triangle $< v_0, v_2, v_3 >$. The left tetrahedron is split into three parts, the right tetrahedron is split into six new tetrahedrons (v_{start} creates for new tetrahedrons, of which one is split into three by v_{end})*

tetrahedrons into three new tetrahedrons. This results in a TEN increase of +2 nodes, +9 edges, +14 triangles, +7 tetrahedrons

c. v_{end} lies on one of the four newly formed edges. In the illustration in figure 6.20 v_{end} lies on edge $< v_{start}, v_2 >$. As a result the three tetrahedrons that contain this edge are each split into two new tetrahedrons, thus resulting in an increase in TEN elements of +2 nodes, +8 edges, +12 triangles and +6 tetrahedrons

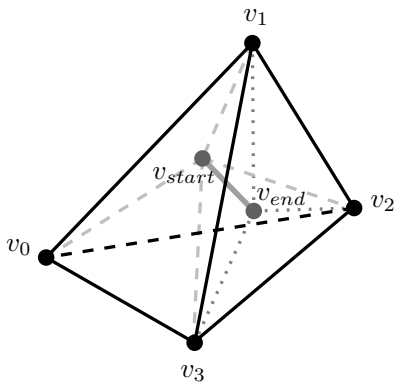


Figure 6.18: *$I_{33}a$.: insertion of v_{start} splits tetrahedron $< v_0, v_1, v_2, v_3 >$ into four tetrahedrons, of which tetrahedron $< v_1, v_2, v_3, v_{start} >$ is split into four new tetrahedrons by the insertion of v_{end}*

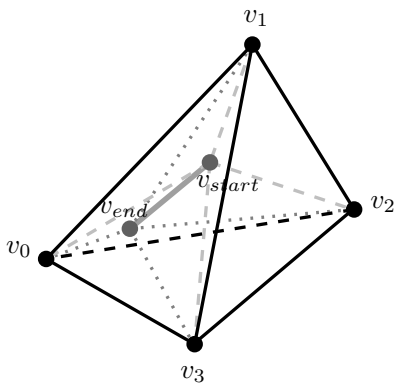


Figure 6.19: *$I_{33}b$.: insertion of v_{start} splits tetrahedron $< v_0, v_1, v_2, v_3 >$ into four tetrahedrons, tetrahedrons $< v_0, v_2, v_3, v_{start} >$ and $< v_0, v_1, v_3, v_{start} >$ are both split into three new tetrahedrons by the insertion of v_{end} in triangle $< v_0, v_3, v_{start} >$*

Although the described cases are exhaustive and mutually exclusive, the solution for each case is not unique. This can be observed for instance in case I_{23}, as shown in figure 6.17. An alternative approach can be followed, see figure 6.21. In this case, both tetrahedrons are first split into three parts, of which the one with v_{end} is split into four new tetrahedrons. Note that with this approach several sub-cases are needed, since v_{end} may fall in the interior of a newly created tetrahedron, but also in the interior of a newly created triangle or edge!

By applying these insertion operators according to the previously introduced principle (see figure 6.5), constrained edges can be inserted in a TEN as a set of segments. By traversing the constrained edge and inserting each segment, these operators guar-

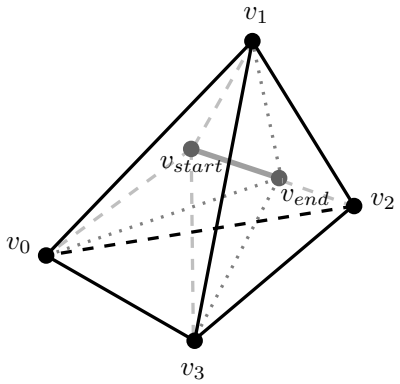


Figure 6.20: $I_{33}c.$: v_{end} *lies on edge* $< v_{start}, v_2 >$, *thus splitting the three neighbouring tetrahedrons each into two new tetrahedrons*

	Node	**Edge**	**Triangle**	**Tetrahedron**
(I_{00})	(+0)	(+0)	(+0)	(+0)
I_{01}	+1	+n+1	+2n	+n
I_{02}	+1	+5	+8	+4
I_{03}	+1	+4	+6	+3
$I_{11}a.$	+2	+2n+2	+4n	+2n
$I_{11}b.$	+2	+n+m+2	+2(n+m)	+n+m
$I_{11}c.$	+2	+n+m+3	+2(n+m)+2	+n+m+1
$I_{12}a.$	+2	+n+6	+2n+8	+n+4
$I_{12}b.$	+2	+n+6	+2n+8	+n+4
$I_{13}a.$	+2	+n+5	+2n+6	+n+3
$I_{13}b.$	+2	+n+6	+2n+8	+n+4
$I_{22}a.$	+2	+10	+16	+8
$I_{22}b.$	+2	+10	+16	+8
I_{23}	+2	+9	+14	+7
$I_{33}a.$	+2	+8	+12	+6
$I_{33}b.$	+2	+9	+14	+7
$I_{33}c.$	+2	+8	+12	+6

Table 6.3: *Overview of all edge insert operations with the increases in TEN size*

antee the (splitted) constrained edge's presence in the structure. Common operators –inserting two nodes and finding the constrained edge by edge recovery using flips– might fail in case flips are not allowed due to other nearby constrained edges. This would typically happen in TEN structure with many features (and thus many constraints), of which topographic data sets are an excellent example. By minimising the impact of the operators on the TEN structure, conflicts with nearby constraints are

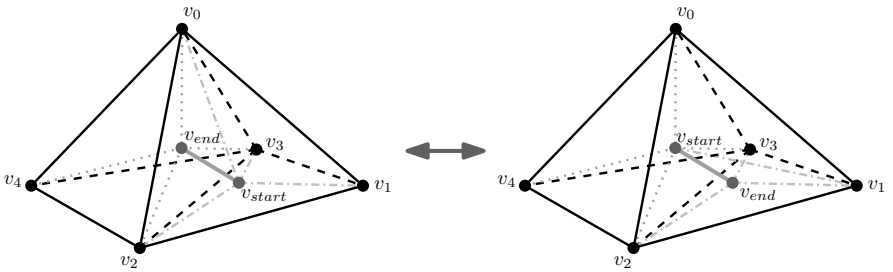


Figure 6.21: *Alternative to I_{23} (compare with figure 6.17): This time (right), v_{end} lies in the interior of newly formed tetrahedron $< v_0, v_1, v_2, v_3 >$ and v_{start} lies on triangle $< v_0, v_2, v_3 >$. Now both tetrahedrons are split into three parts, of which the one with v_{end} is split into four new tetrahedrons. Note that with this approach several sub-cases are needed, since v_{end} may fall in the interior of a newly created tetrahedron, but also in the interior of a newly created triangle or edge!*

avoided. This advantage comes at the price of a resulting tetrahedronisation that will not meet the Delaunay empty-circumcircle criterion. Obviously this might lead to decreasing quality of the TEN network in terms of quality criteria like the shortest-to-longest edge ratio, as introduced in section 2.5.3. To overcome this drawback, additional quality improvements are required. These improvement operators will be introduced in section 6.3.

6.1.4 Step 3. Ensuring presence of constrained triangles

With this set of operations, all constraint edges can be inserted. In case constraint edges are longer, i.e. the edges are crossing multiple tetrahedrons, the constraint edges can be split at the intersection points with the tetrahedron boundaries. These segments can be treated separately by one of the previous operations. However ensuring the presence of all constraint edges is not sufficient, since one has no guarantee that the corresponding triangles are also present. This is illustrated in figure 6.22.

In this illustration, constrained triangle $< v_0, v_1, v_2 >$ lies more or less horizontal and edge $< v_3, v_4 >$ more or less vertical. In this example, the grey edges $< v_0, v_1 >, < v_1, v_2 >$ and $< v_2, v_0 >$ are constrained edges, but since the tetrahedrons are defined (in the middle figure) as $< v_0, v_1, v_3, v_4 >, < v_0, v_2, v_3, v_4 >$ and $< v_1, v_2, v_3, v_4 >$, the required constrained triangle $< v_0, v_1, v_2 >$ is missing. Therefore an additional test is required that checks whether each constraint triangle from the boundary triangulation is also present in the TEN, a task that is quite straightforward due to the node-based notation of all simplexes. A prerequisite of this testing method is that the TEN structure is valid, see chapter 4 for more details. To obtain the constrained triangle, a flip T_{32} has to be performed, which result is shown in the right figure. This flip can be performed as long as (see also figure 2.27)$< v_3, v_4 >$ intersects $< v_0, v_1, v_2 >$, but if this is not the case, constrained triangle $< v_0, v_1, v_2 >$ would have been present

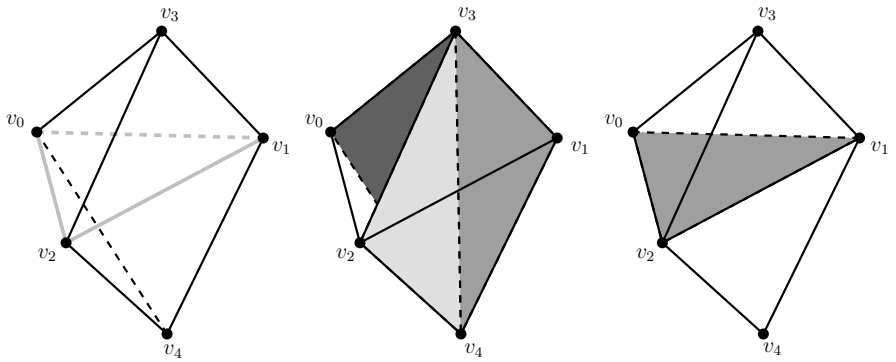


Figure 6.22: *With constrained edges* $< v_0, v_1 >$, $< v_1, v_2 >$ *and* $< v_2, v_0 >$ *inserted, the tetrahedronisation might still look like the one in the middle. Performing flip* T_{32} *(see section 2.5.4) will result in the creation of the wanted constrained triangle* $< v_0, v_1, v_2 >$.

already in the structure.

However, one should not only test for 1:1 relations between triangles in the TEN and triangles in the previously calculated boundary triangulation. The edge insertion operators might have separated the constrained edges into multiple segments and thus the initial constrained triangles might be split into several parts as well. Therefore one has to test for 1:m relationships as well, since several constrained triangles in the TEN might form one constrained triangle from the boundary triangulation. Here an additional test is required, to see whether the m triangles lie in the same plane, most likely performed by comparing the triangle's normal vectors.

6.1.5 Step 4. Modelling the feature's interior and reclassifying tetrahedrons

Two approaches can be followed in modelling the feature interior:

- The first option is the most straight forward approach of enforcing the presence of the complete outer boundary triangulation and detect and reclassify the resulting internal tetrahedrons. The reclassification is required since the new feature replaces (part of) an existing feature. If one thinks of inserting a new building, the interior tetrahedrons will be reclassified from the previous feature (most likely an 'air' feature) into the new feature (a specific building ID). Reclassification will always be involved, since the topographic data model is a full 3D decomposition of space. Note that by reclassifying its outer boundary will automatically appear in the view with constrained triangles, since this view is specified by a function that detects different object identifiers at either side of a triangle (see section 5.4.2 for more details on deriving constraints). The

boundary edges will appear in the view with constrained edges, since this view contains all edges from constrained triangles.

- The second option is to delete the existing tetrahedral structure within the boundary triangulation and replace it with a separately calculated tetrahedronisation. To do so one needs to keep record of all segmentation of the prior boundary triangulation (since the inserted constrained edges might be split into separate parts) and use this refined boundary triangulation as input for a separate tetrahedronisation. This process will result in an optimal (in terms of number and shape of tetrahedrons) internal tetrahedronisation. To each of these tetrahedrons the relevant object identifier will be added, similar to the first approach.

Both approaches require detection of the tetrahedrons within the newly inserted boundary (either for reclassification or deletion). The second option also contains a tetrahedronisation step, thus increasing overall runtime, but this additional step will result in an optimal tetrahedronisation of the interior. However, since regular quality checks and improvements (see section 6.3 for more details) are required due to the non-optimal edge insertion operators, this advantage exists only temporarily.

6.2 Incremental update: feature deletion

Deleting features might be arranged by a very pragmatic operation that does not influence the TEN structure itself. If for instance a building needs to be removed the tetrahedrons that represent this building have to be reclassified, most likely with the 'air' object. By changing the object identifier that is added to the tetrahedron codes the constrained triangles and constrained edges from the feature boundary will disappear from their views, since these triangles and edges do not longer bound two tetrahedrons with different object identifiers. Effectively this operation removes the constraints from these former boundary triangles and edges.

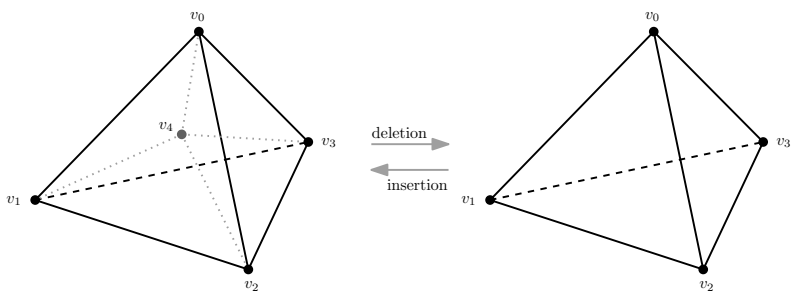


Figure 6.23: *A 'flip41' operation (Edelsbrunner* and *Shah 1996) will modify the four initial tetrahedrons* $< v_0, v_1, v_2, v_4 >$, $< v_0, v_1, v_3, v_4 >$, $< v_0, v_2, v_3, v_4 >$ *and* $< v_1, v_2, v_3, v_4 >$ *into a single tetrahedron* $< v_0, v_1, v_2, v_3 >$, *thus deleting node* v_4.

Although the feature would be deleted by this operation, the structure does not decrease in size, since no TEN elements are deleted. However from the beginning of this research managing data volume is regarded as one of the key elements in obtaining a feasible new data modelling approach. Therefore an operator that deletes unnecessary nodes (and thus edges, triangles and tetrahedrons) is required. In case of feature deletion, all nodes that were part of the deleted feature, are candidates for deletion. First a check is needed whether these nodes are part of any other boundary (i.e. whether they are still part of a constrained edge). Each resulting node can be deleted by the robust operator described by Ledoux *et al.* (2005). This approach, which is described in more detail by Ledoux (2006, paragraph 4.4), uses flips to modify the TEN configuration until a situation as shown at the left in figure 6.23 is reached. In this situation Ledoux *et al.* (2005) apply a so-called 'flip41' to remove the node from the TEN.

Although this deletion operator is referred to as a bistellar flip, within this dissertation the term flips (see section 2.5.4) is reserved for operations that affect only a tetrahedronisation T, whereas the set of input vertices V remains unchanged. Such deletion operators will be annotated as D. The 'flip41' will therefore be annotated as D_3, which is the inverse operator of I_3.

6.3 Quality improvement of TEN structure

As the model will grow over time due to ongoing refinements and incremental updates, a set of tools for quality maintenance of the data structure is necessary. Due to series of delete and insert operations ill-shaped triangles and tetrahedrons might appear, as well as unneeded TEN elements. To reduce data storage and ensure numerical stability these triangles and tetrahedrons should be repaired. Several different approaches might be used complementary. Three categories of quality improvement operators can be distinguished (in practice a mixture of operators of all three types will be used):

I: Operators that *add nodes* to the TEN. A good example is the method for TEN improvement proposed by Shewchuk (1997b). His method is based on the circumradius-to-shortest edge ratio of triangles or tetrahedrons. The circumsphere is a sphere through the nodes of a tetrahedron, the circumradius the radius of this circumsphere. The ratio between the circumradius and the shortest edge is (in most cases) a good indicator for how evenly the tetrahedron is shaped. This concept is illustrated in figure 6.24. It shows an unevenly shaped tetrahedron at the left. The shortest edge in this tetrahedron (edge $< v_1, v_2 >$) is far shorter than the circumradius R_{circ}. In a more evenly shaped tetrahedron (shown at the right side) the shortest edge (edge $< v_1, v_3 >$) is more or less comparable to the circumradius R_{circ}.

In case ill-shaped tetrahedrons are detected, adding a node to the TEN in the circumcentre (the centre of the circumsphere) is an adequate cure in most cases. This approach is described in section 2.5.3 and illustrated (in 2D) in figure 2.25.

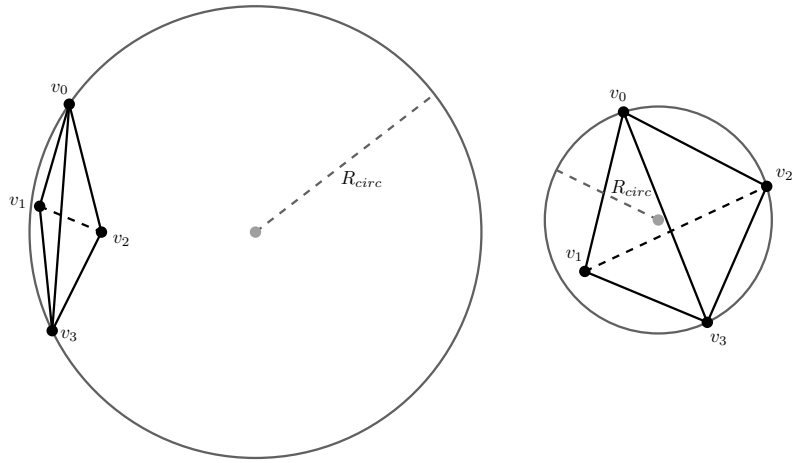


Figure 6.24: *The circumradius-to-shortest edge ratio used by Shewchuk (1997b) is a useful measure for how evenly a tetrahedron is shaped*

Most sliver tetrahedrons will be removed by this procedure. In case of an ill-shaped tetrahedron that can not be improved by adding the circumcentre as node (for instance due to the presence of constraint edges), using Steiner points to partition constrained edges might be very useful.

D: Operators that *remove nodes* from the TEN. Obviously the deletion of unnecessary nodes from the TEN is very welcome from a data volume point of view, as this will increase overall performance. Less obvious is its influence on the overall quality of the TEN. Vertex deletion can also be seen as a precaution operation: if one thinks of a house that is being demolished and replaced by a new house at almost the same location, omitting vertex deletion might result in near-parallel edges (from the old and the new building), causing situations as illustrated in figure 2.18. These near-parallel lines can cause the creation of numerous very small tetrahedrons. Removing nodes (and thus edges, triangles and tetrahedrons) helps decreasing the risk of such unwanted configurations. The robust vertex deletion algorithm as described in the previous section can be used to achieve this.

T: Operators that *modify the TEN configuration*. A good example of modifying a TEN structure is flipping. Flipping (as described in chapter 2) uses the fact that there are two ways to tetrahedronise a convex set of five points (assuming this is not prevented by constraints): one with two tetrahedrons and one with three tetrahedrons. One of these two tetrahedronisations must be the Delaunay tetrahedronisation of five points (Cavalcanti and Mello 1999). Therefore flipping from 2 to 3 tetrahedrons (T_{23}) or vice versa (T_{32}) can improve the quality of the data structure. In special cases (more points need to be co-planar) also T_{22}

or T_{44} operations can improve the quality.

6.4 Initial bulk loading and bulk rebuild

6.4.1 Bulk loading to create a new data set

Although it is possible to create a complete tetrahedronised 3D topography data set with incremental algorithms, initial bulk loading speeds up the process. Although the basic data structure is implemented in the Oracle DBMS, the required tetrahedronisation algorithms are currently not implemented within the DBMS due to practical limitations. As a temporary work around TetGen (TetGen 2007) is used to externally perform a tetrahedronisation. The input is a Piecewise Linear Complex (PLC), see figure 6.25 and section 2.5.3. A PLC (Miller *et al.* 1996) is a set of vertices, segments and facets, where a facet is a polygonal region. Each facet may be non-convex and have holes, segments and vertices in it, but it should not be a curved surface. A facet can represent any planar straight line graph (PSLG), which is a popular input model used by many two-dimensional mesh algorithms. A PSLG is (MathWorld 2007) a graph embedding of a planar graph (i.e. a graph without graph edge crossings) in which only straight line segments are used to connect the graph vertices (as introduced in section 2.5.1). Compared to a polyhedron, a PLC is

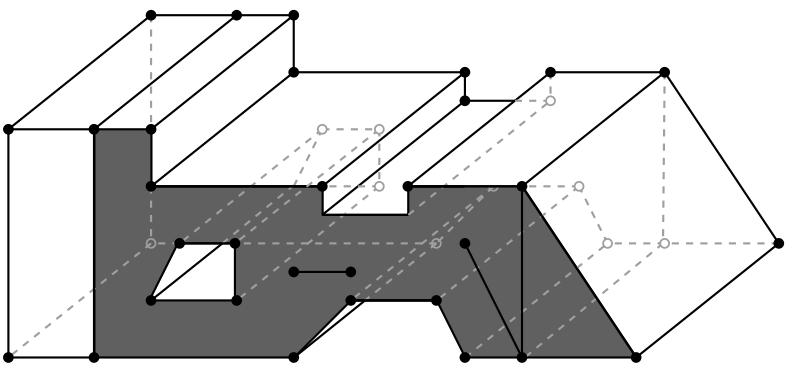


Figure 6.25: *A Piecewise Linear Complex (PLC), input for the tetrahedronisation algorithm (from (Si 2006c))*

a more flexible format. If one looks at the shaded facet in figure 6.25, one can see that this facet cannot be described by a polygon because there are loose and dangling line segments. However, in our application situations like these will be rare or not appearing at all. Nevertheless TetGen creates a constrained Delaunay tetrahedronisation based on an input PLC. A PLC may consist of several disjoint volumes. As a result, all topographic features are represented in one PLC in bulk tetrahedronisation.

TetGen requires an `*.poly` file as input data. Such a file represents a piecewise

linear complex (PLC) as well as some additional information. This input file contains four types of information, which are a list of points, a list of facets, a list of (volume) hole points and (optionally) a list of region attributes (see appendix IV for a short example). The list of points consists of x, y and z coordinate and a point identifier. The facets are described by a list of these point identifiers. Hole points are used to identify holes, but since TetGen does not tetrahedronise holes, this feature is not useful for the new approach. Regions are separate (3D) parts of the PLC, each with their own identifier. Region attributes (identifiers) are assigned to the volume that contains the given point in the file. These region attributes are used in bulk loading to assign feature identifiers.

Running TetGen results in a `*.ele` file, a text file containing all tetrahedrons and a `*.node` file with all nodes (see appendix IV for examples). The `*.ele` file consists for each tetrahedron of its four nodes and the relevant region attribute. Note that this output format is really close to the identifier concatenation as introduced in section 5.3.2, since concatenating the four node identifiers and the volume attribute will result in the desired tetrahedron code.

By loading the `*.node` file into the node table and the concatenated codes into the tetrahedron table the tetrahedronisation is converted into the Poincaré-TEN model. If one prefers coordinate concatenation (also introduced in section 5.3.2), one needs to replace the node identifiers in the tetrahedron codes by their concatenated coordinates and load these altered tetrahedron codes into the tetrahedron table.

6.4.2 Bulk rebuilding to optimise the tetrahedronisation

Besides the regularly performed quality updates as suggested in the previous section, one might consider rebuilding the updated data set every now and then. This operation might fix some problems that are difficult to overcome by the use of the previously introduced quality update operations. To do so, one needs to create an input data set first. When using TetGen, this means converting all feature boundaries into a PLC in a `*.poly` file. The constrained triangle view contains all required PLC facets. Note that the number of facets will be larger compared to an initial PLC with the same features, since facets derived from the view are already split up in triangular parts. By selecting all nodes, used in the constrained triangle view, one can create the required list of nodes in the input file. Note that by selecting the nodes from the constrained triangle view one ignores all other nodes that might be part of the tetrahedronisation, thus 'nominating' these nodes for elimination in order to minimise the number of elements in the TEN. The results from the tetrahedronisation process can be loaded similar to the bulk loading procedures. An advantage of bulk rebuilds outside the database is that one can easily benefit from the most recent improvements in the field of computational geometry with respect to tetrahedronisation, for instance regarding TEN quality or performance. If for instance Pyramid (see section 2.5.4) proves to be faster or to obtain better tetrahedron shapes one can easily adopt these new techniques without the need to modify the actual data modelling approach.

Part III

Evaluation and conclusions

Chapter 7

Evaluation and discussion

The previous part – consisting of chapters 4-6 – of this dissertation introduced a simplicial complex-based data structure for 3D topography. It was shown that this approach has several theoretical favourable characteristics, both due to the TEN approach and due to the application of operators from simplicial homology. A proof-of-concept implementation is needed to determine feasibility. Although a full implementation (in terms of the most time-efficient or storage-efficient implementation) is out of scope of this research, proof-of-concept implementations of the coordinate and identifier concatenation approaches (as introduced in section 5.3.2) were created. This chapter presents the valuable experiences gathered by these implementations.

In order to be able to test the data structure, tetrahedronised data is needed. Several data sets have been created by extrusion and were loaded into the database. One data set was further extended with 3D features, such as a viaduct and a geological layer. Section 7.1 describes these data sets, after which section 7.2 continues with the tetrahedronisation of this data in TetGen. The different data sets are used in section 7.3 to evaluate storage requirements, by comparing the coordinate concatenation approach with the identifier concatenation approach and comparing these approaches with storage as polyhedrons in Oracle Spatial 11*g*. Section 7.4 evaluates some visualisation approaches for tetrahedronised data sets in databases. Based on the experiences with three data sets, section 7.5 elaborates on requirements for new, proper 3D data, that should already be taken into account during the data acquisition phase. The chapter ends with identifying several future developments in section 7.6.

7.1 Evaluation material: three different data sets

Proper testing of a 3D volumetric data structure requires 3D volumetric data sets. Unfortunately 3D volumetric topographic data sets –especially good quality, error free ones– are hard to find. Originally it was planned to use data developed within the larger 3D topography project consortium (van Oosterom 2005a,b). However, due

to an asynchronous start of the separate parts of this larger research project, the data is not yet available. As a work-around several data sets have been created, usually by extrusion of buildings from existing 2D data sets, sometimes extended with data from other sources. This section will describe three data sets: the first one is completely made-up and was even tetrahedronised by hand at first. The second one consists of almost 1800 buildings in Rotterdam, extruded from the TOP10 data set (a topographic data set which paper predecessor had scale 1:10,000 and is now used at a scale range of 1:5,000 - 1:25,000 (Kadaster 2007)). The third one contains about 300 buildings on the campus of Delft University of Technology and is based on extrusion of the GBKN, the large scale base map with scale 1:1,000, combined with additional geological data and a terrain including viaducts, thus introducing real 3D situations in this data set. The extrusion of buildings in both the second and third data set is based on AHN heights (as introduced in section 2.1.2).

7.1.1 Initial 'toy' data set

A small 'toy' data set is created by hand. It consists of an earth surface with a road surface feature on top and a single building with a saddle roof. This dataset was tetrahedronised by hand to get more insight in a tetrahedron-based structure. In order to get 'air' and 'earth' tetrahedrons two extreme points were added, one on top and one at the far bottom. Figure 7.1 shows the tetrahedronisation of the small data set, with the building and the road in front of it (see figure 7.12 for an impression of the data set). This small data set, consisting of three volume features (building, air, earth), is composed by 56 tetrahedrons, 120 triangles (no opposites included), 83 edges and 20 nodes.

The very first experiences, as presented by Penninga *et al.* (2006), were based on this model. In a later stage TetGen (with its default settings) is also used to generate a tetrahedronisation of this very basic data set. This constrained tetrahedronisation results (TetGen output) in 117 tetrahedrons and 31 nodes. This second version of the 'toy' data set is loaded into the database structure (which includes triangle opposites), thus resulting in a set of 117 tetrahedrons, 468 (of which 140 constrained) triangles, 155 edges and 31 nodes. The increase in the number of nodes is caused by Steiner points that were added to improve tetrahedron shapes, see figure 7.2 for the eleven Steiner points.

Depending on the chosen concatenation method (see section 5.3.2) only the tetrahedrons or the tetrahedrons and nodes are stored explicitly, all other elements will be derived in views. The significant increase in the number of elements (compared to the first hand-made version) is due to the Delaunay tetrahedronisation (standard TetGen settings were used), whereas the first set was just a valid tetrahedronisation, without any guarantees on empty circumspheres or quality parameters like a minimised radius-to-edge ratio.

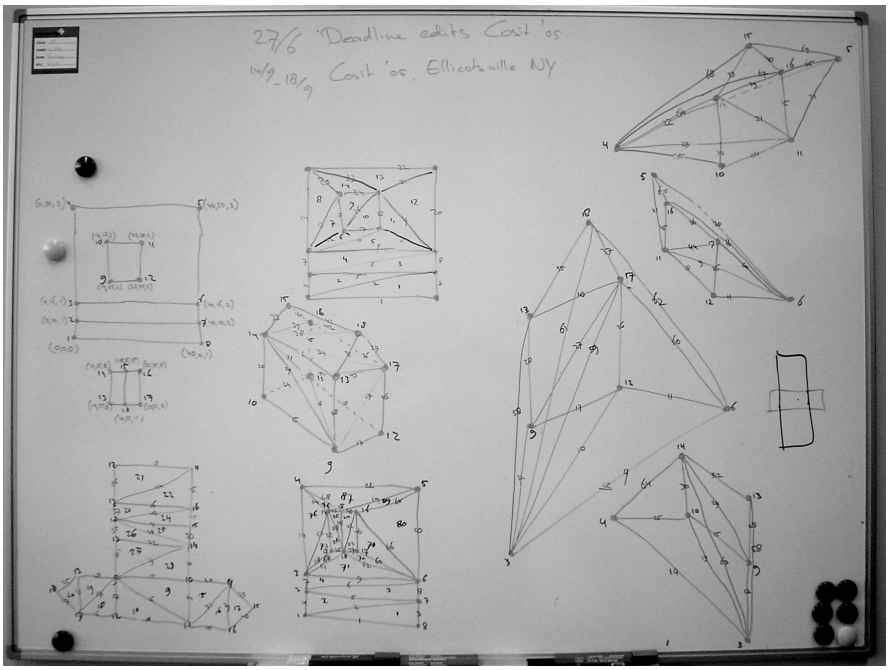


Figure 7.1: *First tetrahedronisation of the 'toy' dataset, created by hand.*

7.1.2 Rotterdam buildings data set

The first real world data set covers a part of Rotterdam. It contains 1,796 buildings in the north-western part of the city, see figures 7.3. As described by Penninga and van Oosterom (2008b), it covers an area of about seven square kilometres. Other topographic features like a realistic earth surface, roads, tunnels etc. are still missing due to a lack of appropriate 3D data. Complex situations like multiple land use and highway interchanges lack as well. However, this dataset will provide more insight into the number of elements of a TEN and into the resulting storage requirements. The data set is created by extruding polygons from a topographic data set. Since this topographic data set is originally created for usage with scale of about 1:10,000, the buildings do not contain much detail.

7.1.3 Delft University of Technology campus data set

Within the 3D topography research project, a test data set of the campus of Delft University of Technology is created (see figure 7.4 for the first version of this data set) to compare different approaches to 3D data modelling. Although the campus data set contains less buildings than the Rotterdam data set, this set is particularly interesting because the inclusion of real 3D features, such as viaducts and geological

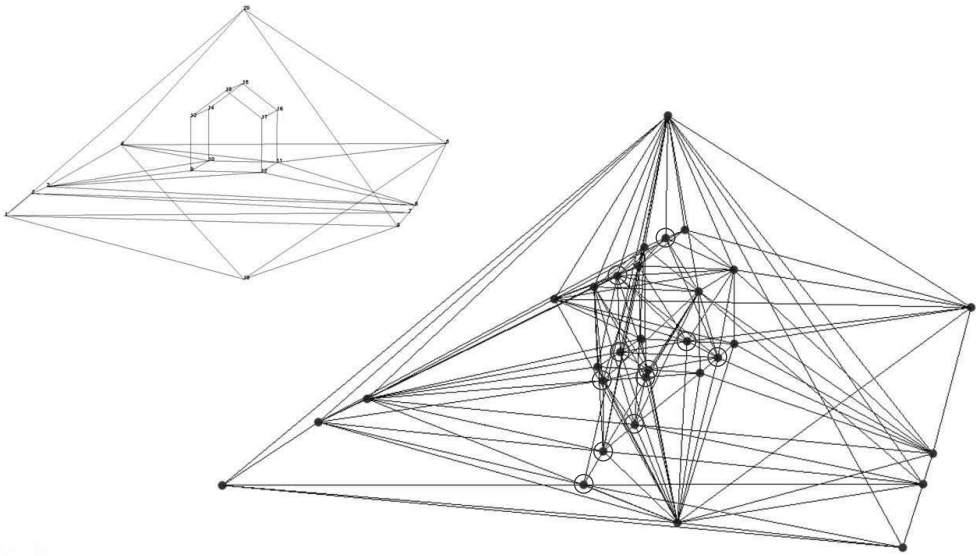


Figure 7.2: *Eleven Steiner points are added (circled), see also figure 5.1 for another view of this data set*

Figure 7.3: *Aerial picture of the two neighbourhoods of the Rotterdam data set*

layers. Beside that, the source data for building extrusion is of larger scale, since the GBKN (Dutch large scale base map, 1:1,000) is used, compared to TOP10 (scale 1:10,000) data for the Rotterdam data set. As a result the building polygons contain more detail. At the same time this data set contains also several unwanted points, since apparently each surveyed point is part of building's footprint, regardless whether this inclusion contributes to the overall shape of this footprint or not. Furthermore, the original version of the DUT campus data set contained numerous topological errors (mainly intersections) and thus failed to tetrahedronise in TetGen. As a result only 306 of the original 550 buildings are present in the data set used in this chapter. However, this data set is still very useful in the context of this evaluation, since it does include real 3D situations and many details in the building geometries.

Figure 7.4: *DUT campus data set as* `*.kml` *file, displayed in Google Earth*

Table 7.1 compares the different input data sets. The average numbers of input nodes and input faces per building can be seen as a measure for the amount of detail in the data set.

7.2 Evaluating bulk tetrahedronisation process

During this research TetGen software is used to tetrahedronise the data sets. As described in section 6.4, TetGen requires a `*.poly` file as input data, containing a list of points, a list of facets, a list of volume holes (not applicable for this approach since a full tetrahedronisation of space is required) and a list of region attributes (see appendix IV for example). Since the facets are described by point identifiers, prepar-

	'Toy' data set	Rotterdam data set	DUT campus data set
No. of buildings	1	1796	306
No. of input nodes (buildings)	10	26,650	8,034
No. of input nodes (terrain)	10	6	38
No. of input nodes (geology)	-	-	110
No. of input faces (buildings)	7	16,928	4,251
No. of input faces (terrain)	1	1	36
No. of input faces (geology)	-	-	216
Average no. of nodes per building	10	14.8	23.7
Average no. of faces per building	7	9.4	12.5

Table 7.1: *Comparison of input data set characteristics. Note that the small number of input terrain nodes indicates that the terrain is simplified: apart from the viaduct in the DUT campus data set, the terrain is flat. All input faces of the buildings and most input faces of the terrain are polygons, the geology faces and terrain faces near the viaduct are triangles*

ing such an input file is non-trivial in case the available data sets lack such explicit topological relationships. TetGen is selected since it is the only available tetrahedronisation software capable of constrained (Delaunay) tetrahedronisation. Pyramid, the long-awaited 3D successor of the popular 2D mesh generator Triangle, is still not released (Shewchuk 2008). However, TetGen incorporates many of the ideas published by Shewchuk (TetGen 2007), including Shewchuk's fast and robust predicates (Shewchuk 1997a). Table 7.2 provides more insight in the performance of TetGen. Note the large number of flips (section 2.5.4) that are required to obtain a tetrahedronisation that meets the constrained Delaunay and minimal angle criteria.

Table 7.3 contains some characteristics of the tetrahedronised data sets. Due to the incompleteness of these sample data sets, they can not be seen as a fully representative sample for topographic data sets. As a result, one should be careful with drawing final conclusions from these results, although the data sets do provide some useful insights. One can see that the average number of tetrahedrons per building is very high in the DUT campus data set. This is mainly caused by the presence of surveyed points in the building footprints. Figure 7.5 illustrates such a case. The depicted building footprint contains additional points (that do not really contribute to the overall shape) near

	'Toy' data set	**Rotterdam data set**	**DUT campus data set**
Input	Points: 20 Facets: 27 Regions: 3	Points: 26,656 Facets: 16,928 Regions: 1,796	Points: 8,182 Facets: 4,503 Regions: 309
Running time	0.063 sec.	16.641 sec.	248.392 sec.
Flips	T_{23}: 37 T_{32}: 26 T_{22}: 0 T_{44}: 5	T_{23}: 243,589 T_{32}: 178,680 T_{22}: 2 T_{44}: 616	T_{23}: 55,380 T_{32}: 39,106 T_{22}: 65 T_{44}: 18,382
Output	Points: 31 Constr. tr: 70 Tetra: 117	Points: 30,877 Constr. tr: 54,566 Tetra: 167,598	Points: 23,260 Constr. tr: 87,316 Tetra: 131,437

Table 7.2: *TetGen tetrahedronisation results*

the centre of the dotted circle. These additional points are located close to another, thus resulting in very small wall facets in the automatic extrusion. The presence of these parallel lines trigger the creation of vast numbers of tetrahedrons.

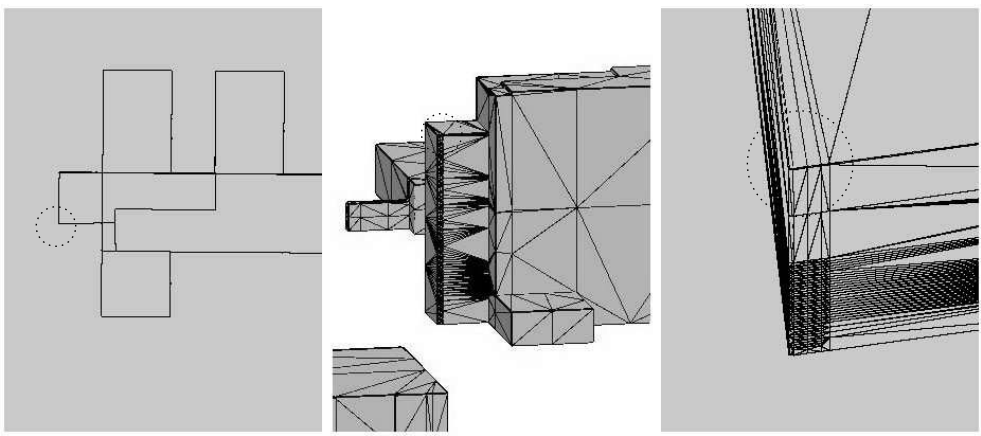


Figure 7.5: *The results of improper data cleaning: unnecessary points result in close parallel lines after extrusion, thus triggering the creation of many unwanted tetrahedrons. The unnecessary points lie on the edge in the circle, but are not visible in the top view (left: top view, mid: bird's eye view, right: detail)*

Another observation is that, by comparing the number of input nodes with the number of TEN nodes, the number of Steiner points is quite limited, unlike the potential threat of virtually unlimited addition of Steiner points as described in section

2.5. In this observation the increase in Steiner points of the DUT campus data set is partially ignored, since the creation of a lot of these points is unnecessary, as previously described and shown in figure 7.5.

The third observation is that the percentage of air and earth tetrahedrons is very large (over 70%). It appears that the price of following a full volume partition approach lies in an increase in data volume by a factor of four. Of course this number might vary for other data sets, but nevertheless it is a significant cost to obtain a fully connected network and the possibility for future extensions with more subtypes of air and earth features.

The fourth observation is that a node is part of a lot of tetrahedrons. The average for the 'toy' data set is strongly influenced by the fact that a lot of tetrahedrons have faces on the convex hull of the tetrahedronisation, thus decreasing the average number of tetrahedrons spanned by a node. The 15 tetrahedrons per node is thus a lower bound of the range one could expect. Okabe *et al.* (2000, table 5.11.2) report a theoretical number of slightly over 27 tetrahedrons per node for a tetrahedronisation of an evenly distributed set of points (a Poisson Delaunay diagram in $\mathbb{R}^3$). Most likely this number acts as an upper bound in most cases. This number is especially interesting in section 7.3, since this number is a first indication of the number of geometry repetitions in the tetrahedron table, since tetrahedrons are encoded by their nodes.

	'Toy' data set	**Rotterdam data set**	**DUT campus data set**
No. of buildings	1	1,796	306
No. of tetrahedrons	117	167,598	131,457
No. of constrained triangles	70	54,566	87,316
No. of input faces	*27*	*16,928*	*4,503*
No. of TEN nodes	31	30,877	23,260
No. of input nodes	*20*	*26,656*	*8,182*
% air / earth tetrahedrons	83.8%	77.7%	70.9%
Average no. of tetrahedrons per building	19	20.8	125.1
Average no. of tetrahedrons a node is part of	15	21.7	22.6

Table 7.3: *Comparison of tetrahedronised data sets*

One should note that the tetrahedronisation results in one network, i.e. the space in between the buildings is tetrahedronised as well. The increase in the number of nodes is caused by the addition of Steiner points; additional points required to

either enable tetrahedronisation or to improve tetrahedronisation quality (in terms of altering ill-shaped triangles and tetrahedrons to avoid numerical instability). The results for the 'toy' data set were already illustrated in figure 5.1, since it shows the original input (the facets), the resulting tetrahedronisation and a derived visualisation of all feature boundaries (i.e. constrained triangles). The results of the Rotterdam data set tetrahedronisation are illustrated in figure 7.6. Due to the large number of tetrahedrons, illustrating all tetrahedrons will result in one big black spot. As an alternative only the constrained triangles are drawn. Note that this figure clearly underlines the advantage of using an irregular data structure: the big apartment buildings in the upper part of the figure are modelled with a few large tetrahedrons (and thus large triangles can be seen in the facades), while smaller and more complex houses are modelled with much more smaller tetrahedrons.

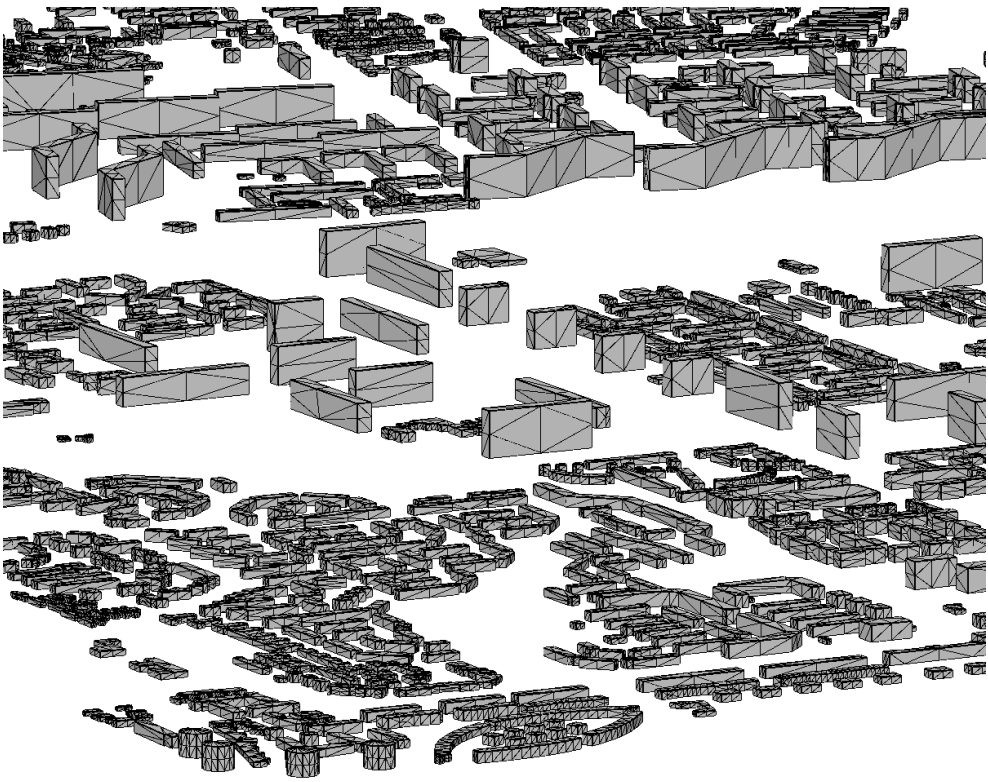


Figure 7.6: *Tetrahedronisation of the Rotterdam data set (only constrained triangles are drawn)*

Figure 7.7 shows a small selection of buildings with more complex shapes compared to the big apartment buildings. Tetrahedrons can also be used to approximately model circular buildings and buildings with small extensions. Note that this figure

is an enlargement of the foreground of figure 7.6, seen from the opposite site. To illustrate the concept of the full volume partition, figure 7.8 shows a small part of the TEN network. The outlines of the four circular buildings, that can be seen in the background of figure 7.7, are marked in grey in figure 7.8. Note that despite its already apparent complexity this figure still is only a view from above, thus obscuring all edges in vertical planes. For instance, one can recognise only the roof of the circular buildings, all edges in its walls and the floor outline are not visible.

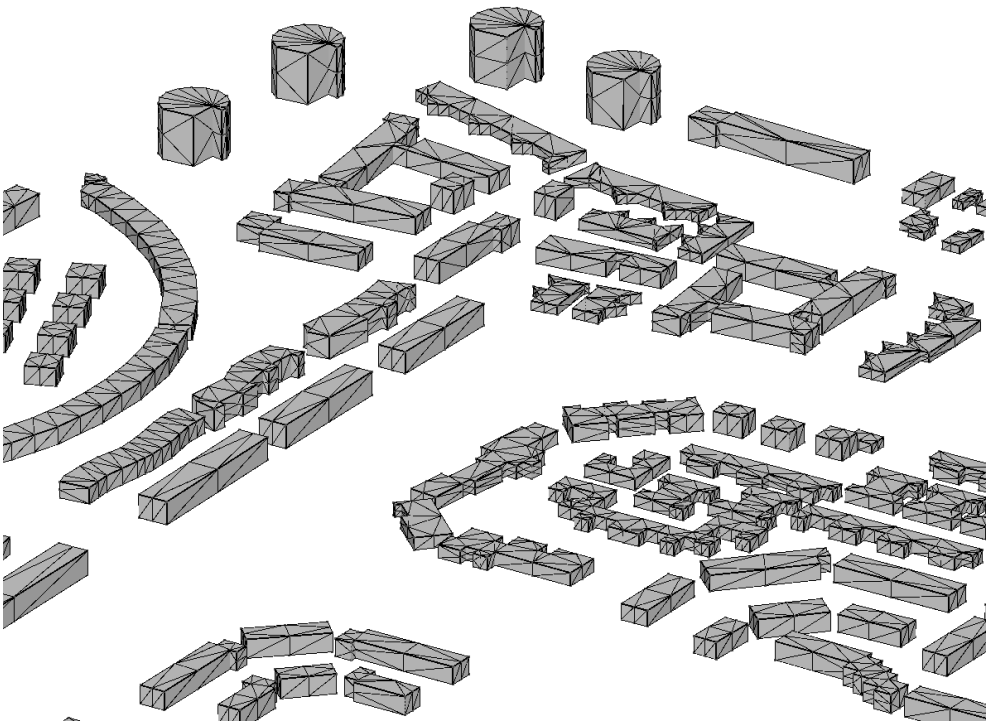


Figure 7.7: *Tetrahedronisation results for more complex shapes: circular buildings and buildings with small extensions*

Figures 7.9 and 7.10 show the results of the tetrahedronisation of the Delft campus data set. Only constrained triangles are shown in these figures, since the full tetrahedronisation including all earth and air tetrahedrons would be hard to interpret. Both illustrations demonstrate that the tetrahedronisation can deal with real 3D situations. The geological feature is visible in both illustrations (the sub-surface feature), and figure 7.10 shows also the viaduct (for Delft locals: the Kruithuiswegviaduct crossing the Mekelweg).

After tetrahedronisation the results are loaded into the simplicial complex-based structure. First the two output files from TetGen (`*.ele` file: a text file describing all

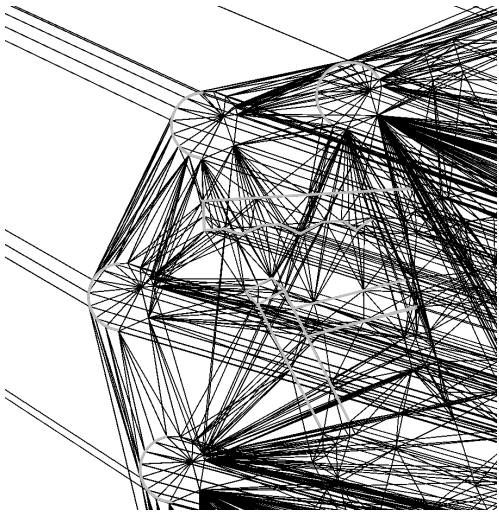


Figure 7.8: *Impression of TEN complexity: top view of TEN network. The four circular buildings from figure 7.7 are marked in grey*

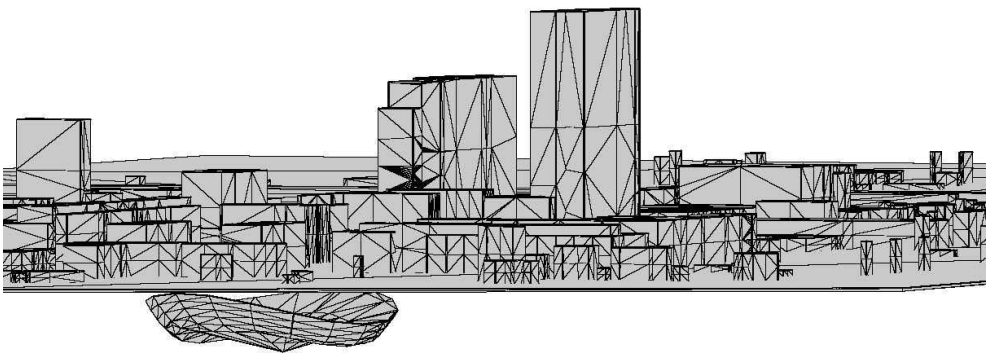


Figure 7.9: *Tetrahedronisation results of the Delft campus: view from the north-west*

tetrahedrons by their node id's and object id's and `*.node` file: a file with all nodes) are loaded into temporary tables with SQL*Loader. Depending on the selected concatenation method the appropriate tetrahedron codes have to be constructed and loaded into the tetrahedron table. In case of identifier concatenation the nodes are loaded into the node table. For instance in case of the Rotterdam data set the tetra-

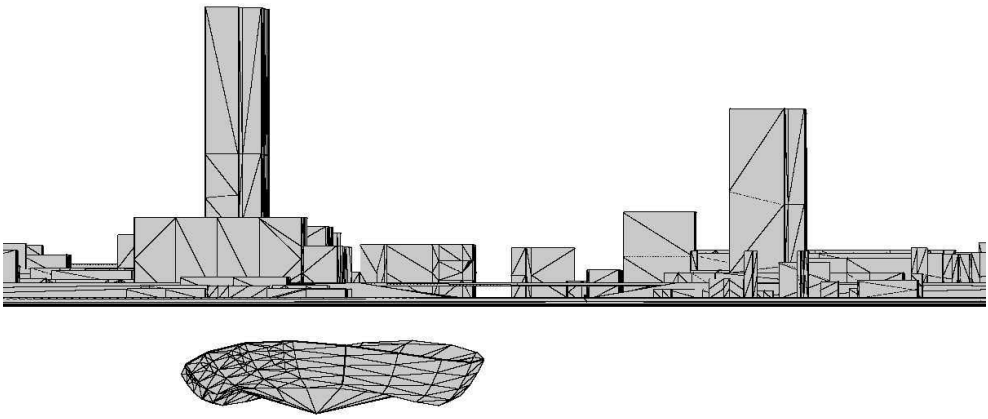


Figure 7.10: *Tetrahedronisation results of the Delft campus: view from the south-east. Note that the viaduct is visible in the centre of the image.*

hedron table consists of 167.598 tetrahedrons. For each tetrahedron $< a, b, c, d, oid >$ in this table it is ensured (as described in section 5.4) that it is sorted according to the arbitrary criterion $a < b < c < d$ and that it has positive orientation (i.e. all normal vectors of boundary triangles point outwards, achieved if necessary by swapping c and d). After that, views are created with triangles, constrained triangles, edges and, in case of coordinate concatenation, nodes by repeatedly applying the boundary operator. Since the simplicial complex-based structure contains opposites of all triangles, the numbers of TEN elements differ from the initial tetrahedronisation. From the 167,598 tetrahedrons 670,392 (4 $\times$ number of tetrahedrons) triangles are derived, of which 109,120 are constrained triangles. This number of constrained triangles slightly differs from multiplying the original 54.566 constrained triangles by two (because of inclusion of the opposite), since the outer boundary of the TEN consists of twelve triangles without a opposite. The edge view provides information for the 198,480 edges. Note that edges are described by their nodes alone, so without inherited object id, opposite or sign, otherwise this number would have been significantly higher.

7.3 Evaluating storage requirements

As stated before, the current implementation is very straightforward: obtaining a working implementation was strongly favoured over minimising storage requirements or optimising performance. Nevertheless, two major comparisons of storage requirements need to be made: first the coordinate concatenation vs. identifier concatenation (the two variants of the simplicial complex-based approach) and second the simplicial

complex-based approach vs. the polyhedron approach. This section will make these two comparisons. For the polyhedron comparison, features are converted to Oracle Spatial 11*g* polyhedrons (see appendix III for the conversion to polyhedrons) based on their boundary triangulations. Polyhedrons in Oracle Spatial 11*g* are described as a set of polygonal faces (in this case only triangular faces are used), each described by their vertices. Since the air and earth tetrahedrons are not exported into polyhedrons, polyhedron storage is also compared to an estimation of the storage space that the identifier concatenation approach would take if air and earth are not explicitly modelled (and the model would thus consist of separate TENs). Table 7.4 provides a small overview of storage requirements and will act as a basis for comparison. It contains only the required tables, since views do not take any storage space.

	'Toy' data set	**Rotterdam data set**	**DUT campus data set**
Oracle 11 polyhedrons	0.20301 MB	3.90 MB	0.95 MB
Coordinate concatenation	0.0234 MB	87.30 MB	58.04 MB
Identifier concatenation	0.0234 MB *node: 0.0078 MB* *tetr.: 0.0156 MB*	20.73 MB *node: 1.46 MB* *tetr.: 19.27 MB*	16.17 MB *node: 0.95 MB* *tetr.: 15.22 MB*
Estimation id concatenation without air/earth	0.0105 MB *17.2 %*	5.76 MB *22.3 %*	5.34 MB *29.1 %*

Table 7.4: *Comparison of storage requirements. Note that the estimation of storage requirements of identifier concatenation without air and earth is only a rough calculation, based on the percentage of air/earth tetrahedrons (as reported in table 7.3). The percentage of non air/earth tetrahedrons is applied to the storage requirements for the tetrahedron table and this is added to the node table storage requirement.*

7.3.1 Coordinate concatenation vs. identifier concatenation

The storage requirements for the Rotterdam data set, stored using identifier concatenation, were previously published by Penninga and van Oosterom (2008b). In that publication the assumption was made that coordinate concatenation would require more storage capacity. Table 7.4 clearly confirms this assumption for the Rotterdam data set, since coordinate concatenation requires about four times more storage space. The explanation for this difference lies in the coordinate repetition within the data structure. All tetrahedrons, regardless the selected concatenation method, are described by their nodes. However, since each node is part of multiple tetrahedrons, this node will appear multiple times in the tetrahedron table. Repeating the node coordinates several times requires more storage space than repeating the node identifier. Table 7.3 provided an indication of how often these coordinates or identifier are

repeated. For the 'toy' data set this was on average 15 times, while the Rotterdam and DUT campus data set show an average of around 22. As mentioned before, the number of 15 is likely to be too low to be representative, due to the large amount of tetrahedrons that lie on the outer boundary of the TEN. The theoretical number of (slightly more than) 27 tetrahedrons per node, as reported by Okabe *et al.* (2000, table 5.11.2) for a tetrahedronisation of an evenly distributed set of points, might act as an upper bound.

Although the repetition explains the significant differences in storage requirements for the Rotterdam data set, one can not generalise these outcomes. Table 7.4 reports equal storage requirements for both concatenation methods for the 'toy' data set. This is caused by the unrealistic coordinates used in this data set. Since it is a fake data set, all coordinates are small integers. Obviously redundant storage of these concatenated coordinate integers does not differ from redundant storage of concatenated identifier integers. The Rotterdam buildings are described by their coordinates in the Dutch national grid and thus require more storage space than the identifiers. However, if one extends this to a national data set at the same level of detail, a tetrahedron in coordinate concatenation form will still require the same amount of storage space, whereas the identifier concatenation storage requirements will increase with the increase in the TEN size, due to the required very large node identifiers. This will lead to differences that are less than the factor four reported for the Rotterdam and the DUT campus data set. Furthermore, switching to a binary representation (see section 7.6) or adapting a coordinate difference approach for three of the four tetrahedron nodes (see section 7.6.2) will reduce data storage requirements for the coordinate concatenation approach. Nevertheless, for 'local' data sets one might prefer (from a storage requirements point of view) identifier concatenation.

However, reducing data storage might deteriorate performance, as additional operations are necessary to perform geometrical operations on top of simplexes. If one thinks for instance of the operation that checks whether a tetrahedron is oriented positively or negatively, one needs the node coordinates to calculate a normal vector on one of the triangles and calculate the angle between this normal vector and a vector from the triangle to the opposite, fourth node to determine whether the normal points inwards or outwards. To perform this operation in the identifier concatenation approach, one has to search the node table first to obtain the node geometries. It should be noted that this effect is probably smaller that the currently required ASCII to binary conversions. Furthermore, coordinate concatenation might offer possibilities for spatial clustering (Penninga and van Oosterom 2008b). In order to enable efficient implementation of spatial queries such as the rectangle (or box) selections, large real world data sets will require spatial organisation of the data. Spatial organisation includes spatial clustering (things close in reality are also close in computer memory, which is tricky given the one dimensional nature of computer memory) and spatial indexing (given the spatial selection predicate, the addresses of the relevant objects can be found efficiently). If the current coding of the tetrahedrons (first x, then y, then z) is replaced by bitwise interleaving, the tetrahedron code itself may be used for spatial clustering (similar to the Morton code) and also be used (together with

the TEN structure) for spatial indexing without using additional structures, such as quad-trees or r-trees (see section 7.6.1 for more details). Only the coboundary references of the triangles might need functional indexes to improve performance.

7.3.2 Simplicial complex-based storage vs. polyhedrons

Table 7.4 shows storage requirements for the different data sets in different formats. Apparently the polyhedron approach is significantly smaller. However, to perform a fair comparison, one should take into account that the polyhedron approach only covers the 'classical' topographic features, thus omitting the air and earth. In the previous section, table 7.3 showed that the percentage of air and earth tetrahedrons can be higher than 75%. In order to make a fair comparison between the tetrahedronised data sets and the polyhedron storage, one has to take this factor into account. When these tetrahedrons are ignored, the comparison shows some interesting results. The Rotterdam data set in identifier concatenation form requires 20.73 MB of storage space (19.27 Mb for the tetrahedron table and 1.46 MB for the node table). It turned out that about 77.7% of the tetrahedrons represent either air or earth. As a result the tetrahedronised buildings require about 5.76 MB (22,3% of 19.27 MB plus 1.46 MB). This estimation shows storage requirements of about the same order of magnitude as the polyhedron approach. Unfortunately this conclusion does not hold for the DUT campus data set. Based on closer visual inspection of the tetrahedronisation results of this data set, it is assumed that this different outcome is mainly caused by the unnecessary insertion of large number of tetrahedrons due to uncleaned data as described earlier. Figure 7.5 illustrated the case with two close parallel lines in a facade, which triggered the creation of a lot of additional tetrahedrons. The presence of small details will not necessarily trigger these additional tetrahedrons, as can be seen in figure 7.11. The difference between the two situations (figure 7.5 and figure 7.11) is that in the latter case the small details are not coplanar, whereas in the former case these small faces are coplanar. Nevertheless, generally speaking an increase in scale will increase the number of tetrahedrons, thus decreasing the compactness of the simplicial complex-based approach.

Still one needs to take into account that the current implementation of identifier concatenation uses character strings, whereas bit strings would be more storage efficient. Using binary representations in the test implementation will save storage space (estimated between a factor 2 to 3). Furthermore switching to binary format will increase performance and ASCII to binary conversions will become obsolete.

Although significant differences exist in the ratio between a polyhedron and a simplicial complex-based approach, the overall result is still interesting. It is plausible that this difference will be smaller as soon as the negative influence of unwanted artefacts in the DUT campus data set on the storage requirements is excluded. So, if the volume partition approach is ignored, optimising the simplicial complex-based approach (in terms of switching to binary format, similar to the polyhedron format) may lead to a comparable or, in some cases, even more compact representation than polyhedrons. This result contradicts with the prevailing view that tetrahedrons are

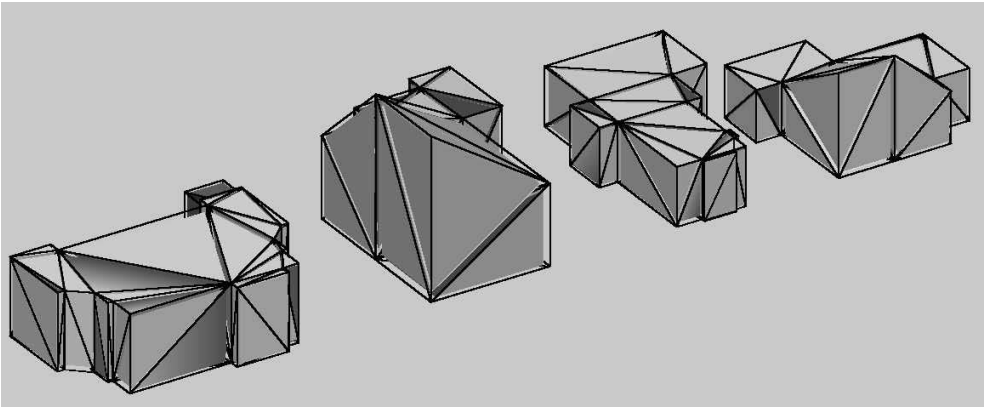


Figure 7.11: *Small details such as the building extensions (less than one metre) cause less problems in case the parallel lines are not all coplanar (compared with figure 7.5)*

more expensive in terms of storage than polyhedrons, as expressed for instance by Zlatanova *et al.* (2004, p. 424): 'An additional disadvantage of TEN is its much larger database size compared with other representations'. Obviously this outcome is also influenced by the rather redundant implementation of polyhedrons in Oracle Spatial 11*g*, since nodes are repeated for each face. An alternative implementation, for instance the one suggested by Arens *et al.* (2005), that eliminates this redundant storage of nodes, might be smaller.

7.4 Evaluating initial visualisation tools

Some basic visualisation tools have been developed to visualise the tetrahedronised topographic features. Penninga *et al.* (2006) describe a realisation of a simple 3D viewer, based on an available 2D viewer (i.e. Oracle AS MapViewer (Kothuri *et al.* 2004)) and the implementation of one 3D rotation function `rotate_geom` (suitable for any type of Oracle spatial geometry). This rotation will result in a new 2D view that effectively looks like a bird's-eye view of a 3D data set. Figure 7.12 shows the first version of the 'toy' data set as described in section 7.1.1, with a house and a road in front of it. Note that the SQL select statement, that selects and rotates edges from view full_edge, is visible in the bottom of this screendump. The queries resulting in the coloured road and roof top triangles are not visible. Penninga *et al.* (2006) suggest further improvements, such as implementing hidden line/hidden surface removal by depth sorting after rotation, adding semi-transparency and developing a GUI that defines several views with different rotation angles, thus enabling viewing from different viewpoints. This additional functionality will transform existing 2D viewers into useful '3D' viewers.

True 3D visualisation of tetrahedronised data sets, using formats such as X3D,

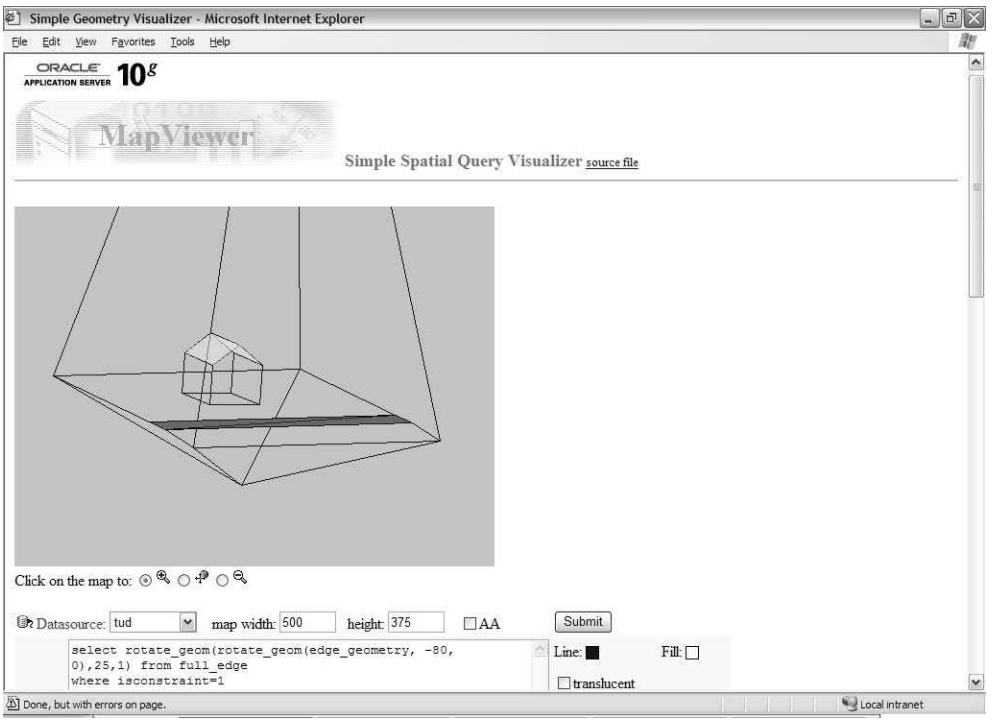


Figure 7.12: *Combining Oracle AS MapViewer (a 2D viewer) and a 3D rotation function results in an initial '3D' viewer*

CityGML and KML, is described by Wesselingh (2007). In this research a web-based viewer is developed (Webviewer Wesselingh 2007), capable of visualising data directly from the database. Depending on the preferred visualisation either the tetrahedron table or the constrained triangle view are used to collect all required data. An example of a visualisation derived from the constrained triangle view is shown in figure 7.13. In this visualisation only the buildings were selected, thus showing no earth surface. Such visualisations can become more realistic by the addition of textures. Since the data model uses opposites one can even use different textures for different sides of for instance a wall, thus enabling more realistic exterior and interior modelling of buildings.

7.5 Discussing requirements for 3D data sets with correct topology

The previously described tests with data sets did not only result in insights on the capabilities of the new method, but also in insights on additional requirements for 3D

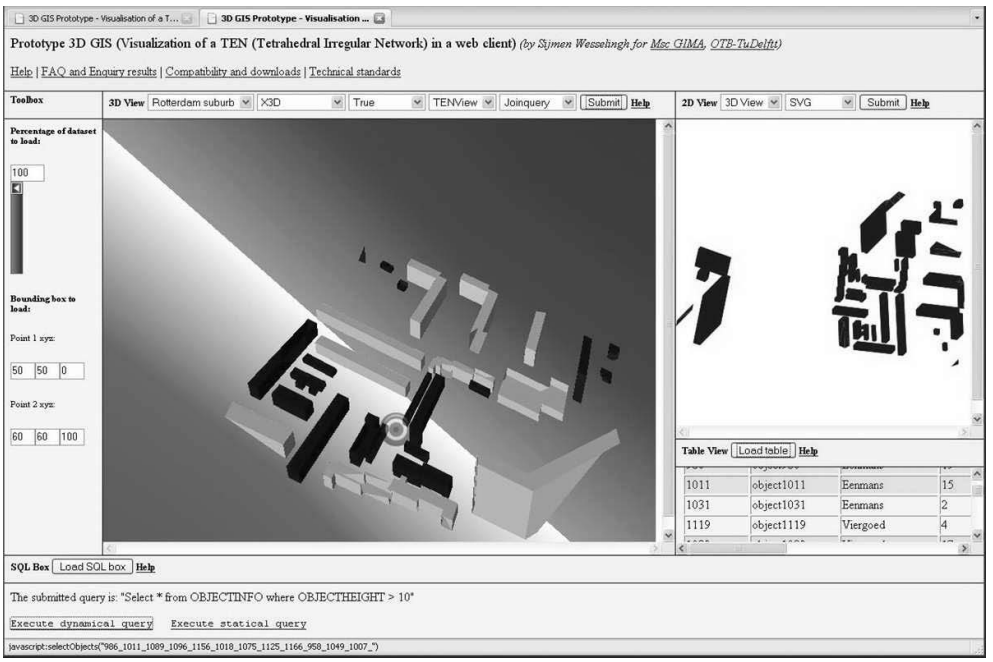


Figure 7.13: *Visualisation in X3D of part of the Rotterdam data set, with different colours for buildings taller than 10 meters. Note that the data set is shown in its rotated form, as described in section 7.2 (source: Webviewer Wesselingh (2007), Wesselingh (2007))*

data sets. As mentioned in the beginning of this chapter, most test data sets were created (partially) by extrusion. The extrusion algorithms are supposed to result in valid PLC's in order to be able to be used as input in the tetrahedronisation software. However, just extruding a 2D map polygon-by-polygon will usually result in a topologically incorrect PLC. Figure 7.14 shows some potential errors in extrusion algorithms. At the left the 2D map is displayed. Two adjacent buildings (polygons described as list of boundary points) share boundary $< v_1, v_2 >$. First this adjacency has to be detected to make sure that the shared points in both buildings refer to the same node (thus preventing the existence of multiple nodes at the location of v_1 and v_2). Now each building can be extracted (figure 7.14: middle). Extruding the first building (with h=5) results in the creation of nodes v_7, v_8, v_9 and v_{10}. During extraction of the second building one should not only create nodes $v_{11}, v_{12}, v_{13}, v_{14}$ and v_{15} and the connecting walls, but also detect that the newly created nodes v_7 and v_8 intersect these newly created walls. In this case three separate faces are needed to correctly describe the wall between the two buildings (figure 7.14: right).

Another important requirement is that the polygons are closed and do not intersect other polygons. The same criteria are also important in case a 3D data set in

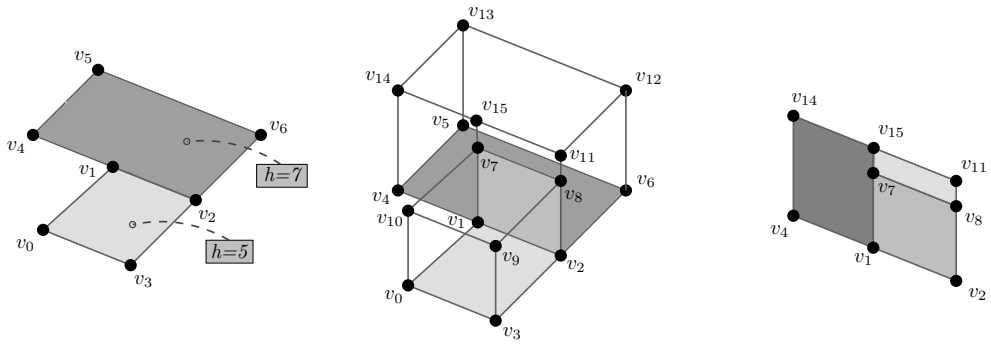


Figure 7.14: *Possible topological errors introduced by feature-by-feature extrusion: the shared boundary $< v_1, v_2 >$ has to be detected to avoid the creation of multiple nodes on the locations of v_1 and v_2 and the intersecting planes in the shared wall have to be detected and split to create a topologically correct description.*

geometric form needs to be transformed into a correct topological description of the same data set. Furthermore, when transforming for instance a polyhedron data set into a topological format, not only each individual polyhedron has to be valid, but also the resulting set of polyhedrons. For instance, two watertight polyhedrons might still intersect each other (beyond tolerance values) and this needs to be avoided.

Although not topologically incorrect, situations as described in section 7.2 (where multiple unnecessary points were located on a line) will also have a negative impact on the usability of a data set in the simplicial complex-based data structure. Another requirement might be that for instance the earth surface, including building footprints, forms a planar partition.

7.6 Identifying future developments*

Within this research not all ideas have been implemented and some of the ideas were triggered by the evaluation tests. This section will present some of these ideas to illustrate the possibilities of the simplicial complex-based approach.

7.6.1 Improving performance: spatial clustering and indexing

Section 7.3 already mentioned the possibility to utilise the tetrahedron codes (based on coordinate concatenation) for spatial clustering. This idea is based on continuing a line of argument from the observation in section 5.3.2 that a node geometry in itself is already an unique identifier. Since node identifiers might be obsolete, no additional

*This section is based on the open issues 0, 2 and 4, as described in Penninga and van Oosterom (2008b). These open issues were identified after initial tests (presented in the same paper). Therefore this section is included within this chapter.

index on top of these indexes is required either, as long as some sort of sorting is possible. Bitwise interleaving of node coordinates will result in such a sortable code. The concept is to represent geometry (for instance node *(4,1,5)*) in binary format (*x=100, y=001, z=101*) and then interleave these binary representations into a single code (*101000011*). With this sortable node code, the concatenation of nodes can be sorted as well. This will result in a 3D Morton code-like spatial clustering of tetrahedrons. This tetrahedron encoding within a TEN structure can also be used for spatial indexing without additional structures, such as quad-tree or r-trees. Since these structures require a significant amount of storage space and maintenance during updates (van Oosterom and Vijlbrief 1996), this will be a big advantage, especially since one saves also the storage space for meaningless unique identifiers.

7.6.2 Dealing with storage requirements: storing all coordinates vs. storing differences

Assuming that one opts for the coordinate concatenation approach, storage requirements can be reduced by avoiding storage of the full coordinates. Since the four nodes are relatively close to each other, one might choose to store the coordinates of one node and only give difference vectors to the other three nodes: $x_1y_1z_1x_2y_2z_2x_3y_3z_3x_4y_4z_4$ would change into $xyz\delta x_1\delta y_1\delta z_1\delta x_2\delta y_2\delta z_2\delta x_3\delta y_3\delta z_3$. Switching to coordinate difference may lead to an estimated storage reduction with factor two or three. As with the choice between coordinate concatenation and identifier concatenation, reducing data storage will come at a price. Again additional operators are required to reconstruct the four node geometries when necessary. However, if these can be implemented efficiently (and there is no reason why this can not be done), these could be used in a view translating the compact physical storage representation in the more verbose full representation. Also the bitwise interleaving approach may still work well with this approach, as it is sufficient to apply bitwise interleaving only to the first coordinate.

7.6.3 Improving edit functionality: snapping

The current implementation requires 3D data sets to load into the database. Although research efforts are made to increase availability of such datasets (Oude Elberink and Vosselman 2006), dependence of the availability of such data sets seriously limits applicability of the data structure at this time. Therefore additional functionality is required to switch from importing 3D data sets into importing 3D data from different sources. One can imagine that creation of a 3D topographic model starts with the creation of the earth surface, followed by inclusion of 3D buildings. In general, buildings are build on top of the earth surface. As the earth surface and building data originates from different sources, these objects are not likely to fit together perfectly. To cope with such situation, one needs a snap-to-earth-surface operator. Such an operator will project the buildings footprint onto the terrain and determine the distance between terrain and building undersides. If this distance is smaller than a certain pre-set tolerance, the building will be placed on the terrain by applying a vertical displacement, thus ensuring a tight fit. Two options exist for this, as one

can either adjust the buildings underside to fit the terrain or adjust the terrain to fit the (usually flat) underside of the building. The snapping operator can also be utilised for inclusion of for instance infrastructural objects and land coverage objects. Although snapping to the earth surface is probably the most obvious snapping target, usability of such an operator is not limited to the earth surface. For instance, one can also think of snapping a building to another building, thus ensuring the presence of a shared wall.

Chapter 8

Conclusions

The objective of this research was to develop a data model (i.e. both a conceptual model and a data structure) for 3D topography, capable of handling large data volumes, that offers support for loading, updating, querying, analysis, and validation. The reasons for the development of such a 3D model are twofold: on the one hand it is demand-driven due to increasing importance of sustainability and environmental issues, thus requiring more accurate 3D analysis, on the other hand it is supply-driven since recent sensor technology developments resulted in both increasing availability and increasing point density of 3D data. The results of this research are summarised in section 8.1. Section 8.2 draws conclusions from these results and these results are discussed in section 8.3. The chapter ends with an outlook of future research in section 8.4.

8.1 Results

The main objective of this research is expressed in the main research question:

How can a 3D topographic representation be realised
in a feature-based triangular data model?

Within this main research question, two key questions were distinguished. The first one is how to develop a conceptual model that describes the real world phenomena (the topographic features), regarding the general purpose-characteristic of topographic data sets. This question is answered in section 8.1.1. The second one is how to implement this conceptual model within a DBMS environment, in other words, how to develop a suitable data structure? Section 8.1.2 presents such a suitable data structure.

8.1.1 A conceptual model for 3D topography

One of the basic assumptions within this research is the use of triangular data models. These structures are selected due to computational advantages, the flatness of the faces (well defined by three points) and the presence of well-known topological relationships. As a result, topographic features will be described as sets of triangles and these features will be connected by triangles as well, thus creating a fully connected triangular network. Chapter 3 introduced two different approaches to triangular modelling of 3D topography. The first one is a very pragmatic hybrid approach that combines a 2.5D surface with 3D objects for those cases where 2.5D modelling is not sufficient. In terms of triangular data structures, this approach combines a TIN with several TENs. This irregular data structure not only allows varying point density (depending on local model complexity), but extends this irregularity into varying even model dimensionality, thus offering the ultimate fit-for-purpose approach. Unfortunately, connecting TIN and TEN networks appeared to be very difficult at design level. The second approach avoids these problems, since it is a full 3D approach using only a TEN.

With respect to the selected full 3D modelling approach, two fundamental observations are of great importance:

- Physical objects have by definition a volume. In reality, there are no point, line or polygon objects; only point, line or polygon representations exist (at a certain level of abstraction/simplification).
- The real world can be considered a volume partition (within one theme): a set of nonoverlapping volumes that form a closed (i.e. no gaps within the domain) modelled space. Objects like 'earth' or 'air' are thus explicitly included in the model.

In topographic data models, planar features like walls or roofs are obviously very useful. They can be part of the volumetric data model as 'derived features', i.e. these features depend on the relationship between volume features. For example, the earth surface is the boundary between earth and non-earth features, while a wall or a roof are the result of adjacent building an air features. In terms of UML, these surface features are modelled as association classes. As a result, surface features are lifetime dependent from the association between two volume features.

A number of advantages of a full volumetric modelling approach can be identified. First of all, explicit inclusion of air and earth extends the analytical capabilities of the model, since the air and earth are often subject of analysis. In case of the air, one can think of modelling noise propagation, air pollution or particulate matter distribution. Second, these general air and earth types might be subclassified into air traffic or telecommunication corridors, geological layers or oil and gas reservoirs, thus increasing the data richness of topographic data sets. Integrating more types of information extends the number of possible applications and analyses, since all these different sources can be related to each other. The third and last advantage is that all features are well-connected within the volumetric data set, thus enabling topological

approaches to analysis and validation (although the actual benefits will depend on the implementation of the accompanying data structure).

As a result, topographic features will be modelled in a TEN. Each feature will be represented by a set of tetrahedrons. The presence of correct feature boundaries is ensured by the use of constrained edges and triangles. Constrained edges and triangles are preserved under edit or refinement operations on the TEN. Since the real world is considered as a volume partition, each tetrahedron will represent one (part of a) feature. In other words, this representation is a 3D single valued vector map. Less dimensional features, such as the earth surface, can be modelled as derived features and are defined by the neighbour relation between volume features, for instance between earth and air. Database views might be defined to provide access to such special features, thus resulting in a TEN subset that is basically a TIN.

8.1.2 A data structure for 3D topography

A solid mathematical foundation

One of the key characteristics of the presented TEN data structure is that it is based on Poincaré simplicial homology. The operators and definitions from this field, as presented in chapter 4, offer a solid mathematical foundation for the data structure. Simplexes are well defined, ordered and constructed of simplexes of lower dimension. The boundary operator can be used to derive these less dimensional simplexes. Based on the ordering of simplexes one can determine orientation. One might consider orientation as not a real mathematical characteristic of simplexes, since the concept of orientation is based on arbitrary conventions; nevertheless it enables useful operations within a GIS context, for instance in defining inside and outside. Within this context, simplicial homology offers some favourable characteristics. First, the zero homomorphism results in consistent orientation of faces of a simplex. For instance, the normal vectors of the four boundary triangles of a tetrahedron will either all point inwards or all point outwards. Second, obtaining a simplex with opposite orientation requires only a transposition of its nodes.

A very important concept from simplicial homology is the simplicial complex, since such a set of connected simplexes will be used to model 3D topography. In terms of simplicial homology, a TEN can be defined as a simplicial complex of homogeneous dimension of three that consists of face-connected *3*-simplexes. In addition to the requirements following from this definition (no self-intersections, no dangling edges, faces or tetrahedrons), a TEN is also supposed to consist of positively oriented *3*-simplexes within the scope of this dissertation. In other words, the normal vectors of the boundary faces of each *3*-simplex point outwards, which is common practice in computer graphics and GIS. Since topographic features will be represented as set of tetrahedrons, it is important that the boundary operator can also be applied to simplicial complexes, thus able to derive feature boundaries.

A DBMS data structure

Applying definitions and operators from simplicial homology enables one to store a constrained TEN in a relatively compact way. The new simplicial complex-based method (as introduced in chapter 5) requires only explicit storage of tetrahedrons, while simplexes of lower dimensions (triangles, edges, nodes), constraints and topological relationships can be derived in views. In this implementation, simplexes are encoded by their vertices. The boundary operator is used to derive views with the simplexes of lower dimension. In order to avoid explicit storage of constrained triangles and edges, simplex encoding is extended with a feature identifier. In other words, a tetrahedron is encoded as $S_3 =< v_0, v_1, v_2, v_3, fid >$. The boundary operator is implemented such that oriented triangles inherit the feature id from the tetrahedron they partially bound. Since the implementation uses triangles and opposite triangles ('halftriangles'), neighbouring tetrahedrons are separated by a triangle and its opposite. In case both tetrahedrons represent the same topographic feature, both the triangle and its opposite have the same inherited feature id. However, if the triangle is a constrained triangle, the tetrahedrons will represent different features and thus the triangle and its opposite will have different inherited feature identifiers. As a result, non-constrained triangles cancel out in pairs (since the triangle and its opposite will have identical triangle codes apart from the sign), while constrained triangles remain present. Topological relationships can also be derived, both at feature and at TEN level, for instance the neighbour relationship between tetrahedrons. This is achieved by creating a view with triangle opposites and defining the triangle view such that the coboundary is known. Neighbouring tetrahedrons can be identified by deriving the boundary triangles and querying their opposites first and their coboundaries afterwards, thus resulting in the four neighbouring tetrahedrons.

Two variants in simplex encoding have been developed. The *coordinate concatenation* is the most rigorous attempt to eliminate redundancy. By concatenating the x, y and z coordinate as node identifier, the node geometry in itself becomes an unique identifier. Simplexes of higher dimension are described by a concatenation of node codes. The alternative approach, *identifier concatenation*, requires an additional node table. Separate (meaningless) node identifiers are used to encode simplexes. This approach reduces the number of coordinate repetitions, since the coordinates of a specific node will be repeated in multiple tetrahedrons. Despite the substantially reduced storage requirements of identifier concatenation, this is not automatically the best solution. Applying coordinate concatenation yields the potential to sort bitwise-interleaved tetrahedron codes, such that this ordering acts as a spatial location code, thus eliminating the need for additional spatial index creation. Not only would this affect the overall storage requirements, but it will also affect performance. A clean comparison, based on multiple realistic data sets, is needed to determine the most suitable concatenation approach.

Validation can be performed at TEN level as well as at feature level. To validate a volume feature, three checks need to be performed. First of all a valid TEN is

required. A valid TEN meets the criteria described in the definition of a TEN as a simplicial complex (as given earlier), including the positive orientation. Secondly, the volume feature boundary should be watertight (which is the case if the zero homomorphism holds). The last check is to see whether the interior of the volume feature is face-connected. An overall validation of topographic features in a simplicial complex-based structure requires therefore three steps: first the TEN has to be valid, second all individual features have to be valid and third and last the set of all features has to be consistent, meaning that there are no gaps or overlays between features.

An editable data structure

A critical factor in the creation of data structure feasible for topographic data sets, is the possibility to perform incremental updates. Complete rebuilds of the TEN structure will be time-consuming due to the expected data volume and should therefore be avoided. Furthermore, edits are usually related to local changes, so edit operations are required that affect the TEN structure as locally as possible, thus trying to avoid 'domino' effects of alterations. Four steps can be distinguished in the feature insertion process: first a surface boundary triangulation of the feature is created. The resulting edges are then inserted as constrained edges into the TEN. Next one has to ensure the presence of constrained faces and as the fourth and last step the feature's interior is modelled, including reclassifying the tetrahedrons involved. In order to demonstrate the usability of the simplicial complex-based approach for 3D topographic purposes, one need to guarantee successful update operations, since the frequent updates are a key characteristic of topography. Chapter 6 presented a new approach to the insertion of feature edges that guarantees successful insertion of these edges. Nine exhaustive and mutually exclusive cases were distinguished, depending on the type of intersection between the two nodes of a feature edge and the existing TEN structure. The following annotation is used: I_{ij}, where i and j indicate the dimension of the simplexes in which interior the two nodes of the constrained edge lie. As a result, the nine cases I_{01}, I_{02}, I_{03}, I_{11}, I_{12}, I_{13}, I_{22}, I_{23} and I_{33} should be interpret as (in case of I_{01}) inserting an edge of which one node is already present and the other lies on an existing edge, as (in case of I_{02}) inserting an edge of which one node is already present and the other lies on an existing triangle and so on. In each case the edge is inserted with a pure local impact on the TEN structure.

8.2 Main conclusions

The previous section summarised the main results of this research. Based on these results, the following conclusions can be drawn:

- This dissertation presents a new topological approach for 3D topography, based on a tetrahedral network. Operators and definitions from the field of simplicial homology are used to define and handle this structure of tetrahedrons. Applying simplicial homology offers full control over orientation of simplexes and

enables one to derive substantial parts of the TEN structure efficiently, instead of explicitly storing all primitives.

- Operators and definitions from simplicial homology provide a solid mathematical foundation for the data structure.
- The simplicial complex-based approach and the vertex encoding including feature identifiers eliminate the need for explicit storage of triangles, edges, nodes (in case of coordinate concatenation), constrained triangles, and constrained edges. As a result, it is a relatively compact data structure.
- The prevailing view that tetrahedrons, compared to polyhedrons, are more expensive in terms of storage, proved to be outdated. Storage requirements for 3D objects in tetrahedronised form (excluding the space in between these objects) and 3D objects stored as polyhedrons, are in the same order of magnitude. Switching to a binary representation and storing coordinate differences instead of coordinates will lead to a significant storage reduction (about factor six) for the coordinate concatenation approach (including air and earth), thus justifying the full decomposition approach.
- A TEN has favourable characteristics from a computational point of view. All elements of the tetrahedral network consist of flat faces (important for clear inside/outside decisions), all elements are convex and they are well defined, thus allowing relatively easy implementation of operations, such as validation of 3D objects.
- DBMS characteristics as the usage of views, functions and function-based indexes are extensively used and contribute to the compactness and versatility of the data structure. Furthermore a database is capable of managing large data volumes, which is an essential characteristic in handling large scale 3D data.
- A full volumetric approach contributes not only to improved analytical and validation capabilities, but also enables future integration of topography and other 3D data within the same volume partition, like geological layers, polluted regions or air traffic and telecommunication corridors. Although the price of this approach in terms of storage space is high (about 75% of the tetrahedrons models air or earth), this approach is likely to become justifiable due to current developments like the trend towards sustainable urban development and increased focus on environmental issues. The impact of these factors on GI-developments is significant, as illustrated by for instance the environmentally-driven INSPIRE project (INSPIRE 2007).
- Constrained tetrahedronisation algorithm implementations are scarce and usually not developed with GI or DBMS applications in mind. TetGen is used in this research and the TetGen website (2007) defines TetGen's main goal as 'to generate suitable meshes for solving partial differential equations by finite element or finite volume methods'. As a result these algorithms are usually not tested with large numbers of features and large numbers of incremental changes.

- The simplicial complex-based approach is not limited to 3D applications, since the underlying mathematical definitions and operators apply in general dimension. For instance, 4D simplexes can be used for spatio-temporal modelling. Although it is hard to imagine or visualise a 4D simplex, simplicial homology shows that five 4D nodes define a 4D simplex, 120 permutations of this simplex code exist (of which 60 will have an orientation opposite to the other 60 permutations), the boundary operator can be used to derive its five boundary tetrahedrons in 4D and the Cayley-Menger determinant may be used to calculate its volume (although it might not be a meaningful).

- Nine exhaustive and mutually exclusive cases can be distinguished in constrained edge insertion (I_{01}, I_{02}, I_{03}, I_{11}, I_{12}, I_{13}, I_{22}, I_{23} and I_{33}), based on the type of intersection between the two nodes of the constrained edge and the existing TEN structure. In each case the edge is inserted with a pure local impact on the TEN, thus enabling incremental updates that affect the TEN structure very locally. These operators guarantee successful update operations, a crucial prerequisite for 3D topography.

- Since the edit operations act as locally as possible, the resulting tetrahedronisation is not necessarily of the best quality. To overcome this drawback, periodical quality improvements need to be made. Three types are distinguished: operators that add vertices, operators that remove vertices and operators that modify the TEN configuration through flips. Every now and then a complete TEN rebuild might be feasible to optimise TEN quality.

8.3 Discussion

The first important discussion topic is to identify the innovative aspects of the proposed method. As discussed in section 2.4.3, neither the idea to use a TEN data structure for 3D data nor the idea to use simplexes (in terms of simplicial homology) in a DBMS implementation is new. However, the proposed approach reduces data storage and eliminates the need for explicit updates of both topology and simplexes of lower dimension. By doing so, the approach tackles common drawbacks as TEN extensiveness and laboriousness of maintaining topology. Furthermore, applying simplicial homology offers full control over orientation of simplexes, which is a significant advantage especially in 3D. In addition to this aspect, the mathematical theory of simplicial homology offers a solid theoretical foundation for both the data structure and data operations. Integrating these concepts with database functionality results in a new innovative approach to 3D data modelling.

Second, the often raised objection of complexity of a TEN approach can be questioned. Obviously, a 1:n relation exists between features and their representations, which might confuse the user as the link between tetrahedrons and the real world features can be hard to recognise. However, one can think of a setup in which the user handles only features (as polyhedrons), while the implemented algorithms translate

these polyhedrons into constrained triangles and use these to construct or update the constrained tetrahedronisation. In other words, the user interface determines the perceived complexity, not the internal representation.

This separation between the internal data representation and a data representation used in the user interface, can easily be extended to the next level: since a TEN structure requires about the same amount of storage space as a polyhedron data type, one might argue that a polyhedron data type in a DBMS should be represented internally as set of tetrahedrons. In this case each individual polyhedron is represented as a TEN, so no full space partition will exist. By doing so, one can benefit from the well-defined character of the TEN and simplify –amongst others– object validation significantly.

The most important issue is to analyse whether the proposed method is feasible. The plea for provable correct modelling methods goes back for two decades (Frank and Kuhn 1986), but is still valid (Frank 2008, Requirement 1). The simplicial complex-based modelling approach provides such a method. This is not only shown by this research, but also by independent research by Hui *et al.* (2006), whose Double-Level Decomposition (DLD) data structure stores a decomposition of a 3D complex as a collection of tetrahedrons, dangling triangles and wire edges. Another important result is that the prevailing view that tetrahedrons are always much more expensive in terms of storage than polyhedrons has been falsified in this research. Obviously, including air and earth within the model comes at a price (approximately 75% of tetrahedrons model air or earth), but –as stated earlier in the conclusions– this approach is likely to become justifiable, due to current sustainability and environmentally-driven developments. The decision to develop the data structure as a database structure contributes to the overall feasibility, since a database will become indispensable due to the expected data volumes.

8.4 Future research

At this point the simplicial complex-based data structure is available in its basic form. Several future extensions of this approach look very promising and deserve further investigation, as well as some improvements of the current implementation:

- Within this research vario-scale representations or generalisation on top of the TEN were out of scope, but this is a very interesting field of research. Generalisation algorithms for TIN structures are well-known and often used to reduce the vast amount of height data points obtained by LIDAR. Also for generalising integrated terrain and 2D object models algorithms are available, for instance the one described by Stoter *et al.* (2005). In general these kind of generalisation algorithms try to detect characteristic points or break lines in a triangulation, followed by a point removal operation that preserves these characteristic features. One can think of similar approaches in 3D, where break lines or characteristic faces have to be detected and preserved. Currently no examples of such 3D TEN generalisation algorithms are known to the author.

- An topic that was briefly mentioned, but out of scope of this research, is temporal modelling with simplicial complexes. Since all underlying mathematical theory applies in general dimension, the possibility of spatio-temporal modelling with 4D simplexes has to be explored. A 4D spatio-temporal partition of space enables consistency checks. Some may argue that 4D is not the way to go given the completely different nature of space and time, which leads to different units along the axis of the 4D space. Nevertheless, interesting spatio-temporal relationships can be detected in such a 4D space. Answering questions like 'where were you when JFK was shot' or 'yesterday I went to this pub, have you ever been there?' boils down to detecting intersecting 4D simplicial complexes. Distance calculations in 4D might be meaningless due to the different units, but become relevant when projected in space (distance) or time (time difference). Both possibilities and limitations of such an approach have to be investigated.

- Closely related to temporal modelling is dynamic modelling in TEN structures. Ledoux (2008) describes a kinetic 3D Voronoi diagram, but all calculations on a moving point are performed within a tetrahedronisation. Based on this part of the algorithm, algorithms capable of handling moving lines, faces and volumes might be developed.

- The modelling of continuous fields (continuous in space, not in time) is also very interesting. One way to go might be an approach similar to the one described by Ledoux and Gold (2007), who argue that one should use both Voronoi diagrams (the primal) and Delaunay tetrahedronisation (the dual) in 3D. In this case the simplicial complex-based structure might be seen as the primal, whereas specific continuous phenomena can be represented in the dual, the Voronoi diagram. Another nice characteristic of applying this primal-dual approach is that the Voronoi diagram will contain information on proximity of topographic features.

- Indoor topography is also a very interesting extension for 3D topography, especially due to the possible applications, for instance in the field of disaster management. Adding indoor features will require a change in the conceptual model, but the data structure is likely to be feasible. Since triangle opposites (halftriangles) are available, separate information can be linked to these two triangles, thus enabling for instance a distinction between the inner and the outer side of a wall.

- Although more applied research, further integration with the subsurface world is recommended. Whereas tetrahedrons are still rather unusual in the GIS world, they are more common in the field of geological modelling, where tetrahedrons are often used as the irregular counterpart of voxels, see for instance (Pouliot *et al.* 2008). Integrating topography with subsurface date might lead to the integration of object and field data, in which subsurface nodes can be seen as sample (i.e. measured) data.

- To edit features within the simplicial complex-based data structure, a set of tetrahedron-based edit operations is required. Snapping functionality is already

described in section 7.6, but other operators are useful as well. One can think of cleaning operators, that remove unnecessary nodes from lines (thus avoiding situations as illustrated in figure 7.5), union operators (that union two separate features), split operators (reverse of union operators) and move operators, that adjust geometrical attributes of features, but not necessarily the topological structure, for instance in case a new, more accurate survey cause small shifts in the location of features. All these operators should work directly on the TEN structure and preserve the Euler-Poincaré characteristic. A nice example of using only Euler operators to update a data structure is presented by Tse *et al.* (2004).

- Several techniques that can reduce storage requirements have to be implemented and tested, such as the switch to binary form including spatial clustering through bitwise interleaving, and the switch to coordinate difference for three of the four nodes of a tetrahedron. Both ideas are described in section 7.6.

- As mentioned in section 8.1.1, the current conceptual model is a single valued vector map. The option to store multi valued vector maps (for instance for the integration of 3D topography and cadastral data) has to be explored.

Bibliography

3D Topography, 2006, *website:* http://www.rgi-otb.nl/3dtopo.

Actueel Hoogtebestand Nederland, 2006, *website:* http://www.ahn.nl.

Arens, C., 2003, "Maintaining Reality; Modelling 3D spatial objects in a Geo-DBMS using a 3D primitive", Master's thesis, Delft University of Technology.

Arens, C., Stoter, J. and van Oosterom, P., 2005, Modelling 3D spatial objects in a geo-DBMS using a 3D primitive. *Computers & Geosciences*, **31**, 165–177.

Beers, B.J., 1995, FRANK - the design of a new landsurveying system using panoramic images. PhD thesis, Delft University of Technology.

Borst, H., 2001, The role of Urbis noise and noise effects maps in local policy. In: *2001 International Congress and Exhibition on Noise Control Engineering*.

Bowyer, A., 1981, Computing Dirichlet Tesselations. *Computer Journal*, **24**, 162–166.

Carlson, E., 1987, Three-dimensional conceptual modeling of subsurface structures. In: *Auto-Carto 8*, pp. 336–345.

Cavalcanti, P.R. and Mello, U., 1999, Three-dimensional Constrained Delaunay Triangulation: a Minimalist Approach. In: *8th International Meshing Roundtable*, pp. 119–129.

Center for Advanced Gaming and Simulation (AGS), 2007, *website:* http://www.gameresearch.nl/research.html.

Chew, L.P., 1993, Guaranteed-Quality Mesh generation for Curved Surfaces. In: *Ninth Annual Symposium on Computational Geometry* (Association for Computing Machinery), pp. 274–280.

Chew, P.L., 1989, Constrained Delaunay Triangulation. *Algorithmica*, **4**, 97–108.

Colins, K.D., 2003, Cayley-Menger Determinant. From Mathworld – A Wolfram Web Resource. *website:* http://mathworld.wolfram.com/Cayley-MengerDeterminant.html.

Comer, D., 1979, The ubiquitous B-tree. *ACM Computing Surveys*, **11**, 121–137.

Coumans, F., 2007, Actueel Hoogtebestand Nederlnad wordt veel gedetailleerder (in Dutch). *VI Matrix*, **15**, 24–27.

Cowen, D., 1988, GIS versus CAD versus DBMS: what are the differences. *Photogrammetric Engineering and Remote Sensing*, **54**, 1551–1555.

Date, C.J., 1986, *An Introduction to Database Systems*, Fourth Edition , Vol. I (Addison-Wesley Publishing Company).

de Vries, J., 2001, "Three dimensional buffering based on Tetrahedron Networks, storage and analysis in a 3D GIS (in Dutch)", Master's thesis, Delft University of Technology.

Delaunay, B., 1934, Sur la sphère vide. *Izvestia Akademia Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, **7**, 793–800.

Edelsbrunner, H. and Shah, N., 1996, Incremental topological flipping works for regular triangulations. *Algorithmica*, **15**, 223–241.

Egenhofer, M. and Frank, A., 1989, PANDA: An Extensible DBMS Supporting Object-Oriented Software Techniques. In: *Datenbanksysteme in Büro, Technik und Wissenschaft. Proceedings of GI/SI Fachtagung, Zürich, 1989*, Informatik Fachberichten (Springer-Verlag), pp. 74–79.

Egenhofer, M., Frank, A. and Jackson, J., 1989, A Topological Data Model for Spatial Databases. In: *Design and Implementation of Large Spatial Databases: First Symposium SSD'89*, 409 of *Lecture Notes on Computer Science* (Springer), pp. 271–286.

El-Harbawi, M., Mustapha, S., Rashid, S.A., Choong, T.S. and AL-Shalabi, M., 2004, Using geographic information systems in assessment of major hazards of liquefied petroleum gas. *Disaster Prevention and Management*, **13**, 117–129.

Ellul, C. and Haklay, M., 2006, Requirements for Topology in 3D GIS. *Transactions in GIS*, **10**, 157–175.

Ford, A. and James, P., 2005, Integration of 3D petroleum datasets in commercial GIS. In: *AGILE 2005, 8th Conference on Geographic Information Science. Conference Proceedings. Estoril, Portugal, May 26-28*, F. Toppen and M. Painho (Eds), pp. 411–418.

Forta, B., 2004, *SQL in 10 Minutes*, Third Edition (Sams Publishing).

Frank, A.U., 2008, Requirements for 3D in Geographic Information Systems Applications. In: *Advances in 3D Geoinformation Systems*, P. van Oosterom, S. Zlatanova, F. Penninga and E.M. Fendel (Eds), Lecture Notes in Geoinformation and Cartography (Springer), pp. 419–423.

Frank, A.U. and Kuhn, W., 1986, Cell Graphs: A provable Correct Method for the Storage of Geometry. In: *Proceedings of the 2nd International Symposium on Spatial Data Handling, Seattle, Washington*.

Giblin, P., 1977, *Graphs, Surfaces and Homology, An Introduction to Algebraic Topology* (Chapman and Hall).

Gold, C.M., 2006, What is GIS and what is not. *Transactions in GIS*, **10**, 505–519.

Gold, C.M., Ledoux, H. and Dzieszko, M., 2005, A data structure for the construction and navigation of 3D Voronoi and Delaunay cell complexes. In: *13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, pp. 21–22.

Guibas, L. and Stolfi, J., 1985, Primitives for the Manipulation of General Subdivisions and the Computation of Voronoi Diagrams. *ACM Transactions on Graphics*, **4**, 74–123.

Guttman, A., 1984, R-trees: A dynamic index structure for spatial searching. In: *Proceedings of 13th ACM SIGMOD Conference*, pp. 47–57.

Hatcher, A., 2002, *Algebraic Topology* (Cambridge University Press) (Available at http://www.math.cornell.edu/∼hatcher).

Heerd, R., Kuijlaars, E., Teeuw, M. and van t Zand, R., 2000, Productspecificatie AHN 2000 (in Dutch). Meetkundige Dienst, Rijkswaterstaat.

Hilbert, D., 1891, Über die stetige Abbildung einer Linie auf ein Flächenstück. *Mathematische Annalen*, **38**, 459–460.

Hui, A., Vaczlavik, L. and Floriani, L.D., 2006, A decomposition-based representation for 3D simplicial complexes. In: *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing* (Eurographics Association), pp. 101–110.

INSPIRE, 2007, Directive 2007/2/EC of the European Parliament and of the Council of 14 March 2007 establishing an Infrastructure for Spatial Information in the European Community (INSPIRE). In: *Official Journal*, April 25, 2007. Entered into force: May 15, 2007. Available at http://www.ec-gis.org/inspire/.

ISO 19101:2002, 2002, Geographic Information – Reference Model. International Organization for Standardization.

ISO 19107:2003(E), 2003, Geographic Information - Spatial Schema. International Organization for Standardization.

Joe, B., 1991, Construction of Three-dimensional Triangulations using Local Transformations. *Computer Aided Geometric Design*, **8**, 123–142.

Joe, B., 1993, Construction of k-dimensional Delaunay Triangulations using Local Transformations. *SIAM Journal on Scientific Computing*, **14**, 1415–1436.

Joe, B., 1995, Construction of three-dimensional improved-quality triangulations using local transformations. *SIAM Journal on Scientific Computing*, **16**, 1292–1307.

Kadaster, 2007, Product leaflet TOP10vecotr. Kadaster Available at http://www.kadaster.nl/pdf/TOP10vector.pdf.

Kazar, B.M., Kothuri, R., van Oosterom, P. and Ravada, S., 2008, On Valid and Invalid Three-Dimensional Geometries. In: *Advances in 3D Geoinformation Systems*, P. van Oosterom, S. Zlatanova, F. Penninga and E.M. Fendel (Eds), Lecture Notes in Geoinformation and Cartography (Springer), pp. 19–46.

Kolbe, T., Gröger, G. and Plümer, L., 2005, CityGML: Interoperable Access to 3D City Models. In: *Geo-information for Disaster Management (GI4DM)* (Springer), pp. 884–899.

Kothuri, R., Godfrind, A. and Beinat, E., 2004, *Pro Oracle Spatial: The essential guide to developing spatially enabled business applications* (Apress).

Kraak, M. and Ormeling, F., 1996, *Cartography, Visualization of spatial data* (Longman).

Kroonenberg, S., 2006, *De menselijke maat – de aarde over tienduizend jaar* (Amstel Uitgevers) (In Dutch).

Kwan, M.P. and Lee, J., 2005, Emergency response after 9/11: The potential of real-time 3D GIS for quick emergency response in micro-spatial environments. *Computers, Environment and Urban Systems*, **29**, 93–113.

Lawson, C., 1977, Software for C1 Surface Interpolation. In: *Mathematical Software III*, J. Rice (Ed.), pp. 161–194.

Ledoux, H., 2006, Modelling Three-dimensional Fields in Geoscience with the Voronoi Diagram and its Dual. PhD thesis, University of Glamorgan/Prifysgol Morgannwg.

Ledoux, H., 2008, The Kinetic 3D Voronoi Diagram: A Tool for Simulating Environmental Processes. In: *Advances in 3D Geoinformation Systems*, P. van Oosterom, S. Zlatanova, F. Penninga and E.M. Fendel (Eds), Lecture Notes in Geoinformation and Cartography (Springer), pp. 361–380.

Ledoux, H. and Gold, C.M., 2007, Simultaneous storage of primal and dual three-dimensional subdivisions. *Computers, Environment and Urban Systems*, **31**, 393–408.

Ledoux, H., Gold, C.M. and Baciu, G., 2005, Flipping to robustly delete a vertex in a Delaunay tetrahedralization. In: *Proceedings International Conference on Computational Science and its Applications-ICCSA 2005*, 3480 of *Lecture Notes on Computer Science* (Springer), pp. 737–747.

Lee, D.T. and Lin, A.K., 1986, Generalized Delaunay triangulations for planar graphs. *Discrete and Computational Geometry*, **1**, 201–217.

Liu, A. and Baida, M., 2000, How far flipping can go towards 3D conforming/constrained triangulation. In: *Proceedings of the 9th International Meshing Roundtable* (Sandia National Laboratories), pp. 307–313.

Louwsma, J., Zlatanova, S., Lammeren, R. and van Oosterom, P., 2006, Specifying and Implementing Constraints in GIS - with Examples from a Geo-Virtual Reality System. *GeoInformatica*, **10**, 531–550.

Mäntylä, M., 1988, *An Introduction to Solid Modeling* (Computer Science Press).

MathWorld, 2007, *website:* http://mathworld.wolfram.com/PlanarStraightLineGraph.html.

Meshing Research Corner, 2008, *website:* www.andrew.cmu.edu/user/sowen/mintro.html.

Miller, G.L., Talmor, D., Teng, S.H., Walkington, N. and Wang, H., 1996, Control Volume Meshes using Sphere Packing: Generation, Refinement and Coarsening. In: *5th International Meshing Roundtable* (Sandia National Laboratories), pp. 47–62.

Molenaar, M., 1990a, A Formal Data Structure for Three Dimensional Vector Maps. In: *4th International Symposium on Spatial Data Handling, Zürich*, July (Columbus, OH: International Geographical Union IGU), pp. 830–843.

Molenaar, M., 1990b, A Formal Data Structure for Three Dimensional Vector Maps. In: *Proceedings First European Conference on GIS (EGIS'90), Volume 2*, Amsterdam, pp. 770–781.

Molenaar, M., 1992, A topology for 3D vector maps. *ITC Journal*, **2**, 25–33.

Morton, G.M., 1966, A computer oriented geodetic data base and a new technique in file sequencing. IBM Ltd.

Murray, C., 2007, Oracle Spatial Developer's Guide 11*g* Release 1 (11.1) B28400-02. Oracle corporation.

Nobbe, H., Elberink, S.O., Penninga, F., Verbee, E. and Zuidema, G., 2006, Gebruikerswensen 3D Topografie (in Dutch). 3D Topography consortium. Available at http://www.gdmc.nl/3dtopo.

Okabe, A., Boots, B., Sugihara, K. and Chiu, S.N., 2000, *Spatial tessellations: Concepts and applications of Voronoi diagrams*, Second (John Wiley and Sons).

Oude Elberink, S. and Vosselman, G., 2006, Adding the Third Dimension to a Topographic Database Using Airborne Laser Scanner Data. In: *Photogrammetric Computer Vision 2006. IAPRS, Bonn, Germany*, pp. 92–97.

Paoluzzi, A., Bernardini, F., Cattani, C. and Ferrucci, V., 1993, Dimension-independent modeling with simplicial complexes. *ACM Trans. Graph.*, **12**, 56–102.

Peano, G., 1890, Sur une courbe, qui remplit toute une aire plane. *Mathematische Annalen*, **36**, 157–160.

Penninga, 2008, *website:* http://www.frisopenninga.nl.

Penninga, F., 2005a, 3D Topographic Data Modelling: Why Rigidity Is Preferable to Pragmatism. In: *Spatial Information Theory, Cosit'05*, A.G. Cohn and D.M. Mark (Eds), 3693 of *Lecture Notes on Computer Science* (Springer), pp. 409–425.

Penninga, F., 2005b, Towards 3D Topography using a Feature-based Integrated TIN/TEN Model. In: *AGILE 2005, 8th Conference on Geographic Information Science. Conference Proceedings. Estoril, Portugal, May 26-28*, F. Toppen and M. Painho (Eds), pp. 373–381.

Penninga, F. and van Oosterom, P., 2006a, Editing Features in a TEN-based DBMS approach for 3D Topographic Data Modelling. Delft University of Technology Available at http://www.gdmc.nl/publications/reports/GISt43.pdf.

Penninga, F. and van Oosterom, P., 2006b, Updating Features in a TEN-based DBMS approach for 3D Topographic Data Modelling. In: *Geographic Information Science, Fourth International Conference, GIScience 2006, Münster, Germany, September 2006, Extended Abstracts*, M. Raubal, H.J. Miller, A.U. Frank and M.F. Goodchild (Eds), 28 of *IfGI prints*, pp. 147–152.

Penninga, F. and van Oosterom, P., 2007, A Compact Topological DBMS Data Structure For 3D Topography. In: *Geographic Information Science and Systems in Europe, Agile Conference 2007*, S. Fabrikant and M. Wachowicz (Eds), Lecture Notes in Geoinformation and Cartography (Springer), pp. 455–471.

Penninga, F. and van Oosterom, P., 2008a, A Simplicial Complex-based DBMS Approach To 3D Topographic Data Modelling. *Accepted for publication in the International Journal of Geographical Information Science* Date of acceptance: August 18, 2007. Publication scheduled for 2008.

Penninga, F. and van Oosterom, P., 2008b, First implementation results and open issues on the Poincaré-TEN data structure. In: *Advances in 3D Geoinformation Systems*, P. van Oosterom, S. Zlatanova, F. Penninga and E.M. Fendel (Eds), Lecture Notes in Geoinformation and Cartography (Springer), pp. 177–198.

Penninga, F., van Oosterom, P. and Kazar, B.M., 2006, A TEN-based DBMS approach for 3D Topographic Data Modelling. In: *Progress in Spatial Data Handling, 12th International Symposium on spatial Data Handling*, A. Riedl, W. Kainz and G. Elmes (Eds) (Springer), pp. 581–598.

Pigot, S., 1995, A topological model for a 3-dimensional Spatial Information System. PhD thesis, University of Tasmania, Australia.

Pigot, S., 1992, A Topological Model for a 3D Spatial Information System. In: *Proceedings of the 5th International Symposium on Spatial Data Handling*, pp. 344–360.

Pilouk, M., 1996, Integrated Modelling for 3D GIS. PhD thesis, ITC Enschede, Netherlands.

Poincaré, H., 1895, Analysis Situs. *Journal de l'Ecole Polytechnique*, **1**, 1–123.

Poincaré, H., 1899, Complément á l'Analysis Situs. *Rendiconti del Circolo Matematico di Palermo*, **13**, 285–343.

Pouliot, J., Badard, T., Desgagné, E., Bédard, K. and Thomas, V., 2008, Development of a Web Geological Feature Server (WGFS) for sharing and querying 3D objects. In: *Advances in 3D Geoinformation Systems*, P. van Oosterom, S. Zlatanova, F. Penninga and E.M. Fendel (Eds), Lecture Notes in Geoinformation and Cartography (Springer), pp. 115–130.

Rasmussen, K.B., 1998, On the development of approximate models for outdoor sound propagation. In: *Eighth International Symposium on Long Range Sound Propagation at Penn State.*

Requieha, A. and Voelcker, H., 1980a, Construcvive Solid Geometry. .

Requieha, A. and Voelcker, H., 1980b, Mathematical Foundations of Constructive Solid Geometry: General Topology of Closed Regular Sets. .

Requieha, A. and Voelcker, H., 1983, Solid Modeling: Current Status and Research Directions. *IEEE Computer Graphics and Applications*, **3**.

Rossmann and Bücken, 2008, Using 3D Laser Scanners and Image Recognition for Volume Based Single Tree-Delineation and -Parameterization for 3D-GIS-Applications. In: *Advances in 3D Geoinformation Systems*, P. van Oosterom, S. Zlatanova, F. Penninga and E.M. Fendel (Eds), Lecture Notes in Geoinformation and Cartography (Springer), pp. 131–145.

Ruppert, J., 1995, A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation. *Journal of Algorithms*, **18**, 548–585.

Schönhardt, 1928, Über die Zerlegung von Dreieckspolyedern in Tetraeder. *Mathematische Annalen*, **98**, 309–312.

Shewchuk, 2008, *website:* http://www.cs.berkeley.edu/∼jrs/.

Shewchuk, J.R., 1996, Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In: *Applied Computational Geometry: Towards Geometric Engineering*, M.C. Lin and D. Manocha (Eds), 1148 of *Lecture Notes in Computer Science* (Springer-Verlag), pp. 203–222.

Shewchuk, J.R., 1997a, Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates. *Discrete & Computational Geometry*, **18**, 305–363.

Shewchuk, J.R., 1997b, Delaunay refinement mesh generation. PhD thesis, Carnegie Mellon University.

Shewchuk, J.R., 1999, Lecture Notes on Delaunay Mesh Generation. Department of Electrical Engineering and Computer Science, University of California at Berkeley. Available at http://www.cis.upenn.edu/∼cis610/shewchuk1999.pdf.

Shewchuk, J.R., 2002, Delaunay Refinement Algorithms for Triangular Mesh Generation. *Computational Geometry: Theory and Applications*, **22**, 21–74.

Shewchuk, J.R., 2003, Updating and Constructing Constrained Delaunay and constrained Regular Triangulations by Flips. In: *Proceedings of the 19th Annual Symposium on Computational Geometry, San Diego*, pp. 181–190.

Shewchuk, J.R., 2004, General-Dimensional Constrained Delaunay and Constrained Regular Triangulations I: Combinatorial Properties. *To appear in: Discrete & Computational Geometry* Available at http://www-2.cs.cmu.edu/∼jrs.

Si, H., 2006a, On Refinement of Constrained Delaunay Tetrahedralizations. In: *15th International Meshing Roundtable, Birmingham, USA*.

Si, H., 2006b, TetGen - A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator. Version 1.4 Users Manual. Wias Berlin.

Si, H., 2006c, TetGen, A Quality Tetrahedral Mesh Generator and Three-Dimensional Delaunay Triangulator. User's Manual. Weierstrass Institute for Applied Analysis and Stochastics, Berlin, Germany Available at http://tetgen.berlios.de/files/tetgen-manual.pdf.

Simonse, M., Verbree, E., van Asperen, P. and van der Vegt, J.W., 2000, Construction of the 3DTOP10 - Integration of Countrywide Planimetric Data and Laseraltimetry Data to Support 3D-Visualisation and Analysis. In: *XIXth Congress ISPRS 2000, Amsterdam*, M. Molenaar and K.J. Beek (Eds), XXXIII of *International Archives of Photogrammetry and Remote Sensing*, pp. 995–1002.

Stoter, J., 2004, 3D Cadastre. PhD thesis, Delft University of Technology.

Stoter, J., de Kluijver, H. and Kurakula, V., 2008, Towards 3D environmental impact studies: example of noise. In: *Advances in 3D Geoinformation Systems*, P. van Oosterom, S. Zlatanova, F. Penninga and E.M. Fendel (Eds), Lecture Notes in Geoinformation and Cartography (Springer), pp. 341–360.

Stoter, J., Penninga, F. and van Oosterom, P., 2005, Generalization of integrated terrain elevation and 2D object models. In: *11th International Symposium on Spatial Data Handling: Developments in spatial data handling*, P.F. Fisher (Ed.) (Springer), pp. 527–546.

Tangelder, J., Ermes, P., Vosselman, G. and van den Heuvel, F., 2003, CAD-Based Photogrammetry for Reverse Engineering of Industrial Installation. *IEEE Computer Graphics and Applications*, **18**, 264–274.

TetGen, 2007, *website:* http://tetgen.berlios.de/.

Thompson, R., 2007, Towards a rigorous logic for spatial data representation. PhD thesis, Delft University of Technology.

Thompson, R.J., 2006, 3D Framework for Robust Digital Spatial Models. (Taylor & Francis CRC Press), pp. 177–209.

Thompson, R.J. and van Oosterom, P., 2006, Implementation issues in the storage of spatial data as regular polytopes. In: *UDMS'06 Aalborg, Denmark May 15-17, 2006*, E. Fendel and M. Rumor (Eds), pp. 2.33–2.46.

Tse, R.O. and Gold, C., 2004, TIN Meets CAD - Extending the TIN Concept in GIS. *Future Generation Computer systems (Geocomputation)*, **20**, 1171–11849.

Tse, R., Gold, C. and Kidner, D., 2004, An original way of building a TIN with complex urban structuress. In: *Proceedings of ISPRS 2004 - XXth Congress, Istanbul, Turkey.*

van der Most, A., 2004, "An algorithm for overlaying 3D features using a tetrahedral network", Master's thesis, Delft University of Technology.

van Essen, R., 2008, Maps Get Real: Digital Maps evolving from mathematical line graphs to virtual reality models. In: *Advances in 3D Geoinformation Systems*, P. van Oosterom, S. Zlatanova, F. Penninga and E.M. Fendel (Eds), Lecture Notes in Geoinformation and Cartography (Springer), pp. 3–18.

van Oosterom, P., 2005a, Space for geo-information - 3D Topography - project proposal fase 1 RGI - 011. Delft University of Technology. Available at http://www.rgi-otb.nl/3dtopo.

van Oosterom, P., 2005b, Space for geo-information - 3D Topography - project proposal fase 2 RGI - 011. Delft University of Technology. Available at http://www.rgi-otb.nl/3dtopo.

van Oosterom, P., Vertegaal, W., van Hekken, M. and Vijlbrief, T., 1994, Integrated 3D Modelling within a GIS. In: *International GIS workshop AGDM'94*, pp. 80–95.

van Oosterom, P. and Vijlbrief, T., 1996, The Spatial Location Code. In: *Advances in GIS research II; proceedings of the seventh International Symposium on Spatial Data Handling - SDH'96*, M.J. Kraak and M. Molenaar (Eds) (Taylor and Francis).

Verbree, E., 2002, Driedimensionale Topografische Terreinmodellering op basis van Tetrader Netwerken: Top10-3D (in Dutch). Delft University of Technology. Available at http://www.gdmc.nl/publications/reports/GISt16.pdf.

Verbree, E. and van Oosterom, P., 2003a, Better surface representations by Delaunay Tetrahedronized Irregular Networks. In: *ESRI International User Conference, San Diego.*

Verbree, E. and van Oosterom, P., 2003b, STIN Method: Surface TIN Representation by Delaunay TENs constrained by observation lines. In: *6th AGILE Conference, Lyon, April 24-26 2003*, pp. 409–419.

Verbree, E., 2006, Piecewise Linear Complex Representations through Conforming Delaunay Tetrahedronization. In: *Geographic Information Science, Fourth International Conference, GIScience 2006, Münster, Germany, September 2006, Extended Abstracts*, M. Raubal, H.J. Miller, A.U. Frank and M.F. Goodchild (Eds), 28 of *IfGI prints*, pp. 385–387.

Verbree, E., van der Most, A., Quak, W. and van Oosterom, P., 2005, Towards a 3D Feature Overlay through a Tetrahedral Mesh Data Structure. *Cartography and Geographic Information Science*, **32**, 303–314.

Verbree, E., Zlatanova, S. and Smit, K., 2004, Interactive navigation services through value-added CycloMedia panoramic images. In: *ICEC '04: 6th international conference on Electronic commerce* (ACM Press), pp. 591–595.

Watson, D.F., 1981, Computing the n-dimensional Delaunay Tesselation with Application to Voronoi Polytopes. *Computer Journal*, **24**, 167–172.

Webviewer Wesselingh, 2007, *website:* http://www.3dwebgis.nl.

Wei, G., Ping, Z. and Jun, C., 1998, Topological Data Modelling for 3D GIS. In: *ISPRS Commission IV Symposium on GIS Between Visions and Applications*, D. Fritsch, M. Englich and M. Sester (Eds), 32 of *The International Archives of Photogrammetry and Remote Sensing*, pp. 657–661.

Wesselingh, S., 2007, "Visualization of a TEN (Tetrahedral Irregular Network) in a web client", Geographical Information Management and Applications (GIMA), Utrecht University.

Winther, M., Kousgaarda, U. and Oxbølb, A., 2006, Calculation of odour emissions from aircraft engines at Copenhagen Airport. *Science of the Total Environment*, **366**, 218–232.

Worboys, M. and Duckham, M., 2004, *GIS: A Computing Perspective*, Second Edition (CRC Press).

Wordnet, 2007, *website:* http://wordnet.princeton.edu.

Zlatanova, S., Abdul Rahman, A. and Shi, W., 2004, Topological models and frameworks for 3D spatial objects. *Computers & Geosciences*, **30**, 419–428.

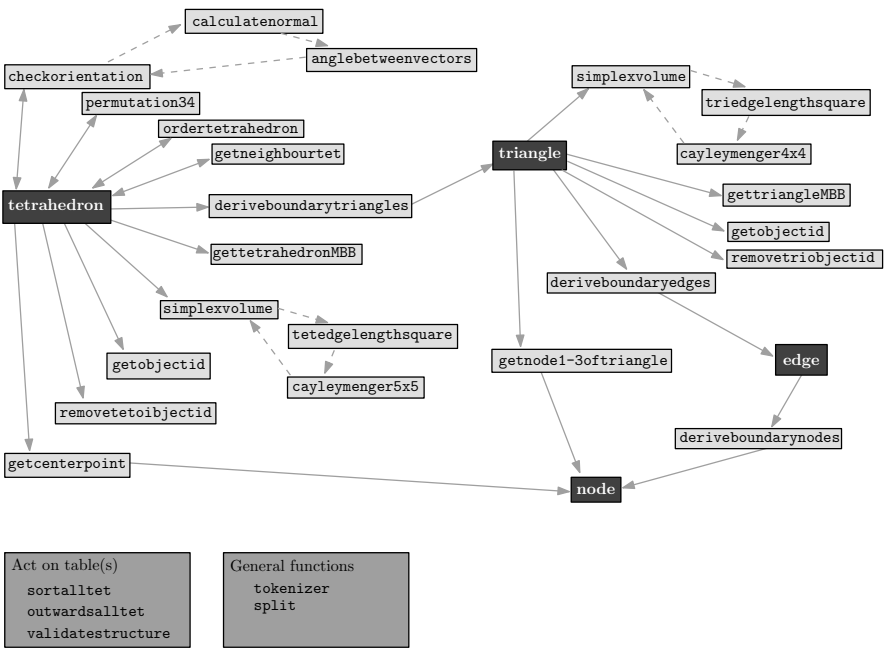
Zlatanova, S., 2000, 3D GIS for urban development. PhD thesis, Graz University of Technology.

Zlatanova, S., Abdul Rahman, A. and Pilouk, M., 2002, 3D GIS: Current Status and Perspectives. In: *Proceedings of Joint Conference on Geo-Spatial Theory, Processing and Applications, Ottawa, Canada*.

Appendix I
Implementation:
Functions and procedures

This appendix contains most Oracle PL/SQL functions and procedures that are used in the proof-of-concept implementation of the simplicial complex-based DBMS data structure, as described in chapter 5. Note that some procedures are simplified to improve readability and that the declaration section might be reformatted to reduce the length of this appendix. The full code is available online through the 3D Topography (2006) and Penninga (2008).

The functions and procedures in this appendix are used in the coordinate concatenation implementation. Slightly altered versions were used in the identifier concatenation implementation. The figure below illustrates the relationships between the functions and procedures.



```
-- The narray type will be used frequently in the following functions and procedures
CREATE OR REPLACE TYPE narray IS VARRAY(13) OF NVARCHAR2(1000);
/

/*
================================================================================
Procedure name : tokenizer
Description : procedure used in procedure Split to derive tokens (in this case
              individual coordinate values from the simplex codes)
================================================================================
*/

CREATE OR REPLACE PROCEDURE tokenizer(iStart IN NUMBER, sPattern IN NVARCHAR2,
                     sBuffer IN NVARCHAR2, sResult OUT NVARCHAR2, iNextPos OUT NUMBER)
AS
   nPos1 NUMBER;
   nPos2 NUMBER;
BEGIN
   nPos1 := Instr (sBuffer ,sPattern ,iStart);
   IF nPos1 = 0 THEN
      sResult := NULL ;
   ELSE
      nPos2 := Instr (sBuffer ,sPattern ,nPos1 + 1);
      IF nPos2 = 0 THEN
         sResult := Rtrim(Ltrim(Substr(sBuffer ,nPos1+1)));
         iNextPos := nPos2;
      ELSE
         sResult := Substr(sBuffer ,nPos1 + 1 , nPos2 - nPos1 - 1);
         iNextPos := nPos2;
      END IF;
   END IF;
END;
/


/*
================================================================================
Procedure name : Split
Description : procedure used to split the simplex code (nvarchar2) in separatate
              coordinate values (v_array)
================================================================================
*/

CREATE OR REPLACE PROCEDURE Split(sbuf IN NVARCHAR2, sepr IN NVARCHAR2,
                                  node_array OUT NARRAY)
AS
   sres NVARCHAR2(500);
   sbuf2 NVARCHAR2(500);
   pos NUMBER;
   istart NUMBER;
   i NUMBER;
BEGIN
   sbuf2 := 'x'||sbuf;
   node_array := narray('0','0','0','0','0','0','0','0','0','0','0','0','0');
   istart := 1;
```

```
   i := 0;
   tokenizer(istart,sepr,sbuf2,sres,pos);
   IF (pos <> 0) THEN
      i := node_array.first;
      node_array(i) := sres;
   END IF;
   WHILE (pos <> 0)
   LOOP
      istart := pos;
      i := i+1;
      tokenizer (istart,sepr,sbuf2,sres,pos);
      node_array(i) := sres;
   END LOOP;
END;
/

/*
================================================================================
Function name : cayleymengerdeterminant5x5 / 4x4
Description : wrapper for an existing Java function to calculate the determinant of a
              5x5/4x4 matrix. The Cayley-Menger determinant calculates the volume of a
              tetrahedron or a triangle, based on the lengths of their edges (and is
              thus independent of the dimension of space)
================================================================================
*/

CREATE OR REPLACE FUNCTION cayleymengerdeterminant5x5(input1 NUMBER, input2 NUMBER,
                          input3 NUMBER, input4 NUMBER, input5 NUMBER, input6 NUMBER)
   RETURN NUMBER
   AS LANGUAGE JAVA
      NAME 'Friso.cayleymengerdeterminant5x5(double, double, double,
                                             double, double, double) return double';
/

CREATE OR REPLACE FUNCTION cayleymengerdeterminant4x4(input1 NUMBER, input2 NUMBER,
                                                      input3 NUMBER)
   RETURN NUMBER
   AS LANGUAGE JAVA
      NAME 'Friso.cayleymengerdeterminant4x4(double, double, double) return double';
/


/*
================================================================================
Procedure name : filltettable
Description : uses temporary tables (in which TetGen results (nodes and tetrahedrons)
              were loaded) to create the tetrahedron table of the simplicial complex-
              based data structure. It uses coordinate concatenation. Separate
              functions will later check orientation and ordering of these codes
================================================================================
*/

CREATE OR REPLACE PROCEDURE filltettable(empty IN NUMBER)
IS
   CURSOR tetcur IS
      SELECT tid FROM temp;
```

```
   temptid INTEGER;
   tetcode NVARCHAR2(500);
   tempn1, tempn2, tempn3, tempn4, tempoid INTEGER;
   x1, y1, z1, x2, y2, z2, x3, y3, z3, x4, y4, z4 NUMBER;
   t1, t2, t3, t4, toid NVARCHAR2(500);
BEGIN
   OPEN tetcur;
   LOOP
      FETCH tetcur INTO temptid;
      EXIT WHEN tetcur%notfound;
      SELECT n1,n2,n3,n4,oid INTO tempn1,tempn2,tempn3,tempn4,tempoid FROM temp
        WHERE (tid=temptid);
      SELECT xcoord,ycoord,zcoord INTO x1,y1,z1 FROM tempnode WHERE (nid=tempn1);
      SELECT xcoord,ycoord,zcoord INTO x2,y2,z2 FROM tempnode WHERE (nid=tempn2);
      SELECT xcoord,ycoord,zcoord INTO x3,y3,z3 FROM tempnode WHERE (nid=tempn3);
      SELECT xcoord,ycoord,zcoord INTO x4,y4,z4 FROM tempnode WHERE (nid=tempn4);
      INSERT INTO tetrahedron VALUES (
          to_char(x1)||'x'||to_char(y1)||'x'||to_char(z1)||'x'||to_char(x2)||'x'||
          to_char(y2)||'x'||to_char(z2)||'x'||to_char(x3)||'x'||to_char(y3)||'x'||
          to_char(z3)||'x'||to_char(x4)||'x'||to_char(y4)||'x'||to_char(z4)||'x'||
          to_char(tempoid));
   END LOOP;
   CLOSE tetcur;
END;
/

/*
================================================================================
Function name : getnode 1-3 oftriangle
Description : returns respectively the 1st, 2nd or 3rd node of a triangle
================================================================================
*/

CREATE OR REPLACE FUNCTION getnode1oftriangle(tricode IN NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   node NVARCHAR2(500);
   node_array narray := narray();
BEGIN
   Split(tricode,'x',node_array);
   node := node_array(2)||'x'||node_array(3)||'x'||node_array(4);
   RETURN node;
END;
/

-- Similar to getnode1oftriangle, functions getnode2oftriangle and getnode3oftriangle
-- are created


/*
================================================================================
Procedure name : getcenterpoint
Description : calculates centrepoint of tetrahedron (centrepoint is later used in
              orientation determinination)
================================================================================
*/
```

```
CREATE OR REPLACE PROCEDURE getcenterpoint(tetcode IN NVARCHAR2, x OUT NUMBER,
                                           y OUT NUMBER, z OUT NUMBER)
IS
   node_array narray := narray();
BEGIN
   Split(tetcode,'x',node_array);
   x := (node_array(1)+node_array(4)+node_array(7)+node_array(10))/4;
   y := (node_array(2)+node_array(5)+node_array(8)+node_array(11))/4;
   z := (node_array(3)+node_array(6)+node_array(9)+node_array(12))/4;
END;
/


/*
================================================================================
Procedure name : calculatenormal
Description : calculates normal vector of signed triangle
================================================================================
*/

CREATE OR REPLACE PROCEDURE calculatenormal(tricode IN NVARCHAR2, vx OUT NUMBER,
                                            vy OUT NUMBER, vz OUT NUMBER)
IS
   node_array narray := narray();
   sign NVARCHAR2(500);
BEGIN
   Split(tricode,'x',node_array);
   sign := node_array(1);
   vx := (node_array(6)-node_array(3))*(node_array(10)-node_array(4))-
         (node_array(7)-node_array(4))*(node_array(9)-node_array(3));
   vy := (node_array(7)-node_array(4))*(node_array(8)-node_array(2))-
         (node_array(5)-node_array(2))*(node_array(10)-node_array(4));
   vz := (node_array(5)-node_array(2))*(node_array(9)-node_array(3))-
         (node_array(4)-node_array(3))*(node_array(8)-node_array(2));
   IF (vx<0) THEN
      IF (sign='-') THEN
         vx := -1*vx;
      END IF;
   ELSE
      IF (sign='-') THEN
         vx := -1*vx;
      END IF;
   END IF;
   IF (vy<0) THEN
      IF (sign='-') THEN
         vy := -1*vy;
      END IF;
   ELSE
      IF (sign='-') THEN
         vy := -1*vy;
      END IF;
   END IF;
   IF (vz<0) THEN
      IF (sign='-') THEN
         vz := -1*vz;
```

```
      END IF;
   ELSE
      IF (sign='-') THEN
         vz := -1*vz;
      END IF;
   END IF;
END;
/


/*
================================================================================
Procedure name : anglebetweenvectors
Description : calculates the angle between two 3D vectors. The resulting angle is in
              radians! Used in orientation determination
================================================================================
*/

CREATE OR REPLACE PROCEDURE anglebetweenvectors(normx IN NUMBER, normy IN NUMBER,
   normz IN NUMBER, vecx IN NUMBER, vecy IN NUMBER, vecz IN NUMBER, angle OUT NUMBER)
IS
   dotproduct NUMBER;
   length1 NUMBER;
   length2 NUMBER;
BEGIN
   dotproduct := normx*vecx + normy*vecy + normz*vecz;
   length1 := power(normx,2) + power(normy,2) + power(normz,2);
   length2 := power(vecx,2) + power(vecy,2) + power(vecz,2);
   angle := ACOS(dotproduct/((SQRT(length1))*(SQRT(length2))));
END;
/


/*
================================================================================
Procedure name : checkorientation
Description : supposes poincare tetrahedron, so orientation of all four boundary
      triangles is identical (either all inwards or all outwards). From tetrahedron
      <v0,v1,v2,v3> it takes boundary <v1,v2,v3> (orientation is positive). The normal
      vector of this triangle is calculated. A second vector is constructed, from <v0>
      to <v1>. The angle between these two vectors is calculated. if it is smaller
      than 90 degrees (0.5*PI), the orientation is inwards, otherwise outwards.
================================================================================
*/

CREATE OR REPLACE PROCEDURE checkorientation(tetcode IN NVARCHAR2,
                                             isoutwards IN OUT NUMBER)
IS
   node_array narray := narray();
   angle NUMBER;
   tricode NVARCHAR2(500);
   difx, dify, difz  NUMBER;
   normx, normy, norm  NUMBER;
BEGIN
   Split(tetcode,'x',node_array);
   difx := node_array(1) - node_array(4);
```

```
   dify := node_array(2) - node_array(5);
   difz := node_array(3) - node_array(6);
   tricode := '+x'||node_array(4)||'x'||node_array(5)||'x'||node_array(6)||'x'||
              node_array(7)||'x'||node_array(8)||'x'||node_array(9)||'x'||
              node_array(10)||'x'||node_array(11)||'x'||node_array(12);
   calculatenormal(tricode,normx,normy,normz);
   anglebetweenvectors(normx,normy,normz,difx,dify,difz,angle);
   IF (angle>(3.14159265358979323846264338327795/2)) THEN
      isoutwards := 1;
   ELSE
      isoutwards := 0;
   END IF;
END;
/


/*
================================================================================
Procedure name : permutation34
Description : Performs a permutation of the last two vertices in the tetrahedron code.
      This single permutation causes a chance of orientation (i.e. inwards instead of
      outwards or vice versa) of the tetrahedron. For simplexes of dimension < 3,
      usage of signed and ordered encodings is preferred above permutations
================================================================================
*/

CREATE OR REPLACE PROCEDURE permutation34(tetcode IN NVARCHAR2,
                                          tetcodeperm OUT NVARCHAR2)
IS
   node_array narray := narray();
BEGIN
   Split(tetcode,'x',node_array);
   tetcodeperm := node_array(1)||'x'||node_array(2)||'x'||node_array(3)||'x'||
                  node_array(4)||'x'||node_array(5)||'x'||node_array(6)||'x'||
                  node_array(10)||'x'||node_array(11)||'x'||node_array(12)||'x'||
                  node_array(7)||'x'||node_array(8)||'x'||node_array(9)||'x'||
                  node_array(13);
END;
/

/*
================================================================================
Procedure name : ordertriangle
Description : Orders a triangle, based on the coordinate code of each vertex, from
      small to large, in order to ensure that each triangle and its opposite have the
      same code (apart from the sign) and not one of their equivalent permutations
================================================================================
*/

CREATE OR REPLACE PROCEDURE ordertriangle(tricode IN NVARCHAR2,
                                          orderedtricode OUT NVARCHAR2)
IS
   node_array narray := narray();
   sign NVARCHAR2(500);
   test1, test2, test3 INTEGER;
BEGIN
```

```
   Split(tricode,'x',node_array);
   IF (node_array(2) < node_array(5)) THEN
      test1 := 1;
   ELSE
      IF (node_array(2) = node_array(5)) THEN
         IF (node_array(3) < node_array(6)) THEN
            test1 := 1;
         ELSE
           (...)

-- The rest of the code is omitted since it is is long and full of repetitions. It
-- determines the current order and rewrites triangle < a, b, c> such that (a < b < c)
-- holds. For two nodes a and b, a<b holds if x_{a} < x_{b}. In case of equal
-- x-coordinates the test is performed on y- or z-coordinates.


/*
================================================================================
Procedure name : ordertetrahedron
Description : Orders a tetrahedron based on the coordinate code of each vertex, from
      small to large, in order to ensure that each triangle and its opposite have the
      same code (apart from the sign) and not one of thheir equivalent permutations
================================================================================
*/

-- This code is omitted, since it is is long and full of repetitions. It acts similar
-- to the ordertriangle procedure, desecribed above


/*
================================================================================
Procedure name : deriveboundarytriangles
Description : Derives the four boundary triangles of a tetrahedron by applying the
      boundary operator of the simplical complex homology. All resulting triangles
      are ordered (v0<v1<v2) and signed(+/-)
================================================================================
*/

CREATE OR REPLACE PROCEDURE deriveboundarytriangles(tetcode IN NVARCHAR2,
                                      tricode1 OUT NVARCHAR2, tricode2 OUT NVARCHAR2,
                                      tricode3 OUT NVARCHAR2, tricode4 OUT NVARCHAR2)
IS
   node_array narray := narray();
BEGIN
   Split(tetcode,'x',node_array);
   ordertriangle('+'||'x'||node_array(4)||'x'||node_array(5)||'x'||node_array(6)||
     'x'|| node_array(7)||'x'||node_array(8)||'x'||node_array(9)||'x'||node_array(10)
     ||'x'||node_array(11)||'x'||node_array(12)||'x'||node_array(13), tricode1);
   ordertriangle('-'||'x'||node_array(1)||'x'||node_array(2)||'x'||node_array(3)||
     'x'||node_array(7)||'x'||node_array(8)||'x'||node_array(9)||'x'||node_array(10)
     ||'x'||node_array(11)||'x'||node_array(12)||'x'||node_array(13), tricode2);
   ordertriangle('+'||'x'||node_array(1)||'x'||node_array(2)||'x'||node_array(3)||
     'x'||node_array(4)||'x'||node_array(5)||'x'||node_array(6)||'x'||node_array(10)
     ||'x'||node_array(11)||'x'||node_array(12)||'x'||node_array(13), tricode3);
   ordertriangle('-'||'x'||node_array(1)||'x'||node_array(2)||'x'||node_array(3)||
     'x'||node_array(4)||'x'||node_array(5)||'x'||node_array(6)||'x'||node_array(7)||
```

```
     'x'||node_array(8)||'x'||node_array(9)||'x'||node_array(13), tricode4);
END;
/

-- Functions deriveboundarytriangle1, deriveboundarytriangle2, deriveboundarytriangle3
-- and deriveboundarytriangle4, result in 1 of the 4 boundary triangles, as described
-- in procedure deriveboundarytriangles



/*
================================================================================
Procedure name : deriveboundaryedges
Description : Derives the three boundary edges of a signed triangle by applying the
      boundary operator of the simplical complex homology. All resulting edges are
      ordered (v0<v1, as the triangles are ordered) and signed(+/-)
================================================================================
*/

CREATE OR REPLACE PROCEDURE deriveboundaryedges(tricode IN NVARCHAR2,
                  edcode1 OUT NVARCHAR2, edcode2 OUT NVARCHAR2, edcode3 OUT NVARCHAR2)
IS
   sign NVARCHAR2(500);
   node_array narray := narray();
BEGIN
   Split(tricode,'x',node_array);
   sign := node_array(0);
   IF (sign='+') THEN
      edcode1 := '+'||'x'||node_array(5)||'x'||node_array(6)||'x'||node_array(7)
                 ||'x'||node_array(8)||'x'||node_array(9)||'x'||node_array(10);
      edcode2 := '-'||'x'||node_array(2)||'x'||node_array(3)||'x'||node_array(4)
                 ||'x'||node_array(8)||'x'||node_array(9)||'x'||node_array(10);
      edcode3 := '+'||'x'||node_array(2)||'x'||node_array(3)||'x'||node_array(4)
                 ||'x'||node_array(5)||'x'||node_array(6)||'x'||node_array(7);
   ELSE
      edcode1 := '-'||'x'||node_array(5)||'x'||node_array(6)||'x'||node_array(7)
                 ||'x'||node_array(8)||'x'||node_array(9)||'x'||node_array(10);
      edcode2 := '+'||'x'||node_array(2)||'x'||node_array(3)||'x'||node_array(4)
                 ||'x'||node_array(8)||'x'||node_array(9)||'x'||node_array(10);
      edcode3 := '-'||'x'||node_array(2)||'x'||node_array(3)||'x'||node_array(4)
                 ||'x'||node_array(5)||'x'||node_array(6)||'x'||node_array(7);
   END IF;
END;
/

-- The functions deriveabsboundaryedge1, deriveabsboundaryedge2 and
-- deriveabsboundaryedge3 each result in 1 of the 3 edge codes, as described in
-- procedure deriveboundaryedgese.


/*
================================================================================
Procedure name : deriveboundarynodes
Description : Derives the two boundary nodes of a signed (or not) edge by applying the
              boundary operator of the simplical complex homology.
================================================================================
```

```
*/

CREATE OR REPLACE PROCEDURE deriveboundarynodes(edcode IN NVARCHAR2,
                                    odecode1 OUT NVARCHAR2, nodecode2 OUT NVARCHAR2)
IS
   node_array narray := narray();
BEGIN
   Split(edcode,'x',node_array);

   IF (node_array(1)='+') OR (node_array(1)='-') THEN
      nodecode1 := node_array(2)||'x'||node_array(3)||'x'||node_array(4);
      nodecode2 := node_array(5)||'x'||node_array(6)||'x'||node_array(7);
   ELSE
      nodecode1 := node_array(1)||'x'||node_array(2)||'x'||node_array(3);
      nodecode2 := node_array(4)||'x'||node_array(5)||'x'||node_array(6);
   END IF;
END;
/

-- The functions deriveboundarynode1 and deriveboundarynode2 result in 1 of the 2
-- node codes, as described in procedure deriveboundarynodes above.



/*
================================================================================
Procedure name : gettetrahedronmbb
Description : Derives the minimum bounding box of a tetrahedron by selecting the min
              x, y, z and max x, y, z.
================================================================================
*/

CREATE OR REPLACE PROCEDURE gettetrahedronmbb(tetcode IN NVARCHAR2, minx OUT NUMBER,
                                  miny OUT NUMBER, minz OUT NUMBER, maxx OUT NUMBER,
                                  maxy OUT NUMBER, maxz OUT NUMBER)
IS
   node_array narray := narray();
BEGIN
   Split(tetcode,'x',node_array);
   minx := node_array(1);
   maxx := node_array(1);
   miny := node_array(2);
   maxy := node_array(2);
   minz := node_array(3);
   maxz := node_array(3);

   FOR i IN 1..3
   LOOP
      IF (node_array(3*i+1)<minx) THEN
         minx := node_array(3*i+1);
      END IF;
      IF (node_array(3*i+1)>maxx) THEN
         maxx := node_array(3*i+1);
      END IF;
      IF (node_array(3*i+2)<miny) THEN
```

```
         miny := node_array(3*i+2);
      END IF;
      IF (node_array(3*i+2)>maxy) THEN
         maxy := node_array(3*i+2);
      END IF;
      IF (node_array(3*i+3)<minz) THEN
         minz := node_array(3*i+3);
      END IF;
      IF (node_array(3*i+3)>maxz) THEN
         maxz := node_array(3*i+3);
      END IF;
   END LOOP;
END;
/


/*
================================================================================
Procedure name : gettrianglembb
Description : Derives the minimum bounding box of a tetrahedron by selecting the min
              x, y, z and max x, y, z.
================================================================================
*/

-- Function similar to procedure gettetrahedronmbb, as described above


/*
================================================================================
Procedure name : tetedgelengthsquare
Description : Calculates the square of the lengths of the six edges of a tetrahedron.
              Used as input for Cayley-Menger determinant
================================================================================
*/

CREATE OR REPLACE PROCEDURE tetedgelengthsquare(tetcode IN NVARCHAR2, a OUT NUMBER,
                 b OUT NUMBER, c OUT NUMBER, d OUT NUMBER, e OUT NUMBER, f OUT NUMBER)
IS
  node_array narray := narray();
BEGIN
  Split(tetcode,'x',node_array);
  a := power((node_array(4)-node_array(1)),2)+power((node_array(5)-node_array(2)),2)
       +power((node_array(6)-node_array(3)),2);
  b := power((node_array(7)-node_array(1)),2)+power((node_array(8)-node_array(2)),2)
       +power((node_array(9)-node_array(3)),2);
  c := power((node_array(10)-node_array(1)),2)+power((node_array(11)-node_array(2)),2)
       +power((node_array(12)-node_array(3)),2);
  d := power((node_array(7)-node_array(4)),2)+power((node_array(8)-node_array(5)),2)
       +power((node_array(9)-node_array(6)),2);
  e := power((node_array(10)-node_array(4)),2)+power((node_array(11)-node_array(5)),2)
       +power((node_array(12)-node_array(6)),2);
  f := power((node_array(10)-node_array(7)),2)+power((node_array(11)-node_array(8)),2)
       +power((node_array(12)-node_array(9)),2);
END;
/
```

```
/*
================================================================================================
Procedure name : triedgelengthsquare
Description : Calculates the square of the lengths of the three edges of a triangle.
              Used as input for Cayley-Menger determinant
================================================================================================
*/

-- Function similar to procedure tetedgelengthsquare, as described above


/*
================================================================================================
Procedure name : simplexvolume
Description : Calculates the volume of a 3- or 2-simplex, using the Cayley-Menger
              determinant
================================================================================================
*/

CREATE OR REPLACE PROCEDURE simplexvolume(simplexcode IN NVARCHAR2,
                                          simplexvolume OUT NUMBER)
IS
   a, b, c, d, e, f, det NUMBER;
BEGIN
   IF (((SUBSTR(simplexcode,1,1)) = '+') OR ((SUBSTR(simplexcode,1,1)) = '-')) THEN
      triedgelengthsquare(simplexcode,a,b,c);
      det := cayleymengerdeterminant4x4(a,b,c);
      simplexvolume := SQRT(det/-16);
   ELSE
      tetedgelengthsquare(simplexcode,a,b,c,d,e,f);
      det := cayleymengerdeterminant5x5(a,b,c,d,e,f);
      simplexvolume := SQRT(det/288);
   END IF;
END;
/


/*
================================================================================================
Procedure name : outwardsalltet
Description : ckecks for all tetrahedrons in the tetrahedron table whether they are
              oriented outwards. if not, the tetrahedrons are replaced by an outwards
              oriented permutation (permutation of v2 and v3)
================================================================================================
*/

CREATE OR REPLACE PROCEDURE outwardsalltet(changes OUT NUMBER, nochanges OUT NUMBER)
IS
   CURSOR tetcur IS
      SELECT tetcode FROM tetrahedron FOR UPDATE;
   tetcode tetrahedron.tetcode%type;
   currenttetcode NVARCHAR2(500);
   newtetcode NVARCHAR2(500);
   bool NUMBER;
   a NUMBER;
```

```
BEGIN
   a := 0;
   changes := 0;
   nochanges := 0;
   OPEN tetcur;
   LOOP
      FETCH tetcur INTO tetcode;
      EXIT WHEN tetcur%notfound;
      checkorientation(tetcode,bool);
      a:= a+1;
      IF (bool = 0) THEN
         permutation34(tetcode,newtetcode);
         UPDATE tetrahedron SET tetcode=newtetcode WHERE CURRENT OF tetcur;
         changes := changes+1;
      ELSE
         nochanges := nochanges+1;
      END IF;
   END LOOP;
   CLOSE tetcur;
END;
/



/*
================================================================================
Procedure name : sortalltet
Description : ckecks for all tetrahedrons in the tetrahedron table whether they are
              properly ordered: a<b<c<d. if not, the ordering is altered
================================================================================
*/

CREATE OR REPLACE PROCEDURE sortalltet(changes OUT NUMBER, nochanges OUT NUMBER)
IS
   CURSOR tetcur1 IS
      SELECT tetcode FROM tetrahedron FOR UPDATE;
   tetcode tetrahedron.tetcode%type;
   newtetcode NVARCHAR2(500);
   bool NUMBER;
   a NUMBER;
BEGIN
   a := 0;
   changes := 0;
   nochanges := 0;
   OPEN tetcur1;
   LOOP
      FETCH tetcur1 INTO tetcode;
      EXIT WHEN tetcur1%notfound;
      ordertetrahedron(tetcode,newtetcode);
      UPDATE tetrahedron SET tetcode=newtetcode WHERE CURRENT OF tetcur1;
          changes := changes+1;
   END LOOP;
   CLOSE tetcur1;

END;
```

```
/


/*
================================================================================
Function name : get tet/tri objectid
Description : Returns object ID, to be used in SQL select statements, for instance for
              finding all tetrahedrons or triangles of a specific object.
================================================================================
*/

CREATE OR REPLACE FUNCTION gettetobjectid(simplexcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   node_array narray := narray();
BEGIN
   Split(simplexcode,'x',node_array);
   RETURN node_array(13);
END;
/

CREATE OR REPLACE FUNCTION gettriobjectid(simplexcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   node_array narray := narray();
BEGIN
   Split(simplexcode,'x',node_array);
   RETURN node_array(11);
END;
/


/*
================================================================================
Function name : remove tet/tri objectid
Description : Returns simplexcode without object ID (so only geometrical part)
================================================================================
*/

CREATE OR REPLACE FUNCTION removetetobjectid(simplexcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   node_array narray := narray();
BEGIN
   Split(simplexcode,'x',node_array);
   RETURN node_array(1)||'x'||node_array(2)||'x'||node_array(3)||'x'||
          node_array(4)||'x'||node_array(5)||'x'||node_array(6)||'x'||
          node_array(7)||'x'||node_array(8)||'x'||node_array(9)||'x'||
          node_array(10)||'x'||node_array(11)||'x'||node_array(12);
END;
/

CREATE OR REPLACE FUNCTION removetriobjectid(simplexcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   node_array narray := narray();
```

```
BEGIN
   Split(simplexcode,'x',node_array);
   RETURN node_array(1)||'x'||node_array(2)||'x'||node_array(3)||'x'||
          node_array(4)||'x'||node_array(5)||'x'||node_array(6)||'x'||
          node_array(7)||'x'||node_array(8)||'x'||node_array(9)||'x'||
          node_array(10);
END;
/


/*
================================================================================
Procedure name : validatestructure
Description : Performs several checks:
  check1: Euler count (num of nodes - num of edges + num of triangles - (num of
          tetrahedrons + external volume)
  check2: number of triangles equals number of unique triangles
  check3: number of triangles equals four times number of tetrahedrons
================================================================================
*/

CREATE OR REPLACE PROCEDURE validatestructure(result OUT NVARCHAR2)
IS
   numnode, numedge, numtri, numtri1, numtri2, numtet, check1, check2, check3 NUMBER;
BEGIN
   SELECT COUNT(*) INTO numnode FROM node;
   SELECT COUNT(*) INTO numedge FROM edge;
   SELECT COUNT(DISTINCT ABS(removetriobjectid(tricode))) INTO numtri FROM triangle;
   SELECT COUNT(*) INTO numtet FROM tetrahedron;
   check1 := numnode - numedge + numtri - (numtet+1);

   SELECT COUNT(*) INTO numtri1 FROM triangle;
   SELECT COUNT(DISTINCT tricode) INTO numtri2 FROM triangle;
   check2 := numtri1 - numtri2;

   check3 := numtri1 - 4*numtet;

   IF (check1+check2+check3 = 0) THEN
     result := 'Validation result: OK';
   ELSE
     IF (check1 <> 0) THEN
        result := 'Euler condition not satisfied';
     END IF;
     IF (check2 <> 0) THEN
        result := result||'-'||'Triangles not unique';
     END IF;
     IF (check3 <> 0) THEN
        result := result||'-'||'Error in deriving boundary triangles';
     END IF;
  END IF;

END;
/


/*
```

```
================================================================================
Function name : getneighnourtet1-4
Description : Returns neighbouring tetrahderon by using the triangle and
oppositetriangle view
================================================================================
*/

CREATE OR REPLACE FUNCTION getneighbourtet1(tetcode NVARCHAR2)
RETURN NVARCHAR2 DETERMINISTIC
IS
   neighbourtet NVARCHAR2(100);
BEGIN
   SELECT fromtetcode INTO neighbourtet FROM triangle
     WHERE tricode=(SELECt dt.oppositetricode FROM oppositetriangle dt
     WHERE dt.tricode=deriveboundarytriangle1(tetcode));
  RETURN neighbourtet;
END;
/

-- Functions getneighnourtet2-4 are similar, but use deriveboundarytriangle2-4
```

Appendix II
Implementation:
Creating the data structure

This appendix contains the procedures used for the creation of the simplicial complex-based data structure, as described in chapter 5. The procedures used in this appendix are from the coordinate concatenation implementation and differ slightly from the identifier concatenation implementation.

```
================================================================================

                    Initial table creation and data loading

================================================================================

-- Create and fill temporary tables with tetrahedronisation results from TetGen

CREATE TABLE tempnode(nid INTEGER, xcoord NUMBER, ycoord NUMBER, zcoord NUMBER);

LOAD DATA
INFILE 'data/rotterdam_b.node'
APPEND
INTO TABLE tempnode
fields terminated by '\t'
  (nid integer external,
   xcoord float external,
   ycoord float external,
   zcoord float external)

CREATE TABLE temp(tid INTEGER, n1 INTEGER, n2 INTEGER, n3 INTEGER, n4 INTEGER,
                  oid INTEGER);

LOAD DATA
INFILE 'data/rotterdam_b.ele'
APPEND
INTO TABLE temp
fields terminated by '\t'
   (tid,n1,n2,n3,n4,oid)

-- Create empty table with tetrahedrons

CREATE TABLE tetrahedron(tetcode NVARCHAR2(300));
```

```
================================================================================

                  Loading tetrahedron table and deriving views

================================================================================

-- Load data into tetrahedron table (tetrahedron codes are created)

EXEC filltettable(10);


-- Rewrite tetrahedron table to achieve correct ordering and outwards orientation

VAR a NUMBER;
VAR b NUMBER;
EXEC sortalltet(:a,:b);
COMMIT;

VAR c NUMBER;
VAR d NUMBER;
EXEC tettableoutwards(:c,:d);
COMMIT;


-- Creating function based indexes to support view triangle

CREATE INDEX deriveboundarytriangle1_idx ON
             tetrahedron(deriveboundarytriangle1(tetcode));
CREATE INDEX deriveboundarytriangle2_idx ON
             tetrahedron(deriveboundarytriangle2(tetcode));
CREATE INDEX deriveboundarytriangle3_idx ON
             tetrahedron(deriveboundarytriangle3(tetcode));
CREATE INDEX deriveboundarytriangle4_idx ON
             tetrahedron(deriveboundarytriangle4(tetcode));

-- Creating view triangle with fields tricode and tetcode
-- (tetrahedron of which the triangle is one of the boundary triangles)

CREATE OR REPLACE VIEW triangle AS
   SELECT deriveboundarytriangle1(tetcode) tricode, tetcode fromtetcode
   FROM tetrahedron
   UNION ALL
   SELECT deriveboundarytriangle2(tetcode) tricode, tetcode fromtetcode
   FROM tetrahedron
   UNION ALL
   SELECT deriveboundarytriangle3(tetcode) tricode, tetcode fromtetcode
   FROM tetrahedron
   UNION ALL
   SELECT deriveboundarytriangle4(tetcode) tricode, tetcode fromtetcode
   FROM tetrahedron
;

-- Creating view oppositetriangle (two columns: triangle and its opposite,
-- both encoded, including an inherited objectid

CREATE OR REPLACE VIEW oppositetriangle AS
```

```
   SELECT t1.tricode tricode, t2.tricode oppositetricode
   FROM triangle t1, triangle t2
   WHERE removetriobjectid(t2.tricode) = -1 *removetriobjectid(t1.tricode)
;


-- Creating view constrainedtriangle (with inherited object id's)

CREATE OR REPLACE VIEW constrainedtriangle AS
   SELECT t1.tricode tricode FROM triangle t1
   WHERE NOT EXISTS (SELECT t2.tricode FROM triangle t2
                     WHERE t1.tricode = t2.tricode*-1)
;


--  Creating view edge (without inherited object id's and orientation)

CREATE OR REPLACE VIEW edge AS
   SELECT DISTINCT deriveabsboundaryedge1(tricode) edcode FROM triangle
   UNION
   SELECT DISTINCT deriveabsboundaryedge2(tricode) edcode FROM triangle
   UNION
   SELECT DISTINCT deriveabsboundaryedge3(tricode) edcode FROM triangle
;


-- Creating view constrainededge (without inherited object id's and orientation)

CREATE OR REPLACE VIEW constrainededge AS
   SELECT DISTINCT deriveabsboundaryedge1(tricode) edcode FROM constrainedtriangle
   UNION
   SELECT DISTINCT deriveabsboundaryedge2(tricode) edcode FROM constrainedtriangle
   UNION
   SELECT DISTINCT deriveabsboundaryedge3(tricode) edcode FROM constrainedtriangle
;


--  Creating view node (without inherited object id's)
CREATE OR REPLACE VIEW node AS
   SELECT DISTINCT deriveboundarynode1(edcode) nodecode FROM edge
   UNION
   SELECT DISTINCT deriveboundarynode2(edcode) nodecode FROM edge
;


-- Creating view tetrahedronneighbours

CREATE OR REPLACE VIEW tetrahedronneighbours AS
   SELECT tetcode tetcode, getneighbourtet1(tetcode) ntet1,
                           getneighbourtet2(tetcode) ntet2,
                           getneighbourtet3(tetcode) ntet3,
                           getneighbourtet4(tetcode) ntet4
   FROM tetrahedron
;
```

Appendix III
Converting to
Oracle Spatial 11g polyhedrons

This appendix contains the script for converting objects into Oracle Spatial 11g polyhedrons. It exports the constrained triangles into polyhedrons with triangular faces. Note that this is not the optimal solution: this would require merging triangles that are coplanar (within a tolerance). This polyhedron representation is used in chapter 7 to compare storage requirements.

```
DROP TABLE solids PURGE;

CREATE OR REPLACE VIEW vistemp AS
   SELECT tricode, to_number(getobjectid_tt(10,tricode)) oid
   FROM constrainedtriangle_tt
   WHERE getobjectid_tt(10,tricode)<>0
;


================================================================================
Creating table with solids
================================================================================

CREATE TABLE solids (oid number, geom mdsys.sdo_geometry);

DECLARE
   tmp_geom mdsys.sdo_geometry;
   x1 number;
   y1 number;
   z1 number;
   x2 number;
   y2 number;
   z2 number;
   x3 number;
   y3 number;
   z3 number;
   ni number;              -- ni is element number of elem_info array
   pos number;              -- pos is position in ordinate array

   tricode vistemp.tricode%TYPE;
   type object_list is varray(100000) of vistemp.tricode%TYPE;
   tmp_objects object_list;
   t number;
```

```
BEGIN
   FOR a IN (SELECT DISTINCT oid FROM vistemp)
   LOOP
      ni       := 1;
      pos      := 1;

      tmp_geom := mdsys.sdo_geometry(3008,null,null,mdsys.sdo_elem_info_array(),
                                     mdsys.sdo_ordinate_array());

      SELECT tricode BULK COLLECT INTO tmp_objects FROM vistemp c WHERE c.oid=a.oid;

      FOR b IN tmp_objects.first..tmp_objects.last
      LOOP
         tmp_geom.sdo_elem_info.extend(3);

         tmp_geom.sdo_elem_info(ni)   := pos;
         tmp_geom.sdo_elem_info(ni+1) := 1007;
         tmp_geom.sdo_elem_info(ni+2) := 1;
         ni := ni + 3;

         tmp_geom.sdo_ordinates.extend(9);

         getnodecoords_tt(getnode1oftriangle_tt(10,tmp_objects(b)),x1,y1,z1);
         getnodecoords_tt(getnode2oftriangle_tt(10,tmp_objects(b)),x2,y2,z2);
         getnodecoords_tt(getnode3oftriangle_tt(10,tmp_objects(b)),x3,y3,z3);

         tmp_geom.sdo_ordinates(pos)   := x1;
         tmp_geom.sdo_ordinates(pos+1) := y1;
         tmp_geom.sdo_ordinates(pos+2) := z1;
         tmp_geom.sdo_ordinates(pos+3) := x2;
         tmp_geom.sdo_ordinates(pos+4) := y2;
         tmp_geom.sdo_ordinates(pos+5) := z2;
         tmp_geom.sdo_ordinates(pos+6) := x3;
         tmp_geom.sdo_ordinates(pos+7) := y3;
         tmp_geom.sdo_ordinates(pos+8) := z3;
         pos := pos + 9;

      END LOOP;

      INSERT INTO SOLIDS VALUES (a.oid,tmp_geom);
      COMMIT;
   END LOOP;
END;
/
```

Appendix IV
TetGen files

This appendix contains example TetGen files. It shows the input `*.poly` file of the 'toy' data set (see chapter 7) and the resulting `*.node` (including Steiner points) and `*.ele` files (note that for reasons of compactness a part of this file is omitted) with respectively the nodes and the tetrahedrons.

Input file: `*.poly`

```
# node list
#    First line: <# of points> <dimension (3)> <# of attributes><boundary markers)>
#    Remaining lines list # of points: <point #> <x> <y> <z>

20 3 0 0
1 0 0 0
2 0 10 1
3 0 16 1
4 0 50 2
5 40 50 3
6 40 16 0
7 40 10 0
8 40 0 0
9 14 25 2
10 14 35 2
11 22 35 2
12 22 25 2
13 14 25 8
14 14 35 8
15 18 35 11
16 22 35 8
17 22 25 8
18 18 25 11
19 20 25 -10
20 20 25 25

#facet list
#   One line: <# of facets> <boundary markers (0 or 1)>
#   Following lines list # of facets: <facet #>
#      Format for facets:
#      One line: <# of polygons> [# of holes]
#      Following lines list # of polygons:
#      <# of corners> <corner 1> <corner 2> ... <corner #>
```

```
#      ...
#      Following lines list # of holes:
#      <hole #>
#
#(note that in this example facets have no holes and exist of one polygon)

27 0
1 0
3 1 2 7
1 0
3 1 7 8
1 0
3 2 3 6
1 0
3 2 6 7
1 0
3 3 12 6
1 0
3 3 9 12
1 0
3 3 4 10
1 0
3 3 9 10
1 0
3 4 10 11
1 0
3 4 11 5
1 0
3 5 11 6
1 0
3 11 12 6
1 0
4 12 11 10 9
1 0
4 9 10 14 13
1 0
5 10 11 16 15 14
1 0
4 11 12 17 16
1 0
5 9 13 18 17 12
1 0
4 14 15 18 13
1 0
4 15 16 17 18
1 0
5 4 3 2 1 19
1 0
5 8 7 6 5 19
1 0
3 5 4 19
1 0
3 1 8 19
1 0
5 1 2 3 4 20
1 0
```

```
5 5 6 7 8 20
1 0
3 4 5 20
1 0
3 8 1 20

# list of volume holes

0

# list of region attributes
#    One line: <# of region>
#    Following lines list # of region attributes:
#    <region #> <x> <y> <z> <region number>

3
1 20 25 70 1
2 20 25 -8 2
3 20 30 5 3
```

Output file: *.node

```
31  3  0  0
   1    0  0  0
   2    0  10  1
   3    0  16  1
   4    0  50  2
   5    40  50  3
   6    40  16  0
   7    40  10  0
   8    40  0  0
   9    14  25  2
  10    14  35  2
  11    22  35  2
  12    22  25  2
  13    14  25  8
  14    14  35  8
  15    18  35  11
  16    22  35  8
  17    22  25  8
  18    18  25  11
  19    20  25  -10
  20    20  25  25
  21    20.014135583320211  10  0.49964661041699471
  22    20.015935021083745  16  0.49960162447290635
  23    14  30.004904  2
  24    22  29.995671000000002  2
  25    18.001438  35  2
  26    18.003170000000001  25  2
  27    14  29.997526000000001  8
  28    18  29.999835999999998  11
  29    20.599983760302624  5.1499959400756561  0
  30    18.304005323787162  23.488002177912929  1.8320002419903254
  31    15.418360166248526  22.307510977101668  1.7008345530112963
# Generated by tetgen -pAV miniset_tetgen.poly
```

Output file: `*.ele`

```
117  4  1
    1      23    10    14    25    3
    2      26    13    27     9    3
    3      18    22    17    21    1
    4      25    27    24    16    3
    5      27     9     3    23    1
    6      20    17    18    28    1
    7      25    27    16    14    3
    8       2    31     3    13    1
    9      26    31    13     9    1
   10      27    25    24    23    3
   11      27    20    28    14    1
   12      10    14     3    23    1
   13      18    22    21    13    1
   14      16     4    15     5    1
   15      19    30    12    26    2
   16      21    20     7    17    1
   17      16    20     5    15    1
   18       4    15     5    20    1
   19      20     6     7    17    1
   20      19    31     9     3    2
   21       6    11     5    19    2
   22      23    19    10    25    2
   23       7    22    17     6    1
   24       7    29    20    21    1
   25      10    19     3     4    2
   26      29     2    19     1    2
   27      18    31    22    13    1
   28       2    29    19    21    2
   29       6    22    12    19    2
   30       3    21     2    19    2
   31       6    11    19    24    2
   32      10     3    14     4    1
   33       4    27    14    20    1
   34       4    27    20     3    1
   35       4    16    15    14    1
   36       4    14    10    25    1
   37      25    11    16    24    3
   38      29     7    20     8    1
   39      15    20    28    16    1
   40      20     3    13     2    1
   41      25    11    24    19    2

                 ⋮

  111      21    31    13    22    1
  112      17    31    18    30    1
  113      19    31     3    22    2
  114       2    31    21     3    1
  115       2    31    13    21    1
  116      26    31     9    19    2
  117      17    31    22    18    1
# Generated by tetgen -pAV miniset_tetgen.poly
```

Summary

3D Topography

A Simplicial Complex-based Solution in a Spatial DBMS

Current topographic products are limited to a real world representation in only two dimensions, with at best some additional point heights and contour lines. Modelling the real world in two dimensions implies a rather drastic simplification of three dimensional real world elements. By representing these elements in two dimensions, loss of information is inevitable. Due to this simplification, accuracy of analysis results is limited and a meaningful, insightful representation of complex situations is hard to obtain. Environmental issues like high concentrations of particulate matter along highways in urban areas, the effects of noise and odour propagation and risk analysis of liquefied petroleum gas storage tanks are random examples of current issues in 3D urban planning in which more precision is required than 2D analyses can offer. In a time with increasing attention for these kind of environmental and sustainability issues, limitations of 2D models become real problematic and trigger the demand for 3D topography.

The development of 3D topography is also supply-driven, especially by the increasing availability of high density laser scan data. Height data becomes available with point densities –multiple height points per square meter– that were previously unthinkable with traditional photogrammetric stereo techniques. Direct 3D data acquisition by terrestrial laser scanning is emerging, thus providing detailed measurements of facades, tunnels and even indoor topography. The fast developments in this field are partly triggered by the emerging popularity of personal navigation devices, which will use 3D models in the future to simplify user interpretation of the (map) display.

Objective and research question

The objective of this research is to develop a data structure that is capable of handling large data volumes and offers support for loading, updating, querying, analysis and especially validation. To achieve this, a triangular approach will be used, due to its advantages in maintaining consistency, its robustness and editability. This triangular approach creates a network of triangles (in 2D) or tetrahedrons (in 3D), in which topographic features are represented by sets of triangles or tetrahedrons. Such a network is an example of an irregular tessellation, in which the real world is decomposed into smaller (triangle/tetrahedron-shaped) building blocks. The resulting networks are called TINs (Triangular Irregular Networks) or TENs (TEtrahedronised irregular Networks). The presence of boundaries of topographic features are ensured by the use of constraints, preventing the deletion of crucial boundary edges and triangles. Algorithms exist to calculate these constrained triangulations and constrained tetrahedronisations of topographic data.

In this research a two-step approach will be adopted. First one has to decide how real-world objects should be modelled into features, secondly one needs to store these features in such a way that the requirements in terms of querying, analysis and validation are met. An obvious step in dealing with large volumes of geographically referenced data, is to use a spatial database.

This objective is expressed in the main research question:

How can a 3D topographic representation be realised in a feature-based triangular data model?

Note that the term 'triangular' is used here in general dimension, so both triangle- and tetrahedron-based models will be considered. As mentioned before, a two-step approach will be adopted to achieve a solution to the main research question. In accordance with the two steps, two key questions can be distinguished:

- How to develop a conceptual model that describes the real world phenomena (the topographic features), regarding the general purpose-characteristic of topographic data sets?
- How to implement this conceptual model, i.e. how to develop a suitable DBMS data structure?

The results of this research will be summarised according to this two-step approach.

A conceptual data model for 3D topography

One of the basic assumptions within this research is the use of triangular data models. As a result, topographic features will be described as sets of triangles and these features will be connected by triangles as well, thus creating one triangular network. This research explored two different approaches to triangular modelling of 3D topography.

- The first one is a very pragmatic hybrid approach that combines a 2.5D* surface with 3D objects for those cases where 2.5D modelling is not sufficient. In terms of triangular data structures, this approach combines a TIN with several TENs. These irregular data structures not only allow varying point density (depending on local model complexity), but extend this irregularity into varying even model dimensionality, thus offering the ultimate fit-for-purpose approach. Unfortunately, connecting TIN and TEN networks appeared to be very difficult at design level and during prototype implementation.

- The second approach avoids these problems, since it is a full 3D approach using only a TEN. Two fundamental observations are of great importance:

  - Physical objects have by definition a volume. In reality, there are no point, line or polygon objects; only point, line or polygon representations exist (at a certain level of abstraction/generalisation).
  - The real world can be considered a volume partition: a set of nonoverlapping volumes that form a closed (i.e. no gaps within the domain) modelled space. Objects like 'earth' or 'air' are thus explicitly included in the model.

  In topographic data models, planar features like walls or roofs are obviously very useful. They can be part of the volumetric data model as 'derived features', i.e. these features depend on the relationship between volume features. For example, the earth surface is the boundary between air and earth features, while a wall or a roof are the result of adjacent building and air features. In terms of UML, these planar features are modelled as association classes. As a result, planar features are lifetime dependent from the association between two volume features.

Among the advantages of the full volumetric approach are its explicit inclusion of air and earth (often subject of analysis), its extensibility (geology, air traffic/telecommunication corridors, etc.) and its strong mathematical definition (full connectivity enables the use of topology for query, analysis and validation). As a result, topographic features will be modelled in a TEN. Each feature will be represented by a set of tetrahedrons.

A data structure for 3D topography

The newly developed data structure has three important characteristics:

- It has a *solid mathematical foundation*. Operators and definitions from the mathematical field of Poincaré simplicial homology (part of algebraic topology) are used to handle simplexes†, the basic elements in a triangular data structure. Simplexes are well defined, ordered and constructed of simplexes of lower dimension. The boundary operator can be used to derive these less dimensional

*See section 2.2 for an overview of often-used dimension indicators

†A simplex can loosely be defined as the simplest shape in a dimension, in which simplest refers to minimising the number of points required to define such a shape, for instance a point, a line, a triangle and a tetrahedron. See section 4.1 for a proper mathematical definition

simplexes. Based on the ordering of simplexes, one can determine orientation, a useful concept in GIS. Another important concept from simplicial homology is the simplicial complex, since such a set of connected simplexes will be used to model 3D topographic features.

- It is developed as a *spatial database data structure*. Applying definitions and operators from simplicial homology enables one to store a TEN in a relatively compact way. The new simplicial complex-based method requires only explicit storage of tetrahedrons, while simplexes of lower dimensions (triangles, edges, nodes), constraints (which guarantee feature boundary presence) and topological relationships can be derived in views. Using functions to derive views from a table is typical database functionality. In this implementation, simplexes are encoded by their vertices, similar to the annotation in simplicial homology. These simplex encodings are extended with a feature identifier, indicating which topographic feature is (partly) represented by this simplex. So, a tetrahedron is encoded as $S_3 =< v_0, v_1, v_2, v_3, fid >$. Two variants in simplex encoding have been developed: coordinate concatenation and identifier concatenation. The concept of coordinate concatenation is to concatenate x, y and z coordinates as node identifiers and to concatenate the resulting unique node codes to describe simplexes of higher dimension. The alternative approach, identifier concatenation, uses separate (meaningless) node identifiers to encode simplexes to reduce the number of coordinate repetitions, since a specific node will be part of multiple tetrahedrons. This approach requires an additional node table to store node geometries.

- It is an *editable data structure*, which is a crucial prerequisite to be a feasible approach for topographic data storage. Incremental updates are required, since complete rebuilds of the TEN structure will be time-consuming due to the expected data volumes. Whereas most existing update algorithms for constrained tetrahedronisations use node insertions, followed by edge reconstruction, this research presents edge insertion operators. Nine exhaustive and mutually exclusive cases are distinguished, based on the location in the TEN of the inserted edge's nodes. These operators guarantee the constrained edge's presence in the structure. Existing operators might fail to recover these edges, due to the presence of nearby constrained edges, which would typically happen in topographic data sets.

Conclusions

This dissertation presents a new topological approach to data modelling, based on a tetrahedral network. Operators and definitions from the field of simplicial homology are used to define and handle this structure of tetrahedrons. Simplicial homology provides a solid mathematical foundation for the data structure and offers full control over orientation of simplexes and enables one to derive substantial parts of the TEN structure efficiently, instead of explicitly storing all primitives. DBMS characteristics as the usage of views, functions and function-based indexes are extensively used to

realise this potential data reduction. A proof-of-concept implementation was created and tests with several data sets show that the prevailing view that tetrahedrons are more expensive in terms of storage when compared to polyhedrons, is not correct when using the proposed approach. Storage requirements for 3D objects in tetrahedronised form (excluding the space in between these objects) and 3D objects stored as polyhedrons, are in the same order of magnitude.

A TEN has favourable characteristics from a computational point of view. All elements of the tetrahedral network consist by definition of flat faces, all elements are convex and they are well defined. Validation of 3D objects is also simplified by tetrahedronisation. Furthermore, a full volumetric approach enables future integration of topography with other 3D data like geological layers, polluted regions or air traffic and telecommunication corridors. The price of this full volumetric approach in terms of storage space is high (about 75% of the tetrahedrons models air or earth); nevertheless this approach is likely to become justifiable due to current developments towards sustainable urban development and increased focus on environmental issues.

Now the innovative aspects of the proposed method has to be identified. Neither the idea to use a TEN data structure for 3D data nor the idea to use simplexes (in terms of simplicial homology) in a DBMS implementation is new. However, the proposed approach reduces data storage and eliminates the need for explicit updates of both topology and simplexes of lower dimension. By doing so, the approach tackles common drawbacks as TEN extensiveness and laboriousness of maintaining topology. Furthermore, applying simplicial homology offers full control over orientation of simplexes, which is a significant advantage, especially in 3D. In addition to this aspect, the mathematical theory of simplicial homology offers a solid theoretical foundation for both the data structure and data operations. Integrating these concepts with database functionality results in a new innovative approach to 3D data modelling.

An often raised objection to a TEN approach is its presumed complexity. Obviously, a 1:n relation exists between features and their tetrahedron representations. However, as long as a user handles only features (as polyhedrons) and implemented algorithms translate these polyhedrons into operations on the TEN, one can overcome the perceived complexity. Furthermore, the prevailing view that tetrahedrons are more expensive in terms of storage than polyhedrons has been falsified in this research.

Overall, the simplicial complex-based modelling approach provides a provable correct modelling method. The use of tetrahedrons is justified by the mathematical benefits and the acceptable storage requirements. Obviously, including air and earth within the model comes at a price, but –as stated earlier– this approach is likely to become justifiable, due to current sustainability and environmentally-driven developments. The decision to develop the data structure as a database structure contributes to the overall feasibility, since a database will become indispensable due to the expected data volumes.

Samenvatting

3D Topography

A Simplicial Complex-based Solution in a Spatial DBMS

De representatie van topografische objecten beperkt zich in de huidige generatie topografische producten meestal tot twee dimensies, hooguit aangevuld met een aantal hoogtepunten of -lijnen. Het modelleren van de werkelijkheid in twee dimensies impliceert een behoorlijk drastische simplificatie van de werkelijkheid, die zelf driedimensionaal is. Door deze werkelijkheid in 2D te representeren, is verlies aan (detail)informatie onvermijdelijk. Door deze simplificaties wordt niet alleen de nauwkeurigheid van analyses op deze data beperkt, maar wordt een inzichtelijke representatie van complexe 3D situaties ook zo goed als onmogelijk. Het modelleren van concentraties fijn stof langs snelwegen in stedelijke gebieden, geluids- en stankanalyses en risicoanalyses van LPG opslagtanks zijn een aantal willekeurige voorbeelden van 3D planning waarin de nauwkeurigheden van 2D analyses niet toereikend zijn. Met de huidige aandacht voor milieu- en duurzaamheidgerelateerde vraagstukken komen de beperkingen van 2D modellen steeds nadrukkelijker aan het licht. Deze beperkingen zijn een belangrijke aanleiding voor de groeiende vraag naar 3D topografie.

Tegelijkertijd kunnen de ontwikkelingen richting 3D topografie ook worden verklaard door het groeiende aanbod van laserscan datasets met zeer hoge puntdichtheden. Hoogtebestanden komen op de markt met puntdichtheden –meerdere punten per vierkante meter– die ondenkbaar waren met traditionele inwintechnieken zoals stereofotogrammetrie. Daarnaast maakt ook de directe 3D inwinning met terrestrische laserscanning een stormachtige ontwikkeling door, waarmee gedetailleerde modellen van gevels, tunnels of indoor topografie beschikbaar komen. De snelle ontwikkelingen op het gebied van data-acquisitie worden deels veroorzaakt door de snel groeiende populariteit van navigatiesystemen, die in de toekomst steeds vaker 3D modellen zullen gebruiken om zo een makkelijker te interpreteren ‘kaart’beeld te bieden.

Doel en onderzoeksvraag

Het doel van dit onderzoek is om een datastructuur te ontwikkelen die om kan gaan met grote hoeveelheden data en die ondersteuning biedt voor het laden, bijhouden, bevragen, analyseren en nadrukkelijk ook valideren van de data. Hiertoe zal een triangulatiemethode gebruikt worden, met name vanwege de voordelen op het gebied van consistentie bewaking, robuustheid en behoudbaarheid. Deze triangulatiemethode resulteert in een netwerk van driehoeken (in 2D) of tetraëders (in 3D), waarin de topografische objecten worden gerepresenteerd door een verzameling driehoeken of tetraëders. Dergelijke netwerken zijn een voorbeeld van een onregelmatige ruimteverdeling (mozaïek), waarin de werkelijkheid wordt opgedeeld in een verzameling (driehoekige/tetraëdervormige) bouwstenen. De op deze wijze verkregen netwerken staan bekend als TINs (Triangulated Irregular Networks) of TEN (TEtrahedronised irregular Networks). De aanwezigheid van begrenzingen van topografische objecten in deze netwerken wordt gegarandeerd door zogenaamde constraints (condities), die voorkomen dat cruciale grenszijdes of -driehoeken worden verwijderd uit de netwerken. Deze constrained triangulaties en tetraëdrisaties kunnen berekend worden met speciale algoritmes.

In dit onderzoek wordt een tweestaps aanpak gehanteerd. Eerst wordt bepaald op welke wijze fysieke objecten moeten worden gemodelleerd en vervolgens moet worden bepaald hoe deze objecten opgeslagen worden, zodat aan de eisen op het gebied van bevragen, analyse en valideren wordt voldaan. Aangezien er met grote hoeveelheden geografische gegevens moet worden omgegaan, is het gebruik van een ruimtelijk database management systeem een voor de hand liggende keuze.

Deze doelstelling van het onderzoek komt tot uitdrukking is de onderzoeksvraag:

Hoe kan een 3D topografische representatie worden gerealiseerd in een objectgericht, triangulatie-gebaseerd data model?

Merk op dat de term 'triangulatie' hier in algemene zin wordt gebruikt, dus zowel driehoeks- als tetraeder-gebaseerde aanpakken komen in aanmerking. Zoals eerder opgemerkt, zal een tweestaps aanpak gehanteerd worden om de onderzoeksvraag te kunnen beantwoorden. Voor elk van deze twee stappen kan een deelvraag worden geformuleerd:

- Hoe kan een conceptueel data model worden ontwikkeld voor fysieke objecten (de topografie), gezien het feit dat topografische bestanden voor een breed scala aan toepassingen wordt gebruikt?
- Hoe kan dit conceptuele model worden geïmplementeerd of, met andere woorden, hoe ziet een bijpassende DBMS datastructuur eruit?

De resultaten van dit onderzoek zullen aan de hand van deze twee stappen worden samengevat.

Een conceptueel datamodel voor 3D topografie

Eén van de uitgangspunten van dit onderzoek is het gebruik van triangulaties. Dit houdt in dat topografische objecten gerepresenteerd zullen worden door verzamelingen driehoeken en dat ook de tussenliggende ruimte bestaat uit driehoeken, wat samen resulteert in een netwerk van driehoeken. In dit onderzoek zijn twee verschillende aanpakken voor een dergelijke modellering van 3D topografie onderzocht.

- Eerst is gekeken naar een pragmatische hybride aanpak, waarin een 2,5D[‡] terreinoppervlak wordt gecombineerd met afzonderlijke 3D modellen in die gevallen dat een 2,5D representatie niet voldoet. In termen van triangulaties bestaat deze aanpak uit het combineren van een TIN met diverse TENs. Deze aanpak wordt niet alleen gekenmerkt door de wisselende puntdichtheden (afhankelijk van de lokale complexiteit van het model), maar ook door de uitbreiding naar wisselende modeldimensies, waarmee maatwerk geleverd wordt. Helaas bleek de daadwerkelijke verbinding tussen het TIN en de TENs op ontwerpniveau uitermate lastig.
- De tweede aanpak vermijdt dit probleem, doordat het een volledig 3D model is, waarin alleen een TEN gebruikt wordt. Twee fundamentele uitspraken zijn in dit kader van groot belang:
  - Fysieke objecten hebben per definitie een volume. In de werkelijkheid bestaan geen punt-, lijn- of vlakobjecten; alleen punt-, lijn- of vlakweergaves bestaan (afhankelijk van de mate van abstractie/generalisatie).
  - De fysieke werkelijkheid kan beschouwd worden als een volume partitie: een set van niet-overlappende volumes die samen de gehele ruimte vullen (gaten komen dus niet voor binnen het model). Objecten zoals 'lucht' of 'aarde' maken dan ook expliciet deel uit van het model.

  In topografische modellen zijn vlakobjecten zoals muren en daken natuurlijk uitermate bruikbaar. Dergelijke vlakobjecten kunnen in een volumemodel voorkomen als 'afgeleide objecten', d.w.z. dat deze objecten afhankelijk zijn van een relatie tussen twee volumeobjecten. Zo kan het aardoppervlak bijvoorbeeld gedefinieerd worden aan de hand van de buurrelaties tussen 'aarde' en 'lucht' objecten, terwijl muren en daken het resultaat zijn van de buurrelatie tussen een gebouw en de lucht. In UML termen worden vlakobjecten gemodelleerd als 'association classes', waarmee het bestaan van elk vlakobject dus volledig afhangt van het bestaan van de relatie tussen twee volumeobjecten.

Voordelen van het volledig 3D model zijn o.a. de expliciete aanwezigheid van lucht en aarde (vaak onderwerp van analyse), de uitbreidbaarheid (in de toekomst met geologie, luchtvaartroutes, straalpaden, enz.) en de sterke wiskundige onderbouwing (de volledige connectiviteit in de volumepartitie maakt topologische bevragingen, analyses en validatie mogelijk). Hierom is er voor gekozen om 3D topografie te modelleren in een TEN. Elk object wordt gerepresenteerd door een verzameling tetraëders.

[‡]Zie sectie 2.2 voor een overzicht van veelgebruikte dimensie-aanduidingen

Een datastructuur voor 3D topografie

De nieuw ontwikkelde datastructuur heeft drie kenmerkende eigenschappen:

- De datastructuur heeft een *solide wiskundige onderbouwing*. Operatoren en definities uit het wiskundige deelgebied van de Poincaré simpliciale homologie (onderdeel van de algebraïsche topologie) worden gebruikt om simplexen[§] te hanteren, de bouwstenen in een tetraëderstructuur. Simplexen zijn goed gedefinieerd, geordend en bestaan uit simplexen van lagere dimensies. De grensoperator kan gebruikt worden om deze simplexen van lagere dimensies af te leiden. Op basis van de ordening van simplexen kan de oriëntatie bepaald worden; een bruikbaar concept in GIS toepassingen. Een ander belangrijk concept uit de simpliciale homologie is het simpliciale complex, aangezien zo'n verzameling van verbonden simplexen gebruikt zal worden voor het modelleren van 3D topografische objecten.

- Het is een *ruimtelijke database datastructuur*. Het gebruik van operatoren en definities uit de simpliciale homologie maakt het mogelijk om een TEN relatief compact op te slaan. De nieuwe simpliciale complex-gebaseerde methode vergt alleen expliciete opslag van tetraëders, terwijl simplexen van lagere dimensies (driehoeken, zijden, punten), constraints (die de aanwezigheid van de objectgrenzen garanderen) en topologische relaties kunnen worden afgeleid in views (virtuele tabellen). Het gebruik van functies om views af te leiden uit een tabel is typische database functionaliteit. In de implementatie zijn simplexen gecodeerd aan de hand van hun punten, vergelijkbaar met de notatie die in de simpliciale homologie wordt gebruikt. Aan de simplexcode wordt nog een objectnummer toegevoegd, die aangeeft welk topografisch object (gedeeltelijk) wordt gerepresenteerd door deze simplex. Een tetraëder wordt dus als volgt gecodeerd: $S_3 =< v_0, v_1, v_2, v_3, oid >$. Binnen deze codering zijn twee varianten ontwikkeld: coördinaat concatenatie en objectnummer concatenatie. Het idee achter coördinaat concatenatie is om elk punt te coderen aan de hand van het x, y en z coördinaat en vervolgens deze unieke puntnummers te concateneren om zo simplexen van hogere dimensie te coderen. Het alternatief, puntnummer concatenatie, gebruikt betekenisloze puntnummers om simplexen mee te coderen om zo herhaling van coördinaten te vermijden. Deze herhaling wordt veroorzaakt door het gegeven dat elk punt deel uitmaakt van meerdere tetraëders. Dit alternatief vereist wel een aanvullende punttabel waarin voor elk punt het puntnummer en de geometrie wordt opgeslagen.

- Het is een *bijhoudbare datastructuur*, een cruciale vereiste voor een datastructuur voor topografie. Het incrementeel bijhouden van de structuur is noodzakelijk, aangezien het volledig herbouwen van de TEN structuur te veel tijd zal kosten

[§]Een simplex kan losjes gedefinieerd worden als de eenvoudigste geometrie in een dimensie, waarbij eenvoudigst slaat op het minimaliseren van het aantal punten dat nodig is om die geometrie te definiëren, bijvoorbeeld een punt, een lijn, een driehoek en een tetraëder. Zie sectie 4.1 voor de zuiver wiskundige definitie

gezien de te verwachten hoeveelheden data. Alhoewel de meeste bestaande bijhoudoperaties voor constrained tetraëdrisatie gebruik maken van het invoegen van punten, waarna zijdes worden gereconstrueerd, gaan de bijhoudoperaties in dit onderzoek uit van het invoegen van zijdes. Afhankelijk van de locatie van begin- en eindpunt van de zijde, kunnen negen uitputtende en niet overlappende gevallen worden onderscheiden. Deze set operatoren garandeert de aanwezigheid van de ingevoegde zijde in de datastructuur. Bestaande operatoren bieden deze garantie niet altijd, aangezien het reconstrueren van zijdes kan mislukken door de nabije aanwezigheid van andere 'constrained' zijdes, iets wat typisch het geval zal zijn in topografische bestanden.

Conclusies

Dit proefschrift beschrijft een nieuwe topologische aanpak van datamodellering die uitgaat van tetraëdernetwerken. Operatoren en definities uit de simpliciale homologie worden toegepast in het definiëren en bewerken van deze structuur van tetraëders. De simpliciale homologie biedt een solide wiskundige onderbouwing van de datastructuur, volledige controle over oriëntatie van simplexen en maakt het mogelijk om grote delen van de datastructuur af te leiden in plaats van expliciet op te slaan. Databasefunctionaliteit zoals het gebruik van views, functies en functiegebaseerde indexen worden toegepast om het potentieel van de datastructuur te realiseren. Een experimentele implementatie is ontwikkeld en tests met verschillende datasets tonen aan dat de heersende opvatting dat tetraëders meer opslagruimte vergen dan polyëders, achterhaald is. De vereiste opslagruimte voor 3D objecten in getetraëdriseerde vorm (het modelleren van tussenliggende ruimtes niet meegerekend) en voor 3D objecten in polyëderformaat, liggen in dezelfde orde van grootte.

Het gebruik van een TEN biedt een aantal rekenkundige voordelen. Elk element in een TEN bestaat uit platte vlakken, elk element is convex en alle elementen zijn goed gedefinieerd. Validatie van 3D objecten wordt ook eenvoudiger wanneer er gebruik gemaakt wordt van tetraëdernetwerken. Daarnaast biedt de volledig 3D modellering ruimte voor toekomstige integratie van topografie met andere 3D data zoals geologie, vervuilde aardlagen en straalpaden. De prijs van het volledig in 3D modelleren is hoog (ongeveer 75% van de tetraëders beschrijft lucht of aarde); toch kan deze aanpak gerechtvaardigd worden, gezien de huidige ontwikkelingen richting duurzame stedelijke ontwikkeling en toegenomen aandacht voor milieukwesties.

Nu kunnen ook de vernieuwende aspecten van de voorgestelde methode worden bepaald. Noch het idee om een TEN structuur te gebruiken voor 3D data, noch het idee om simplexen (zoals in de simpliciale homologie) in een database structuur te gebruiken, is nieuw. Desondanks brengt de voorgestelde methode de vereiste opslagruimte terug en maakt het expliciete bijhouding van zowel topologie als van simplexen van lagere dimensie overbodig. Hiermee ondervangt de nieuwe methode de 'klassieke' nadelen van een tetraëderstructuur. Daarnaast biedt het toepassen van simpliciale homologie volledige controle over oriëntatie van simplexen, wat met name in 3D een enorm voordeel is. Bovendien biedt simpliciale homologie een solide

wiskundige onderbouwing voor zowel de datastructuur als voor de operaties erop. Het integreren van deze concepten in een databaseomgeving resulteert in een vernieuwende aanpak van 3D datamodellering.

Een vaakgenoemd nadeel van een TEN aanpak is de veronderstelde complexiteit. Immers, er bestaat een 1:n relatie tussen objecten en hun tetraëderrepresentatie. Hoe dan ook, zolang een gebruiker alleen objecten manipuleert en algoritmes zorgdragen voor de bijbehorende wijzigingen in de datastructuur, wordt dit nadeel volledig ondervangen. Daarnaast is aangetoond dat een TEN aanpak niet meer opslagruimte vergt dan een polyëderaanpak, waarmee omvang dus niet meer bijdraagt aan eventuele complexiteit.

Afsluitend kan gesteld worden dat de simpliciale complex-gebaseerde aanpak een bewijsbaar correcte modelleermethode is. Het gebruik van tetraëders wordt gelegitimeerd door wiskundige voordelen en acceptabele opslagvereisten. Natuurlijk leidt het expliciet opnemen van lucht en aarde tot een aanzienlijke toename in datavolume, maar –zoals eerder reeds opgemerkt– toch kan deze aanpak gerechtvaardigd worden, gezien de huidige ontwikkelingen op duurzaamheid- en milieugebied. De keuze om de datastructuur in een ruimtelijke database te implementeren draagt bij aan de algehele bruikbaarheid, aangezien het gebruik van een database onvermijdelijk zal worden gezien de verwachte datavolumes.

Curriculum Vitae

Friso Penninga was born at August 5, 1978 in Zevenhuizen. In 1996 he obtained his high school diploma (in Dutch: atheneum) from Coenecoop College in Waddinxveen, after which he started his study Geodesy at Delft University of Technology. He obtained his MSc degree in 2003 with a thesis called 'Generating a 6-position Postal Code Map based on the Cadastral Registration'. The main results of this research were published in GeoInformatica[¶].

In 2004 he joined the Section GIS Technology of the OTB Research Institute at the same university. First he worked on a temporary basis, after which he switched to a PhD-position in April 2004. He was awarded the Geo-Info prize 2005 for Best Article in April 2006 for an article describing his PhD research. The 3D Topography research project, of which this PhD research is one of the key components, was awarded the Dutch Geo-Innovation Award 2007 in the category Science. In 2007 he was one of the organisers of the 3D GeoInfo '07 Workshop, held at the Conference Centre of Delft University of Technology (December 12-14, 2007). In April 2008 he started as an assistant professor within the Section GIS Technology. A more comprehensive overview of his professional activities can be found at http://www.frisopenninga.nl.

[¶]Friso Penninga, Edward Verbree, Wilko Quak and Peter van Oosterom. *Construction of the Planar Partition Postal Code Map Based on Cadastral Registration* In: GeoInformatica, Volume 9, 2, 2005, pp. 181-204