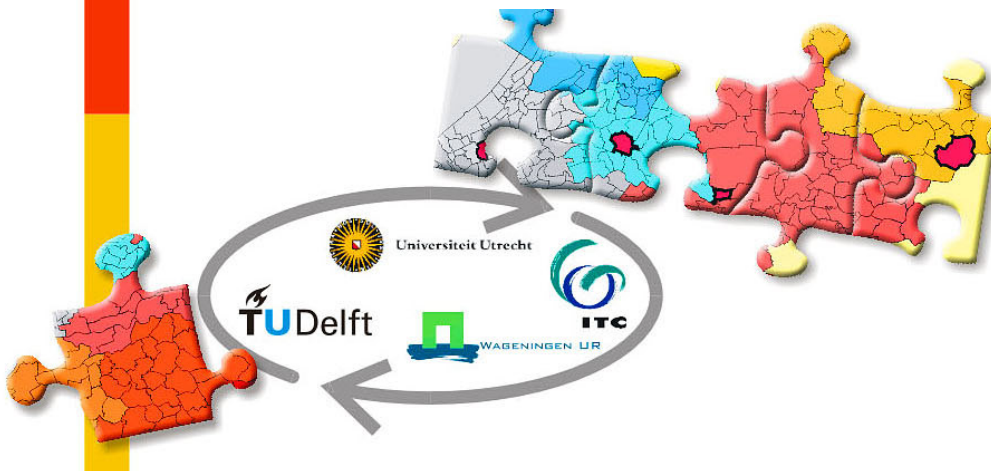


# GIMA

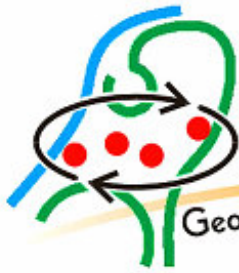
Geographical Information Management and Applications

Visualization of a TEN (Tetrahedral Irregular Network) in a web client.

Sijmen Wesselingh







# GIMA

Geographical Information Management and Applications

## Visualization of a TEN (Tetrahedral Irregular Network) in a web client.

GIMA MSc thesis 8

By

Sijmen Wesselingh

In Partial Fulfillment of the Requirements for the Degree  
“Geographical Information Management and Applications”  
at Utrecht University, Delft University of Technology, Wageningen University and the International  
Institute for Geo-Information Science and Earth Observation.

Research Institute for Housing, Urban and Mobility Studies  
Delft University of Technology  
2007

Copyright 2007 by Sijmen Wesselingh

Professor:	Prof. Dr. Peter van Oosterom, OTB, TUDelft
Supervisor:	Drs. Marian de Vries. OTB, TUDelft
2nd supervisor:	Dr. Friso Penninga. OTB, TUDelft
Reviewer:	Drs. Barend Kobben, ITC, Enschede
Studentname and number:	Sijmen Wesselingh / 0320714
Submittal data and place:	Utrecht, 02-11-2007



## Acknowledgements

First of all I would like to thank my first supervisor Marian de Vries for always willing to help during the Msc project (and earlier during my internship and module six) even sometimes during weekends and for always be open to discussion. Then I want to thank Professor Peter van Oosterom and my second supervisor Friso Penninga for their help and advice during the Msc. Project and Barend Kobben for being the reviewer of my work.

Secondly I would like to thank people at my workplace at the (TNO Built Environment and Geosciences- Geological Survey of the Netherlands) for giving advice sometimes and allowing me to work at my thesis whenever that was required. Thirdly I want to thank Rekha Khurmi from England, temporarily working at TNO, for reviewing my work on English spelling and grammar. Fourthly I want to thank Stan Geertman, the former director of GIMA, for always willing to answer questions about GIMA in general.

In the personal space I like to thank my family (my parents Laurens en Truus and my brother Jasper) for getting me through this period and for reviewing my work. Secondly I want to thank ABC and Helios for keeping me in good shape mentally and physically. Last but not least I want thank my friends for keeping my mind off the work during the free time.

# Table of Contents

<b>LIST OF FIGURES .....</b>	<b>8</b>
<b>LIST OF TABLES .....</b>	<b>9</b>
<b>LIST OF TEXT INSERTMENTS .....</b>	<b>9</b>
<b>LIST OF ABBREVIATIONS .....</b>	<b>10</b>
<b>SUMMARY .....</b>	<b>11</b>
<b>SAMENVATTING.....</b>	<b>12</b>
<b>1. INTRODUCTION .....</b>	<b>13</b>
1.1 Overview.....	13
1.2 Background of the research .....	14
1.3 Objectives of the research .....	15
1.4 Requirements for the prototype development .....	16
1.5 Research Methodology .....	17
<b>2. DATA MODELS FOR 3D GEO-INFORMATION .....</b>	<b>18</b>
2.1 Introduction.....	18
2.2 Feature components.....	18
2.3 Data representations.....	21
2.4 Data models .....	21
2.5 TEN, 3D TIN and ‘TEN view’ data model.....	22
2.6 TEN dataset build-up .....	28
2.7 Topology.....	29
2.8 Database formats .....	30
<b>3. PROTOTYPE RELATED TECHNOLOGY .....</b>	<b>34</b>
3.1 Introduction.....	34
3.2 HTML for website design.....	34
3.3 XML based standards for visualization .....	35
3.4 Standards for database access.....	39
3.5 Applications and plugins .....	41

3.6 Programming languages.....	45
<b>4. REVIEW OF 3D GIS APPLICATIONS ON THE WEB .....</b>	<b>50</b>
4.1 Introduction.....	50
4.2 Review of 3D GIS implementations.....	50
4.3 Feature comparison of prototypes.....	60
4.4 Feasibility review of prototypes.....	62
4.5 Conclusion .....	64
<b>5. PROTOTYPE ARCHITECTURE.....</b>	<b>65</b>
5.1 Introduction.....	65
5.2 Preliminary draft .....	65
5.3 Realized architecture .....	67
5.4 Prototype development process .....	71
<b>6. PROTOTYPE COMPONENTS, DATASETS, AND TECHNIQUES .....</b>	<b>74</b>
6.1 Introduction.....	74
6.2 Prototype components .....	74
6.3 Datasets.....	82
6.4 Implemented techniques.....	85
6.5 Non-implemented techniques.....	99
<b>7. EVALUATION AND CONCLUSION .....</b>	<b>102</b>
7.1 Discussion / evaluation.....	102
7.2 Conclusion / summary .....	104
7.3 Future research and recommendations.....	107
<b>REFERENCES .....</b>	<b>109</b>
<b>APPENDICES .....</b>	<b>116</b>

## List of figures

Figure 1.1: A tetrahedron	
Figure 1.2: A network of tetrahedra (Si, 2007)	13
Figure 1.3: Design architecture	15
Figure 2.1: Convex and concave faces.	19
Figure 2.2: Planar and non-planar face	20
Figure 2.3: Sample of a fully partitioned TEN	23
Figure 2.4: UML TEN model (1 <sup>st</sup> variant), taken over from (Penninga, 2006)	24
Figure 2.5: UML 3D TIN model adapted from TEN model (1 <sup>st</sup> variant)	26
Figure 2.6: UML ‘TEN view’ model adapted from TEN model (1 <sup>st</sup> variant)	27
Figure 2.7: Schematical representation of the TEN, 3D-TIN and ‘TEN view’	28
Figure 2.9: Internal topology	29
Figure 2.10: External topology	30
Figure 3.1: 3D visualization of a TEN dataset in X3D (transformed from GML 2)	38
Figure 4.1: VRML / JAVA web GIS architecture by (Morcrette, 1999)	51
Figure 4.2: VRML / JAVA 3D web GIS prototype architecture by (Kim et al, 1998)	52
Figure 4.3: Web GIS prototype by (Zlatanova, 2000)	53
Figure 4.4: 3D Web GIS Prototype [VRML implementation] (de Vries et al, 2003)	54
Figure 4.5: VRML / JAVA 3D web GIS prototype architecture by (Zu et al, 2004)	54
Figure 4.6: 3D Web GIS prototype [X3D implementation] (Vries et al, 2003) (Held et al, 2004)	56
Figure 4.7: 3D Web GIS Prototype based on SDO and WFS (de Vries et al, 2004)	57
Figure 4.8: 3D Web GIS Prototype with AJAX client model by (Ninsawat et al, 2006)	58
Figure 4.9: X3D/ JAVA 3D web GIS architecture by (Kumke et al, 2006)	59
Figure 5.1: Preliminary draft of prototype architecture for TEN visualization	65
Figure 5.2: Realized architecture of TEN visualization	68
Figure 5.3: Prototype web site structure	69
Figure 5.4: Prototype web site layout	69
Figure 6.1: Web application preview (empty) (from prototype website)	75
Figure 6.2: Web application preview (with data) (from prototype website)	76
Figure 6.3: Titlebar (from prototype website)	76
Figure 6.4: Toolbox (from prototype website)	77
Figure 6.5: 3D visualization in X3D (from prototype website)	78
Figure 6.6: 3D visualization in KML / Google Earth	78
Figure 6.7: 3D visualization in CityGML in Aristoteless (without air tetrahedra)	79
Figure 6.8: 3D Visualization in CityGML in Aristoteless (with transparent air tets)	79
Figure 6.9: 2D visualization in SVG (from prototype website)	80
Figure 6.10: SQL query inputbox (from prototype website)	81
Figure 6.11: SQL query result (from prototype website)	81
Figure 6.12: Schematic representation of an ideal TEN dataset	82
Figure 6.13: Representation small test dataset (right image from prototype website)	83
Figure 6.14: Selection of buildings longer > 10 m in 3D (from prototype website)	84
Figure 6.15: Selection of buildings longer > 10 m in 2D (from prototype website)	84
Figure 6.16: The Original / Faster algorithm in a schema.	90
Figure 6.17: The Joinquery algorithm in a schema.	91
Figure 6.18: House (from prototype website)	94
Table 6.6: Attribute data of small test dataset	95
Figure 6.19: Small dataset with selection of earth tets (from prototype website)	96
Figure 6.20: Textured X3D model	99
Figure 6.21: Sample of Level of Detail implementation in X3D	100



## *List of tables*

Table 2.1: Comparison of datatypes	33
Table 4.1: Comparison of prototypes	60
Table 6.1: Table visualization in HTML (from prototype website)	80
Table 6.2: Table visualization of one record in HTML (from prototype website)	81
Table 6.3 : Selection of buildings longer > 10 m in table (from prototype website)	85
Table 6.4: 'Animal data' table	93
Table 6.5: Attribute data of terrain feature	94
Table 6.7: Attribute data of small test dataset [2]	96
Table 6.8: Transparency and colour attributes of small test dataset	97

## *List of text insertments*

Text insertment 3.1: Convert to integer to string in Java	45
Text insertment 3.2: Convert integer to string in Python	46
Text insertment 3.3: Print "hello" in Java	46
Text insertment 3.4: Print "hello" in Python	46
Text insertment 3.5: Assign four coordinates in Python	46
Text insertment 3.6: Define face of two dimensions in Python	46
Text insertment 3.7: Append coordinates in Python	46
Text insertment 3.8: Append a new face in Python	47
Text insertment 3.9: Read out coordinate in Python	47
Text insertment 6.1: SQL select queries	86
Text insertment 6.2: Joined SQL query to select constrained triangles	87
Text insertment 6.3: Joined SQL query to select tetrahedra	87
Text insertment 6.4: Index number per object ID	89
Text insertment 6.5: Array of faces with node references	89
Text insertment 6.6: IndexedFaceSet string	89
Text insertment 6.7: Viewpoint string	91
Text insertment 6.8: GML poslist string	92
Text insertment 6.9: SVG polygon tag	93
Text insertment 6.10: HTML code for sample table	93
Text insertment 6.11: Jump to a viewpoint with an URL	98
Text insertment 6.12: Jump to a viewpoint with JavaScript	98

## List of abbreviations

2.5D	2.5 Dimensional	ODBC	Open Database Connectivity
3D	3 Dimensional	OGC	Open Geospatial Consortium
3D FDS	3D Formal Data Structure	Oracle 9i	(database software)
3D GIS	3D Geographical Information System	PERL	(programming language)
3D TIN	3D Triangulated Irregular Network	PHP	Hypertext Preprocessor
AJAX	Asynchronous JavaScript And XML	PL/SQL	Procedural Language / Standard Query Language
Apache	(server application)	PLC	Piecewise Linear Complex
API	Application Programming Interface	PyODBC	(ODBC access module for Python)
ASP	Active Server Pages	Python	(programming language)
Blender	(3D design software)	SAI	Scene Acces Interface
BsContact	BsContact (software) (Bs stand for Bitmanagement Software)	SDO	Oracle Spatial Data Option
		SIG	Special Interest Group
CAD	Constructive Automated Drawing	SOMAS	Solid Object Management System
CGI	Common Gateway Interface	SQL	Standard Query Language
CityGML	City Geography Markup Language	SSM	Simplified Spatial model
CSG	Constructive Solid Geometry	SSS	Simplified Spatial Structure
DEM	Digital Elevation Models	SVG	Scalable Vector Graphics
DOM	Document Object Model	TEN	Tetrahedral Irregular Network
EAI	External Authoring Interface	TetGen	(software name)
FHHB	Basel University of Applied Sciences	TIN	Triangulated Irregular Network
Flux	(X3D player)	TNO	Dutch Research Institute for Applied Sciences
GDI NRW	Geo-Data Infrastructure North-Rhine Westphalia	TUdelft	Delft University of Technology
GDMC	Geo-Database Management Center	U3D	Universal 3D standard
GIMA	Geographical Information Management and Applications	UDM	Urban Data Model
		UML	Unified Modelling Language
GIS	Geographical Information System	UMN	University of Minnesota
GML	Geography Markup Language	UU	University of Utrecht
GRASS	(GIS application)	VRML	Virtual Reality Modelling language
GUI	General User Interface	W3C	World Wide Web Consortium
HTML	Hyper Text Markup Language	W3DS	Web 3D Service
HTTP	Hypertext Transfer Protocol	WFS	Web Feature Service
IIS	microsoft Internet Information Server	WMS	Web Map Service
IST	Information Services and Technology	WU	Wageningen University
ITC	International Institute for Geo- Information Science and Earth Observation	X3D	Extensible 3D
		XML	Extensible Markup Language
JAVA	(programming language)	XMML	The Exploration and Mining Markup Language
JDCB	Java Database Connectivity		
KML	Keyhole Markup Language	XSLT	Extensible Stylesheet Language Transformation
LoD	Level of Detail		
NETGEN	(software name)		

## *Summary*

This thesis is about the visualization of a Tetrahedral Irregular Network (TEN) in a web client. A tetrahedron is the simplest geometric form in 3D space, a pyramid with a triangular ground face. Its four points can lie anywhere in the 3D space as long as they do not lie on one plane. 3D objects can be composed from multiple tetrahedra. In a fully partitioned TEN even air and soil are modeled with tetrahedra. Most GIS analyses on a TEN can be deduced to an operation on the triangle simplex, therefore 3D analysis on A TEN is relatively easy compared to analysis on another 3D data structures. Chapter one (Introduction) gives an overview of the background of the research, the objectives, the requirements and the research methodology. In Chapter two (Data models for 3D geo-information) the theory around 3D data modeling is given. It starts with an explanation of the 3D feature components. In the next part, there is a comparison of 3D data representations and 3D data models and a conclusion is drawn why the TEN is the best data model. In the next paragraphs, theory on the TEN data models is given as well as theory on the derived 3D TIN and 'TEN view' data model. In addition, this chapter gives information how to build up a TEN dataset, about 3D topology and a comparison is made how 3D spatial (TEN) data can be stored in a database. Chapter three (Prototype related technology) gives information about XML based standards to exchange and visualize 3D and 2D data and about any other technologies that are used in the prototype made for this thesis. In Chapter four (Review of 3D GIS applications on the web) a number of 3D web GIS prototypes are discussed which have been created in the last decennium. In addition to a description of them, the prototypes are compared on a number of features and a feasibility review is done of each prototype. Some prototypes are picked out and a conclusion is made which are the best prototype examples for the development of the prototype of this thesis. In Chapter five (Prototype architecture) first the preliminary draft of the prototype is discussed. Secondly, the chapter gives a schematic overview of the realized architecture and a description of the main components and techniques of the prototype. Thirdly, the steps that have been taken over time to develop the prototype are listed. In Chapter six (Prototype components, dataset and techniques) first each component of the prototype web application is described which are a 3D view, a 2D view, a Table view, a Titlebar and an SQL box. Secondly, information is given about the datasets that have been used. Thirdly, each technique used in the prototype is explained in detail. In addition some techniques that are not implemented in the prototype are discussed too. Chapter seven (Evaluation and conclusion) starts with an evaluation. First, a comparison is made which XML based language is best for 3D TEN visualization: X3D, KML, or CityGML. Secondly is reviewed in which way the data can best be stored in the database. Thirdly is compared whether it is best to visualize the TEN, 3D TIN, or TEN view. As fourth, the architecture is discussed. In the next part of Chapter six (Summary and conclusion) each sub question as stated in the Chapter one is answered. In addition, an answer is given on the main research question. Finally, remarks are made regarding feature research and recommendations.

## *Samenvatting*

Deze scriptie gaat over de weergave van een Tetrahedral Irregular Network (TEN) in een web client. Een tetrahedron is de simpelste geometrische vorm in de 3D ruimte, namelijk een pyramide met een driehoekig grondvlak. De vier punten van de tetrahedron kunnen op een willekeurige plek in de 3D ruimte liggen zolang ze niet alle vier op één vlak liggen. 3D objecten kunnen samengesteld worden uit meerdere tetrahedra. In een volledig gepartitioneerde TEN zijn zelfs lucht en grond gemodelleerd met tetrahedra. De meeste GIS analyses op een TEN kunnen versimpelt worden tot een bewerking op de driehoek vandaar dat 3D analyse op een TEN relatief makkelijk is in vergelijking met analyse op andere 3D data structuren. Hoofdstuk een (Introductie) belicht de achtergrond van het onderzoek, de doelen, de vereisten en de onderzoeksmethode. Hoofdstuk twee (Data modellen voor 3D geo-informatie) beschrijft de theorie rond 3D data modelering. Het hoofdstuk begint met een uitleg van de 3D data componenten. Vervolgens komt er een vergelijking van 3D data weergaven en 3D data modellen en er is beschreven waarom de TEN het beste data model is. In de daaropvolgende paragrafen zijn de theorie van het TEN, en het afgeleide 3D TIN and 'TEN view' data model weergegeven. Tenslotte geeft dit hoofdstuk weer hoe een TEN dataset moet worden opgebouwd, het geeft informatie over 3D topologie en een vergelijking wordt gemaakt tussen de verschillende manieren waarop ruimtelijke 3D (TEN) data kan worden opgeslagen in een database.

Hoofdstuk drie (Prototype gerelateerde technologie) beschrijft o.a. op XML gebaseerde talen voor het uitwisselen en visualiseren van 3D geo-informatie. Ook elke andere technologie die gebruikt is voor het bouwen van het prototype dat gemaakt is voor deze scriptie wordt hier beschreven. In Hoofdstuk 4 zijn een aantal 3D web GIS applicaties beschreven die in het laatste decennium zijn gemaakt. Deze applicaties worden vergeleken op een aantal kenmerken en als aanvulling hierop wordt er een geschiktheidsanalyse gedaan van elke web applicatie. Tenslotte wordt er geconcludeerd welke 3D web GIS applicaties als beste voorbeeld dienen voor de ontwikkeling van het prototype dat gemaakt is voor deze thesis. In Hoofdstuk 5 (Prototype architectuur) wordt de architectuur van the prototype dat gemaakt is voor deze thesis beschreven. Ten eerste wordt het voorontwerp besproken. Ten tweede wordt een overzicht gegeven van de gerealiseerde architectuur met een beschrijving van de belangrijkste componenten en technieken. Tenslotte zijn de stappen die gedaan zijn om het prototype te realiseren chronologisch weergegeven. In hoofdstuk zes (Componenten, datasets en technieken) zijn ten eerste zijn alle componenten van de web applicatie beschreven (de 3D view, de 2D view, de Tabel View, het Titelgedeelte en de SQL box). Ten tweede wordt er informatie gegeven over de datasets die gebruikt zijn. Ten derde wordt elke gebruikte techniek in detail beschreven. In hoofdstuk zeven (Evaluatie en de conclusie) wordt er ten eerste geevalueerd wat de beste op XML gebaseerde taal voor 3D visualisatie is: X3D, KML of CityGML. Ten tweede wordt er besproken wat de beste manier is om data op te slaan in de database. Als derde wordt gekeken of de TEN, 3D TIN of 'TEN view' het beste gevisualiseerd kan worden en de beste architectuur wordt besproken. In het volgende gedeelte van hoofdstuk zeven wordt een antwoord gegeven op elke deelvraag zoals geformuleerd in de introductie. Daaropvolgend wordt er antwoord gegeven op de hoofdvraag. Tenslotte worden er opmerkingen en aanbevelingen gemaakt die betrekking hebben op toekomstig onderzoek.

# 1. Introduction

## 1.1 Overview

This thesis named “Visualization of a TEN (Tetrahedral Irregular Network) in a web client” is written for the master degree Geographical Information Management and Applications (GIMA). GIMA is actually a joint program of four Dutch universities: the University of Utrecht (UU), Delft University of Technology (TUDelft), Wageningen University (WU), and the International Institute for Geo-Information Science and Earth Observation in Enschede (ITC).

The project is carried out for the Research Institute for Housing, Urban and Mobility Studies (OTB), TUDelft. This research forms part of a wider research about collection, storage, and visualization of 3D data of which the main partners are OTB -TUDelft and ITC (RGI, 2007).

The thesis explores the possibility to visualize 3D geographical information in the form of a Tetrahedral Irregular Network (TEN), by means of a fusion of Python, database and Extensible 3D (X3D) and other 3D XML based technology. Python is a programming language; a database is used to store data, and X3D or other XML based languages are used to visualize or exchange 3D information on the internet.

A prototype of a web application has to be developed for visualization of 3D geographical information in the form of a TEN that is stored in a database.

A TEN is the 3D variant of the TIN; both are a form of geographical information. TEN is translated as Tetrahedral Irregular Network (Penninga et al [2], 2006) or Tetrahedronized Irregular Network (Penninga et al [1], 2006), depending on the literature picked. In this report, it is always translated as Tetrahedral Irregular Network. A tetrahedron is the simplest geometrical shape in 3D space, a pyramid with a triangular ground face. Its four points can lie anywhere in space as long as they do not lie on one plane. In figure 1.1, a sample tetrahedron is displayed. From multiple tetrahedra ‘GIS objects on the terrain’ and ‘3D terrain’ can be composed: the space can be partitioned in tetrahedra, which form a network together. When only the surface of objects is stored and not the internal tetrahedronization, it is a 3D TIN, which consists of closed surfaces of adjacent triangles. In figure 1.2 a sample of a network of tetrahedra is given.

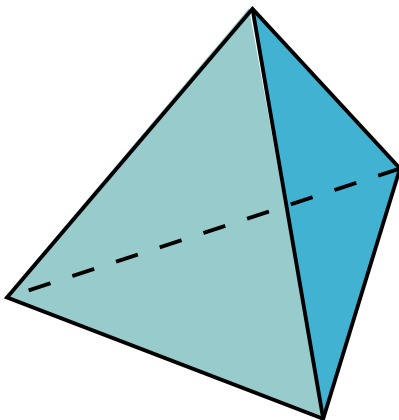


Figure 1.1: A tetrahedron

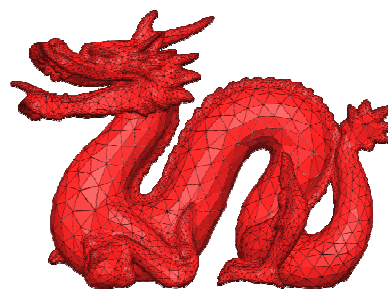


Figure 1.2: A network of tetrahedra (Si, 2007)

The first (C1 Introduction) describes the context and the different goals of the research. The chapter starts with background and objectives of the research. The following parts describe the requirements and strategy.

The second (C2 General 3D Theory) covers the theory of data models about 3D geo-information. First, some general 3D data modeling theory is given about feature components, data representation, and data models. Secondly, the TEN, 3D TIN, and 'TEN view' data models are described. Thirdly, the built-up of a TEN dataset, 3D topology, and database formats are described. Chapter three (C3 Prototype Related Theory) describes the technologies that are used for developing the prototype. First, the technology used for website design is described, followed by a description of XML based languages for 2D and 3D visualization. After, 3D and 2D plug-ins and database software are described. Finally information about the used programming languages is given.

Chapter four (C4 Review of 3D GIS applications on the web) gives an overview of different related research projects, which have been carried out in the past. A comparison of the different prototypes is made on the number of features and on the strong and weak points. A few prototypes are finally picked out and are used to form an example for the development of the prototype of this thesis. Chapter five (C5 Prototype Architecture) describes the architecture for working out the prototype for TEN visualization on the web. In the first paragraph, the preliminary draft of the architecture is given which was made before the prototype was developed. Secondly, the realized design of the prototype is described. Thirdly, the design steps that are taken to develop the prototype are summarized shortly.

The contents of Chapter six (C5 Prototype Components, Datasets and Techniques) is going in the direction of a results chapter. In the first part of Chapter six, the different components of the prototype are described. In the second part, samples are given of the datasets that are used. Finally, the techniques that are used in the prototype are described quite detailed.

The next part of Chapter six contains the most important outcomes of the thesis: the sub questions are answered, an answer is given on the main question, and suggestions are done for future research.

## 1.2 Background of the research

This research forms part of a wider research on 3D topography about collection, storage and visualization of 3D data (van Oosterom et al, 2006). This research is a joint venture of different parties, of which are the most important OTB – TUDelft and ITC (RGI, 2007).

The research is focused on new possibilities for visualization and analysis of 3D GI information on the web. TEN data has not been visualized on the web before, though other types of 3D data have been visualized on the web. For example, building models or Digital Elevation Models (DEMs) have been visualized in a web client but the results are mostly outdated and not online anymore. There are very few prototypes available on the web to visualize any kind of 2.5 or 3D geographical information with a live connection to a geo-database.

The reason to describe GI information in 3D is in the first place: the increase of multiple-space-use, which results in more adjacent and hooked constructions (Penninga, 2006).

In the second place, it is the increase in computer power and development of technology, which allows more and more 3D visualization instead of 2D visualization at reasonable speed.

In the third place, it is the increase in the demand for 3D information. For example, citizens want to check if their house falls into a noise pollution zone of airplane routes. The government wants to see which houses and areas have been affected by a fireworks explosion. An oil company wants to see where oil is located between geological layers and a citizen may want to check what the shortest route is at a combined underground metro and train station.

In the fourth place 3D data collection, for example laser scanning and (digital) aerial photography, is growing. Finally, Geographic Information Systems are moving to the web and 3D.

There are possibilities to display 3D information on the web, however GIS functionality is then missing most of the times, or there is GIS functionality but it can only be viewed in 2D in a web service.

Preceding this research (Wesselingh et al, 2005) have created a prototype of a 3D web GIS where attributes can be read from the database, geometry is not in the database but already available as an X3D file. An ArcIMS interface has been used for the 2D map view. When an object is clicked, the 3D view is activated. Some other prototypes for visualization of 3D data are described in Chapter four.

In comparison with 3D representations, 2.5D representations are good for 3D visualization but lack 3D analysis as volume calculations, sound and smell analysis and optimization of mobile networks. One of the best reasons for storing 3D data in the form of a TEN, is that geographical queries are relatively easy in comparison to other storage structures, also on the web (Stoter et al, 2002) (Stoter et al, 2003 [1]). The tetrahedron primitive heavily reduces complexity of geometrical and topological operations, that is why the tetrahedron primitive is used and no other. For example if the polyhedron or CAD data structure is used, topology would be too complex. If a 3D raster or voxel is used, storage space would be too big (Oosterom et al, 2006).

Finally, the prototype made for this thesis at URL (<http://www.3dwebgis.nl>) may be used to gain wider support for the TEN data structure and prosper its development.

### 1.3 Objectives of the research

**The goal of the research is to develop a prototype of a web application that shows an interactive 3D live visualization of 3D geographical information in the form of TEN data from a database via a web service in a web client.**

In a 'live' visualization, when something changes in the database, the visualization is directly updated (after a refresh of the browser). Vice versa in the visualization can be interacted with the database, for example an object can be identified by clicking on it. A web service takes care of the communication between server and client. For example, the 3D visualization is prepared on the server, and returned to the client by the web service. The 3D information can finally be visualized in the web client possibly with a 3D plugin. A preliminary draft of the architecture of the web application is illustrated in the figure 1.3.

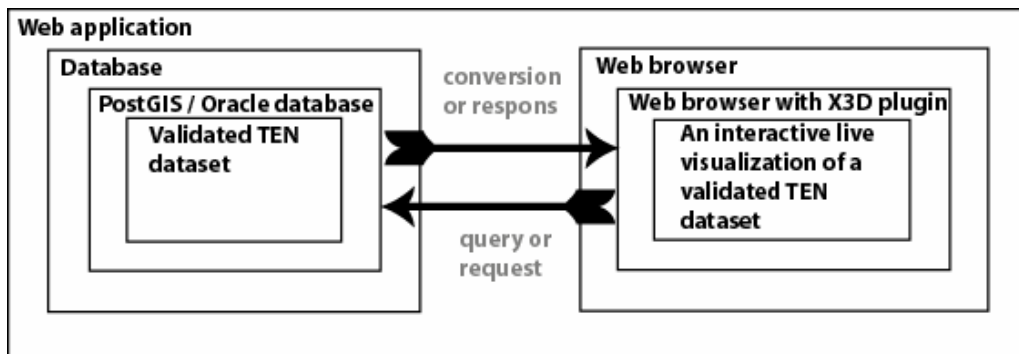


Figure 1.3: Design architecture

**The main question is: How is a TEN dataset visualized in a web client, and what options are there to enhance the visualization and to add GIS functionality?**

The sub questions answered in this thesis are:

- 1 What architecture is used to store and visualize the data?
- 2 In which output formats is the data visualized on the client?
- 3 How can a connection be made between the database and the web client?
- 4 What options do X3D and other XML based languages offer for visualization of TEN geo-information?
- 5 How is data prepared for visualization?
- 6 How a dataset is designed that exploits all the possibilities of a TEN in comparison to a dataset in another data model?
- 7 How can interaction be established between a visualization of the 3D model on the client and storage of the 3D model in the database on the server?
- 8 Is it quicker to visualize a TEN, 3D TIN or 'TEN view' model, and what visualization looks better?
- 9 How is dealt with the fully partitioned space so for example how are air tetrahedra hidden?
- 10 How to identify a feature or how to query attribute information from the data, how to make a selection on the data based on attribute information. For example, a query on the landuse can be executed: "select all objects where landuse = 'grass'."
- 11 How are colour and transparency added to the data visualization?

## 1.4 Requirements for the prototype development

In addition to the sub questions, from interviews with the supervisor, some GIMA students, and people that are in some way (or have been) busy with 3D web GIS some additional requirements for the prototype have been derived. The results of the interview can be found under the help at the website of the prototype (<http://www.3dwebgis.nl>). These requirements are:

- 1 To be able to display features by making a request to a Web Feature Service (WFS), which returns features in the Geography Markup Language (GML), which subsequently can be transformed to X3D using a stylesheet (implemented for the small dataset).
- 2 To be able to output data in different formats, so not only in X3D, but also in the Keyhole Markup Language for Google Earth (KML), and in the Geography Markup Language for city models (CityGML) (done).
- 3 When a selection is made on the data, only the selected part of the dataset should be adjusted, so the geometry of the selected part has to be coloured yellow, instead of generating the whole visualization file again (done).
- 4 For large datasets, the data should be streamed. The data is displayed to the user bit by bit by inserting the geometry in the 3D file, instead of returning the whole dataset at once and to the client (not possible).
- 5 Navigation should be easy, and tools must be available in the toolbox (help file added for navigation, some tools in toolbox).
- 6 There should be explanation about the purpose of the application, and the control of the application in the interface (help file added).
- 7 The interface should have a professional look with not too many colours (done).
- 8 The application should be working in multiple browsers; at least it should function in Firefox and Internet Explorer (done).

All requirements were implemented except for streaming. This is motivated in the subparagraph streaming under the non-implemented techniques (subparagraph 6.5.7).



## 1.5 Research Methodology

A strategy was chosen to go through the thesis process. The detailed plan of the strategy as described in the project proposal formed the basis for writing this thesis, especially for writing Chapter five (Prototype components, datasets, and techniques). The global research steps are:

- Literature study and interviews
- Project proposal
- Collect data
  - Getting or designing a suitable validated TEN dataset (a small and a big test dataset have been provided).
- Practical work and note down results
  - Getting User Requirements.
  - Implementing and testing a XML based structure whereby the dataset can be visualized in a web client.
  - Implementing and testing a structure in the database whereby TEN and or 3D TIN data can be stored (the structure for storing TEN data provided by Penninga has been used).
  - Creating and implementing connections or conversions that are required to come from storage in a database to visualization.
  - Developing the web application, whereby there can be interaction with the database and the visualization can be updated live, meaning it is directly refreshed from the database.
  - Insert the TEN model into the database (dataset already inserted in the database by Penninga)
  - Displaying visualization and testing the application
  - Feedback
  - Making final prototype.
- Writing thesis with results, evaluation, and discussion

## ***2. Data models for 3D geo-information***

### **2.1 Introduction**

This chapter describes general theory about 3D data modeling. The first part gives some information about 3D data modeling in general. The 3D feature components are described in paragraph 2.2. In paragraph 2.3 and 2.4 the 3D data representations and the 3D data models are given. Then in paragraph 2.5 the TEN, 3D TIN, and ‘TEN view’ data model are explained in detail and a comparison is made. Paragraph 2.6 gives information how to build up a TEN dataset from scratch. Paragraph 2.7 explains the concepts of 3D topology and paragraph 2.8 list different datatypes or data formats to store 3D data in a database and the 3D data formats are compared.

### **2.2 Feature components**

#### **2.2.1 Introduction**

In this paragraph, the feature components of generic vector 3D data models are described. The most common 3D features are described, which are a node (subparagraph 2.2.2), an edge (subparagraph 2.2.3), a face or FaceSet (subparagraph 2.2.4) and a body (subparagraph 2.2.5). A node is point in space. An edge is a line composed of one or more line segments. A face is a polygon composed of a number of nodes; a face may be a triangle. A body is a polyhedron composed of a number of polygons glued together forming a closed surface, or it may be a volume object composed out of a number of tetrahedra.

#### **2.2.2 Node**

Though in the Extensible Markup Language (XML), a node may be a group tag and all the tags placed under it, within a 3D data model a node is intended as a point or a vertex in the space. A node forms a single point in space, and endpoint of an arc (line) or a connection point of two or more arcs. In the TEN data model (Penninga et al, 2005) (see paragraph 2.5), the node cannot be a single entity in space. It forms a part of the TEN itself, for example, one tetrahedron consists of four nodes. In the TEN data model, LineFeatures with a different ID are divided by PointFeatures, which consist of one node. If a node shaped feature has to be modeled in a TEN, for example to indicate a well, it might be using relatively small tetrahedra or a different datamodel has to be used which allows point features as they are not in Penninga’s current datamodel (as displayed in figure 2.5).

In the TEN geometry tables, nodes are stored in a separate table with a node ID, x, y and z coordinate. This table is loaded entirely when the 3D visualization is calculated or the nodes are first joined with the face table, in this case, only the nodes are loaded which belong to the faces the user has selected. X3D offers the possibility to visualize nodes or points using the <PointSet> tag. In this way point clouds can be visualized to get a vague idea of what the object looks like, or points can be added to the polygon view to emphasize the visualization of the object. It is easier to use the X3D viewer BsContact that has an option to visualize face data as a point cloud (by only visualizing the connection points).

### 2.2.3 Edge

An edge or line usually forms the border of an object and is composed out of one or more arcs or line segments. In the TEN data model (Penninga et al, 2005) (see paragraph 2.5) also the edge cannot be a single entity in space. It forms part of the TEN itself, a tetrahedron consists of four triangles, and a triangle consists of three edges. In addition, AreaFeatures with a different ID are divided by LineFeatures, which consist of one or more arcs. If an edge shaped feature has to be modeled, for example on a certain scale a road is represented by a line, the road should still be considered as a volume for example a block of 10 kilometers long, 10 meters width and 10 cm thick which is divided in tetrahedra or a different data model has to be used which does allow line features.

In the TEN database implementation edges are not explicitly stored; the same is valid for triangles. A triangle table can be derived from the tetrahedron and node table; subsequently an edge table can be created of the derived triangle table and the node table. Edges can be used to show the wireframe structure of a 3D model, though the X3D viewer BsContact offers an easier solution, it has an option to show only the edges of a 3D model.

### 2.2.4 Face / Faceset

In geometry, a face of a polyhedron is any of the polygons that make up its boundaries. Three or more nodes bound a face.

A face is convex if for every pair of points within the object every point on the straight-line segment that joins them is also within the object, and each internal angle is at most 180 degrees. Anything that is hollow or has a dent in it is not convex. A face is strictly convex if every internal angle is less than 180 degrees. Equally, a polygon is strictly convex if every line segment between two vertices of the polygon is interior to the polygon except at its endpoints. A triangle is always strictly convex. If at least one of the interior angles  $> 180$  degrees or one of the joined line segments falls outside the object, a face is concave. Figure 2.1 gives a sample of a convex and a concave face.

**Convex (all angles  $< 180$ , all lines within face)    Concave (one interior angle  $> 180$ , three lines outside face)**

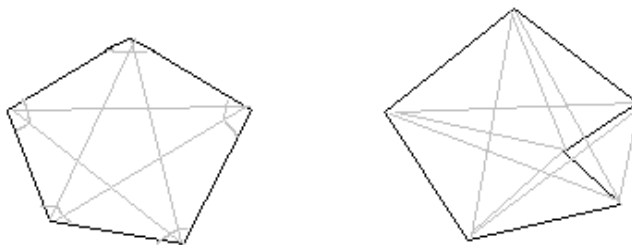


Figure 2.1: Convex and concave faces.

A face is planar if all nodes of the face lie within one flat plane. A face is not planar, if not all nodes of the face lie on one (flat) plane. Figure 2.2 displays a sample of a planar and a non-planar face.

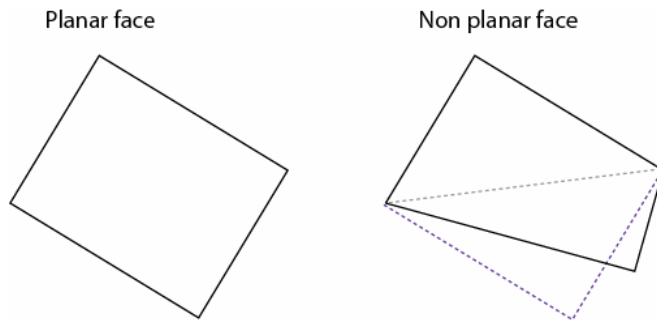


Figure 2.2: Planar and non-planar face

A face is described by listing its coordinates in anti-clockwise direction from the viewer's side. For the backside of a polygon (interior), the coordinates are listed in clockwise direction

A linear ring is in fact the same as a face. It is defined as a list of coordinates, where in between linear interpolation is applied. However, the first coordinate is now listed again at the end of the linear ring. Again the coordinates are listed anti-clockwise for the outside of the faces (outer-rings), and clockwise for the back of the faces (inner rings).

An (Indexed) Faceset is a composition of one or more faces. In the prototype made for this thesis, all the faces that belong to one object (with a unique ID) are listed in one IndexedFaceSet.

In the TEN data model (paragraph 2.5) only triangular faces are used. Triangular faces are always planar and convex, which makes creating valid objects and analysis easier. AreaFeatures in the TEN data model consisting of multiple triangles can be modeled using an IndexedFaceSet. In addition the IndexedFaceSet can be used to model the skin of objects in the case of a 3D TIN, to model the all the triangles of the internal tetrahedrization and the triangles on the surface of an object in the case of a TEN or to model surfaces of constrained triangles in the case of the 'TEN view'. The TEN will require most faces, while the 'TEN view' requires the least faces.

To come from a table with points and a table with faces and point references to an IndexedFaceSet, a transformation is used. This transformation is described in subparagraph 6.4.3.

### 2.2.5 Body / polyhedron

A body is a volume object. It can exist as a set of faces enclosing a volume in the boundary representation, or as set of glued tetrahedra in the TEN representation (paragraph 2.5). A polyhedron is a body in the boundary representation (paragraph 2.3). The surface of a body should be closed. Within X3D and CityGML bodies cannot be explicitly instead lists of IndexedFaceSets or lists of linear rings should be used. In GML 3.1 bodies can be topologically as a volume object with references to faces It is however not common practice to model this way in CityGML, which is a profile on GML 3. CityGML by default makes use of geometrical modeling using linear rings (Verbree et al, 2005). An example of topological modeling in GML 3.1 is given in Appendix 2.1.

The TEN and the 'TEN view' (paragraph 2.5) are not very appropriate to be modeled as bodies. The 3D TIN is appropriate as it objects form closed surfaces. To store the 3D TIN topological in GML 3.1 using bodies would result in a compact file just as the IndexedFaceSet for X3D does. It has not been tested though because the 3D visualization is focused on X3D, KML and CityGML. A motivation for these choices can be found in 3.3.2.

In addition 3D TIN data is not stored in the database currently, data is only stored as TEN and the constrained triangles are derived in the 'TEN view'. The constrained triangles mark the boundary between objects in the 'TEN view' and are visible to the user. For example, the triangles between buildings and air or between the soil and the air are present in the view.

## 2.3 Data representations

For modelling 3D objects, several 3D abstractions are possible, these abstractions are called the 3D data representations (Stoter et al, 2003 [1]) or 3D modelling methods (Penninga, 2006) depending on the author who classifies them.

The most important 3D data representations mentioned are Voxels, Constructive Solid Geometry (CSG), and the Boundary representation (BREP):

- A voxel is a repeating 3D volume element. 3D data can be stored as voxel representation. Drawbacks are the rough surfaces and storage space. For example, a 3D raster is implemented in the GRASS GIS (Grass development team, 2007). The Geological Survey of the Netherlands – TNO Built Environment and Geosciences is carrying out research in storing 3D data as voxels.
- With Constructive Solid Geometry (CSG), shapes are modeled as a combination of solid 3D objects. It is hard to model this kind of geometry and topology in a GIS and CSG object do not fit in the current 2D geometry data model of the Open Geospatial Consortium (OGC). CSG are not considered for the near future. CSG is more design oriented.
- In the Boundary representation bounding elements as point, line, polygon, and polyhedra represent the 3D object. A set of polygons enclose a single volume, meaning it should be watertight or closed. Advantage to this approach is that it takes less storage space to model objects; another advantage is that the representation is one of the best simulations of reality; a disadvantage is that constraints on topology and geometry can get very complex. A 3D boundary representation can act as the input for TEN creation (subparagraph 2.5 describes the TEN data model). First, the boundaries are divided into a set of non-overlapping triangles; secondly, the internal tetrahedronization of the model is calculated (Verbree et al, 2005).

The TEN is considered as a data model not a data representation. In the world of 2D geographical information, data can be stored in a vector representation or a raster representation (the voxel representation may be a 3D raster). The 3D data models described in paragraph 2.4 are all topological data models in the vector representation and some are in the Boundary representation. A deeper discussion of classifications and explanation of terms on models and representation is however not considered important to this thesis.

## 2.4 Data models

The 3D data models are neatly described by (Verbree et al, 2005). Existing 3D data often is supplied within one of the data models and existing tetrahedronization software. TetGen and Netgen can input some of these data models (Si, 2007) (Schöberl, 2007). As stated in paragraph 2.3 all these data models are topological and in the vector representation.

For each data model a short explanation is given:

- In the Urban Data Model (UDM), geometry is represented by planar convex faces, which are divided into triangular faces. Nodes exist as connection points, which connect the faces with themselves and other faces. Loose arcs or edges are not permitted.
- In the Simplified Spatial Model (SSM), a node exists as connection point, which connects the faces with themselves and other faces. In the model, faces are stored with reference to nodes and bodies are stored with reference to faces.
- In the Formal Data Structure and (3D FDS) four elementary objects are defined which are point, line, surface and body, and four primitives which are node, arc, face and edge. The Piecewise Linear Complex (PLC) is related to the 3D FDS and consists of a set of vertices, segments, and facets where each facet is a polygonal region possibly with holes, segments (which are one or more arcs) and vertices (or nodes) in it.
- In the Simplified Spatial Structure (SSS), there are again the four basic objects (point, line, surface, and body), but there are only two primitives: node and face. A 3D primitive is not maintained, and convex faces represent the 3D object.
- In the Tetrahedral Irregular Network (TEN) the world is represented by adjacent tetrahedrons, this includes soil and even air. There are four primitives, which are tetrahedron, triangle, arc, and node. The surface of the object is in fact the boundary representation, and called a 3D Triangular Irregular Network (3D-TIN). More detail can be found in paragraph 2.5.

(Verbree et al, 2005) consider the UDM and TEN as the best model choices as they always have a valid boundary (watertight) and the faces are triangular, which means they are always planar and convex which makes analysis relatively easy. The UDM is a kind of a 3D TIN; a TEN can be derived from the UDM. Thus, the TEN seems the best choice for a 3D web GIS prototype, which requires analysis and visualization. However if the pure purpose of a model is a visualization the SSS is a good choice too. XML based visualization languages make use of the node and face and in much lesser degree of volume objects. Graphics cards in computers always render objects as a set of faces, which are divided into triangles (Allen, 2007).

## **2.5 TEN, 3D TIN and 'TEN view' data model**

### **2.5.1 Introduction**

In this paragraph an explanation about the tetrahedron and the Tetrahedral Irregular Network (TEN) is given. Before visualization the TEN data has to be stored in some way. There are different data models to store the 3D data based on the TEN data model. The TEN based data models described, have in common that the nodes are stored with a node ID and an x; y and z coordinate and the geometrical features (for example tetrahedra or triangles) consist of a reference to a number of nodes (three four triangles or four tetrahedra) and an object ID.

In the TEN data model all the TEN data is stored (subparagraph 2.5.3) while in the 3D-TIN (subparagraph 2.5.4) and the 'TEN view' data model (subparagraph 2.5.5), a certain selection of the data of the TEN data model is derived. In the 3D-TIN data model, only the surface triangles of each object are stored and in the 'TEN view', only the constrained triangles are derived from the TEN data model. The constrained triangles mark the boundary between objects.

### 2.5.2 TEN data model

A TEN is a Tetrahedral Irregular Network. It is the 3D variant of the Triangulated Irregular Network (TIN). A TEN consists of multiple tetrahedra and the 3D space is fully partitioned (figure 2.4). The volume objects in the TEN consist of tetrahedra. The tetrahedra are formed from nodes, edges, and triangles.

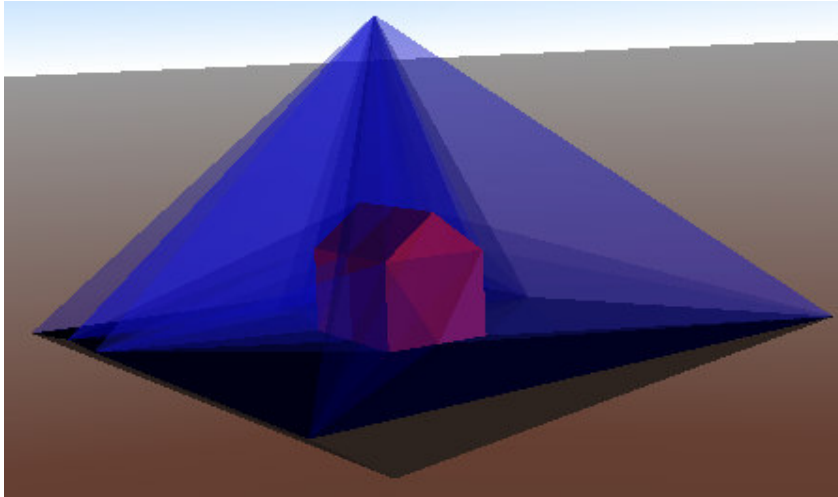


Figure 2.3: Sample of a fully partitioned TEN

(Penninga et al, 2006) have drawn three identical data models for storing TEN data in the Unified Modeling Language (UML). These data models have many similarities, though there are some differences too. The main differences between the data models are, which associations are explicit, and which are derived, and in the case of signed association references (giving the orientation of a face), whether these are modeled with an association class or with an additional undirected primitive. In the TEN data models drawn for the 3D topography project carried out at OTB, TUDelft (Oosterom et al, 2006), an assumption has been made that only volume features exist. Area, line, or point features can only exist as co-boundary of their aggregate, this means as a boundary between their higher dimensional primitive. However, an actual TEN implementation might also support true point, line and area features, which might be good for representing features at smaller scales that are in reality also volumes. For example, a piece of grassland or a road can be represented by an area feature, while a well can be represented by a point feature.

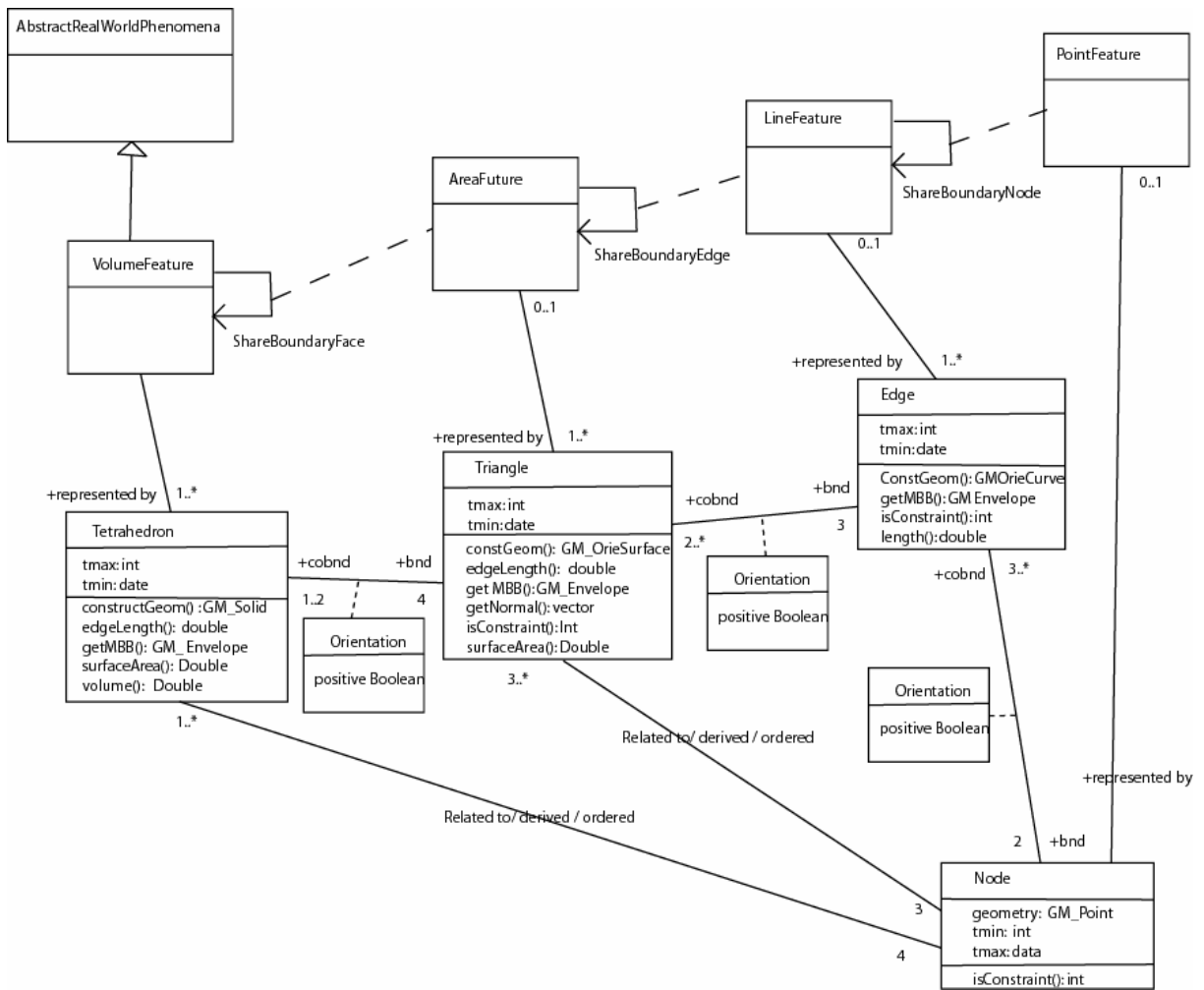


Figure 2.4: UML TEN model (1<sup>st</sup> variant), taken over from (Penninga, 2006)

The first variant of the TEN data model by Penninga et al (figure 2.5) is discussed. First, one to many tetrahedra form a volume feature. Real world phenomena are an aggregate of one or more volume features. A complex hierarchy could describe the relationship between Real World Phenomena as in the XML based language for describing city building models: CityGML. For example, a house is made rooms, a room is made of walls, a wall has doors and windows etc. A more simple hierarchy is present in X3D, where a group is composed of a number of IndexedFaceSets.

As can be seen in the figure each tetrahedron will consist of four triangles and four nodes. A triangle can form the boundary of one tetrahedron or the co-boundary or shared boundary of two tetrahedra. A triangle consists of three nodes and three edges. An edge can form the co-boundary of two to many triangles, because only volume features exist. Each edge consists of two nodes. A node forms the co-boundary of three to many edges. An AreaFeature consists of one to many triangles. An AreaFeature can form the boundary of a volume feature or the co-boundary or shared boundary of two different volume features. A LineFeature consists of one to many edges. A LineFeature can form the boundary of an AreaFeature or the co-boundary of two or more different AreaFeatures. A node feature forms a PointFeature. A PointFeature can be the boundary of a LineFeature or the co-boundary of two or more different LineFeatures.



A tetrahedron has functions to construct its geometrical solid, to calculate the length of its edges, to get its geometrical envelope, to calculate its surface area and its volume. A triangle has functions to construct its geometrical surface, to calculate its edge length, to get its geometrical envelope, to check if the triangle is constrained which means that it stays in the same place when a volume is retetrahedronized, and to calculate its surface area. An edge has functions to construct its geometry, to get its envelope, to check if it is constrained, and to calculate its length. A node has only a function to check if it is constrained. The tetrahedron, triangle, and edge, have a property tmin and tmax to indicate where they exist in time. Initially they do not contain geometry; only the node has a geometry property, which is x, y, and z coordinate. The geometry for the other primitives can be derived from the topology and node geometry.

In the (first) TEN implementation as described in this paragraph, orientation is indicated between edge and node, between triangle and edge, and between tetrahedron and triangle. An efficient implementation of this data model will not explicitly include the association class Orientation, but will use signed (+/-) references to encode the orientation. For example, the inner part of the face is modeled with a negative sign, and the outer side of the face is modeled with a positive sign.

There is no database yet that supports TEN types. Within Penninga's current implementation of TEN, data are stored in native data types in the Oracle database. This means x, y and z coordinates are stored in separate numeric columns, node references and ID for each triangle are concatenated in one column for example like this 050035032014001 (Id is 1, points 50, 35,32 and 14). The pros for the solutions must be sought in access and compactness. The cons of this solution are that each time a triangle is visualized its string has to be unpacked. Another solution would be to store the point IDs in separate columns. A coordinate could be stored in one column as a 3D point. Another solution is to store the data in the form of Simple Feature types (Open Geospatial Consortium, 2007 [2]), namely 3D polygons, which are supported by the latest versions of Oracle. Finally, the closed surface of the 3D TIN data could be stored in the polyhedron data type, which is supported by Oracle 11. In paragraph 2.8 a wider explanation of 3D database formats is given. A choice can be made to visualize a TEN directly; a drawback is that a lot of triangles are stored within the 3D visualization that cannot be seen by the user. This is the result of internal tetrahedronization, which leads to the storage of triangles in the objects. To overcome this, it is recommended to select the 3D TIN or 'TEN view', as described in the next paragraphs. Some tests have been done which goes faster visualizing the TEN directly, or visualizing it in another format. Visualization of the 'TEN view' without air tetrahedra gives the fastest result. Timed results of the visualization of a TEN and 'TEN view' can be found in Appendix 2.2.

### 2.5.3 3D TIN data model

The 3D TIN data model is the 'skin' of the TEN data model. Figure 2.7 displays the UML data model of a 3D TIN. A 3D TIN holds the triangulated surface of objects in 3D, only nodes and triangles on the surface are stored in the database implementation of the model, the other features can be derived from them. As in the TEN data model, an AbstractRealWorldPhenomena (an object for example a house) is formed by an aggregation of volume features. Each volume consists of four to many triangles. Each triangle consists of three arcs and three nodes. An arc forms the co-boundary of two to many triangles, only more than two if the arc forms the co-boundary of more than two objects. The surfaces should enclose a volume. An edge consists of two nodes, while a node forms the co-boundary of three to many edges. Again, an AreaFeature consists of one to many triangles. An AreaFeature can mark the boundary of a VolumeFeature, or mark the co-boundary between two VolumeFeatures. A LineFeature consist of one to many edges and marks the co-boundary of one to many AreaFeatures, while a PointFeature is formed by one node and forms the co-boundary of one to many LineFeatures. The difference with a "normal" 2.5D TIN is that one x; y coordinate can have multiple z-coordinates.

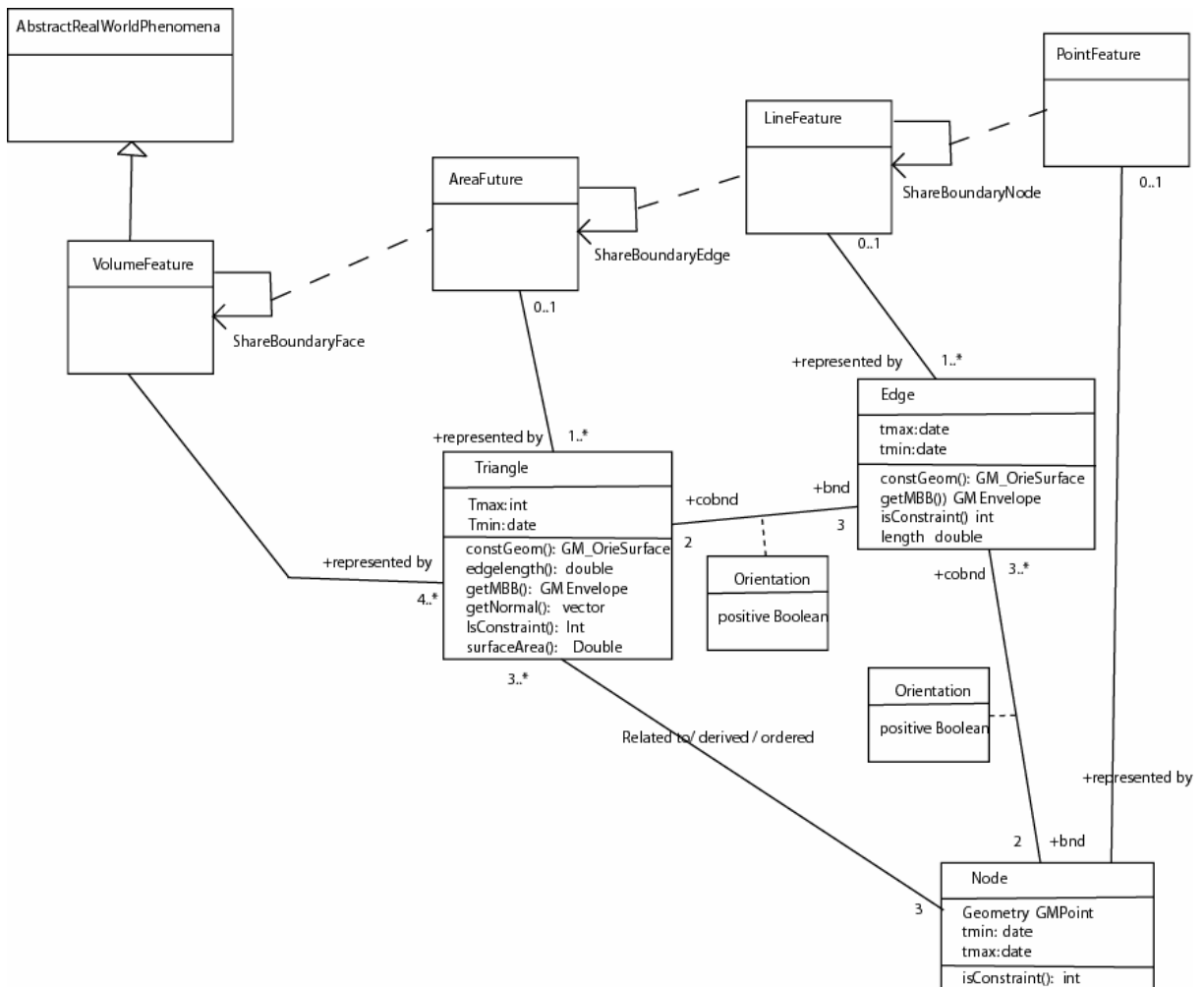


Figure 2.5: UML 3D TIN model adapted from TEN model (1<sup>st</sup> variant)

A significant difference between the TEN and 3D TIN data model is that in the 3D TIN data model each side of a triangle connects to only one other triangle of the same object, while in the TEN data model it can be connected to multiple triangles of the same object. In a 3D TIN, the internal tetrahedronization of objects is not stored in the model unlike in a TEN, so the 3D TIN is more suitable for 3D visualization because it looks better and takes up less space and transfer time.

A 3D TIN is stored /derived in the same way in the database as the TEN using normal data types. The node ID and an x, y and z coordinate are put in separate numeric columns, while the faces with object ID and point references are concatenated in one column. The number of triangles that are in the 3D TIN are 25% - 50% of the number of triangles stored in the TEN.

#### 2.5.4 TEN view

A ‘TEN view’ (personal interview with Penninga F, 2006) only holds the triangles that are visible to the user, which are the constrained triangles. These triangles mark the boundary between buildings and air and the boundary between soil and air. Note that the view is actually dependent on the location of the viewpoint.

If the viewer would be standing in a house, the triangles which mark the boundary between the house and air, and the boundary between the house and soil, would have to be present in the view. The triangles can be connected, but do not have to be. A ‘TEN view’ is very suitable for visualization, because only the items that can be seen are visualized. The triangles in the ‘TEN view’ are also called the constrained triangles. A group of triangles with the same ID can form VolumeFeatures or AreaFeatures.

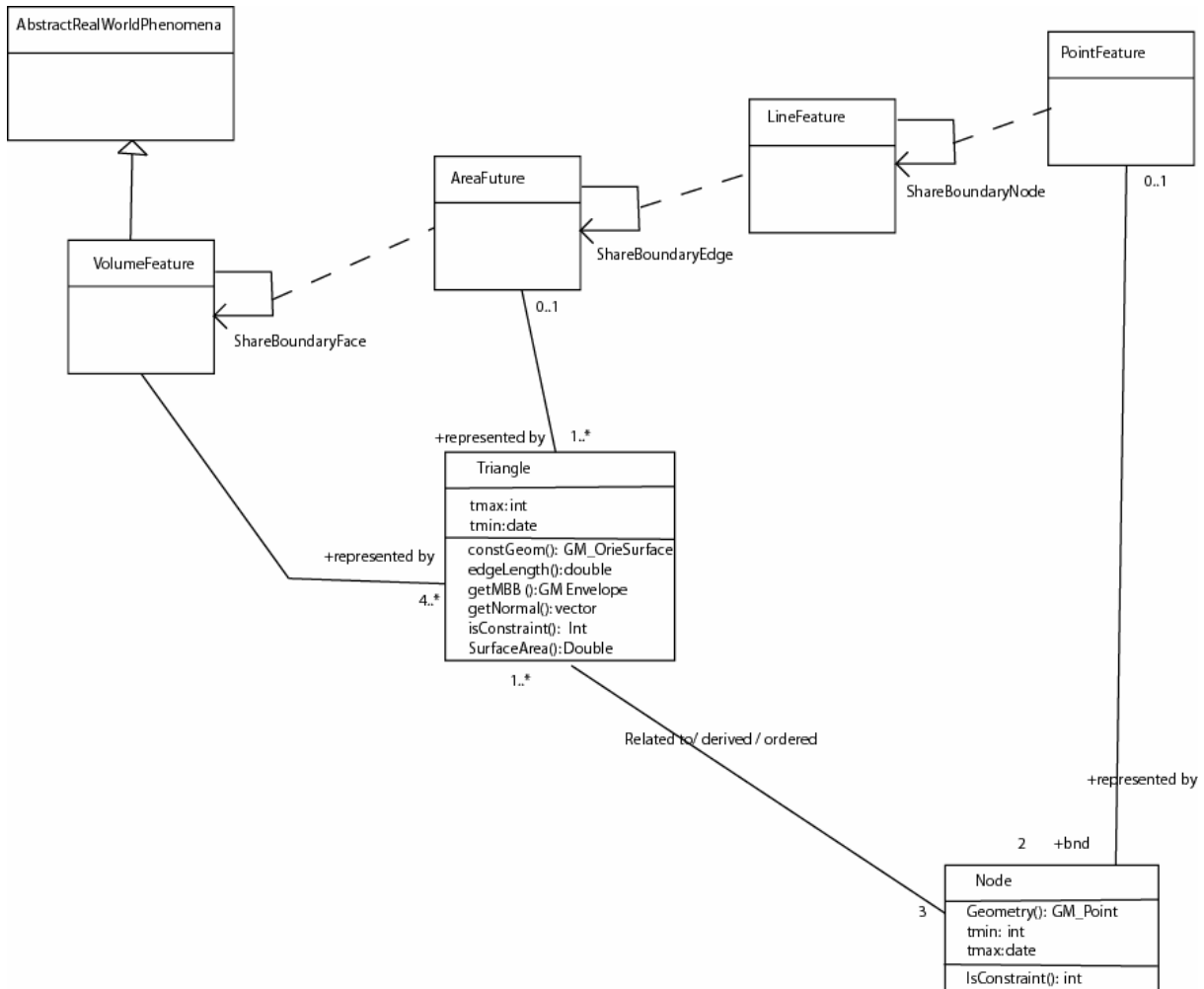


Figure 2.6: UML ‘TEN view’ model adapted from TEN model (1<sup>st</sup> variant)

Figure 2.7 displays the UML data model of the ‘TEN view’. In this data model, each volume feature will exist of four to many triangles. Each area feature exists of one to many triangles. A triangle feature marks the boundary of one, or the co-boundary of two volume features. Each triangle consists of three nodes, while each node can form the co-boundary of one two many triangles. Arcs are not present in this data model because they are not required for visualization and they still could be derived from triangles and nodes.

A ‘TEN view’ is in its original makeup a derived table or ‘view’ of the TEN table. However, access to a “view” is very slow with (Py)ODBC (paragraph 3.4). Therefore, a copy of the view table is stored with the physical tables. Again, data is stored with native data types (numeric, textual, not spatial). A table with points and coordinates with numeric columns for the x, y and z coordinates, and a table with triangles with reference to the points are used. In the latter, the triangle ID and the references to the three points of each triangle are concatenated in one column.

In the 'TEN view', there are the triangles that mark the boundary between buildings and air, air and the soil, and soil and buildings. It is thus not required to hide any of the triangles for visualization, as no volume objects block identification to other objects, and the minimal number of triangles required for visualization is used already (there is an exception to this rule: the air polygons that mark the boundary of the TEN dataset may have to be hidden). In the TEN and 3D TIN storage, it is better to hide the air objects, as the air objects do block identification of other objects.

In fig 2.9, the differences between the TEN, 3D TIN, and 'TEN view' are illustrated to make it clearer.

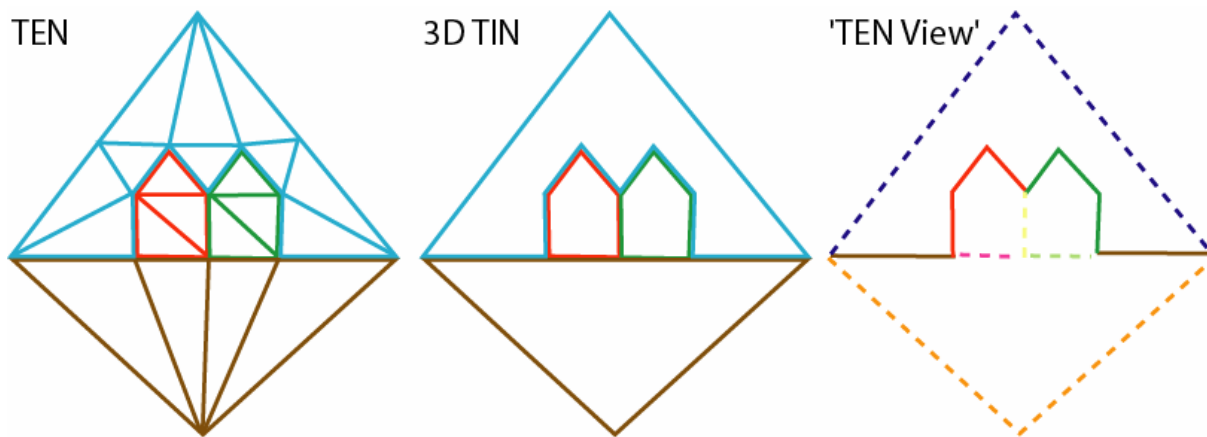


Figure 2.7: Schematical representation of the TEN, 3D-TIN and 'TEN view'

## 2.6 TEN dataset build-up

### 2.6.1 Introduction

In this paragraph first the global steps are described, how to build up a TEN dataset from scratch. Secondly is explained what possibilities there are in the case of edge insertion, in the case a new object is inserted in the TEN, or for example, when a buffer analysis is done. The creation of a TEN dataset falls outside the scope of this thesis. It is however part of the wider research about collection, storage and visualization of 3D data which is the 3D topography project of which the main partners are OTB – TuDelft and ITC (Oosterom et al, 2006) (RGI ,2007).

The process of modeling topographic features in a TEN consists of five steps. First two air and earth tetrahedra are created; secondly, terrain information form a DEM or TEN is inserted. Thirdly, additional (Steiner) points are inserted to create more regular tetrahedra, with less sharper angles. Fourthly, objects are inserted and finally the feature attribute table is updated. In (Penninga, 2005) and (Penninga et al 2006 [1]) the steps are explained in more detail.

For inserting objects in a TEN, constraint edges, edges that cannot be moved or removed, are placed in the dataset. For the insertion of a point in a tetrahedron, four cases can be distinguished. An edge consists of two points. This means for the intersection of an edge with a tetrahedron ten cases can be distinguished. When an edge intersects multiple tetrahedra, the edge is split at the border of the tetrahedra, and the cases are considered for each part. For each of these cases a new fixed number of tetrahedra are created. More details can be found in (Penninga et al, 2006 [1] and Penninga et al, 2006 [2]). The simple shape of the tetrahedron makes the number of cases limited. When modeling with polyhedra there will be a lot more cases, and insertion of data or analysis as buffer and overlay, will become much more complex.

## 2.7 Topology

### 2.7.1 Introduction

While geometry describes the exact location of objects, topology describes spatial relationships between objects and components of objects. 3D topology can be split in internal topology describing the relations between components of an object (subparagraph 2.7.2), and external topology giving the relations between objects (subparagraph 2.7.3). In a topology structure management there is four tables (node, edge, face, and body) and no elements are stored twice avoiding redundancy (subparagraph 2.7.4). The topology for a TEN network has already been developed; the topological support for the polyhedron network will still take time to develop according to (Stoter et al, 2002, Stoter et al, 2003 [1] [2]).

### 2.7.2 Internal topology

Internal topology (Stoter et al, 2002) is most common and describes the relationships between the components of objects. For example, in figure 2.10 tetrahedron I consist of node A, B, C en D, tetrahedron II consist of node B, C, D en F and tetrahedron III consists of node B, C, E and F. In the same manner, the relationships between tetrahedra and arcs and between arcs and nodes can be described.

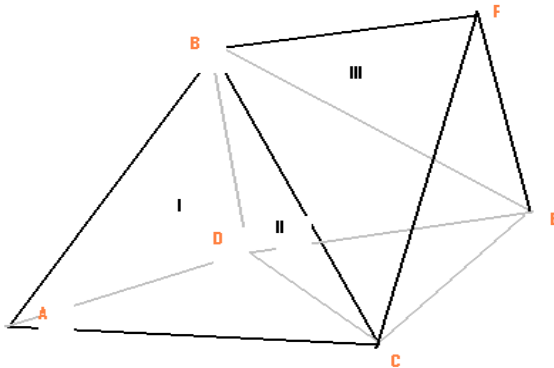


Figure 2.9: Internal topology

TEN data in the databases is stored with the use of internal topology. Tetrahedra are stored with reference to four nodes, triangles have a reference to three nodes, and nodes are stored with an x, y, and z coordinate. The IndexedFaceSet in X3D also makes use of the principle of internal topology. Faces are stored as a list of node references, separated with -1 for every new face, and node coordinates are stored in a separate tag, which follows up the face tag.

### 2.7.3 External topology

External topology (Stoter et al, 2003 [2]) is about the topological relations between objects. In the example of the TEN network with three tetrahedra (figure 2.11) the external topology of the tetrahedra is depicted by the line going through the tetrahedra. The dots represent the midpoints of the tetrahedra, and the slashes mark the boundary faces that connect the tetrahedra.

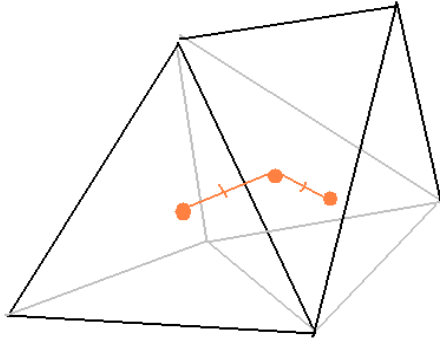


Figure 2.10: External topology

In the TEN data model, external topology can be derived optionally in the database. The result is for example a table with faces, and a reference to one or two of the tetrahedra they belong to.

#### 2.7.4 Topological structure management

In a topology structure management (Stoter et al, 2002) there is four tables (node, edge, face, and body). In a topological structure, no elements are stored twice avoiding redundancy. In the TEN implementation in the Oracle database, each tetrahedron and each node is stored once, so there is a valid topological structure.

However when a 3D TIN or 'TEN view' is derived a triangle can be shared by two different objects, and may be stored twice with a positive and negative orientation. These data models do not fulfill the topological structure.

The topological support of the TEN structure is already developed, as it is relatively easy, compared to other topological structures. This is because the TEN makes only use of one 3D - 2D simplex; the triangle. However, the topological support of the polyhedron primitive will require 6-7 years to develop according to (Stoter et al, 2003 [1] [2]). A difficult question is for example, when do 3D objects touch in the polyhedron model or when they share a common node, edge, or face?

## 2.8 Database formats

There are different ways to store 3D / TEN data in a database.

- First, data can be stored using the spatial Simple Feature Types of the Open Geospatial Consortium (Open Geospatial Consortium, 2007 [2]). The data can then be stored in 2.5D as:
  - Multipolygon: Multiple polygons are stored in one record in the database. Polygons are stored as a list of coordinates. Drawback of the multipolygon there is no relationship between the polygons that define the object (no topology), coordinates are repeated often (for every shared boundary), and validation is impossible (Arens et al 2004). In the case of a TEN, each tetrahedron should be converted to four triangular faces. This means not only there is coordinate repetition but also face repetition as almost any triangle is joined with one other triangle. A 3D TIN and a 'TEN view' can be converted to a multipolygon straight away. In all three data models, a multipolygon is composed of all the faces of one object. A multipolygon is good for visualization as IndexedFaceSet (X3D) and linear rings (GML), because they make use of the same principle

- Multiple polygons: Each object is stored with multiple polygons in the database. For every polygon, a new record is created. The record should have object ID attribute so the different object in the data can be distinguished. There is still no relationships between the polygons (no topology), coordinates are still repeated, though validation of individual polygons is now possible. In the case of a TEN, each tetrahedron is represented by four records in the database, where each of the records contains one triangle. In the case of a 3D TIN or 'TEN view', each triangle is one record in the database.
- Points, Edges and Polygons: In the TEN (and 3D TIN and 'TEN view') data model there are different primitives as a volume object, a tetrahedron, a triangle, and edge and a node. The triangle, edge, and node primitives can be represented by the polygon, edge and point features from the Simple Features. There is no relationship between the features of the same type or between features of different types (thus no topology). There will be coordinate repetition because of neighboring polygons and arcs. The relationships between the triangles, edges, and nodes (topology) could be recorded in the attributes of the record and is thus not automatically recognized by the database.
- As a second option, the data can be stored using 2D / 2.5D spatial topological datatypes introduced in Oracle 10. Polygons are stored with a polygon id and a reference to a number of points. For a TEN implementation, this would mean every tetrahedron is stored as four triangles and the triangles are stored using the topological polygon datatype with a reference to three nodes. The nodes are stored with a node ID and an x, y, and z coordinate (optionally edges could be stored too but this is not useful). For a 3D TIN or a 'TEN view', the triangles of an object are stored as a polygon with a reference to the nodes. As most triangles of one tetrahedron join another triangle, there will a lot of redundant storage especially in the case of a TEN. As there is no volume data type, topology between the volumes and faces should still be recorded in the records. A set of geometrical and topological operators or functions have to be developed, to employ the specific advantages of storing the data in the TEN interior or 3D TIN boundary approaches (Verbree et al, 2005).
- As a third option the data can be stored, using 3D spatial topological datatypes. These have not been implemented in Oracle or another database; however, GML 3 supports 3D topology. Research into a (polyhedral) 3D topological data model is still ongoing according to (Stoter et al, 2003 [1] [2]). A 3D topological TEN specific UML data model has been developed (Arens et al, 2004) but has no database implementation.
  - In a 3D topological data representation, bodies are stored with reference to faces and faces with reference to nodes. In a TEN implementation, the tetrahedra could be stored as bodies with reference to four triangles. Triangles are stored as polygons with reference to nodes and nodes are stored with an x, y, and z coordinate. In the attribute data of the tetrahedra still should be recorded which object they belong to or there should be an object which represent a composition of volume objects. A 3D TIN object can be stored as a volume object with a reference to the triangles, which form the volume object. If the 'TEN view' contains watertight volume objects, these can be stored as a volume object. Otherwise area objects could be saved as a topological multipolygon (if it is implemented) which is a multipolygon with node references.
  - Data could also be stored using 3D spatial topological TEN data types. These data types have not been developed for any database. In this datamodel there is a TENvolume object with have a reference to a number of tetrahedra, there will be a TENtetrahedron object with a reference to four triangles, there will be a TENtriangle object with a reference to three nodes, and there will be a TENnode object with an x, y, and z coordinate. The model could be extended with a TENAreaFuture, TENLineFeature, and TENPoinFeature object as can be seen in the UML diagram of the TEN data model (fig 2.5). The advantage would be that the data can be stored very efficient now (there will hardly be any duplication). Disadvantages are that new data types are brought to the GIS market, which people are not ready to use and all the databases have to be adapted.

- As a fourth option, data can be stored as the 3D spatial polyhedron datatype. This datatype has first been tested in Oracle 9 and is implemented definite in Oracle 11. Object are now stored as a list of multipolygons which should form a watertight volume. In the 3D TIN data model, polyhedra can represent the 3D TIN objects. In the case of a TEN or 'TEN view' a 3D topological multipolygon (as described in the previous bullet) could be used. Because different polyhedra share boundaries, there will be some coordinate repetition in this model. Other topology than relationship between tetrahedra is hard to derive. Validation of the polyhedron is possible. An elegant way for a database and visualization setup would be to store the 3D objects in the DBMS using the polyhedron type, and to visualize them as IndexedFaceSet (subparagraph 2.2.3) (Arens et al, 2004). A topological 3D multipolygon could be derived from the AreaFeatures (features that mark the boundary of or between objects) (subparagraph 2.5.3) too which would results in a compact way of storing.
- As a fifth option XML tags could be stored in the database. For example in the case of a TEN for every object an IndexedFaceSet is derived. The IndexedFaceSet in then stored in the database. This can be done using the Oracle XML Developer Kit (XDK) Oracle, 2007 [2]) or the Procedural Language / Standard Query Language (PL/SQL) for Oracle (The Oracle FAQ, 2007). Further test research has to be done which is quicker: visualizing XML straight from the database or transforming geometry and topology from the database to XML and visualize it. It has to be measured how much extra time and storage space it is going to cost to derive XML or face geometry in the database.
- As a last option, data can be stored in the database with numeric and textual datatypes. Spatial datatypes are not used at all. In the current TEN implementation of Penninga, spatial data types are not used at all. Numeric data types are used to represent the node coordinates (x, y, and z coordinate). The tetrahedra with reference to four nodes are recorded in a string, or in the case of a 'TEN view' or 'TEN', triangles are recorded with reference to three nodes in a string. The triangles are derived from the tetrahedra in a view. Advantages of this approach are that access and manipulation of these kinds of data types is easy. Tetrahedrons and nodes are only stored physically, triangles are derived from them. In this approach, there is no data repetition so it is very compact. Drawbacks are it is hard to validate data, topology is not recognized by the database, and access to native data types is slower than access to spatial data types. Spatial analysis as the "buffer" operation are easier to made on spatial data types than on native data types



		Model designed?	Implemented in database?	Topology?	Validation?
OGC 2.5D Simple Feature Types					
	multipolygon	TRUE	TRUE	FALSE	FALSE
	multiple polygons	TRUE	TRUE	FALSE	TRUE
	point, line, polygon	TRUE	TRUE	FALSE	TRUE
2 / 2.5D Topological datatypes		TRUE	TRUE	TRUE	TRUE
3D Topological datatypes					
	3D Topological datatypes	FALSE	FALSE	TRUE	TRUE
	TEN datatypes	TRUE	FALSE	TRUE	TRUE
Polyhedron datatype		TRUE	TRUE	FALSE	TRUE
XML datatype		TRUE	TRUE	FALSE	FALSE
Numeric and textual datatypes		TRUE	TRUE	FALSE	FALSE

TRUE
FALSE

Table 2.1: Comparison of datatypes

In table 2.1 the different data types that have been mentioned are compared on a number of points. These are:

- Model designed: Has a datamodel been developed for the datatype?
- Implemented in database?: Has the datatype been implemented in a database
- Topology: Does the datatype support topology / Can topology easy be derived.
- Validation: It is easy to validate the data.

For now the 2 / 2.5D Topological datatype seem the best datatype to use to store a TEN (or 3D TIN or 'TEN view') as all points are answered with true. However topology between volumes and faces is not present, in the attribute data should be recorded to which object / volume each face belong too. For the future it might be interesting to implement 3D topological TEN based datatypes in the database. For now has been stucked to the numeric and textual datatypes which are compact and can be accessed easily. Discussion is not extended further as in the development of the prototype, the emphasis has been on developing functionality rather than to do all kinds of research, to see which datamodel performs best etc.

## ***3. Prototype related technology***

### **3.1 Introduction**

This chapter describes the technologies that are used for developing the prototype and in certain cases alternatives for them. Paragraph 3.2 gives information about technology used for website design, the Hypertext Markup Language (HTML). Paragraph 3.3 gives XML based standards for 3D and 2D visualization. These are X3D, (City-) GML and KML for 3D visualization, and SVG for 2D visualization.

Paragraph 3.4 gives standards for database access (ODBC) and for database query SQL.

Paragraph 3.5 gives information about the applications and plugins used. These are BsContact for 3D visualization in X3D, Google Earth for 3D visualization in KML, and LandXplorer for 3D visualization in CityGML. For 2D visualization, the Adobe SVG plugin or the Firefox SVG native implementation is used. Paragraph 3.5.4 gives information about database applications as Oracle and the Oracle Client and gives information about the Python module PyODBC to access databases.

Paragraph 3.5.5 gives information about the server that is used (the Apache HTTP server).

Paragraph 3.6 gives information about the programming languages used, which are Python for scripting, mod\_python to enable Python scripts to be run on the server, and JavaScript for form input and interaction with the 3D and 2D visualization. Finally an explanation about AJAX and Ajax3D technology for dynamical modification of web contents is given.

### **3.2 HTML for website design**

The user interface of the prototype made for this thesis is built with HTML in combination with some plugins and JavaScript.

HTML is the predominant language for the creation of web pages. It provides the means to describe the contents, makeup, and position of text, images, and hyperlinks in a web document. In some degree, HTML can also describe the appearance and semantics of a document. HTML is written in the form of tags, surrounded by < and > signs. HTML is a static language. Interactivity has to be added with other languages as JavaScript (W3 Schools, 2007 [1] and Champion, 2001).

More information on HTML can be found in Appendix 3.1.

A HTML page can be used to layout a GI web application. A page can be split up in different frames. A frame for a 3D visualization, a frame for a 2D visualization, frames for titles and text, a frame for attribute data etc. The HTML page forms the presentation layer in which 3D data that is retrieved from the database is presented to the user on the client side. To enable the visualization of 2D and 3D data in a HTML page, web plugins are used most of the time.

## 3.3 XML based standards for visualization

### 3.3.1 Introduction

In the world of geographic information, XML based standards can be used to exchange and visualize geographical data. This means geographical data is described with nested, descriptive XML tags. 3D data can be described with XML based standards as Extensible 3D (X3D) an XML based 3D format to visualize 3D graphics in a browser, the City Geography Markup Language (CityGML), a XML based profile on the Geography Markup Language (GML 3) to exchange 3D city models, the Keyhole Markup Language (KML) a 3D XML based format to visualize graphics and data in Google Earth and GML 2, an XML based exchange format for 2D / 2.5D geographical data (subparagraph 3.3.2). 2D data can be described with Standard Vector Graphics (SVG), a XML based format to display 2D graphics or animation in a web browser or with 'flat X3D' (subparagraph 3.3.3).

### 3.3.2 Standards for 3D Visualization

#### X3D

The first option is to encode the data in Extensible 3D (X3D) is the ISO/IEC 19775 standard for real-time 3D computer graphics. The semantics of X3D describe an abstract functional behaviour of time-based, interactive 3D, multimedia information. It is the successor to the Virtual Reality Modeling Language (VRML). VRML defines an open file format that integrates 3D graphics and multimedia. In these formats 3D data is stored in a simple hierarchical way, and can directly be visualized in a web browser with a X3D plugin (Web 3D Consortium, 2006 [1] and Web 3D Consortium, 2006 [3]).

The advantages of X3D are that it can handle separate 3D objects, it has a wide range of visualization options for flattening, lightning and Level of Detail (LoD), and scripts or hyperlinks can be attached to objects. This makes interaction with the database possible: by example if an object is clicked, its attributes are shown to the user.

With JavaScript (3.6.4) an X3D file can be altered using the Scene Access Interface (SAI) (Web 3D Consortium, 2006 [2]) without having to reload the whole file. For example, the colour of a building can be changed to yellow if it is selected in a query. Ajax3D also provides this functionality; a part of the 3D scene can be updated by a user request (subparagraph 3.6.5).

People in the field of geo-information have started to use X3D (an in an earlier stage VRML) to visualize 3D geographical data (see subparagraph 4.2.2), and VRML and X3D have been extended with geographical features such as a geo-elevation grid for representing Digital Elevation Models (DEMs) (Web 3D Consortium, 2007 [1]).

X3D features extensions to VRML such as the humanoid animation, Nurbs, GeoVRML, and the ability to be encoded in XML syntax. X3D can be encoded with XML tags or with VRML syntax, coding in XML is most common nowadays (Web 3D Consortium, 2006 [1]).

#### GML 3 and CityGML

The second option is to encode the data in the Geography Markup Language 3 (GML 3). Geographic information is exchanged or stored in a standardized way as Extensible Markup Language (XML) encoding. Descriptive tags and nesting are used to describe the geographic information. GML 3 (and GML 2) are approved by the Open Geospatial Consortium (OGC) to be used as a worldwide standard to describe geographical information. In GML 3, it is possible to represent coverage type data such as remote sensing or distribution functions. In addition, it can describe temporal data, observation data, and dynamic features.

GML 3 also supports a topology model, so that arguments for a network analysis can be specified in GML and the results, a route for example, are returned in GML. In GML 3, it is possible to describe complex 2D and 3D geometries, such as 3D volume features. For 3D data, it means that data can be stored very efficiently. Now only the geometry of the nodes is stored, while arc and faces can be stored as a list of node references, and volumes can be stored as a list of face references (Open Geospatial Consortium [1], 2007). A 3D TIN could be very well stored as GML 3.1 as the objects could be stored as bodies with reference to the faces

The GML formats are more widely used for data exchange than for data visualization or storage. To visualize GML it needs to be transformed with a stylesheet transformer to another format for example KML or X3D although recent software as Aristoteless is able to visualize GML straight from source.

The Web Feature Service (WFS) is a transactional server that will allow database transactions such as insert, delete, or select to be conducted over the web for spatial objects. The GML can be accessed on the web using a WFS; which allows the described kind of transactions. The geometry is then returned on a WFS as a vector representation, or on a Web Map Service (WMS) as an image. Two-dimensional data in the form of GML 2 can be visualized as an image by a Web Map Service or is returned as 2D vector data (in GML 2) by a Web Feature Service. In the last case, it is necessary to transform it to a visualization format as SVG. With 3D data stored / exchanged in GML 3 this is not yet the case (Prins, 2003).

A better option is to describe the data using CityGML, a profile on GML 3. CityGML allows complex relationships between data to be described; for example a city has neighborhoods, a neighborhood has blocks, a block has houses, a house has rooms, a room has doors, windows, and furniture etc, in addition it allows surface appearance as colour to be added to objects which GML does not (Open Geospatial Consortium, 2007 [1] and Kolbe, 2006). CityGML and can be visualized by several applications as LandXplorer and Aristoteless.

CityGML regularly uses a geometrical way of describing objects, so in a boundary representation faces are described with a coordinate list. This makes it easy to manage in a current 3D geodatabase.

Besides semantics, CityGML also adds appearance to objects. As in X3D, now the colour of objects can be described in different component as diffuse, specular, and emissive colour, and the transparency can be described using a value between 0% and 100% (Kolbe, 2006).

CityGML also allows textures to be added to objects. Topology and appearance are optional.

In CityGML, five levels of detail can be specified which are:

- 1 LOD 0, Regional model
- 2 LOD 1, City / site block model
- 3 LOD 2, City / site block model with textures, roof structures
- 4 LOD 3, Detailed architectural city / site model
- 5 LOD 4, Interior model

Another feature of CityGML is the allowance of external reference. An object can point to a record in a database or to another object in the same or another file and reproduce it. Finally, there is some added functionality, which is not explained here in detail as terrain intersection curves, closure surfaces, and Digital Terrain Models with TINs, Grids, 3D Breaklines, and 3D Mass Point. Some of the latter are also available in GML 3 (Kolbe, 2006).

CityGML can directly be generated by the database in the prototype made for this thesis. However, it is returned as XML to the browser because there is not a plugin to visualize it. It can be visualized via a standalone application. There is now the possibility to grab the CityGML from WFS (Aristoteless) but the GML still has to be saved to a local disk to be viewed.

In X3D there are possibilities to adjust a single node without having to reload the whole file (AjaxX3D) (subparagraph 3.6.5). For example, it can be used to colour an object yellow in attribute selection. For CityGML, these kinds of possibilities have not yet been implemented.

The makers of CityGML consider CityGML as the container of the geographic data while languages as KML and X3D are considered the container of graphical features (Kolbe, 2006). They see it this way because GML can hold the data itself, so the attributes and the geometry while KML and X3D only render the geometry. In addition GML is traditionally seen as a data exchange format and not a visualization format. Now CityGML is often used for visualization too (as it can store visualization information as well) and data is stored in a database and translated to CityGML. In addition the KML standard (and in lesser degree the X3D standard) have been and are still extended to include more kinds of geographical data making them more than just a container of graphical elements or a standard just for visualization. CityGML is very well suitable for GIS visualization, and less suitable for some GIS functionality and interaction.

### **KML + KMZ**

The third option is to encode the data in the Keyhole Markup Language (KML). KML is similar to X3D; it can be used to visualize 3D graphics. KML files can be added to Google Earth. KML is an open XML based format to describe 2D and 3D geographic features such as point, lines, polygons, images or ground overlays which are images draped over the earth's terrain, regions and placemarks. Ground overlays are images draped over the earth's terrain. Regions are divisions of an area to decrease the loading time. Placemarks are points indicating features for example a famous landmark, they have a balloon description. As GML or X3D, KML uses a tag-based structure with nested elements and attributes and is based on the XML standard (Google, 2007).

KML can be dynamically generated by applications from a database, and then be opened in Google Earth automatically. This makes the format good for viewing 3D data. As Google Earth is rapidly growing in market share the KML standard may finally become the facto standard, instead of GML3 or X3D. The latter are used by much smaller user groups. Another possibility is that they might be merged in the future.

Besides KML, there is a compressed variant KMZ, which is similar to KML. This format can also be added to Google Earth. This compressed file may hold a KML file and some images. In addition to this, the Collada interchange file format can be used to design buildings with textures and can be imported to Google Earth too. This format is open and XML based too (Google, 2007).

As with CityGML, there is no development ongoing as Ajax KML, where features within the KML can be adjusted without having to reload the whole file again. X3D does allow this kind of functionality (Giger, 2006).

### **GML 2**

GML 2 makes use of 2D Simple Feature types (another OpenGIS standard) such as points, arcs, polygons, and multipolygons (Open Geospatial Consortium, 2007 [2]). As GML 3, GML 2 can be visualized by transforming it to a graphical format as SVG or X3D using a style-sheet transformer (XSLT) or with standalone applications as LandXplorer and Aristoteless, it can be visualized directly.

Drawbacks of GML 2 are that:

- 3D Feature types such as volumes do not yet exist in the standard (Prins, 2003), The latter is not a problem in our case: the TEN data in the Penninga ‘string’ solution polygons are not stored as a 3D type but as a string. The faces are stored as a string and numeric data types are used for the x, y, and z coordinates.
- GML 2 does not support topology.

The advantages of GML 2 are that:

- Compared with GML 3 it has fewer options to model the data. Using GML 3 two different users could model the same dataset in different ways, with GML 2 this is less likely to happen.
- A GML 2 file can be produced by a WFS (Web Feature Service) and then be transformed to X3D, this makes it possible to visualize data without the involvement of Python and a database connection; only JavaScript has to be used. The use of GML 3 by a Web Feature Service is still under development.

An example of a 3D visualization in X3D via GML 2 is displayed by figure 3.1.

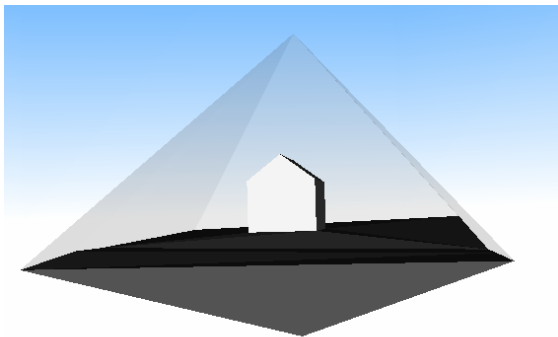


Figure 3.1: 3D visualization of a TET dataset in X3D (transformed from GML 2)

### Other standards for 3D visualization

Finally, a note on standards, which have not been implemented in the prototype created for this thesis:

- XMML is a XML based language for storage of 3D geological data. As it has no viewer it is not being tested (Cox et al, 2006).
- SVG 3D has never been seriously developed; the most advanced 3D visualization is a wireframe visualization of a teapot (Lindsey K., 2006).
- U3D is a binary format (not open) to display 3D industrial (building) models in Acrobat. Because Acrobat does not allow a ‘client-server-setup’ with a database, and the format is not open, it is not tested (3DIF, 2006).

### 3.3.3 Standards for 2D Visualization

#### SVG

A 2D visualization of the data can be done in one of the 3D formats but then in 2D, for example an X3D file without heights, but it is more appropriate to use a XML based format, which is specially devised for visualizing 2D graphics. SVG is such a format. X3D makes use of an index and is much more compact than SVG. However, SVG really gives the idea of a 2D map.

Scalable Vector Graphics (SVG) is a text-based graphics language for describing two dimensional vector graphics and graphical applications in XML with vector shapes, text, and embedded raster graphics. With SVG, visual contents and interactivity can be described through a simple declarative programming model that allows scripting and animation. SVG was introduced as an open standard by the W3C (World Wide Web Consortium) in 1999 for publishing animations, and for interactive applications using vector graphics on the web. SVG is now used in mobile phones, web applications, and Geographic Information Systems (Adobe, 2007).

SVG data can be produced from a database, and is returned to the client. SVG can dynamically be changed with JavaScript; for example an object can be coloured yellow if it is selected, without having to reload the whole SVG file (Carto, 2007).

### **Other methods for 2D visualization**

There are some other options for 2D visualization, which have not been tested.

- A Web Map Service (WMS) reads data in a database or shapefile or GML data and translates it to a raster image, which is returned in the browser (or standalone GIS client). On the site of Map West Virginia Portal, some sample WMS can be found under the tab Web Map Services (Map West Virginia Portal, 2007).
- A Web Feature Service (WFS) reads data in a database or shapefile or GML data and translates it to a vector representation in GML (or HTML) which is returned to the user. The GML could for example be translated to SVG, which is a format to render 2D graphics in a web client, or could be rendered as a vector file in a standalone GIS client.
- A KML file could be visualized in a Google Maps interface in 2D.
- A 2D map of the data can be generated by the UMN Map Server as described in subparagraph 4.2.2 in the prototype of (Ninsawat et al, 2006). Based on the principles of WMS and WFS the UMN Map server can render data from various sources in the PNG image format, which allows identifiable features. There will be more proprietary solutions to visualize 2D spatial data that either use or are based on WMS or WFS standard however these are not listed.

A comparison of WMS, WFS, KML and the UMN map server is not considered important to this thesis. An extended comparison between KML and WFS is made by (Shi, 2007).

## **3.4 Standards for database access**

### **3.4.1 Introduction**

For database access, there are two important standards with a different purpose. The Open DataBase Connectivity (ODBC) is a standard and an Application Programming Interface (API) to access databases independent of the database system, programming language, and operating system (subparagraph 3.4.2). The Standard Query Language (SQL) is a standardized language to query databases (subparagraph 3.4.3).

### 3.4.2 ODBC

ODBC provides a standard Application Programming Interface (API) for using Database Management Systems (DBMS). In the design of ODBC, its aim was to become independent of database systems, programming languages and operation systems.

The ODBC specification offers a procedural API for using SQL queries to access data. An ODBC implementation will contain one or more applications, which are a core ODBC library, and one or more database drivers. The core library that is independent of applications and DBMS systems, acts as an interpreter between the application and the database driver.

The database driver contains the DBMS specific details. ODBC acts as a middle layer between the application and the database. A programmer can write applications that use standard types and features without concern for the specifics of each DBMS that the applications may encounter. Driver implementers thus only need to know how to attach the core library, this makes ODBC modular.

Microsoft mainly develops ODBC. An ODBC driver for accessing an Oracle database is delivered with Windows, or can be downloaded from Microsoft. In the prototype developed for this thesis ODBC is used to access an Oracle database from the Python programming language. The data is then transformed and returned to user on the web client, in a format suitable for visualization (Microsoft, 2007).

### 3.4.3 SQL

The Structured Query Language or SQL is an ANSI/ISO standard language for querying a database management system (DBMS). It is a standardized language, which can be used for tasks like querying and modifying information in a relational database. SQL can be used with almost all modern relational databases. Every DBMS has added its own functionality to SQL; the effect is that a relational database written in SQL in one program cannot always be directly migrated to another DBMS.

Initially SQL was developed as a language for the end user. In practice, SQL is much too complex to be handled by people as managers. In practice a technical specialist executes the SQL while output results may be sent on to the manager probably in a more neat and readable format. In this light SQL has been transformed from a language of end-users to a language that makes a bridge between applications and databases.

People, who have used GIS systems, will have used SQL in a simplified form. For example to select all landlots in an area which has a value > \$200.000 Euro, the following select query can be written in the GIS: *“parcelvalue’ > 200000”*

If this query were executed as an SQL query in a database, it would look like this:

*“Select \* from parcels where parcelvalue > 200000”* which means the rows are requested where the parcel value > \$200.000 and are returned to the user. With \* is meant all columns from this table are returned, for example columns on parcel area, parcel height, parcel owner, land price etc. In this way attribute queries can be executed on one of the attributes mentioned. For example, a question can be executed as; select all landlots with a price below \$200.000 and owned by the municipality of Haarlem.

Other SQL queries can create or delete tables and insert and update values in tables. With a so-called transaction it can be achieved that all queries are executed and saved to the database, or all queries are discarded and the database keeps its original values, the latter is called rollback (Wikipedia EN [2], 2007).



## 3.5 Applications and plugins

### 3.5.1 Introduction

This paragraph lists the used plugins or applications for 2D and 3D visualizations. These are the BsContact plugin for X3D, the Google Earth application for KML, the Aristoteless and LandXplorer application for CityGML for 3D visualization (subparagraph 3.5.2). For 2D visualization, these are the Adobe SVG plugin or Firefox native SVG (subparagraph 3.5.3). In addition information is given about the database the data is stored (Oracle), about the package to make connection to the database possible (Oracle client) and about a module for a specific programming language (Python) to connect to the database called PyODBC (subparagraph 3.5.4). Finally, it gives information about the server application used; Apache HTTP. A server is required to enable the application for 3D visualization to be run on the internet, so other people can access it from any computer connected to the internet (subparagraph 3.5.5).

### 3.5.2 Plugins and applications for 3D visualization

#### Bs Contact X3D Plugin (for Firefox and Internet Explorer)

The 3D plugin used for the prototype for this thesis BsContact (Bitmanagement Software Contact) is a commercial X3D plug in for web browsers. A free version is available but it has a banner of the company. It has been chosen because it can work among multiple browsers, has the most options for visualization, and is stable. The software enables visualizations of applications in Virtual Reality and Augmented Reality through integration of real-time 3D technology. It can visualize X3D and VRML. More recently, it also has been enabled to deal with video in the MPEG4 format however; this functionality is released in another package than the X3D/ VRML support package. The BsContact plugin is the most extensive X3D plugin, unfortunately for users of Apple Macintosh or other operating system it only works with Windows (Bitmanagement, 2007).

BsContact offers a lot of possibilities to navigate around the data in different modes, which are walk, slide, examine, fly, pan, game-like and jump. From the menu can be jumped to different viewpoints in the data.

The scene can be visualized in different modes, which are: wireframe which means only the outlines of the objects are visualized, vertices which means only points are visualized, flat which means sky and objects are visualized as flat faces, smooth which is a normal visualization, solid which is a normal visualization with a wireframe, hidden line which means all hidden lines are visualized in white against a black background and bounding boxes which means bounding boxes of the objects are visualized. Navigating can be done at five different speed levels: very fast, fast, average (default), slow and very slow (Bitmanagement, 2007).

Another strong point of BsContact is that when an X3D file is embedded in the browser, individual X3D nodes can be altered using JavaScript as described in paragraph 3.6.5.

#### Google Earth application

Google has recently launched Google Earth, an application in which the user can zoom in on a globe on any place and view satellite images and detailed aerial photos. The highest detail that can be shown is different for each place: in the Netherlands, data with a resolution of 40 centimeters can be viewed.

With Google Earth, TEN data can be distributed to a large public. The user can open a TEN dataset in Google Earth from the prototype website, or a dataset generated with the prototype can be added to the Google Earth database, so anyone can directly download the KML file (Google, 2007).

Google Earth offers possibility to pan the data which means flying around the earth, to zoom in and out on data which means zooming in and out on the earth and to view data or the earth between an angle of zero degrees (aerial view) and an angle of 90 degrees (men view). Attribute information linked to KML files within Google Earth is displayed in a separate frame where web pages can be loaded. Further Google Earth offers the possibility to turn on all kinds of layers for example roads and 3D buildings, to add KML files from local disk and to search addresses, businesses, and routes (Google, 2007).

The clear advantage is the widespread use of Google Earth; it is more familiar than X3D and CityGML. Disadvantages are that the dataset needs to load in a standalone application and data cannot be dynamically modified (Giger, 2006).

### **LandXplorer**

The application LandXplorer CityGML Viewer of the company 3DGeo can do real-time 3D visualization of CityGML files of up to 15000 city objects. LandXplorer is a standalone application not a plugin for a web browser.

It supports the different Levels of Detail, which are LOD1, LOD2, LOD3, LOD4, but not LOD0 with the most detail, of CityGML. It has a selection function and search function to find and select objects and various navigation techniques such as examine game, fly, pan, and advanced. In addition, the object properties can be browsed with the Object Properties / CityGML browser window. The objects like the compass, sky, navigation settings, the scene, start-up position and the terrain can be turned on or off in the objects window. Transparent and textured object are supported. An advantage is that attribute information can be browsed hierarchy and in detail by the LandXplorer application, as well searches and selection in the data can be carried out easily. A disadvantage is that CityGML cannot be opened automatically in LandXplorer. CityGML now uses the GML mime type, which is not linked to a specific application, or plug-in. CityGML may be given its own mime type and default application so it can be opened automatically (Kolbe, 2006 and 3DGeoGmbH, 2007).

### **Other X3D players**

Flux player (Media Machines, 2007) is another web plugin to display X3D data on the internet. It is open source software so it could be modified to the requirements of the user. It supports Ajax3D as described in subparagraph 3.6.5. Disadvantages are the navigation is less flexible than navigation in BsContact, and embedding of 3D contents needs to be done in a different way then using BsContact.

Octaga player (Octaga, 2007) is another web plugin to display X3D data. It is not very stable with the prototype created for this thesis and therefore not recommended.

The Web 3D consortium (Web3D consortium, 2007) gives a list of other X3D players.

### **Other CityGML visualization software**

The Institute for Cartography and Geoinformation at the University of Bonn has developed an open source GML3 and CityGML 3D viewer application called Aristoteless. It is based on Java 1.5 and Java3D. The Aristoteless viewer project is designed as a framework for applications, which need to develop, demonstrate, or evaluate new features for 3D models. It is made primarily for scientific use. It may not be able to load large file because of memory problems. A connection to WFS option has been implemented, but the GML file requested still first has to be saved to the local disk. Such an option is not available at all in LandXplorer (Aristoteless, 2007).

GoPublisher can translate from the database to CityGML and then save the project (according to Wilkinson, employee of Snowflake software (Snowflake Software, 2007)). In addition, Snowflake software has created a WFS, which extract data from an Oracle database and translates it to CityGML. This WFS service can be accessed with LandXplorer (although not in the version that was tested in the thesis project). This is an interesting option for future research for visualization of TEN data (CityGML Wiki, 2007).

### **3.5.3 Plugins for 2D visualization**

Adobe has the most used and well-known SVG plugin: the Adobe SVG viewer, which can be used in the latest release of Internet Explorer. The functionalities it offers are zooming, switching on or off the sound, switching between high and low quality and the SVG can be copied, viewed, or viewed by source. The Adobe SVG plugin is currently not developed further (Adobe, 2007). Firefox makes uses of it own native SVG, which is more limited. There is no functionality offered to explore SVG, there only the possibility to view it. A list of other SVG viewers is put on a site by the (SVG Foundation, 2007).

Both Internet Explore and Firefox plugins support the adjustment of individual nodes with JavaScript / the SVG Document Object Model (SVG DOM) (to access the elements in the SVG XML tree) (see subparagraph 3.6.5).

### **3.5.4 Database applications**

#### **Oracle database**

Oracle is an extensible and object-relational Database Management System (R-DBMS) made by the Oracle Corporation (Oracle, 2007 [1]). An Oracle database is used in the prototype made for this thesis to store spatial geometric and thematic data. The Oracle database holds tables; tables contain columns, which have a name and a type like integer, string, date etc. Each table has a number of rows holding the data; there may be zero, a few, or 10.000s of rows in a table.

Oracle spatial is an extension on Oracle for holding various spatial data (Wikipedia EN, 2007 [1]). Oracle spatial specifies an SQL schema and functions that facilitate the storage, retrieval, update, and query of collections of spatial features in an Oracle database. In Appendix 3.2, the most important features of Oracle spatial are listed.

From the development from Oracle 9 to 11, first only 2D Simple Features were allowed in Oracle spatial (Oracle 9) (Open Geospatial Consortium, 2007 [2]), while in a later stage topological features were implemented, such as a polygon with reference to nodes (Quack et al, 2006) (Oracle 10). In the latest version (Oracle 11), real volumetric data types are implemented in Oracle spatial; the polyhedron data type (Arens et al 2004).

Using standards as ODBC or JDBC, programming languages can connect to the Oracle database and read out the data (subparagraph 3.4.2). If the programming language supports this, the Oracle data can be transformed, and returned to the web client in a format suitable for visualization; for example X3D (for 3D data), SVG (for 2D data), and HTML (for tabular data). Selections can be made in the Oracle dataset using SQL queries. For example, when all parcels have to be selected with the landuse 'grass' the following statement can be made: "Select \* from mytable where landuse = 'grass'".

#### **Other database applications (not used)**

There are some options to use other databases. These have not been tested as data was already provided in an Oracle database. In alternative one, a PostgreSQL / PostGIS database is used. The advantage of PostGIS is that it is cheap (free) and open source. PostGIS can spatial also store data in the Simple Feature Types as defined by OGC (Refractions Research Inc, 2007) (Open Geospatial Consortium, 2007[2]).

3D data types could be developed for PostGIS, for example a polyhedron, TEnbody, TEntetrahedron, TEntriangle and TEnnode as described in paragraph 2.8.

In alternative two an ArcGIS database is used; it has to be researched if it supports the storage 3D information completely. The information can be stored geometrically in 2.5D but things can go wrong when there are exact vertical rectangular faces. When a straight rectangle is drawn (or programmed in Visual Basic) in ArcGIS, ArcGIS will remove the fourth point and the rectangle will become a triangle. This can only be prevented by moving one of the coordinates a bit out of place so two points are not exactly above each other (Wesselingh, 2006). This issue may be solved in ArcGIS 9.2.

#### **Oracle client**

The Oracle Client software is a package of components that allow communication with Oracle databases in a distributed environment. In this way, a client for example in the form of a Windows computer can communicate with the servers containing the Oracle databases (IST, 2007).

In the prototype made for this thesis, which runs on a Windows computer with an Apache HTTP server, the windows server and the connection module on it (PyODBC for Python) can now talk with the Oracle databases on other (UNIX) servers. Secondly, it enables servers with Oracle databases to talk to other servers with Oracle databases.

The Oracle client is quite a big package (about 500 MB); however, instead of the Oracle client the Oracle instant client can be downloaded, which is a stripped version of the Oracle client and only a few megabytes and offers a database connection as well (Oracle Technology Network, 2007).

Though for a connection with PyODBC currently, the full client has to be installed, the instant client will not do in this case.

#### **PyODBC**

PyODBC is a Python module, which enables access to databases from Python using ODBC. It implements the Python Database API specification v2.0. This API has been defined to encourage similarity between the Python modules that are used to access the database; the interface specification can be found on and is written and maintained by the Python database SIG (Lemburg, 2007). Data that is requested from the database can directly be read into a Python (array) variable. PyODBC is absolutely required as it is a medium to read the data from a database into a programming language. It enables a connection between the web client and the database (Kleehammer, 2007). PyODBC does not have special support for reading spatial datatypes.

### 3.5.5 Server applications

#### Apache HTTP server

A server is required if a web page has to be accessed by other people on the Internet. The user surfs to the server for example by ticking in its address and the requested pages are downloaded.

The Apache HTTP server is an open source HTTP server for modern operating systems including UNIX and Windows NT. The goal of Apache HTTP is to provide a secure, efficient, and extensible server that provides HTTP services in sync with the current HTTP standards. By installing `mod_python`, Python scripts can run on the Apache HTTP server (the Apache software foundation, 2007) (sub paragraph 3.6.3).

## 3.6 Programming languages

### 3.6.1 Introduction

In this paragraph, the programming languages are described that are used to set up the prototype. Subparagraph 3.6.2 gives information about Python, which is used for scripting in general. The advantages of Python in comparison with other programming languages as Java, VB, and C++ are described. The syntax of Python will be described in this paragraph too.

Subparagraph 3.6.3 gives information about the `mod_python` extension, with `mod_python` the Python scripts can run on the server.

Subparagraph 3.6.4 is about JavaScript, JavaScript is used to read out forms elements as fields, drop down lists etc. JavaScript can format a link, which points to a Python script with the form values as input parameters. For example, when the output format is CityGML, JavaScript will pick up the value 'CityGML' and the Python script will display CityGML to the user when the link is executed. JavaScript is used to adapt the 2D or 3D visualization dynamically; this is explained in subparagraph 3.6.5.

### 3.6.2 Python

Python is an object oriented, high-level, prototyping language. Python is a scripting language, which means it is interpreted every time it is run. Advantages of Python over other (scripting) languages are that it is extensible and embeddable in applications. Python is free, open source, easy to learn and well readable (van Rossum, 1999).

Python is a programming language, just as Java, Visual Basic, and C++ are. However, there are some major differences with these languages. Python is dynamically typed. This means the type of a variable never has to be declared. In Python if a name is assigned to an object of one type it may be later assigned to an object of another type. Python container objects can hold objects of any type including numbers and lists. When an object is retrieved from a container, it always remembers its type unlike in Java (Ferg, 2007).

Text insertment 3.1 displays for example how to initialize an integer to zero and then convert it to a string, in Java.

```
Int mycounter = 0;
String myString = Integer.toString(myCounter);
```

Text insertment 3.1: Convert to integer to string in Java

Text insertment 3.2 displays the commands to initialize an integer to zero and then convert it to a string in Python.

```
myCounter = 0
myString = str (myCounter)
```

Text insertment 3.2: Convert integer to string in Python

Python is more concise and less verbose than other programming languages. It means it uses less words and characters. Text insertment 3.3 displays the Java code to print a line 'hello world'.

```
public class HelloWorld
{
    public static void main (String[] args)
    {
        System.out.println("Hello, world!");
    }
}
```

Text insertment 3.3: Print "hello" in Java

Text insertment 3.4 displays the shorter Python code print a line 'hello world'.

```
Print "Hello, world!"
```

Text insertment 3.4: Print "hello" in Python

Python code can be 2-10 times shorter than similar C++ or Java code. Python has a number of smart operations to inspect string for example:

"Hello"[6] would result in "o". "Hello"[3:6] would result in "llo", "Hello"[3:] would result in "Hel" and "Hello"[:3] would result in "llo".

It is easy to deal with arrays in Python, for example if a face object of four coordinate pairs has to be stored first the four points are assigned to variables as displayed by text insertment 3.5.

```
P1 = (0.0, 0.0, 5.0)
P2 = (0.0, 10.0, 5.0)
P3 = (10.0, 10.0, 5.0)
P4 = (10.0, 0.0, 5.0)
```

Text insertment 3.5: Assign four coordinates in Python

In text insertment 3.6 an array of two dimensions is defined.

```
myface = [[]]
```

Text insertment 3.6: Define face of two dimensions in Python

Text insertment 3.7 displays the commands to add the four coordinate pairs to the first item in the array.

```
myface[0].append(P1)
myface[0].append(P2)
myface[0].append(P3)
myface[0].append(P4)
```

Text insertment 3.7: Append coordinates in Python

Text insertment 3.8 displays the command to add another face.

```
myface.append([])
```

Text insertment 3.8: Append a new face in Python

Text insertment 3.9 displays the command to read out the x coordinate of the fourth point of the first object (indices starts at 0).

```
y = myface[0][3][0]
```

Text insertment 3.9: Read out coordinate in Python

Instead of using a select query on a database, Python can be used to filter data from arrays, using if statements. This depends on the type of operations done in SQL and or in the programming language. For example the 'Joinquery' algorithm in the prototype made for this thesis uses a smarter query to request TEN data in one query instead of requesting the data in two queries (for getting and nodes and faces) in the 'Faster' algorithm. The 'Faster' algorithm uses more 'if' statements and performs much better for loading the whole Rotterdam set (50% faster). This may be different with another dataset or with another query or algorithm (see also subparagraph 6.4.2).

Drawbacks of Python are it can run slower than similar C++ or Java Code. The use of indentation for loops and if statements can be confusing, especially when they have to be closed. Python can connect to a database by using a Python module that can connect to a database. PyODBC can be used to connect to an Oracle database via the ODBC protocol.

Concluding Python offers a simple verbose, it is easy to deal with variables, and lists are very flexible. A Python script can prepare data for visualization by formatting it in such a way that it can be read by an application or plugin, which can visualize this format. For example, Python can transform data from a database in X3D XML tags; the XML tags are then combined to form an X3D file. Using the Apache module mod\_python, Python script can run on server (subparagraph 3.6.3), and the results (the X3D file) can be returned to the client (van Rossum, 1999 and Ferg, 2007).

### 3.6.3 Mod\_python

Mod\_python is an Apache module that enables Python scripts to be run on a server. For example a Python script can be activated on the client, read 3D data from a database on server, transform it to X3D and have the data returned to the client. The latter can be done with the special mod\_python commands: 'req.content\_type="model/x3d+xml"' and 'req.write(data)'

There are different mime types for different visualization formats:

- HTML: text/html
- X3D: model/x3d+xml
- KML: application/vnd.google-earth.kml+xml
- GML: application/vnd.ogc.gml.
- SVG: image/svg+xml

Without, Python scripts could not be run on server thus Python could not be used to return data within a web application. Mod\_python enables 3D data that is created on the server, to be displayed to the client in a browser (Apache Software Foundation, 2007).

### 3.6.4 JavaScript

JavaScript is a scripting language just as Python is. The language is mostly used in websites as client side JavaScript, but is also used to enable scripting access to objects embedded in other applications. JavaScript is not the same as Java. It is only distantly related to the Java programming language.

A major use of JavaScript is to write functions that are embedded in or included from HTML pages and interact with the Document Object Model, which is not possible in HTML alone. For example, values filled in on a form can be read out by JavaScript and be returned on another place of the page or in a new page. Functions that can be performed by JavaScript are having a hyperlink that targets multiple frames popping up a window with programmatic control over the size, position, and look of the new window, validation of form input, and the changing of images on a mouseover. Some of this functionality is used in the prototype. There are slight variations for other browsers of JavaScript called Jscript and Ecma-script (W3 Schools, 2007 [1] and Champion, 2001).

JavaScript can be used too to access X3D or SVG visualizations dynamically, which are embedded in the frames that are used for the visualization of the data using the Scene Access Interface (SAI). The individual nodes can be updated. For example, the colour of an object can be changed to yellow if it is selected, without having to reload the whole file again. This way, HTML files can be modified too, for example, the colour of a row in a table can be changed to yellow corresponding to the selected object. JavaScript can make 3D and 2D visualizations and HTML pages dynamic. These techniques are also known as AJAX or AjaxX3D, depending on the way JavaScript is implemented and the players used. Using these techniques JavaScript can support attribute selection of data (Carto: net. 2007 and Bitmanagement, 2007). Subparagraph 3.6.5 gives give more detail on Ajax and Ajax3D.

JavaScript can also be used to transform data and to have it returned in the web client, it is then executed on the client side. There are limitations, for example, it cannot read data from on Oracle database. In the GML2 -> X3D option JavaScript alone is used to retrieve and return data (W3 Schools, 2007 [2]).

### 3.6.5 JavaScript, Ajax, and Ajax3D

On a website with Ajax technology information can be sent and retrieved in a variety of formats, including XML, HTML, and even text files. Ajax's most appealing characteristic, however, is its "asynchronous" nature, which means it can do all of this without having to refresh the page. This allows portions of the page to be updated upon user events. JavaScript is used for the data communication. However in principle both DOM (for accessing web pages dynamically) and SAI (for accessing a 3D visualization dynamically) are independent of programming language (Parisi, 2006). Ajax is not a programming language itself only a technique (Mozilla Developer Center, 2007).

Ajax3D provides the same functionality for an X3D scene embedded in a website as Ajax does for websites. Individual parts of the X3D scene can be updated (for example the colour of a node can be changed, or a shape can be added to the scene) without having to reload the whole X3D file. An Ajax3D application uses the Scene Acces Interface (SAI) (Web 3D Consortium, 2006 [2]) to access a real-time 3D scene and to modify the 3D scene, the Document Object Model (DOM) (W3C, 2005) to manipulate web page contents in response to changes in the 3D scene, and uses server request methods to store and retrieve data in response to or leading to changes in a 3D scene. JavaScript is used again for the data communication (Parisi, 2007).



In addition to adapting HTML with the DOM, X3D with the SAI or SVG with the SVG DOM, AJAX(3D) technology uses server request methods (XMLHttpRequest, createX3DfromURL) to store and retrieve data in response to, and/or leading to, changes in a 3D scene. In this way an Ajax3D application can build an entire 3D world programmatically, for example based on queries to a MySQL database (Parisi, 2006).. In the prototype developed for this thesis the server request methods are not used, only the DOM, SVG DOM and SAI are.

The X3D player BsContact, used in the prototype for this thesis, has its own proprietary interface to modify the scene using JavaScript, which is not Ajax3D although the differences are not big. The X3D Flux player does use Ajax3D. The Adobe SVG plugin and the native Firefox SVG implementation support dynamical adjustment of the scene too, with the SVG DOM. it is not called Ajax again. The 3D visualization in X3D and the 2D visualization in SVG have to be embedded in the browser to enable dynamical access to the scene.

## ***4. Review of 3D GIS applications on the web***

### **4.1 Introduction**

Several prototypes for making a 3D Geographical Information System (GIS) are presented in this paragraph. Some of the prototypes presented in subparagraph 4.2.1 date from almost a decade ago, and use VRML for 3D visualization, a database, and most of the times a Java client to send data from the database to the 3D visualization. The External Authoring Interface (EAI), an addition to the VRML, is used to interact with and modify the VRML visualization.

In the prototypes in subparagraph 4.2.2, X3D is used to present 3D data. The Scene Access Interface (SAI), an addition to X3D, is used to interact with and modify the data. Different programming languages have been used for the prototypes presented. More advanced features are now added, as a link between the 2D and a 3D visualization, and the possibility to execute SQL queries and have the results of that query visualized. In paragraph 4.2.3, some other prototypes are described, which are not web based. In subparagraph 4.3 a feature comparison of the prototypes is done. Paragraph 4.4 a feasibility review of the prototypes is done and the best prototypes are picked out. In the conclusion in paragraph 4.5 is stated which prototype forms the best example for the development of the prototype made for this thesis.

### **4.2 Review of 3D GIS implementations**

#### **4.2.1 Introduction**

Subparagraph 4.2.2 list the VRML based prototypes by Morcrette, Kim et al, Zlatanova, de Vries et al, Zhu et al, Beard and Rancic et al. The VRML based prototype use VRML for 3D visualization, a database and most of the times Java to send and get data from the 3D visualization. The EAI is used to interact with and modify the VRML scene.

Subparagraph 4.2.3 list the X3D based prototypes by de Vries et al, Ninsawat et al and Kumke et al. X3D is used to present 3D data. The Scene Access Interface (SAI), an addition to X3D, is used to interact with and modify the data.

Subparagraph 4.2.4 list some other 3D GIS prototypes, which are not web, based for the sake of completeness.

#### **4.2.2 VRML implementations**

##### **3D GIS Prototype using VRML by Morcrette, 1998**

In the USA (Morcrette, 1999) was one of the first researchers to transform spatial information in a database to a 3D scene in VRML using Java. VRML the Virtual Reality Modeling Language is text based file format or syntax to describe 3D worlds, and the predecessor of X3D based on XML. In Morcrette's prototype, geometrical information is converted to 3D Buildings, but attribute information is also transformed to 3D using stacked columns of blocks. For example, three blocks with a toilet icon on the front and one block with a shower icon on the front mean that a building will have three toilets and one shower. The columns of blocks are placed above the buildings. Images of the project can be found on the site of the Center for Educational Computational Computer Initiatives, Massachusetts Institute of Technology (Bailey et al, 2006). A similar prototype is available from a Msc. research at the University of Michigan (Chirapiwat, 2003).

A distinction is made between two prototype architectures. In the first prototype architecture, two tier (figure 4.1, left image); there is a direct connection between the VRML file and the database. A Standard Query Language (SQL) scripting node within the VRML realizes the connection; this is a

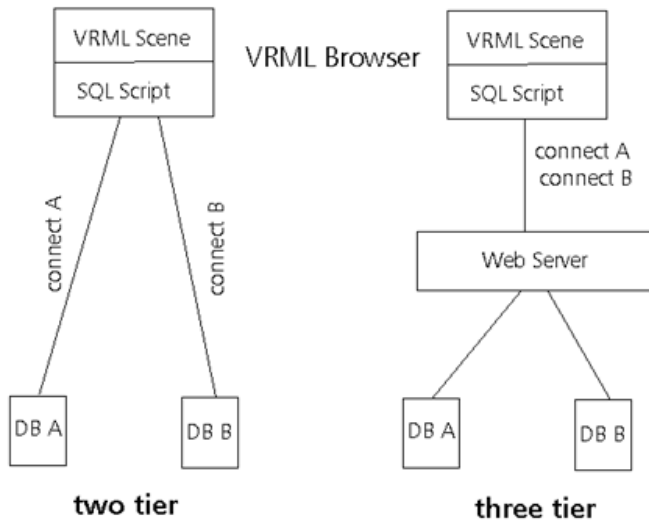


Figure 4.1: VRML / JAVA web GIS architecture by (Morcrette, 1999)

special node that is used to execute normal SQL commands. In the second prototype architecture, three tier (figure 4.1, right image), there is a web server between the 3D VRML visualization and the database. In this architecture, the VRML browser makes contact with the web server in the middle layer, which connects to the database, reads out the data, and sends it back to the VRML browser. The second architecture is more appropriate for the Internet where firewalls are existent, while the first can be used on an intranet.

The client proposed is a Java applet that runs on the client. The database is accessed using the Java DataBase Connectivity protocol (JDBC) (Sun Microsystems, 2007). JDBC is a standardized protocol to connect to different kind of databases with the same technology using Java; similar to the Open DataBase Connectivity protocol (ODBC).

The final prototype architecture is composed of the following three components. The client, a VRML browser (Cosmoplayer), is responsible for displaying (VRML data view) and forwarding input information from the user. A server (unknown brand) is responsible for the data communication, and a Toolset is responsible for the 3D code generation. VRML nodes can be updated individually, for example, the colour can be changed, or a texture can be added. In Appendix 4.1 a sample VRML fragment is given.

Morcrette makes note of 3D XML. This is a VRML language based on XML. This has now evolved into X3D. In this prototype, SQL statements can be included within the individual nodes. The prototype is limited to 2D buildings with only a height connected to them.

### 3D GIS prototype based on VRML and Java by Kim et al, 1998

About the same time in China (Kim et al, 1998), propose a 3D web GIS prototype architecture, based on Java and VRML with more possibilities. 3D GIS functionalities as visualizing, getting non-spatial attributes, 3D feature indexing, 3D analysis such as selection, buffer and near operation, and metric analysis such as distance and statistics, are implemented in Java, while the 3D visualization is done in VRML. Again, the system is implemented in a Java applet.

The EAI allows the Java applet to interact with the 3D visualization in VRML. The EAI is also used in Morcrette's project. The advantage of the EAI is that VRML can be dynamically built, and updated based on data received by the Java applet. The other way around, data of the Java applet can be updated from within the 3D visualization in VRML.

The architecture (figure 4.2) of this prototype is layered. In the middle there is the EAI running in the Java applet. Via the World Manager, it can request 3D features, DEMs, and satellite images from the database (brand not mentioned). The 3D features are made up of four components, which are a geographic primitive as a node, chain, polygon or DEM, visualization information as shape or colour, non-spatial attributes as streetname or owner, and multimedia related information for example a website with a monument database.

It is an advantage that the 3D visualization can be dynamically updated with data from the database, and data in the database can be updated from the 3D visualization. Kim et al created the Spatial Operations to do 3D analysis in Java. The results are visualized in the VRML world by letting the EAI update the individual nodes.

Analyses that can be performed in 3D by the spatial operations manager are feature identification, 3D buffer, 3D near for example find the buildings within 100 meters of a school and Lantern (see article by Kim et al. for more detailed information on Lantern operation).

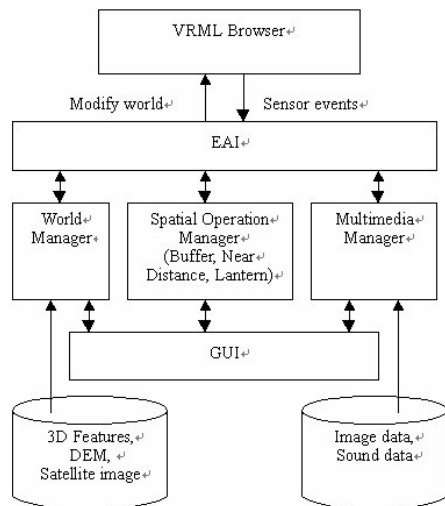


Figure 4.2: VRML / JAVA 3D web GIS prototype architecture by (Kim et al, 1998)

### 3D GIS prototype using VRML and CGI by Zlatanova, 2000

In ITC, the Netherlands (Zlatanova, 2000) created a prototype of a 3D GIS based on CGI / PERL scripting for form input and processing, and VRML for 3D visualization (figure 4.3). CGI is a technology to dynamically request data from a web server, for example with a CGI script, a user on the internet can request certain information from a database and the database receives the request, and passes the results back to the user. Scripting languages as PERL and Python can use CGI (NCSA development team, 1995).

When a spatial query is executed on the data, the results are delivered as VRML. A HTML interface is used for the 3D visualization and the form input. Identification can either be done in forms by filling in the ID of an object, or by clicking an object in the 3D view. When an object is clicked in the 3D view, a short description, and a photo is displayed. Attribute queries can be done on the data by filing in a query in a (CGI based) form.

The data is stored in a MySQL database. Various analyses can be performed and data can even be modified from the web browser. The prototype does offer a solution to handle large datasets by making use of a 3D R-Tree (Zlatanova, 2000).

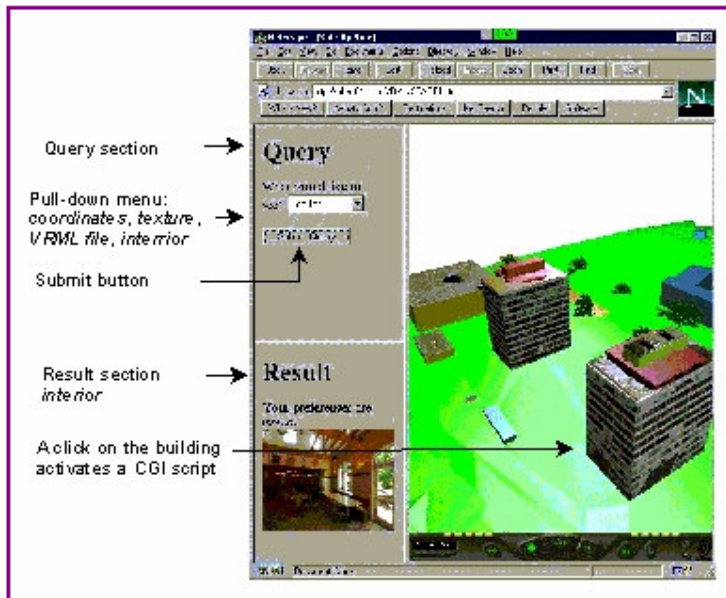


Figure 4.3: Web GIS prototype by (Zlatanova, 2000)

### 3D GIS Prototype based on VRML, ASP and an Access database by de Vries et al, 2003

In OTB, TuDelft, (de Vries et al, 2003), describe two prototypes for 3D visualization. The first prototype uses VRML, and is described here, in the paragraph VRML implementations. The second uses X3D and is described under the X3D implementations (sub paragraph 4.2.3).

The first prototype described by (de Vries et al, 2003) (figure 4.4), uses ASP, VRML and MS Access. Only the attribute data is stored and retrieved from the database. Spatial data is already present in the form of a VRML file on the server. Non-spatial attributes are stored in the MS Access database.

(Wesselingh et al, 2005) built a similar prototype with some more functionality using ArcIMS. The web server used is a Microsoft Internet Information Server (IIS). A server programming language, Active Server Pages (ASP), is used to communicate between the internet client and the web server. ODBC is used to access the database. The user interface holds only two components: a 3D view where the data is viewed in the form of a VRML file, and a data view where the attribute data is shown to the user in the form of an HTML table. When a user clicks an object in the 3D view, the attribute data is shown in the data view. The ASP script reads the attribute data from the Ms Access database using ODBC, transforms it to a HTML table, and returns it to the client. Attribute queries cannot be done (de Vries et al, 2003).

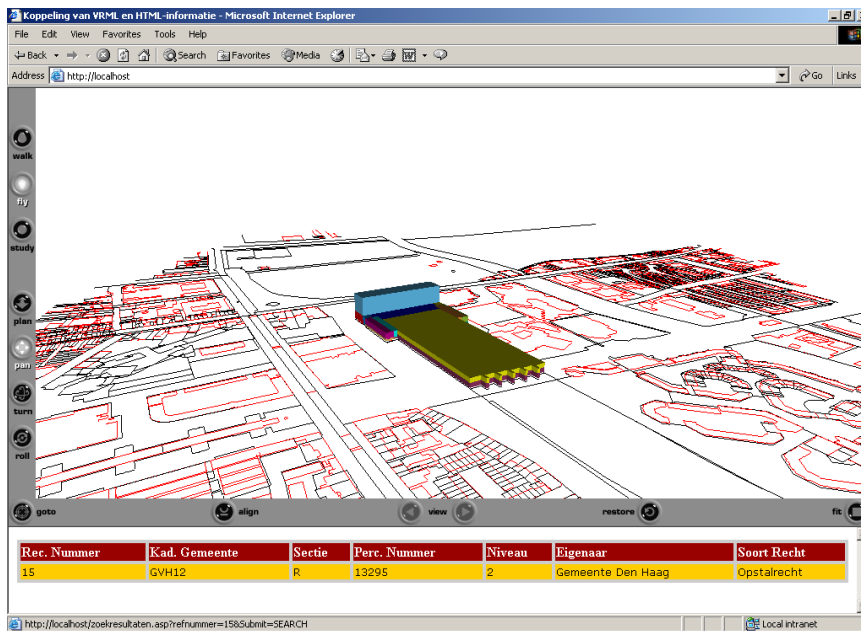


Figure 4.4: 3D Web GIS Prototype [VRML implementation] (de Vries et al, 2003)

#### Advanced 3D GIS prototype based on VRML, Java, JDBC and the EAI by Zhu et al, 2004

Continuing with the work of Kim et al. in Singapore, (Zhu et al, 2004) propose another prototype based on Java, JDBC, and the EAI (figure 4.5). This prototype focuses mainly on terrain visualization, and not on the visualization of other 3D objects.

The prototype exists of two main components, a VRML browser, and a Java user interface. An Apache web server is adopted. In the VRML is made use of script nodes. By using Java in the script, information can be exchanged between the Java General User Interface (GUI) and VRML. Maps are translated to VRML from the database with the EAI component while JDBC is used for input of queries. The query-path in this prototype is limited; it is purely used to get textual results from queries on the database.

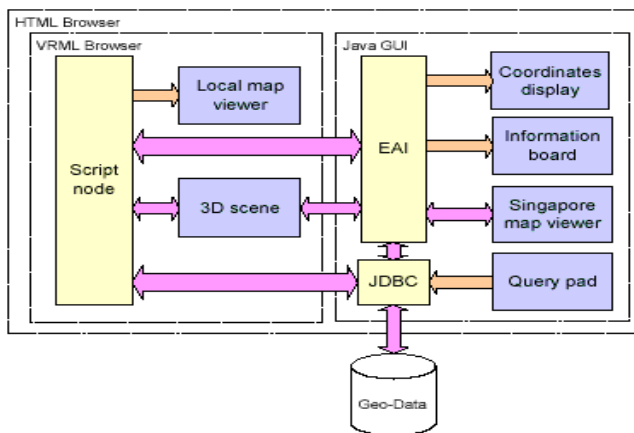


Figure 4.5: VRML / JAVA 3D web GIS prototype architecture by (Zu et al, 2004)

There can be made a distinction between different terrain networks namely TINs (Bossler, 2002 [1]) and DEMs (Bossler, 2002 [2]). Because gridded DEMs are easier to construct and transformation between layers of different resolution is easier with DEM, Zhu et al. chose to visualize a DEM.

Zhu et al. describe two solutions to handle large datasets.

- The first solution is to make use of Level of Detail (LoD); however, two problems occur: In the beginning, the dataset has to be loaded for each Level of Detail making loading time long. If the terrain changes when switching to a different Level of Detail, the features on it have to be adjusted as well This might for example result in a road hanging above and or floating under a piece of terrain.
- The second solution is to only visualize that part of the terrain that can be seen from the viewpoint. The terrain is split up in tiles. When the tile falls inside the view of the user, the DEM data for that tile will be extracted from the database and converted to VRML.

The second solution has been implemented in the prototype by Zhu et al.

On the site of Bitmanagement (Bitmanagement Software, 2007 [2]) an example can be found of a large terrain visualization where probably only that part of the visualization is loaded which is in the view of the user. In this way, the 'tile' selection is done from inside the view. By letting the user input a bounding box the 'tile' selection can be done before visualization of the data (as in fig. 4.6).

This prototype has a few advantages over the prototypes by Morcrette and by Kim et al. There is a 2D-3D Hybrid interface, and analysis and attribute queries can be made on the data, although the result of such a query is only textual. The EAI can update individual nodes in the 3D view. In addition, this prototype can deal with large datasets, by only visualizing the part of the terrain that can be seen from the viewpoints; this is done by splitting up the terrain in different tiles.

### **3D Geological visualization of Tasmania by Beard, 2004**

In Australia, (Beard, 2004), created a geological dataset of Tasmania, Australia using VRML.

Different layers can be added or removed from the scene using a menu on the left. The 3D visualization is altered using JavaScript, which can directly access the VRML data.

This project shows that it is possible to modify a 3D scene with JavaScript without having to reload the full dataset.

A database has not been used for this project; data is already present in the forms of a VRML file (however, a database has been used in a similar project).

Selection queries cannot be made on data, though different 3D layers can be added to or removed from the data depending on what the user wants to see. Disadvantages are that there has not been made a possibility to select features within one layer; there is no solution to deal with large datasets as full layers are provided as VRML nodes thus the user should wait until they are loaded.

### **3D GIS prototype based on VRML / X3D, JAVA, and the EAI by Rančić et al, 2005**

The most recent VRML web GIS implementation is developed in Serbia and Montenegro by (Rančić et al, 2005). They present a prototype for 3D visualization on the web. It is similar to the prototypes of Zhu et al. The prototype can visualize DEMs, 2D, 2.5D, and 3D features that are all stored in different databases.

Technologies that are used are Java, EAI, VRML, and X3D. When zooming in or out on the terrain different LoDs are loaded, in this way can be dealt with large datasets The interface exists of a 3D visualization where the data is loaded as a VRML or X3D file, and a box where the different layers, possible containing different data types, can be turned on and off.

JavaScript is included in objects to make interaction possible.

### 4.2.3 X3D Implementations

#### 3D GIS prototype based on X3D, Java and Oracle XSQL servlet by de Vries et al, 2003

At OTB – TUDelft, the Netherlands, (de Vries et al, 2003), build two prototypes for 3D visualization. The second prototype uses X3D and is described here under the X3D implementations.

The second prototype by (de Vries et al, 2003) is based on an Oracle XSQL servlet, X3D and Java (figure 4.6). A servlet is a Java program that runs on a server. The servlet processes requests on the server, which are made on the client. The server requests geometrical information by means of a XSQL query from the Oracle database. By executing an XSQL query, the data in the database comes available as XML. XSQL is the combination of XML (Extensible Markup Language) and SQL (Structured Query Language) to provide a language and database independent means for storing SQL queries, clauses and query results (Dooling, 2002). The XML is transformed to X3D with a stylesheet transformation (XSLT). Attribute information is requested from the database with another XSQL query. XSQL uses JDBC to make a database connection. The XML stream from the database can be transformed to X3D for a 3D view, to Standard Vector Graphics (SVG) for a 2D view (although this is not done in their prototype), and to HTML to show the attribute information in a table. Queries can be filled in a form and are picked up by JavaScript and then processed by Java and XSQL. Finally the prototype offers the possibility to enter a bounding box from which data is selected.

The prototype has a connection between the 3D visualization on the client and the database on the server in two ways.

- Geometrical information is read from the database and transformed to X3D.
- When an object is clicked, its attribute information is displayed as a HTML table.

The part of the data that is visualized is dependent on the query and bounding box filled in by the user on the form. In this way, loading times for large datasets can be decreased by only requesting a certain area.

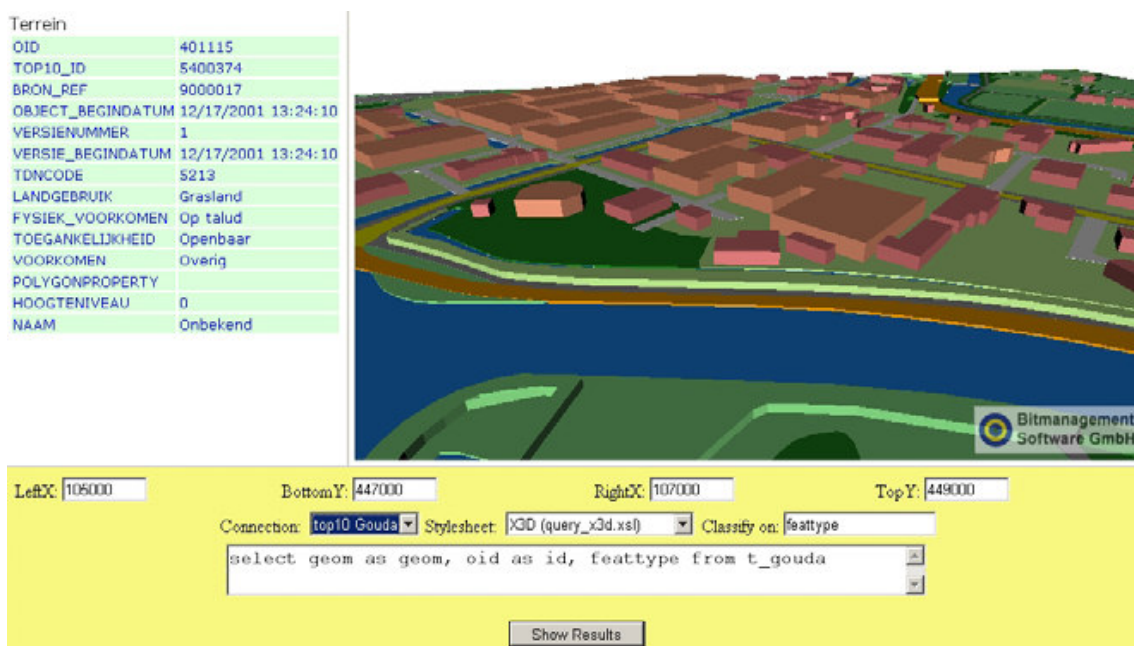


Figure 4.6: 3D Web GIS prototype [X3D implementation] (Vries et al, 2003) (Held et al, 2004)



### Prototype based on X3D, WFS, and Oracle by de Vries et al, 2004

One year later, also at OTB – TUDelft, the Netherlands, (de Vries et al, 2004), present a prototype for the display of 3D geo-information using X3D and a Web Feature Service (figure 4.7). It is a continuation of the work of (Zlatanova, 2000) and (de Vries et al, 2003).

Data is stored in an Oracle database using Oracle Spatial Data Option (SDO) geometry and simple feature types. A Web Feature Service (WFS) reads the data from the database using the JDBC protocol and outputs GML3, which supports 3D geometries (see subparagraph 2.2.4). The GML 3 is processed, transformed, and returned to the client as X3D with client side JavaScript and a stylesheet transformer (XSLT). The X3D is displayed with BsContact X3D plugin. This prototype architecture is implemented in the prototype made for this thesis using an ODBC connection, Oracle data in ‘normal’ data types and GML 2, as the ‘X3D via GML 2’ visualization option (de Vries et al, 2004).

The prototype by de Vries et al. has a connection between the 3D visualization on the client, and the database on the server in two ways. Geometrical information is read from the database, output as GML by the Web Feature Service, and transformed to X3D on the client. In addition, when an object is clicked in the 3D scene, its attribute information is displayed as a HTML table. Queries can be filled in on the client, and are sent to the database using a HTTP Get / Post request and JDBC. The prototype offers a solution to handle large datasets using a WFS filter (de Vries et al, 2004).

This prototype did work well, although with a WFS made by the authors themselves, not an approved WFS.

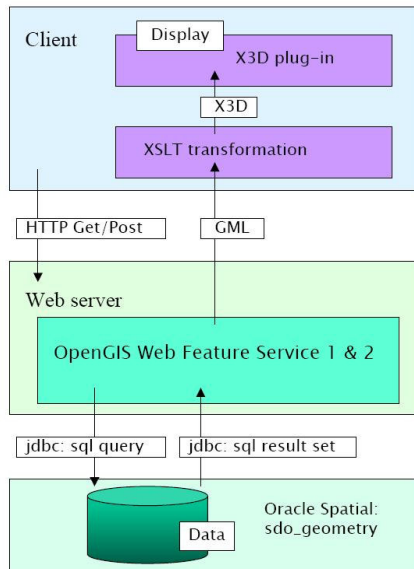


Figure 4.7: 3D Web GIS Prototype based on SDO and WFS (de Vries et al, 2004)

### 3D GIS prototype based on X3D, Flux, and PostgreSQL by Ninsawat et al, 2006

In Japan, (Ninsawat et al, 2006), created an open source solution for the development of a 3D Web GIS system using X3D. It enables terrain visualization using medium resolution DEMs and the display of borehole data or infrastructure network as layers (figure 4.8). All the software used is open source; PostgreSQL is used as the database, Python is used for scripting, X3D is used as the 3D visualization format and Flux player is used for viewing X3D files.

The output X3D file is only of the requested area. Data is returned to the user in a HTML interface with Flux plugin embedded. Queries can be picked up by JavaScript and be sent as HTTP request to the web server, which picks it up, and passes it on to the database. 2D data is presented via a UMN Map server with PHP MapScript (University of Minnesota, 2007).

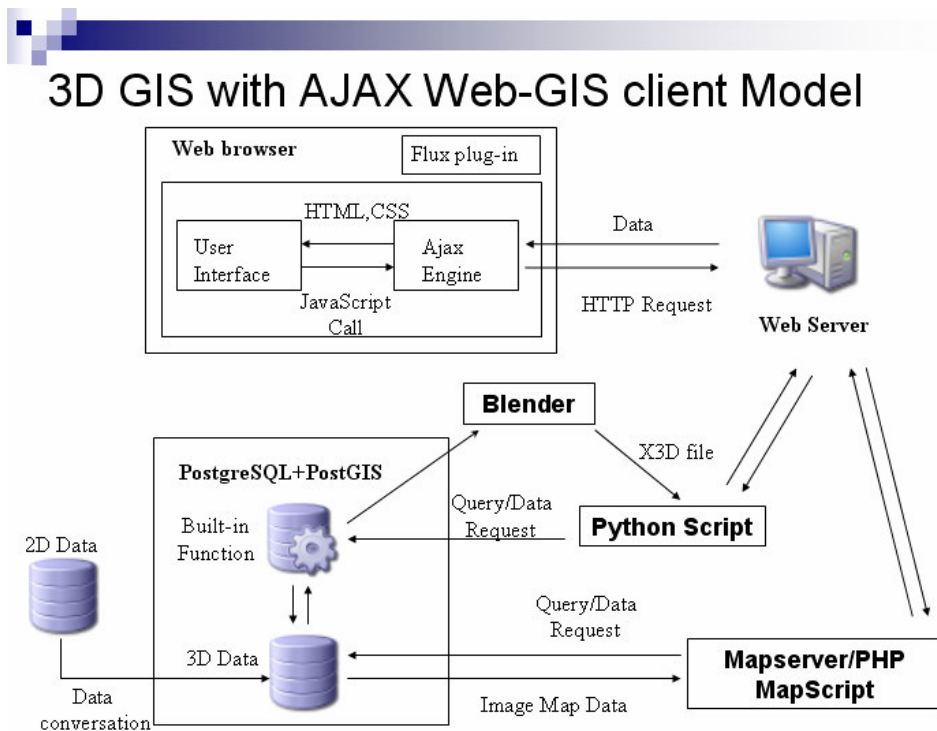


Figure 4.8: 3D Web GIS Prototype with AJAX client model by (Ninsawat et al, 2006)

A Delaunay triangulation using parallel computer technology can be developed to reduce processing time of terrain data and to allow large datasets and multi-user service. New technologies as Ajax and Ajax3D (subparagraph 3.6.5), can provide functionality as adjusting the X3D visualization without having to return the whole 3D file to the server at once. This is also one of the user requirements of this thesis.

In the prototype by Ninsawat et al., analysis and attribute queries can be made, using JavaScript for query input and Python to process the query and to interact with the database. With Ajax X3D (Asynchronous JavaScript and XML) the scene can be updated (see paragraph 3.6.5). There has been a focus on the visualization of terrain or DEM data and 2D / 2.5D features, like boreholes or infrastructure. Several solutions to handle large datasets are presented like visualizing the area that is requested by the user, or to use parallel computed Delaunay triangulations. A 2D-3D hybrid interface is presented to the user with a 2D interface in the form of a 2D map (generated by UMN Map Server), and an X3D visualization in Flux Player for the 3D window. The prototype that is online does currently not work (Ninsawat, 2007), probably because of revision reasons.

### 3D GIS prototype based on X3D, Oracle, and Java by Kumke et al, 2006

In Germany, (Kumke et al, 2006) propose an application for display of 3D building data, and the integration of thermal textures and characteristics. 3D building models and infrared sensor data are stored in a database. The data analysis is split up in analysis of the geometry and analysis of the characteristics and properties. The latter are classified. The results of both analyses are sent back to the database (figure 4.9).

Geometrical data is stored in the Oracle 9i database as 2D polygons with a z-component in the Boundary representation. In addition, images from the infrared and the visible part of the spectrum are stored in the database. The geometry is translated to X3D using Java3D and presented on the client. Images are also sent to the client and referenced to as textures within X3D. The client is made in Java. On the client, queries can be input and sent to the database with JDBC, or the data can be accessed on the client with SQL plus worksheet, which uses PL/SQL to talk to the database. Procedural Language / Standard Query Language (PL/SQL) is an extension to the SQL database language for Oracle (The Oracle FAQ, 2007). The interface is quite similar to the interface of the prototype for this thesis but now for the interface Java has been used instead of HTML. Queries are picked up by Java instead by CGI / JavaScript, and queries are done with the JDBC protocol instead of the ODBC protocol. A Java procedure transforms data in the database and returns it to the client as X3D. It is not clear if a web server is used in this prototype architecture and which web server.

For a more in depth description is referred to the project presentation by (Kumke et al, 2006).

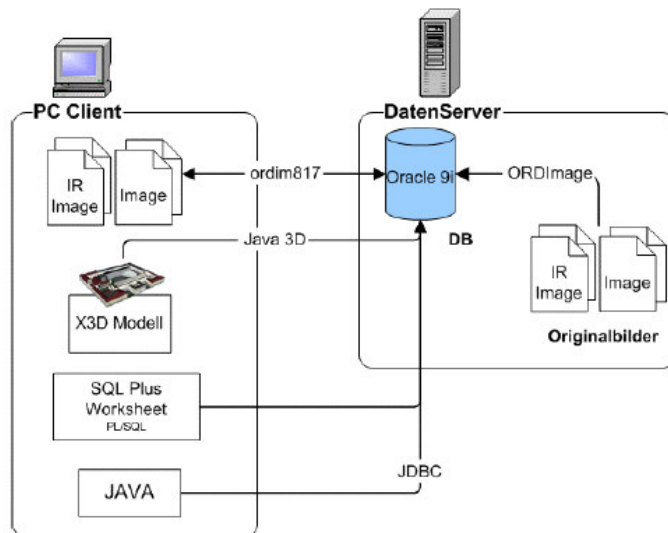


Figure 4.9: X3D/ JAVA 3D web GIS architecture by (Kumke et al, 2006)

There is support for different data types as DEMs, InfraRed image /raster representation and 3D features in the boundary representation. Textures can be stored within the database and be added to the data in the 3D visualization. It is not clear whether individual nodes in the 3D scene can be updated. This prototype focuses more on the attributes too than on the 3D visualization; it is about thermal characteristics. This prototype is not online.

#### 4.2.4 Other 3D GIS prototypes

For the sake of completeness a short description of prototypes is given, which are not based on a web architecture with a server, visualization in VRML or X3D in the web client, and a database.

In Switzerland, (Carosio, 2007) proposes a prototype of a standalone 3D vector GIS called SOMAS (Solid Object Management System). The goal of this project is to implement a suitable geometric, topological, and thematic data model to optimize the data management (storage and access), and to get a tool for investigation of 3D and 4D analysis functions.

(Held G et al, 2004) give a number of 3D (web) GIS solutions. These are Salix (Lammeren et al, 2003,) for interactive landscape planning (a thick client which means most operations take place on the client side), DILAS (Nebiker, 2003), for managing, editing, reconstructing, visualizing and publishing virtual worlds on a standalone client within MicroStation, and GIERS (Kwan et al, 2003), a GIS based intelligent emergency response systems which is implementing 3D routing features up to the inside of buildings for rescue times in real time with a underlying topological data structure. Finally the Special Interest Group (SIG) 3D of the Geo-Data Infrastructure North-Rhine Westphalia Germany (GDI NRW) proposes a 3D prototype based on 3D urban (CityGML) data and OGC and ISO 19107 standards (Panzer et al, 2006). In addition they propose an extension of the Web Terrain Service (WTS) (Lieberman et al, 2003) called Web 3D Service (W3DS) (Homoet et al, 2005).

TNO Built environment and Geosciences - The Geological Survey of the Netherlands (Dinoloket, 2007) has created the Geo3DJViewer. This viewer make a geological 3D visualization out of bottom and top raster of various geological layers stacked on each other. In addition any profile can be drawn in the 3D viewer.

### 4.3 Feature comparison of prototypes

	Year	Format	Client-server connection	Attribute Selection	SQL support	BREP	TIN	Raster	DEM	2.5D	TEN	Large dataset	Hybrid 2D-3D
<b>VRML implementations</b>													
Morcrette	1998	VRML											
Kim et al	1998	VRML											
Zlatanova	2000	VRML											
de Vries et al [1]	2003	VRML		only attributes									
Zhu et al	2004	VRML			only text								
Beard	2004	VRML											
Rančić et al	2005	VRML+X3D											
<b>X3D implementations</b>													
de Vries et al [2]	2003	X3D											
de Vries et al [3]	2004	X3D											
Ninsawat et al	2006	X3D											
Kumke et al	2006	X3D											
<b>Thesis implementation</b>													
Wesselingh	2007	X3D											

**Legend**

feature implemented
not sure if implemented
not implemented

Table 4.1: Comparison of prototypes

In the table (table 4.1) the twelve prototypes that are reviewed are compared, the arguments on which the comparison is made are discussed next.

1. With the 'client-server connection' is meant in the prototype a connection can be made between the 3D visualization on the client and the database on the server. This means the geometry is read from the database, transformed, and returned to the client in a XML based 3D visualization format on one hand and on the other hand, the attributes of objects can be requested by interacting with the objects in the 3D visualization on the other hand.
2. Selection can be done by use of an SQL query or in another way:
  - 2.1 With 'attribute selection' is meant is that questions can be done like "Visualize all the houses owned by Pietersen". The results are displayed in the visualization for example by colouring the objects yellow.
  - 2.2 'With SQL support' is meant that attribute selection can be done based on an SQL query or other kind of analysis (for example buffer, near) can be executed with an SQL query.
3. For each prototype is listed if data in one of the following formats can be visualized:
  - 3.1 Boundary representation or 3D features (BREP)
  - 3.2 Triangulated Irregular Network (TIN)
  - 3.3 Raster
  - 3.4 Digital Elevation Model (DEM)
  - 3.5 Extruded features (2.5D)
  - 3.6 Tetrahedral Irregular Network (TEN)
4. With 'Large dataset' is meant whether the prototypes uses methods to deal with large dataset, for example by only visualizing the data that can be seen from the viewpoint, by using Level of Detail or by using a bounding box to only request a certain area.
5. With 'Hybrid 2D-3D' is meant whether the prototype has a 2D in addition to the 3D view. The two dimensional view can support navigation within the 3D view.

The first conclusions are that all 'newer' prototypes using X3D seem to have SQL support and the possibility to do attribute selection, this is also true for the VRML based prototypes by Morcrette, Kim et al. and Zlatanova (1 point). The prototypes by Ninsawat et al. and Kumke et al support the largest number of data types (four), although the prototypes by Kim et al., Zhu et al., and Rancic et al. still support three different data types too (also 1 point). The prototype of Ninsawat et al., Wesselingh, and Zhu et al. are the only prototypes that have a Hybrid 2D 3D interface (1 point). Many prototypes offer some solution to deal with a large datasets; among them are the prototypes of Zlatanova, Zhu et al., Rancic et al., de Vries et al [2] and [3], Ninsawat et al., and the prototype by Wesselingh (1 point). All the prototypes, except the prototype by Beard offer a client server connection (1 point). The prototype made for this thesis by Wesselingh is finally the only prototype that supports the visualization of data in the TEN structure (1 point). The points are added up and included in the individual review of the prototypes in paragraph 4.4. Prototypes with more than three points will always form an example for the development of the prototype made for this thesis. Prototype with less than three points will be discarded as an example. Prototypes with three points are sometimes used, sometimes not used as an example, for each prototype, this is motivated.

## 4.4 Feasibility review of prototypes

### 4.4.1 Introduction

In the following paragraph, all the prototypes are reviewed again. Four prototypes, which form the best example for the development of the prototype of this thesis, are picked out. For each prototype, arguments are given why it should or should not be used as an example for the prototype made for this thesis. In the conclusion is stated which prototype has formed the best example to develop the prototype made for this thesis.

### 4.4.2 VRML based prototypes

The first prototype discussed by (Morcrette, 1999) is rather limited and is mainly able to visualize an extruded 2D scene with some attribute information; in addition, some simple queries can be executed. The prototype has been given only two points. This prototype is therefore not considered a good candidate to serve as an example for development of the prototype made for this thesis.

The prototype built by (Kim et al, 1998) has much 3D GIS functionality such as selection, buffer, and near. It supports different kinds of data types such as BREP, Raster, and DEM and attribute selection is possible. The prototype has three points, which should be enough to be a feasible example however; the prototype is improved by (Zhu et al, 2004) by adding a 2D view and introducing technologies such as tiling so that only a part of the scene can be loaded. Therefore, the prototype by (Zhu et al, 2004) is a better example than the prototype by (Kim et al, 1998) for the development of the prototype made for this thesis.

The prototype by (Zlatanova, 2000) is very strong in analysis of 3D data. It has functionalities as select common faces and visibility check. Using a 3D R-Tree large datasets can be processed. As the prototype made for this thesis, it uses a scripting language for creating 3D visualizations (called PERL, which is based on CGI). In Zlatanova's prototype server side CGI – PERL is used to read out all kinds of form data, nowadays this is all possible with client-side JavaScript. CGI – PERL can be compared with JavaScript and with mod\_python. Mod\_python, letting Python scripts run on an Apache server, overcomes much of the difficulties of CGI. One of the best arguments in favour of mod\_python is that it runs faster (Lowe, 2004) (The Apache Software Foundation, 2007 [1]).

A drawback which is mentioned by (Zlatanova, 2000) is that a CGI script cannot modify VRML objects, thus it cannot detect the ID of a VRML object. Another drawback of the prototype by (Zlatanova, 2000) is the software that is used to run the prototype. The development on the internet browser Netscape has stopped and the 3D player Cosmoplayer has lost popularity in favour of players as BsContact and Flux. The prototype is thus strong in analysis and has got three points but because of the many drawbacks mentioned it has not been used an example to develop the prototype which is made for this thesis.

In the first prototype by (de Vries et al, 2003) the spatial data is stored on the server in the form of a VRML file in the Boundary representation and the attribute data is stored in a database. The prototype is not considered as a good example for the development of the prototype of this thesis because the GIS functionalities are very limited (only identification) and the geometrical data is not stored in the database, in addition it has got only one point. An interesting set of technologies which haven't been used before in the development of a 3D web GIS prototype (an IIS server, the ASP programming language and an MS Access database) have however been used and the prototype does function well.

The geological visualization of Tasmania by (Beard, 2004) gave proof that VRML (thus X3D) information can be dynamically modified. In Beard's prototype, different geological themes can be turned on or off and are then added or removed from the 3D visualization. It has been given 1 point because of the different file formats it can handle. It does not have a database connection, all the data is already present in the form of VRML file. As it only has one point, it is not used as an example for the development of the prototype for this thesis although it gave proof that a 3D visualization can be dynamically modified.

The VRML / X3D prototype built by (Rančić et al, 2005) which have been given again three points is similar to the prototype of (Zhu et al, 2004). It is mainly suitable for visualization of all kind of layers for example DEMs, 2.5D data, and data in the Boundary representation. To deal with large datasets the data is stored in different Levels of Detail. The prototype does not offer input for SQL queries and does not have a 2D view or tabular data view. Because of the limited interface, the prototype is not considered as an example for the development of the prototype, which is made for this thesis.

#### **4.4.3 X3D based prototypes**

In the second prototype by de (de Vries et al, 2003) geometrical and attribute information are stored in an Oracle database in the Boundary representation. By use of an XSQL query, the data in the Oracle database is transformed to XML.

The XML is transformed to X3D for 3D visualization of the geometrical data or to a HTML table for visualization of the attribute data; both transformations are done using an XSLT. By using anchors in the X3D scene when an object is clicked in the 3D view the corresponding attributes are shown in the attribute data view.

The prototype allows the input of an SQL query so that only the requested part of the data will be visualized and in an extension of the prototype bounding box coordinates (x and y) can be input to serve the same purpose of only retrieving a certain part of the dataset. The prototype has been given three points, which makes it feasible to form an example the development of the prototype of this thesis.

One drawback of the prototype is that because it uses XSLT it will only function well in Internet Explorer and not in Firefox. XSLT is used in a different way in FireFox than in Internet Explorer so the code should be adjusted if the application has to work in Firefox too.

There are however enough positive reasons to mark this prototype as an example prototype to be used for the development of the prototype made in this thesis.

The prototype by (de Vries et al, 2004) using GML 3 is not used as an example for the development of the prototype of this thesis because it is not online now and did not work with an official WFS, although when it would work it would be a feasible example which has been given three points. The issue of combining GML 3 with a Web Feature Service has not been realized yet. Geoserver is working on supporting GML 3 now (Refraction Research Inc, 2004).

The functionality offered in the most recent 3D web GIS prototype developed by (Ninsawat et al, 2006) is extensive; the five points given underline this statement. 3D and 2D data can be extracted from a PostGIS database and are presented in a web browser with an X3D-plugin. The data is converted with a Python script and returned to the client via the web server. On the client queries can be input and are passed back via the web server to the database. Ajax technology (subparagraph 3.6.5) on the client allows the scene to be dynamically modified for example to do a selection.

In addition, the 3D data can be converted to a 2D image map and is presented on the client via a MapServer/PHP MapScript. MapServer (University of Minnesota, 2007) can render various kind of spatial data as maps, images (for example a tiff) and vector data (for example spatial data types in a database or shapefiles), on the web. The prototype is thus seen as a good example for the development of the prototype made for this thesis.

The prototype build or to be built by (Kumke et al, 2006), which have been given three points, is very promising. Various kinds of data are stored in an Oracle database as 3D features in the Boundary representation and raster images. Using Java, they are translated to X3D on the client. In addition, the client allows SQL queries to be input. However the main aim of this project is to study thermal characteristics, to do this 3D visualization is used, its main purpose is not the 3D visualization itself.

In addition, no results of the prototype can be seen online. For these reasons, this prototype has not been chosen as an example for the development of the prototype made for this thesis.

## 4.5 Conclusion

The prototypes by (Zhu et al, 2004)., the second prototype by (de Vries et al, 2003)., the prototype by (Ninsawat et al, 2006) are the best candidates to be an example for the development of the prototype of this thesis. The prototype by (Ninsawat et al, 2006) has been chosen as the main example for the development of this thesis; because it has most working features, is still under development and is the most recent prototype created. In addition it makes use of technologies which have recently become popular as Python and Ajax and has the highest number of points. In the prototype by (de Vries et al, 2003) the dataset is available as XML via a Web Feature Service and is then transformed to a 3D visualization format. This idea has been implemented as the 'X3D via GML 2' option in the prototype made for this thesis. The prototype by (Zhu et al, 2004) (based on same set of technologies as the prototype by (Rančić et al, 2005) and the prototype by (Kim et al, 1998) as JAVA, the EAI and VRML) could have been used as an example but has not been used because some of these technologies seem a bit outdated. VRML and the EAI have been replaced with X3D and the SAI. The prototype by (Beard, 2004) gave inspiration to modify the scene using JavaScript for example to hilite an object when it is selected.



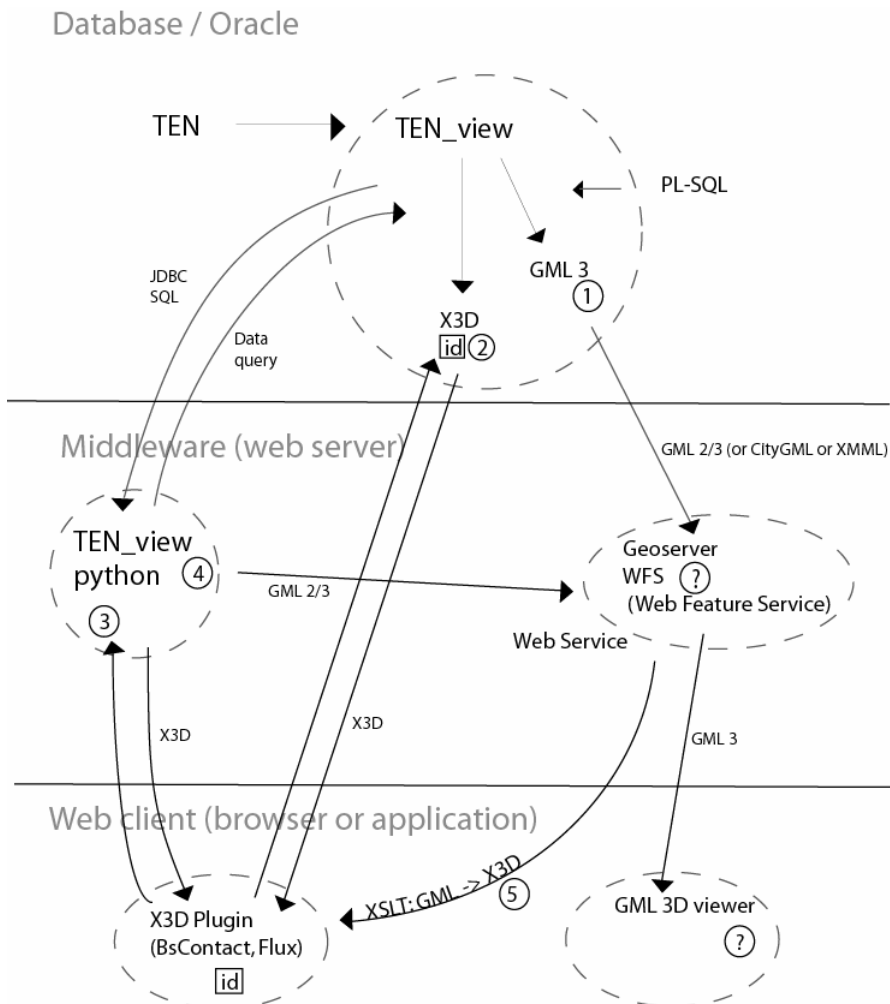
# 5. Prototype architecture

## 5.1 Introduction

Paragraph 5.2 gives the preliminary draft for the architecture of the prototype. These are the possible steps that can be carried out to create a web application for 3D visualization of a TEN in a web client. Paragraph 5.3 gives the realized architecture. The main components and the functionality of the prototype are described and illustrated in figures. Paragraph 5.4 gives a summary of list the different steps that are taken over time to develop the prototype. The techniques that are used in the prototype are described in more detail in Chapter six. An explanation on the technologies mentioned in this chapter can be found in Chapter three. An explanation on the data models mentioned in this chapter can be found in Chapter two. It is recommended to read these chapters first for better understanding of this chapter.

## 5.2 Preliminary draft

Figure 5.1: Preliminary draft of prototype architecture for TEN visualization



### 5.2.1 Introduction

The illustration in figure 5.1 gives a number of options to come from data in the database to the 3D visualization. In option 1, a PL/SQL script outputs GML or a profile on GML (CityGML or XMML) which is picked up by a WFS in the middle layer, which outputs GML. The GML can be shown in a GML viewer or is transformed for example to X3D. In option 2, a PL/SQL script directly outputs X3D. The X3D is returned to the client by a web service. In option 3 A Python script on the server in the middle layer extracts data from the database and transforms it to X3D and returns it to the client. In option 4 and 5 a Python script transforms data in GML 3 which is sent to the WFS, which outputs GML that is transformed to X3D, or the Python script directly outputs CityGML. The options that are using PL/SQL to write out directly 3D data, have not been tested because it has been chosen to expand functionality rather than testing as much different options for producing viewing output. Secondly, an option with a middle layer can easily be adopted to work with different kind of database while PL/SQL scripts are Oracle specific.

### 5.2.2 GML 3 via PL/SQL

① In the first option a PL/SQL script in the database outputs GML, or CityGML, or XMML, the last two are profiles on GML 3. A Web Feature Service in the middle layer, called Geoserver, receives data in the GML format, and returns GML 3. The data returned by the Web Feature Service can be visualized in a GML 3 Viewer. The WFS Geoserver does unfortunately not support input / output of GML 3. It supports only GML 2. As an alternative, the database could produce GML 2 as output.

There are no plugins to view GML in a web browser. This may have been related to the fact that GML traditionally is seen as data or a data exchange format and not so much a format for visualization. This has changed now since CityGML adds visualization to the GML and GML and CityGML can be visualized in applications as LandXplorer and Aristoteless. The application Aristoteless is able to retrieve data from a Web Feature Service; however, when it is called there is prompted to save the GML to the local disk so there can be made no live WFS connection yet. Another option is to let the Web Feature Service output GML 2 containing faces with face coordinates. Either the GML 2 can be viewed by a standalone application, which enables to show a Web Feature Service, or it can be visualized as plain textual XML in the browser. Alternatively, the GML 2 can be transformed to X3D using a stylesheet transformer (XSLT) and be viewed with a web browser with X3D plugin. The last option only requires JavaScript and not Python. This option has been implemented in the final prototype design. The transformation code of the XSLT is different in Internet Explorer as in Firefox. In the prototype made for this thesis it works now in Internet Explorer.

### 5.2.3 X3D via PL/SQL

② In the second option a PL/SQL script in the database directly produces X3D that is picked up by an X3D plugin in the web browser. With the object ID stored within the output X3D, attribute information of objects can be queried from the database. A web server still has to be used to show the data in the web client. The XML developer kit for Oracle (XDK) can be combined with PL/SQL or be used standalone to generate XML within the database (Oracle, 2007 [2]).

### 5.2.4 X3D via Python script on web server

③ In the third option, a Python script running on a web server in the middle layer, extracts data from the database using JDBC or ODBC. Then it transforms the data to X3D and returns it to the web browser on the client side, where it is picked up by an X3D plugin. The object ID is again sent with the X3D, so attribute information can be requested from the database by clicking objects in the X3D visualization. A lot of operations in the programming language Python mirror typical database operations and Python has some pretty good and easy to understand one-liners for composing and decomposing data structures. In PL/SQL it generally will take more effort to produce code (Russel J, 2007). This option has been implemented in the final prototype design.

### 5.2.5 GML 3 / CityGML via Python script on web server (possibly via Web Feature Service)

④ ⑤ In the fourth option, the Python script transforms data in GML 3. This is sent to the Web Feature service and then it is passed on to either a GML 3 viewer or it is transformed into X3D with a stylesheet transformer (XSLT) and picked up by a web browser with a X3D plugin (option 5). Because it has been found out that Geoserver cannot handle yet handle GML 3, an option has been made to have the Python script directly generate CityGML and have the CityGML returned to the web browser as GML / text. Then it can be saved to a local disk to view it a standalone GML application, or when a CityGML plugin comes available, it can be displayed in the web browser. Thus in the prototype, GML 3 can directly be generated from the database and returned to the user as plain XML but the GML 3 cannot be picked up and be produced by a Web Feature Service.

## 5.3 Realized architecture

### 5.3.1 Introduction

The realized architecture has been schematized in an illustration (figure 4.2). The prototype application exists of three main components; a client in this case a Firefox or IE web browser where the web application is shown to the user, an Oracle database where the data is stored and a Apache HTTP web server. An extension to Python called mod\_python is running on the web server to enable Python scripts to be run on server and have results returned to the client. All the scripts as well as the HTML pages, which make up the web application, are stored on the server. The paragraph describes the three main components of the prototype, which are:

1. Prototype website (subparagraph 5.3.2)
2. Database (subparagraph 5.3.3)
3. Web server (subparagraph 5.3.4)

In addition, the main prototype functionalities have been described (subparagraph 5.3.5).

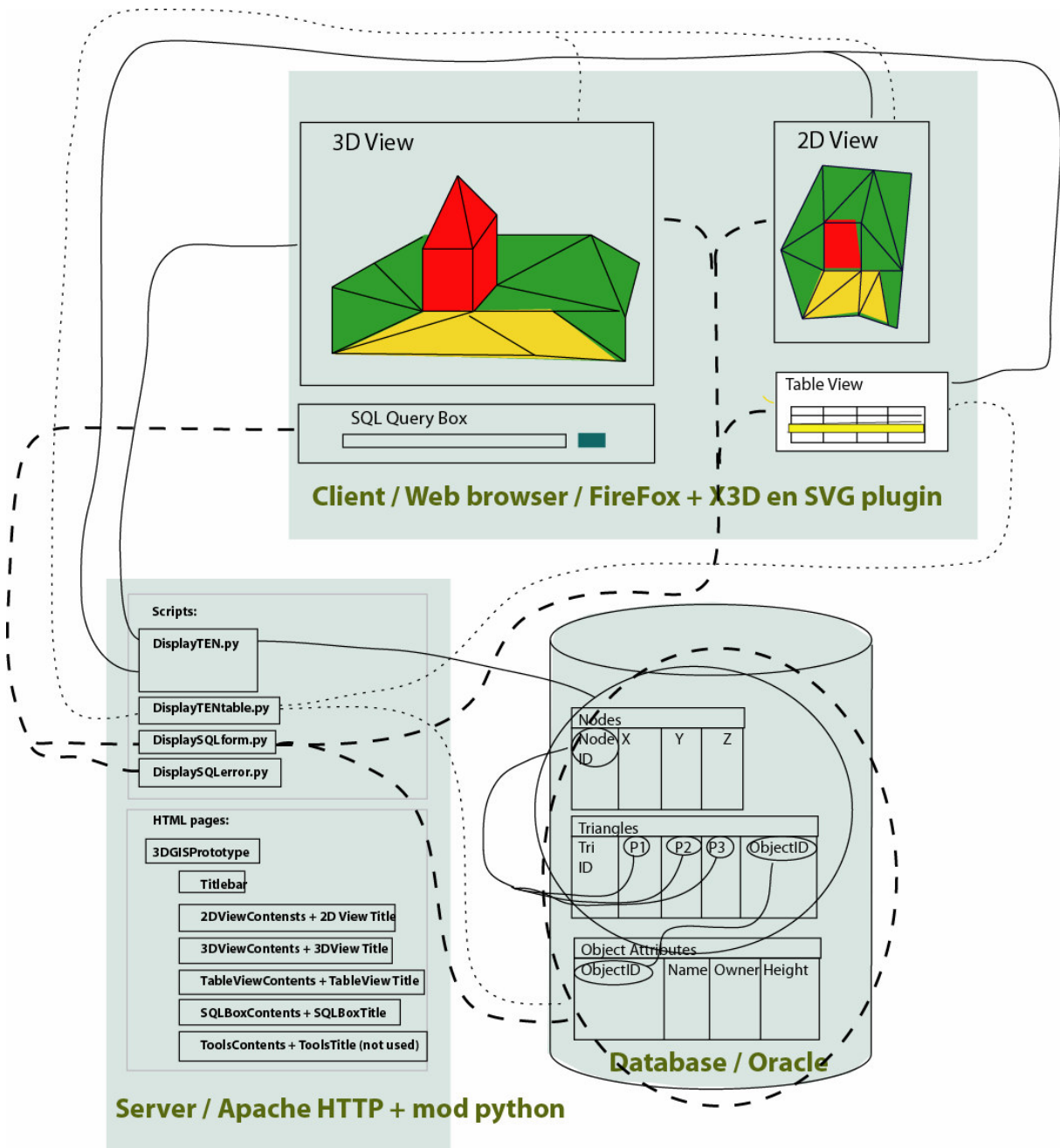


Figure 5.2: Realized architecture of TEN visualization

A short description of the contents of each component of the prototype implemented (as illustrated in figure 5.2) follows here.

### 5.3.2 Prototype website (Client / Web browser / Firefox + X3D and SVG plugin)

The client is a browser. The current web application can work with the Firefox 2 and Internet Explorer 7. The user interface of the prototype made for the thesis is one HTML page, which is split up in different frames. The frame hierarchy is displayed in the figures 5.3 and 5.4.

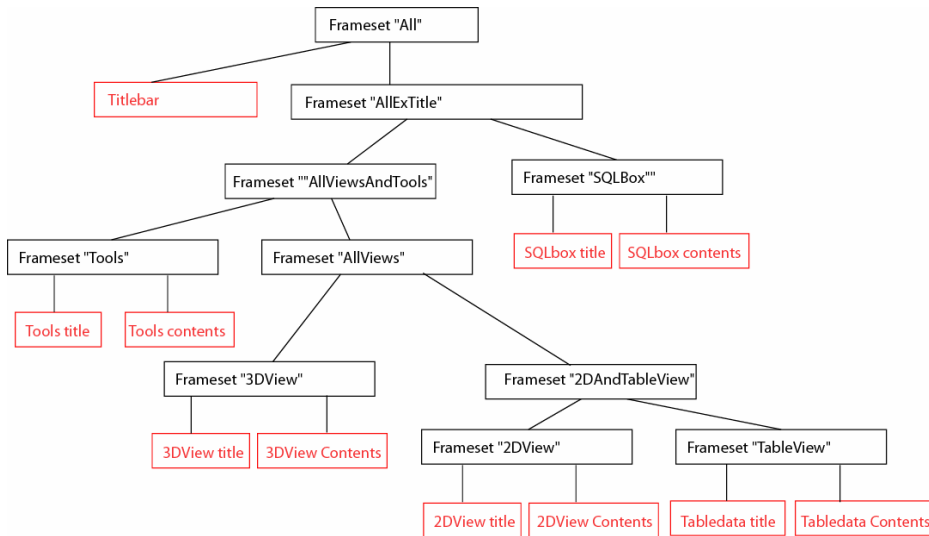


Figure 5.3: Prototype web site structure

The main page holds two frames. The first frame at the top of the page holds the title bar, while the bottom frame holds a frameset containing the rest of the frames. This frameset is divided into two framesets. At the bottom, there is a frameset holding the SQL box for input of the SQL queries and at the top, there is a frameset holding the rest of the frames. This frameset consist of two framesets. The first frameset on the left the holds the toolbox while the frame on the right holds the rest of the frames. This frameset consist of two framesets, the top frameset holds the 2D dataview, where the 2D map data can be showed and the bottom frameset holds the tabular data view, where attribute data can be shown.

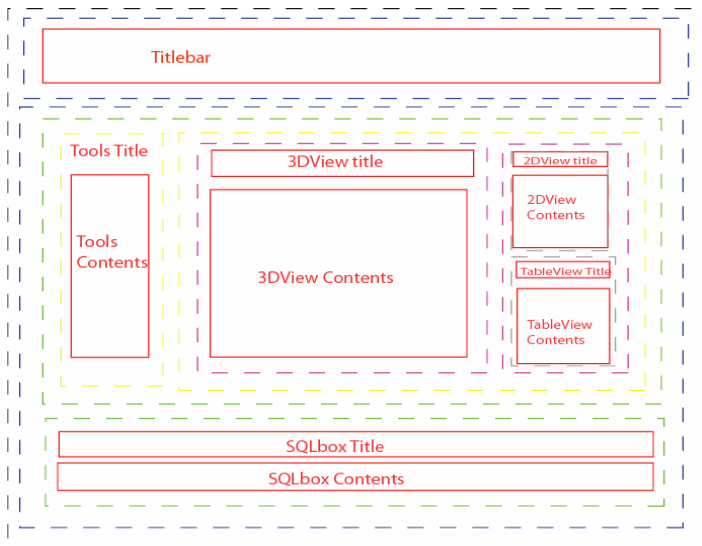


Figure 5.4: Prototype web site layout

When the 2D, 3D, or tabular view is called, a connection to the database is always made. The views are called using buttons in the titleframe of the 2D, 3D or tabular view. The data is then transformed and displayed in the contents view in one of the frames.

Anchors embedded in the 3D and 2D view; add another possibility to interact with the database. When an object is clicked, the attribute data is shown in the tabular data view.

### 5.3.3 Database (Database / Oracle)

The data is stored in an Oracle database (see subparagraph 3.5.4). Tables in the database are accessed from a Python script on the server using the PyODBC module. The Python script can transform the data from the tables into data that is suitable for visualization and return it to the client.

The database consists of three tables. The node table holds all nodes in the dataset. Each node has a unique ID and an X, Y, and Z coordinate. The table with triangles contains all triangles in the TEN dataset; each triangle has an object ID and a reference to three nodes in the object table. The object ID indicates to which object the triangle belongs too. One object consists of many triangles. The object attributes table is a table with the attributes of each object for example name, owner and height. Each object has a unique object ID. The node and triangle table are used to derive the 3D and 2D views from the 3D TIN and the 'TEN view', while the object attribute table is used to derive attribute information and to perform queries based on attribute information. In addition to a table with triangles there is a table with tetrahedra, each tetrahedron has an object ID and a reference to four nodes. The table with tetrahedra is used to derive the 3D and 2D view from the TEN.

For more detail on how the connection is made to the database is referred to 6.4.2.

### 5.3.4 Server (Apache HTTP + mod\_python)

The server is an Apache HTTP server running on a normal PC at home or work. The server directory contains a number of scripts for doing the database queries, transformations, and visualizations and for adding functionality to the web application. Most scripts are written in Python, though sometimes is also made use JavaScript. The Python script runs on the server, this is possible due to installation of the mod\_python module. The JavaScript are stored on the server too, but are executed on the client. More information on the Apache server can be found in subparagraph 3.5.5.

### 5.3.5 Prototype functionality

The black lines (in fig. 5.2) describe the 3D visualization and 2D visualization. The user pushes a button "Submit" on the client. The button calls the script "DisplayTEN.py". This script reads the geometrical data out of the database and transforms it to X3D or SVG. Finally, the data is returned to the client. Alternatively, the 3D data can be transformed to CityGML or KML; and the 2D data can be transformed to X3D. For more detail on these transformations is referred to subparagraph 6.4.3 and subparagraph 6.3.4.

The dotted lines (in fig. 5.2) represent interactivity in the objects. The user can click an object in the 3D or 2D visualization. The script "DisplayTENtable.py" is called with the ID of the object. The script queries the object attributes table with that ID and returns the record corresponding with that ID as a HTML table on the client, in the Table view. For more detail is referred to subparagraph 6.4.6.

The dashed lines (in fig. 5.2) represent visualization of a user query. The results can be visualized in 2D, 3D, or in the form of a table. The user can write a query in the SQL query box. The web application requires the query to be done on the attribute data, and for now not on the geometrical data. For example the user can do a query “Select \* from OBJECTINO where id = 2”, or “Select \* from OBJECTINFO where landowner = “Pietersen”. Python runs the query by a script within the "DisplaySQLform.py" script. The resulting IDs of the query are stored in an array variable by the script on the server. As well, the IDs of the object that are selected, and the IDs of the object that are not selected are stored in a variable. These are passed as arguments to the scripts that create the visualization ("DisplayTEN.py" and "DisplayTENTable.py"). The visualization is re-calculated, and now the selected objects will be coloured yellow. With JavaScript, the selection can be hilted without reloading the whole visualization. The scene is dynamically accessed (subparagraph 3.6.5). More detail on the attribute selection with SQL can be found in 6.4.7.

## 5.4 Prototype development process

### 5.4.1 Introduction

Paragraph 5.4.2 chronological list the steps that were taken top develop the basic functionality of the prototype, paragraph 5.4.3 list the functionalities with which the prototype was extended further.

### 5.4.2 Initial steps

The first steps that were taken over time are done to give the prototype basic functionality:

- 1 working out transformation to generate X3D

The first step taken was to work out on paper how a conversion of a table with triangles with point references and a table with points and coordinates can be transformed to an X3D file. These two tables were provided in a text file. A script was written which uses these two files as input, and outputs an X3D file. For more detail is referred to subparagraph 6.4.3.

- 2 add a database connection

In the next step, several modules were installed to test a database connection. Of these modules, PyODBC was used to connect to an Oracle database. Now the script was adapted to read data out of a database, instead from an X3D file. For more detail is referred to subparagraph 6.4.2.

- 3 work out transformations to generate CityGML and KML

In addition to a script to output X3D, a script was written to output CityGML and KML. For more detail is referred to subparagraph 6.4.3.

- 4 enable the server

In the next step, the Apache HTTP server and mod\_python were installed. Mod\_python enables Python script to be run on the Apache server. Apache is configured for running the Python scripts and Python scripts were adapted to collect the output in a string and return the 3D /2D data to the client, instead of saving it to a file. This functionality is provided by mod\_python too. For more detail is referred to subparagraph 3.5.4 and 3.6.3.

- 5 to make a design of the website of the prototype

A website was created which holds different frames. The frames in the website are a title frame, a frame for viewing the 3D data, a frame to show a 2D map overview, a frame that holds tabular data, and a frame where SQL queries can be input. For more detail is referred to subparagraph 6.3.2.

- 6 to enable feature identification

In the next step, hyperlinks were added to the X3D objects. When the 3D object is clicked, attribute information is shown in the frame with data. A script is written to convert data to an HTML table. For more detail is referred to subparagraph 6.4.6.

- 7 Work out a transformation to generate SVG

In the next step, a script was written which created 2D data. The script extracts the 2D coordinates (x and z) from the data and converts them to SVG. The SVG is output in the 2D map frame. The SVG is also made interactive just as the X3D, when an object is clicked its attribute information is shown in the Table view. For more detail is referred to subparagraph 6.4.4.

- 8 to read colour from attribute data

Up to now random colours were assigned to the objects. This was improved by including colour information in the attribute information of the data. While the output is generated in the form of X3D, KML, or CityGML, the colours are read from the attribute information and assigned to the right objects. Another small improvement was that column names are also added to the HTML table of the attribute information. For more detail is referred to subparagraph 6.4.8.

- 9 to enable SQL queries

In the next step, a SQL query box was implemented in the website. SQL queries can be made based on the attribute information for example `Select * from ObjectInfo where landuse = "grass."` The selection is highlighted in the data by colouring it yellow. For more detail is referred to subparagraph 6.4.7.

#### **5.4.3 More additions to the prototype development**

More steps were taken to improve and extend the prototype:

- 1 GML 2 via WFS visualization added

A feature was added to read the data from a Web Feature Service, which was one of the user requirements. The data is returned as GML 2. Now the data can be transformed to X3D using a stylesheet transformer (XSLT). This option does not involve Python only JavaScript. This option was only implemented for the small dataset and was not further extended as generating data by a Python script is less complex and works as well in Internet Explorer and Firefox. For more detail is referred to subparagraph 3.3.

- 2 Visualization output in X3D, KML and CityGML is further improved

The orientation of the face (inside or outside) is considered in the visualization output. The colour and transparency information are not only added to X3D, but also to KML and CityGML. The KML visualization is transformed to local system in the Netherlands.



Points with balloons are added to KML, to make identification of features easier. Finally, viewpoints are added to X3D to be able to jump to different objects in the data. From the object in the 2D view (SVG) can be jumped to the corresponding object in the 3D view (X3D). For more detail is referred to subparagraph 6.4.8.

### 3 Prototype help added

A help file has been added to the prototype, which gives information about the different options, especially about navigating in 3D. A FAQ and results of the enquiry about the prototype has been added too. A link has been added from where appropriate plugins can be downloaded, and technical standards used can be viewed by clicking another link. Help on the prototype application was requested by test users and thus is one of the user requirements. The prototype help can be accessed from the prototype website (<http://www.3dwebgis.nl>).

### 4 Dynamical selection added

The 3D view and 2D view are embedded to make them accessible from JavaScript. Selection on the 2D and 3D view can now be done with JavaScript, which accesses the trees of X3D and SVG, without having to reload the whole file again; this is one of the user requirements. The jump to viewpoint option has been rewritten in JavaScript too so again the whole file does not have to be reloaded. More details can be found in paragraph 3.6.5.

### 5 The prototype has been enabled to deal with a large test dataset.

A large test dataset is provided of a suburban neighborhood of Rotterdam. To prevent long loading times the user can choose a percentage of the dataset to be loaded on the toolbox, in addition the user can specify a bounding box to request only a specific area of the dataset. The visualized area in the 2D and 3D view and the attribute data that is displayed are made dependent on percentage and bounding box. Different algorithms were written to increase the speed of calculation of the visualization.

Two new algorithms were created. The 'faster' algorithm puts more steps in one loop and takes out some unnecessary loops when reading out points. Loading time has been deduced more than three times. In the 'Joinquery' algorithm, the nodes and triangles are queried in one query (in the other algorithms they are queried in two separate queries). The 'faster' algorithm is about 1.5 times faster than the 'Joinquery' algorithm however, the advantage of the 'Joinquery' algorithm is that options are added to request only a part of the dataset by input of a percentage or a bounding box. In addition should be noted that the algorithms are only tested on one dataset with a fixed size, the speed difference may vary with datasets of different size. In future research, the options to load only a part of the data can be added as well to the 'faster' algorithm and should be measured which algorithm performs better. The bottlenecks of each algorithm can then be detected to create an even faster algorithm than the existing ones. More details can be found in paragraph 6.4 and in Appendix 2.2

### 6 Visualization settings added

Before requesting the dataset the user can input some choices namely which dataset to visualize (small dataset or big dataset), in which format to visualize (X3D, KML, CityGML), to show or hide air tetrahedra, in which model to visualize (TEN, 3D TIN, or 'TEN view') and which algorithm to use. More details can be found in subparagraph 6.2.5.

# ***6. Prototype components, datasets, and techniques***

## **6.1 Introduction**

First, this chapter describes the components of the prototype in paragraph 6.2. Secondly, the used datasets are overviewed in paragraph 6.3. In paragraph 6.4, the techniques are described that are implemented in the prototype. In addition, a few techniques that were in the project proposal but have not been used are described in paragraph 6.5. The main components are the:

- 1 Titlebar,
- 2 Toolbox
- 3 3D view
- 4 2D View
- 5 Table View
- 6 SQL Query box

The used datasets that are described are the:

- 1 Small test dataset (dataset of one house on a plain)
- 2 Large test dataset (dataset of a suburban neighborhood of Rotterdam)

The techniques that are described are:

- 1 Database connection
- 2 3D data transformation
- 3 2D data transformation
- 4 Attribute data transformation
- 5 Feature Identification
- 6 SQL query
- 7 Colour
- 8 Transparency
- 9 Viewpoints
- 10 Navigation

An explanation of the technologies and applications mentioned in this chapter is given in Chapter three. An explanation of the data models mentioned in this chapter is given in Chapter two. It is recommended to read these chapters first for better understanding of this chapter.

## **6.2 Prototype components**

### **6.2.1 Introduction**

The prototype components are a titlebar and a toolbox, the 3D visualization, the 2D visualization, a table with attribute data and a box to enter SQL queries. Subparagraph 6.2.2 shows images of the whole prototype, while subparagraph 6.2.3 till 6.2.8 show an image of each component of the prototype with an explanation.

### **6.2.2 Full website look**

Figure 6.1 shows the prototype made for the thesis on initial loading. The main components are a titlebar with general information, a toolbox, a 3D view to display the 3D visualization of the TEN data, a 2D View for displaying the 2D visualization of the TEN data, a Data view for displaying attribute information and a SQL query box for the input of SQL queries.

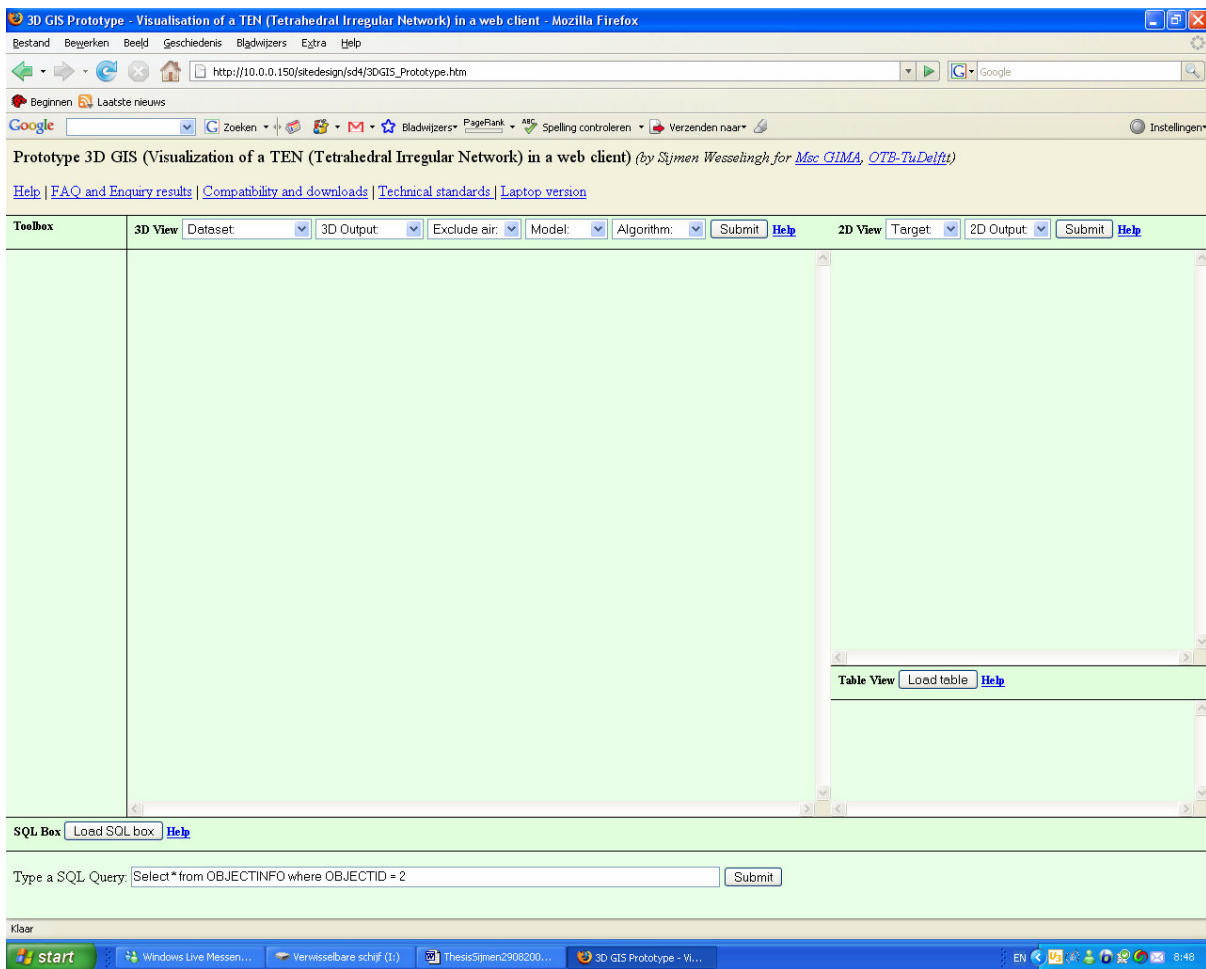


Figure 6.1: Web application preview (empty) (from prototype website)

Figure 6.2 shows what the web application will look like when the visualized in 2D and 3D and the attribute data is loaded.

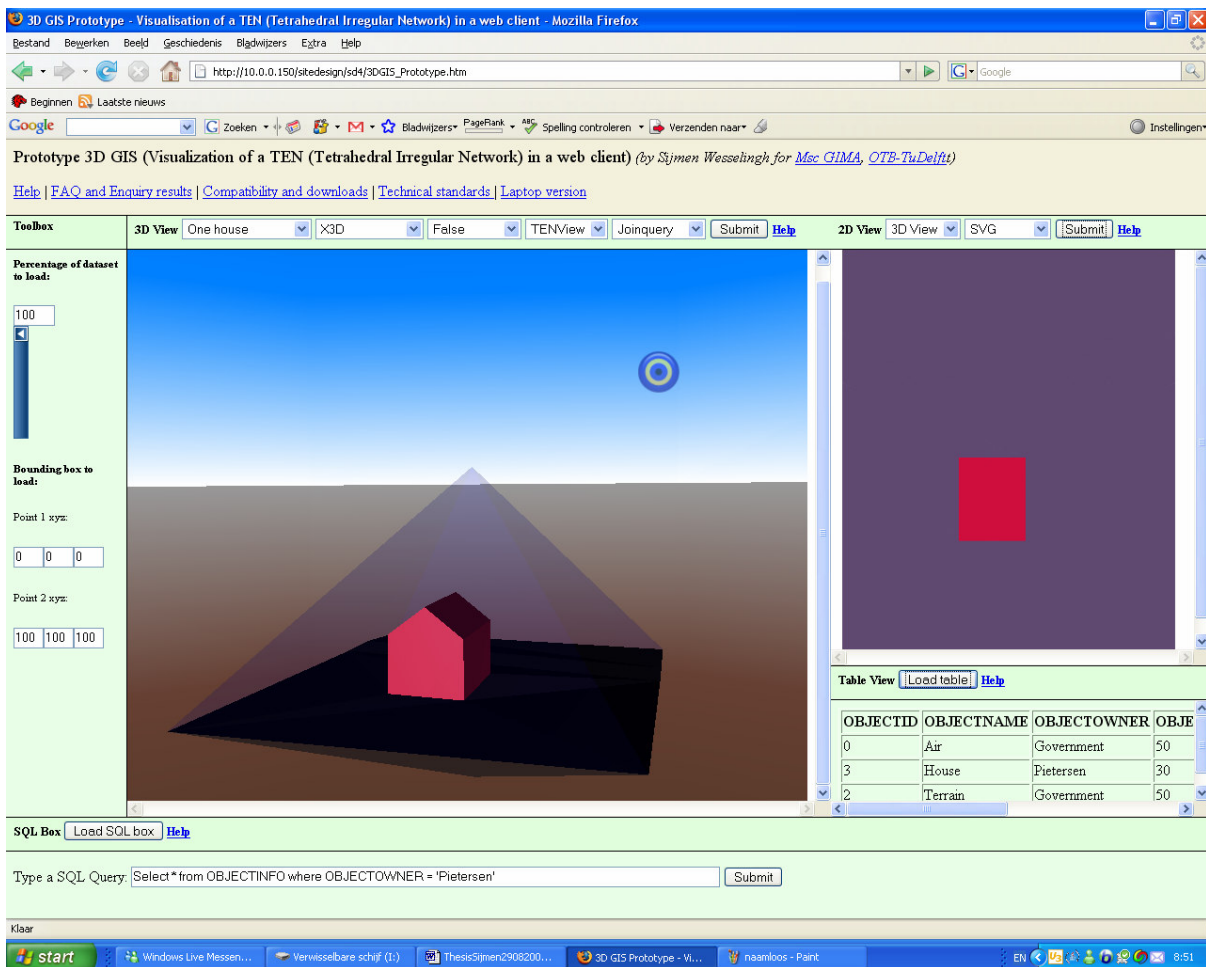


Figure 6.2: Web application preview (with data) (from prototype website)

### 6.2.3 Titlebar (for general information)



Figure 6.3: Titlebar (from prototype website)

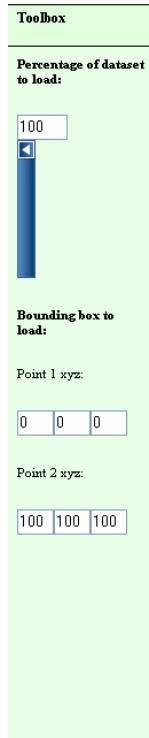
The titlebar (figure 6.3) gives the title, subtitle, and author of the prototype, which are:

1. Prototype 3D GIS
2. Visualization of a TEN (Tetrahedral Irregular Network) in a web client
3. by Sijmen Wesselingh for Msc. GIMA and OTB-TU Delft

In addition, the titlebar has five hyperlinks. They give explanation about the purpose of the application and the control of the application in the interface.

The “Help” gives all kind of information about the use of the different components of the prototype. The most information is given how to navigate the 3D visualization. The “FAQ” are the Frequently Asked Questions and their answers off course. The “Enquiry results” give the results of the enquiry, which had been carried out to improve the prototype. “Compatibility and downloads” indicates which plugins and applications can be downloaded to make the prototype work. “Technical standards” give a list of standards that are used to make the prototype work. In the laptop version of the prototype all frames are scrollable to adhere to the smaller screen.

## 6.2.4 Toolbox



The screenshot shows a web interface titled 'Toolbox'. It contains several input fields and a slider. At the top, there is a 'Percentage of dataset to load:' label followed by a text input field containing '100' and a vertical slider bar. Below this is a 'Bounding box to load:' section with two rows of 'Point xyz:' labels. The first row has three text input fields, each containing '0'. The second row has three text input fields, each containing '100'.

The toolbox (figure 6.4) is used to offer some extra functionality. When the 'Joinquery' algorithm is used (paragraph 6.4), there can be specified how much percent of the dataset has to be loaded, a percentage between 0 and 100 has to be chosen. In addition two coordinates can be input to specify a 3D bounding box, where only the elements that fall inside the box are loaded. The bounding box input should be two coordinates between (0, 0, 0) and (100, 100, 100). To navigate the 3D visualization use the mouse buttons, control buttons and options under right click.

Figure 6.4: Toolbox (from prototype website)

## 6.2.5 3D View for 3D Visualization

Figure 6.5 displays what the 3D visualization of the TEN dataset looks like in X3D. The dataset can be panned, zoomed, and rotated as wished. For orientation and to increase the beauty of the 3D visualization, a horizon with a sky and ground are added. Before displaying the visualization, by using the submit button in the 3D View toolbar in the top of the image, different visualization options can be chosen:

- With the dataset option can be chosen which dataset to visualize. Now there is a small dataset of one house (subparagraph 6.3.3), and a large dataset of a suburb of Rotterdam (subparagraph 6.3.4).
- With the 3D output option can be chosen in which format to output the 3D data; this can be in X3D which is displayed in the web browser (as previewed in figure 6.5), in KML which can be displayed in Google Earth (as previewed in figure 6.6) or in CityGML which can be displayed in a CityGML viewer application (as previewed in figure 6.7 and 6.8) or as textual XML code in the web browser. As a last option the dataset can be visualized in X3D via GML2. The dataset is now requested with a Web Feature Service (works only with the small test dataset for now).
- With the exclude air option can be chosen whether to exclude air tetrahedra when it is set to true. This makes it easier to identify objects, which are surrounded by air, or to include air tetrahedra, so the user will be able to see air, which might enhance the 2D or 3D visualization.
- With the model, option can be chosen to visualize the data from the TEN structure or the 'TEN view' structure. If the TEN structure is chosen, all triangles of the TEN are visualized, if the 'TEN view' structure is chosen only the constrained triangles, the triangles that mark the boundary between different objects, are visualized. If the 3DTIN data model is chosen only the boundary of objects are visualized, a 3D TIN representation of a TEN is quite similar to a 'TEN view' representation (the difference is explained in paragraph 2.5 and can be observed in figure 2.9). The 3D TIN option is disabled currently.
- With the algorithm, option different algorithm can be chosen to derive the 2D or 3D visualization. In the 'Original' and 'Faster' the node and face data are read-in in a separate query and the whole dataset is loaded. If the 'Joinquery' algorithm is chosen, the nodes and faces are read-in in one (joined) query and the user can specify a percentage and a bounding box to load only a part of the dataset.

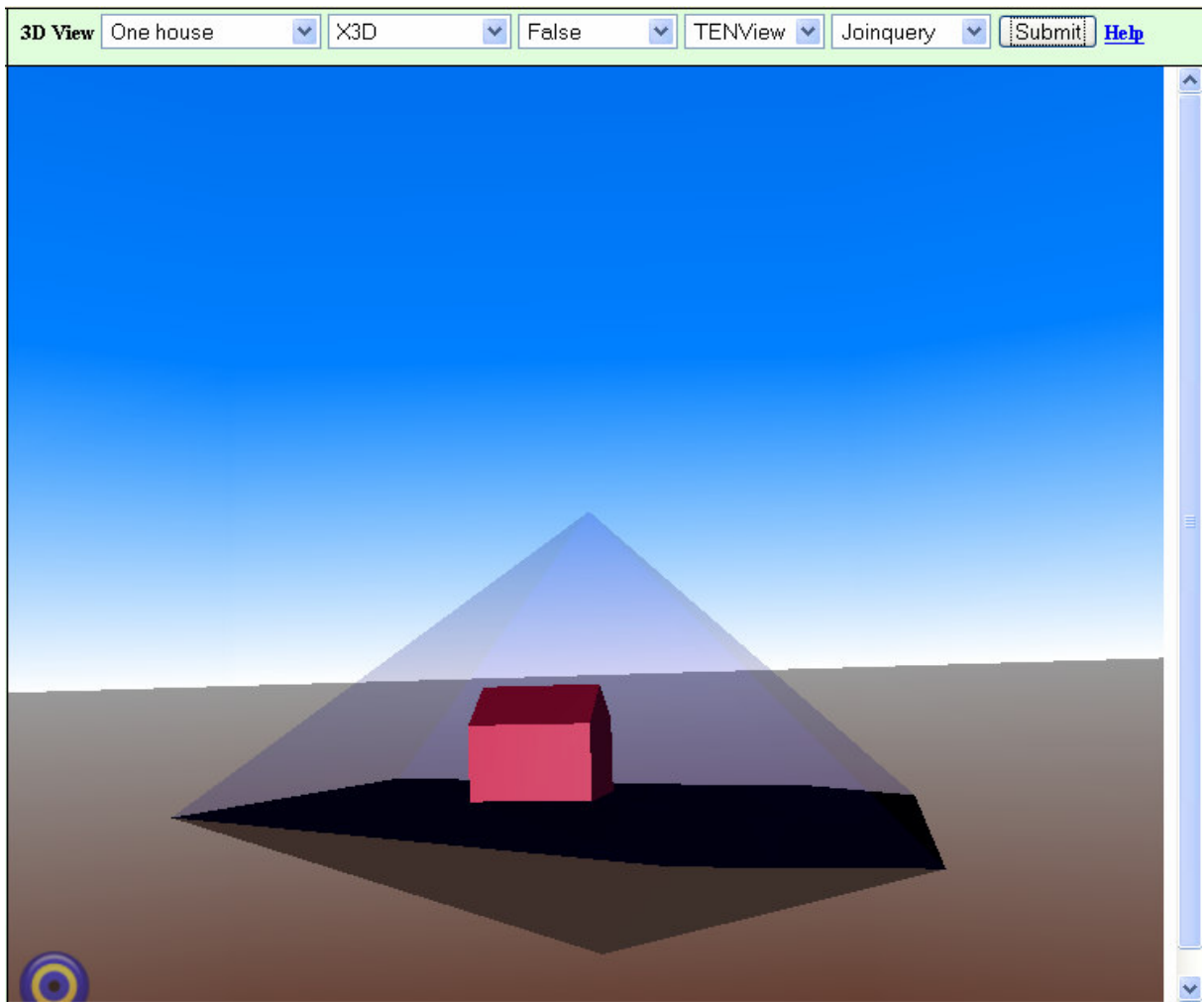


Figure 6.5: 3D visualization in X3D (from prototype website)

In figure 6.6 is displayed what the 3D visualization of the small test dataset looks like in KML. The differences with X3D is, that there is no difference is made between specular, reflective and emissive colour, outlines cannot be completely removed in KML, and satellite imagery will support the visualization within KML, while in X3D there is just a 'standard' background.

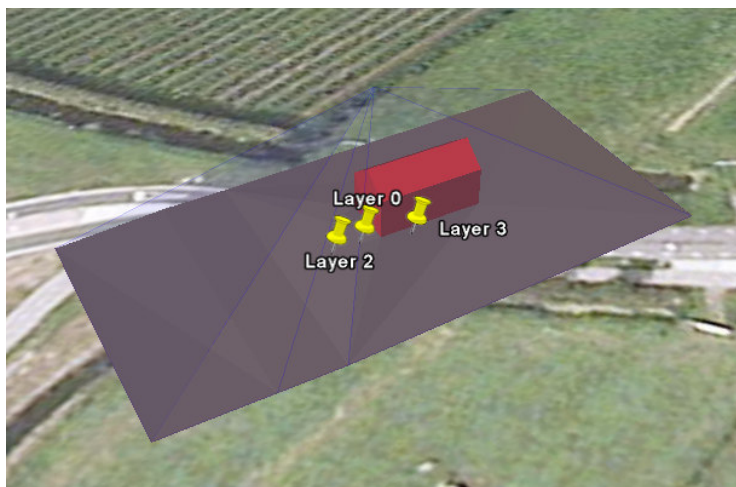


Figure 6.6: 3D visualization in KML / Google Earth

In the figures 6.7 and 6.8 is displayed is what the 3D visualization of the small test dataset looks like in CityGML. The GML is returned as XML text in the browser but can be saved to disk and opened in the CityGML viewer LandXplorer or Aristoteless. CityGML does have the ability to set different kind of colours like specular, reflective, and emissive colour and transparency. In the tested version of LandXplorer (v2.5.0.10925) transparency is not supported (fig 5.7); in the tested version of Aristoteless (1.0 alpha build) transparency is not turned on as default but can be enforced (fig 5.8).



Figure 6.7: 3D visualization in CityGML in Aristoteless (without air tetrahedra)

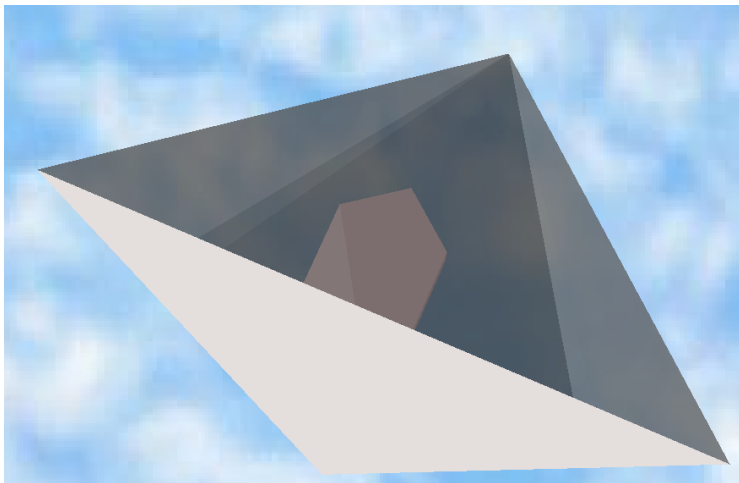


Figure 6.8: 3D Visualization in CityGML in Aristoteless (with transparent air tets)

In the 3D visualization, objects can be identified by clicking on them. In X3D the record is returned in the tabular data view, in KML tabular data is returned as a page within the Google Earth application. When objects are selected using an attribute or analysis query, they are hilited yellow in the 3D visualization. Transparent objects as air can be made very transparent as in the KML and X3D picture (figures 6.5 and 6.6) or are discluded from the visualization as in the CityGML picture (figure 6.7). The last option allows in this example the house to be identified, as the air tetrahedra are not in the way anymore. How data are prepared for 3D visualization is described in subpargraph 6.4.3 about 3D data transformation.

### 6.2.6 2D View for 2D Visualization

In figure 6.9, the 2D visualization is displayed. The 2D visualization of the data is displayed as SVG. The 2D visualization is just a 3D visualization where the z coordinate is not included (height). In the 2D visualization, when some objects are described earlier than others, they can overlap the other object in the 2D visualization. For example, the terrain feature can be placed on the top of the house, which makes the house invisible, therefore an algorithm has been written which looks at the maximum height of each objects and then sorts the SVG objects in order of maximum height.

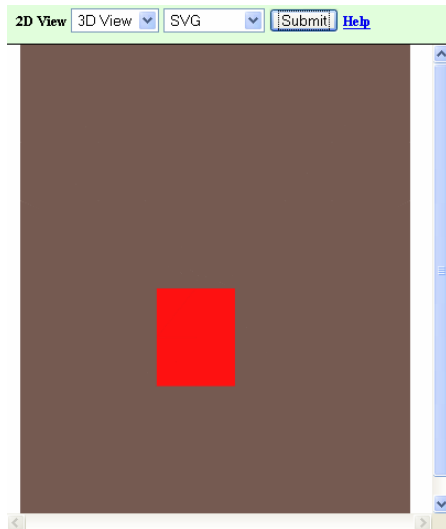


Figure 6.9: 2D visualization in SVG (from prototype website)

In the 2D visualization, objects can be identified by clicking on them and the record is returned in the tabular view, when the target in the 2D view is set to “Data View”. If the target in the 2D view menu is set to ‘3D view’, when an object is clicked in the 2D View, there will be jumped to the corresponding object in the 3D view. If the (2D) output format is set to SVG, the visualization is displayed in the 2D graphics format SVG, which looks the most neat. If the output format is set to X3D the data is showed in X3D. Heights are removed in the 2D X3D visualization to get a flat 2D picture. The 2D visualization in X3D looks less neat, but is more compact because in X3D is made us of the IndexedFaceSet which collects multiple triangles in one group, in SVG a separate is used for every triangle.

How the data is prepared for the 2D visualization is described in subparagraph 6.4.4 called 2D data transformation.

### 6.2.7 Data view for attribute data and feature identification

In table 6.1 the view of the attribute information is displayed. Because the information does not fit entirely in the window, a slider bar is used. The attribute information is displayed in the form of a HTML table. The columns names are made bold. If the ‘load table’ button is clicked all records are loaded into the table.

<b>OBJECTID</b>	<b>OBJECTNAME</b>	<b>OBJECTOWNER</b>	<b>OBJECTHEIGHT</b>	<b>COLOUR</b>	<b>TETTRANSPARENCY</b>
0	Air	Government	50	#0000FF	0.90
3	House	Pietersen	30	#FF1111	0.00
2	Terrain	Government	50	#755A51	0.00

Table 6.1: Table visualization in HTML (from prototype website)



If an object in the 3D view or 2D view is clicked, only the record that corresponds to the object is shown, as displayed in table 6.2.

OBJECTID	OBJECTNAME	OBJECTOWNER	OBJECTHEIGHT	COLOUR	TETTRANSPARENCY
3	House	Pietersen	30	#FF1111	0.00

Table 6.2: Table visualization of one record in HTML (from prototype website)

How data from the database is converted into attribute information, is described in subparagraph 6.4.5 about ‘Attribute data transformation’. How features are identified, when an object is clicked, is described in paragraph 6.4.6 about ‘Feature identification’.

### 6.2.8 Query window for a SQL query

In figure 6.10 the query window is displayed.

Figure 6.10: SQL query inputbox (from prototype website)

An SQL query can be input in the field on the bottom of figure 6.10. It is submitted when the button is pressed. After pressing the button, the query will be printed in the window, and two links will display (figure 6.11), one to execute the query dynamically, which means only queried features are modified with JavaScript. Secondly, the query can be executed statically, which means the entire visualization is recalculated with the query results as input.

Figure 6.11: SQL query result (from prototype website)

A SQL query can be written to query the data based on attribute information, for example select objects that have the landuse ‘grass’. In this case, the query is executed on the attribute information and can be visualized in the 3D view, 2D view and tabular data view. A predefined query is given, this query is formulated as follows: “Select \* from OBJECTINFO where OBJECTOWNER = “Pietersen”. This query makes clear that the name of the attribute table the query is executed on is “OBJECTINFO”. The user can display the attribute data of the selected dataset in the Table View to get an idea on which field names and which field values the query can be done.

In a more advanced prototype, analysis queries as buffer and near and queries on the topological and geometrical data may be supported too.

## 6.3 Datasets

### 6.3.1 Introduction

First is described what an ideal TEN dataset would look like, which means all the advantages of the TEN are exploited (subparagraph 6.3.2). Next is illustrated and described what the small test dataset of one house (subparagraph 6.3.3) and the large test dataset of a suburb of Rotterdam look like (subparagraph 6.3.4).

### 6.3.2 Ideal dataset

For conformance to an ideal complex TEN dataset, the dataset should at least contain an air and ground tetrahedra to fill up the whole space, a soil layer, a tunnel and a bridge, some “flat” topographic “surface” objects like roads and railroads and some elevated objects like some houses and a church. Finally some overhanging cliffs would be nice and some air tetrahedrons containing a plane flight route. Making such a dataset will probably require a lot of work

Objects as the tunnel and bridge and the air corridor cannot be modeled with just 2.5D. They are more complex than that and have multiple layers at different elevations. Rough, leveled terrain can usually be modeled with a 2.5D TIN but in the case of an overhanging cliff there would be multiple z coordinates for one x, y coordinate that is not possible with a 2.5D TIN.

The TEN structure, which divides all of the terrain, air, and objects in tetrahedra, offers a good solution for the problems that occur with 2.5D. Every object is composed out of tetrahedra. Each tetrahedron consists of four 3D coordinates.

Figure 6.1.2 gives a schematic overview of an ideal dataset.

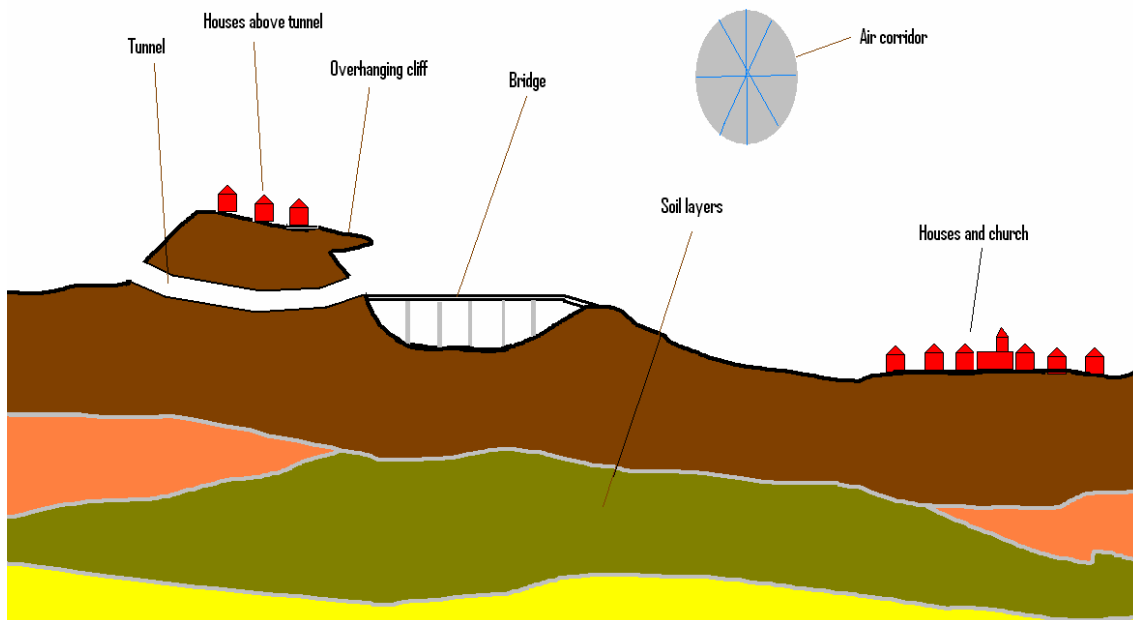


Figure 6.12: Schematic representation of an ideal TEN dataset

### 6.3.3 Small test dataset

The dataset that currently is supplied by the TUDelft is much simpler (fig 6.13). The current small test dataset consist of a rough terrain with a house; it holds tetrahedra for the terrain, the house, and the air. For writing and initial testing of the prototype made for this thesis, this dataset has been used. Because this dataset gave good results, later the results were tested with a more complex dataset that represents a suburban neighborhood of Rotterdam. In this TEN dataset, tetrahedrons are reaching high in the air, and deep underground, like in the right picture of figure 6.13. This dataset has some elements of the ideal TEN dataset as a soil and ground layer and a topographic object. To get a realistic view of the dataset, air tetrahedra can be hidden.

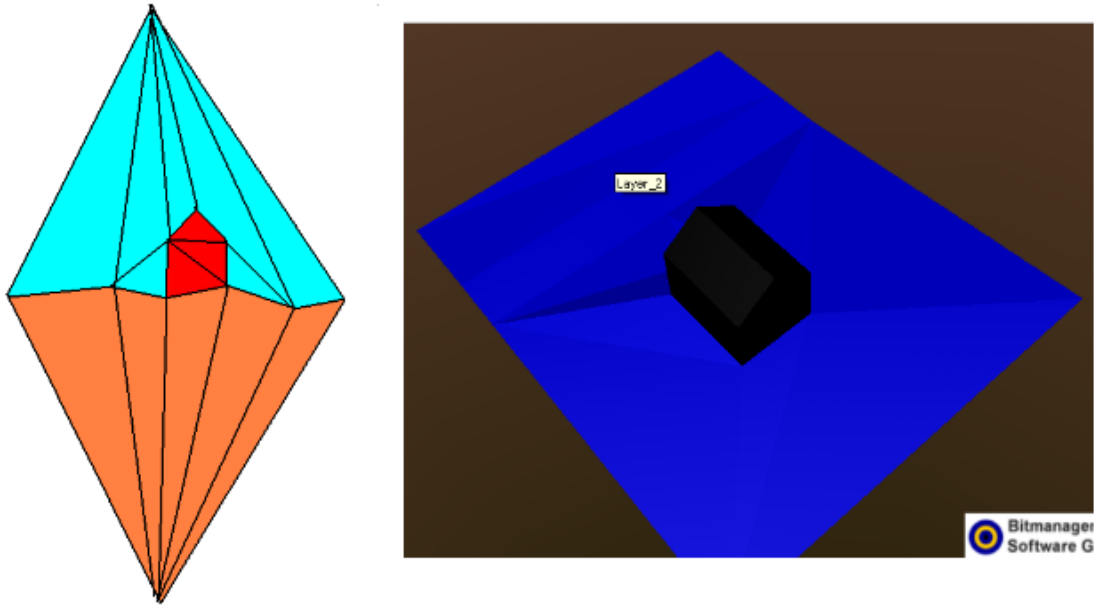


Figure 6.13: Representation small test dataset (right image from prototype website)

### 6.3.4 Large test dataset

Figures 6.14 and 6.15 show a part of the dataset of a suburb of Rotterdam. In the case a 3D box of  $(60, 60, 0) (100, 100, 100)$  is cut out of a box of  $(0, 0, 0) (100, 100, 100)$ . All the buildings which are longer than 10m are selected in the 3D, 2D (figure 6.15), and tabular view (figure 6.14 and 6.15 and table 6.3) (in yellow). Unfortunately, as can be seen in Figure 6.14., the Rotterdam dataset, that was produced using TetGen, is "tilted". In a new version of TetGen this can be remedied, but time was too short to try this out for this thesis.

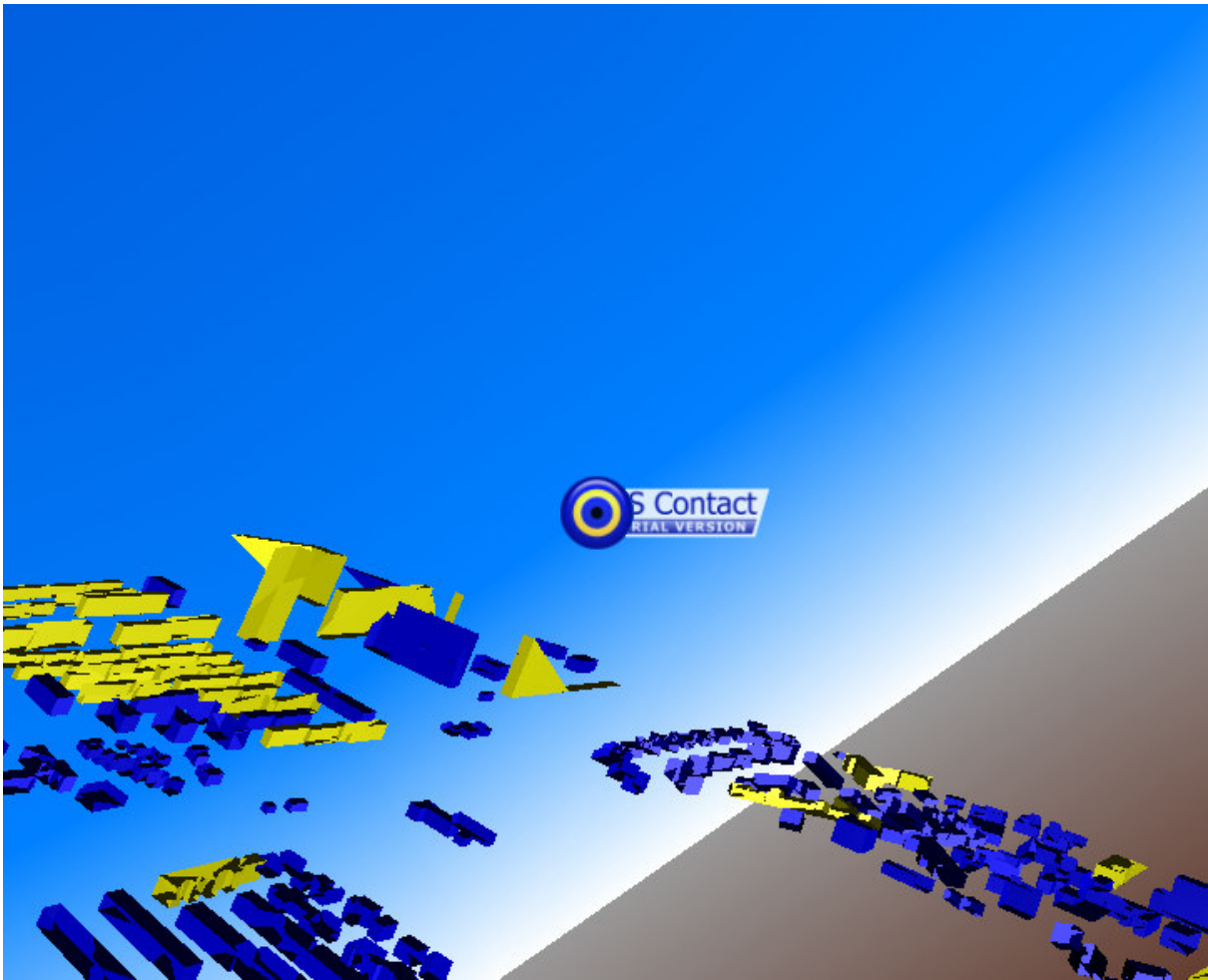


Figure 6.14: Selection of buildings longer > 10 m in 3D (from prototype website)

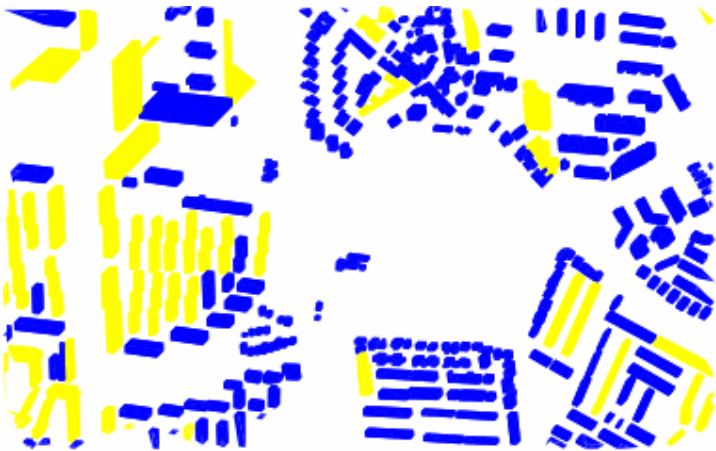


Figure 6.15: Selection of buildings longer > 10 m in 2D (from prototype website)

OBJECTID	OBJECTNAME	OBJECTOWNER	OBJECTHEIGHT	COLOUR	TETTRANSPARENCY	HEIGHT	VOLUME
213	object213	Nogdrie	14	#0000FF	0.00	14.30564598	300297000000.0
216	object216	Eenmans	4	#0000FF	0.00	4.180543891	79.18563015
359	object359	Viergoed	4	#0000FF	0.00	3.844982626	65.79706584
381	object381	Eenmans	3	#0000FF	0.00	2.535574564	43.20900867
402	object402	Tweebrug	3	#0000FF	0.00	2.949514759	989.2059299
417	object417	Tweebrug	4	#0000FF	0.00	4.352819179	3839424900.0
441	object441	Eenmans	3	#0000FF	0.00	2.672004705	44.7133116
470	object470	De Vijfde	3	#0000FF	0.00	2.559105272	39.13429921
471	object471	Eenmans	4	#0000FF	0.00	3.886515375	64.29147657
480	object480	De Vijfde	7	#0000FF	0.00	7.490330775	196916000000.0
514	object514	Viergoed	14	#0000FF	0.00	14.10743011	2261136655.0
603	object603	Nogdrie	5	#0000FF	0.00	5.126709804	674648000000.0
664	object664	Viergoed	2	#0000FF	0.00	2.204597241	268719000000.0
769	object769	Viergoed	3	#0000FF	0.00	2.584864068	40.94928543
783	object783	Nogdrie	3	#0000FF	0.00	3.390978313	84.77019115
825	object825	De Vijfde	4	#0000FF	0.00	4.114655285	450.6799473
867	object867	Tweebrug	2	#0000FF	0.00	1.803471336	354823484.2
948	object948	Nogdrie	6	#0000FF	0.00	5.767953097	53395371368.0
170	object170	De Vijfde	7	#0000FF	0.00	7.088342787	458952000000.0
202	object202	Tweebrug	2	#0000FF	0.00	2.297070361	589700000000.0
253	object253	Nogdrie	3	#0000FF	0.00	2.531093386	1732421096.0
258	object258	Nogdrie	3	#0000FF	0.00	3.105529822	142587000000.0
275	object275	De Vijfde	5	#0000FF	0.00	5.028505352	54.37818861
303	object303	Nogdrie	8	#0000FF	0.00	8.164537732	119384000000.0

Table 6.3 : Selection of buildings longer > 10 m in table (from prototype website)

## 6.4 Implemented techniques

### 6.4.1 Introduction

This paragraph describes the techniques that are used to make the prototype work. Subparagraph 6.4.2 describes how a connection is made with the database. The next paragraphs describe how to transform 3D data from the database into a 3D view on the users screen, in different output format as X3D, CityGML and KML (subparagraph 6.4.3), how to transform the 3D data from the database to a 2D map as SVG (subparagraph 6.4.4) and how to transform attribute data into a HTML table (subparagraph 6.4.5).

The next paragraphs describe how features can be identified in the prototype (6.4.6), how SQL queries can be executed based on attribute data, and how the result of the query is visualized. The last paragraphs describe how colour (subparagraph 6.4.8) and transparency (subparagraph 6.4.9) can be added to the data, how viewpoints can be added to the data and how they can be accessed (subparagraph 6.4.10), and how to navigate around the data (subparagraph 6.4.11).

### 6.4.2 Database connection and data query

A connection is made to the Oracle database using the module PyODBC within Python, which uses the ODBC protocol to connect to the database. The PyODBC-connection-line as well as the PyODBC-connection-lines where the parameters are filled in for connecting to the small and large test dataset are found in Appendix 6.1. After the connection is made, a cursor is created. The data can be requested with three different algorithms the 'Original', 'Faster', and 'Joinquery' algorithm. A general explanation of these algorithms and their differences are explained next.

In the ‘Original’ and ‘Faster’ algorithm at first all the triangles or tetrahedra are read into one array. When the user only would like to select only a subset of this data then still all nodes have to be loaded and the nodes that belong to the requested triangles or tetrahedra have to be filtered from the node array. Alternatively, multiple node queries could be used to request the nodes for each triangle or tetrahedron for example: “Select \* from NODE\_TT where NodeID = 4 or NodeID = 7 or NodeID = 10” and “Select \* from NODE\_TT where NodeID = 12 or NodeID =18 or NodeID = 25” etc. However, multiple queries will slow down the whole operation much. Another alternative would be to try to query all the required nodes in one query, for example “Select \* from NODE\_TT where in (4, 7, 10, 12, 18, 25)” however the number of nodes that can be specified is limited. Therefore, when these queries are used, all the data is requested at once in two queries, one for the faces, and one for the nodes. Text insertment 6.1 displays the SQL queries how to select the constrained triangles (in the case of visualization of the ‘TEN view’) or the tetrahedra (in the case of visualization of the TEN) and the nodes (in both cases).

```

"Select * from CONSTRAINEDTRIANGLE_WFS"
"Select * from TETRAHEDRON_TT"
"Select * from NODE_TT"

```

Text insertment 6.1: SQL select queries

The biggest difference between the ‘Original’ and ‘Faster’ query is that when in the ‘Original’ query a node is being looked up the whole table is scanned for the node with the corresponding node id. In the ‘Faster’ algorithm is made use of the fact that the nodes in the node table are numbered in a normal range (1, 2, 3,4,6,7 etc.), so the node id can be used as an index number on the node table.

In the ‘Joinquery’ algorithm, the triangles or tetrahedra and node data are read in, in one joined query. For each triangle or tetrahedron that is requested its nodes and its coordinates are requested simultaneously. This makes it possible to request only a subset of the data without having to retrieve all the data at once. The user can specify a percentage of the dataset to load. The percentage is translated to a number of triangles or tetrahedra that have to be loaded. In addition, the user can specify a bounding box. Only the faces that fall within this bounding box are loaded (Orlinska, 2005).

.The query is a bit different for the ‘TEN view’ where the triangles are requested, as for the TEN where the tetrahedra are requested. A parameter is added to the query too which indicates whether air tetrahedra are included or not. The air objects are excluded in the sample query in text insertment 6.2 and 6.3, and the full dataset is requested, with a bounding box covering the whole dataset (100%). Text insertment 6.2 displays the query for getting the constrained triangles or the ‘TEN view’ data model.

```

select * from (select orientation, nid1, nid2, nid3, objectid,
    n1.xcoord as x1, n1.ycoord as y1, n1.zcoord as z1,
    n2.xcoord as x2, n2.ycoord as y2, n2.zcoord as z2,
    n3.xcoord as x3, n3.ycoord as y3, n3.zcoord as z3 from
    (select substr(f.tricode, 1,1) as orientation,
    ltrim(substr(f.tricode, 2,10),'0') as nid1,
    ltrim(substr(f.tricode, 12,10),'0') as nid2,
    ltrim(substr(f.tricode, 22,10),'0') as nid3,
    ltrim(substr(f.tricode, 32,10),'0') as objectid
    from constrainedtriangle_wfs f) triangle,
    node_tt n1, node_tt n2, node_tt n3
    where n1.nid = nid1 and n2.nid = nid2 and n3.nid = nid3 and
n1.xcoord > 23226 and n1.ycoord > 67523 and n1.zcoord > -3139 and
n1.xcoord < 24078 and n1.ycoord < 68052 and n1.zcoord < -2255 and
n2.xcoord > 23226 and n2.ycoord > 67523 and n2.zcoord > -3139 and
n2.xcoord < 24078 and n2.ycoord < 68052 and n2.zcoord < -2255 and
n3.xcoord > 23226 and n3.ycoord > 67523 and n3.zcoord > -3139 and
n3.xcoord < 24078 and n3.ycoord < 68052 and n3.zcoord < -2255 and objectid
is not null order by x1, y1) where rownum <= 54554

```

Text insertment 6.2: Joined SQL query to select constrained triangles

With the part of the query “select \* from () where rownum <= aantal” is achieved that a number of records is returned to the user which is determined by the percentage of the dataset the user wants to load.

The orientation, the node IDs (nid1, nid2, nid3) and the object ID are read from the constrained triangle table. The constrained triangles are joined with the nodes to get three coordinates for each triangle. Three coordinates with an x, y and z component (n1.xcoord, n1.ycoord, n1.zcoord, n2.xcoord, n2.ycoord, n2.zcoord, n3.xcoord, n3.ycoord, n3.zcoord) are read from this table. Only the triangles with coordinates within a specific bounding box are requested, the bounding box coordinates are given as absolute numbers in the query, while they were input as a percentage by the user (n1.xcoord > 23226 and n1.ycoord > 67523 etc.).

Text insertment 6.3 displays the query for getting the tetrahedra or the TEN data model.

```

select * from (select nid1, nid2, nid3, nid4, objectid,
    n1.xcoord as x1, n1.ycoord as y1, n1.zcoord as z1,
    n2.xcoord as x2, n2.ycoord as y2, n2.zcoord as z2,
    n3.xcoord as x3, n3.ycoord as y3, n3.zcoord as z3,
    n4.xcoord as x4, n4.ycoord as y4, n4.zcoord as z4 from
    (select ltrim(substr(f.tetcode, 1,10),'0') as nid1,
    ltrim(substr(f.tetcode, 11,10),'0') as nid2,
    ltrim(substr(f.tetcode, 21,10),'0') as nid3,
    ltrim(substr(f.tetcode, 31,10),'0') as nid4,
    ltrim(substr(f.tetcode, 41,10),'0') as objectid
    from tetrahedron_tt f) triangle,
    node_tt n1, node_tt n2, node_tt n3, node_tt n4
    where n1.nid = nid1 and n2.nid = nid2 and n3.nid = nid3 and
n4.nid = nid4 and n1.xcoord > 23226 and n1.ycoord > 67523 and n1.zcoord >
-3139 and n1.xcoord < 24078 and n1.ycoord < 68052 and n1.zcoord < -2255
and n2.xcoord > 23226 and n2.ycoord > 67523 and n2.zcoord > -3139 and
n2.xcoord < 24078 and n2.ycoord < 68052 and n2.zcoord < -2255 and
n3.xcoord > 23226 and n3.ycoord > 67523 and n3.zcoord > -3139 and
n3.xcoord < 24078 and n3.ycoord < 68052 and n3.zcoord < -2255 and
n4.xcoord > 23226 and n4.ycoord > 67523 and n4.zcoord > -3139 and
n4.xcoord < 24078 and n4.ycoord < 68052 and n4.zcoord < -2255 and objectid
is not null order by x1, y1) where rownum <= 37396

```

Text insertment 6.3: Joined SQL query to select tetrahedra

The advancement of the query is similar to the query for getting the constrained triangles but now one more node is included.

#### 6.4.3 3D Data transformation

**For retrieving data from the database and transforming it into the XML based language X3D, is found the following solution:**

The user clicks a submit button in the client. A script reads out the geometric data from the database, transforms it to X3D and returns it to the client.

In the database, either the constrained triangles are stored (present in a view) with references to nodes and nodes are stored with reference to node coordinates, or tetrahedra are stored (stored in a table) with reference to nodes and nodes are stored with reference to node coordinates. A sample of a triangle and node table is given in Appendix 6.2. The data can be retrieved using different algorithms. As described in paragraph 6.4.2 in the 'Original' and 'Faster' algorithm the node and triangle or node and tetrahedra table are each read in by a separate query. In the 'Joinquery' algorithm the node and triangles or node and tetrahedra are read in by one joined query, which results in an array where for each triangular face three coordinates are stored with a coordinate ID an x, y and z coordinate, and in addition for each face an object ID will be stored too.

The triangles and tetrahedrons are stored in the database as a concatenated string. For example, a triangle with object ID 105 and node ids 1501, 1504, and 1510 could be stored like this 001501001504001510000105.

In the case of the 'Original' or 'Faster' algorithm, the triangles or tetrahedra are loaded into an array whereby the object ID becomes the first item of each row; in the database, it is the last item. The node IDs, three in the case of a triangle, four in the case of a tetrahedron, are stored in the next rows. The array is sorted on the object ID. The zeros before the numbers are removed for example 003 becomes 3 (Script Appendix 6.3). The result could look like this: [(3, 9, 23, 27), (3, 10, 14, 23), (3, 14, 15, 16), (3, 10, 14, 25)].

The node (x, y, and z) coordinates are stored in an array, which has as many items as nodes. This table is sorted on the first item, which is the node ID, the other three items are the node x, y and z coordinate. The array could for example look like this: [(9, 14.0, 2.0, 25.0), (10, 14.0, 2.0, 35.0), (14, 14.0, 8.0, 35.0), (15, 18.0, 11.0, 35.0), (23, 14.0, 2.0, 30.0), (25, 18.001438, 2.0, 35.0), (27, 14.0, 8.0, 29.0)]

In the case of the 'Joinquery' algorithm, the triangles or tetrahedra and nodes are loaded into one array. The triangle or tetrahedron table is joined with the node table using node ID as common identifier. When tetrahedra are selected the tetrahedra is split up in four faces. This will result in a face table where each face has an object ID, three node IDs and an x, y and z coordinate for each node.

In Appendix 6.4, a sample is given of the first five faces of the array (object ID 0, node IDs 1, 2, 8, 10, 20, 25 and 29).

The object ID determines the object each triangle belongs to. There will be a few objects and many triangles, each object consist of a number of triangles. In the next steps in both the 'Original' and 'Faster' algorithm:



- An array is created which list the unique nodes per object. For every object each node which is being referred to is put in a list. Even if the node is referred to more than once it is only put in the list once for each object. The array of unique nodes per object could look like this: [[9, 10, 14, 15, 16, 23, 25, 27] [...next object]].
- An array is created which per object list the faces with node references. Each object is now build up from a number of faces. The faces, usually triangles, consist of three nodes. The array of faces with node references could look like this for the sample: [[(9, 23, 27), (10, 14, 23), (14, 15, 16), (10, 14, 25)] [...next object...]] (no id is stored; this is done in the next array).
- An array is created which per object list the object id. The array with object ids could look like this: [2, 3, 5].

(Script in Appendix 6.5)

There is no need to store all the nodes for every object of the IndexedFaceSet. Therefore, the node references have to be localized. For example, node 9 is indexed as first item in the unique node list so it will have 0 as index. Node 23 is indexed as item 6 in the unique node list so it will have 5 as index (Script in Appendix 6.6). While this is an independent step in the ‘Original’ algorithm, in the ‘Faster’ algorithm this step is done in the same loop as the XML X3D creation.

In text insertment 6.4, the index for every number is specified.

```
9(0), 10(1), 14(2), 15(3), 16(4), 23(5), 25(6), 27(7)
```

Text insertment 6.4: Index number per object ID

Text insertment 6.5 displays the array of faces with node references after applying the index.

```
[[ (0, 5, 7), (1, 2, 5), (2, 3, 4), (1, 2, 6) ] [...next object...]]
```

Text insertment 6.5: Array of faces with node references

In the next step of both the ‘Original’ and ‘Faster’ algorithm, the list of unique nodes per object is compared with the node coordinates to create an array of unique node coordinates per object (Script in Appendix 6.7). In the ‘Original’ algorithm this step is taken before the XML creation, in the ‘Faster’ algorithm this step is taken after the XML creation. In the ‘Faster’ algorithm, the step is done more efficient: there has been made use of the fact that the nodes are just numbered as a normal range (1, 2, 3, 4 etc). If a dataset is used where the nodes are not numbered as normal range there should be made use of a dictionary or hash for a quick-lookup of nodes by node ID. A sample of the array of unique nodes coordinates per object is given in appendix 6.8.

In the next step of both the ‘Original’ and ‘Faster’ algorithm, the string for the IndexedFaceSet is derived per object from the array of faces with node references after applying the index (Script in Appendix 6.9). A sample of the CoordIndex of the IndexedFaceSet, which list the faces with node references, is given in text insertment 6.6 as well as the coordinate-list string per object, which is derived from the array of unique node coordinates per object (Script in Appendix 6.10).

```
['<IndexedFaceSet solid="true" CoordIndex="0 5 7 -1, 1 2 5 -1, 2 3 4 -1, 1 2 6 -1, ">', .. Next, object...]
```

```
['point="14 25 2, 14 35 2, 14 35 8, 18 35 11, 22 35 8, 14 30.004904 2, 18.001438 35 2, 14 28.9975260 8, " />', ...next object...]
```

Text insertment 6.6: IndexedFaceSet string

The last two steps described of both the ‘Original’ and ‘Faster’ algorithm are almost identical for the ‘Original’ and the ‘Faster’ algorithm. The difference is that they are carried out in two separate loops in the ‘Faster’ algorithm, and in one loop in the ‘Original’ algorithm. Figure 6.16 schematized the ‘Original’ or ‘Faster’ algorithm.

### Original / faster algorithm

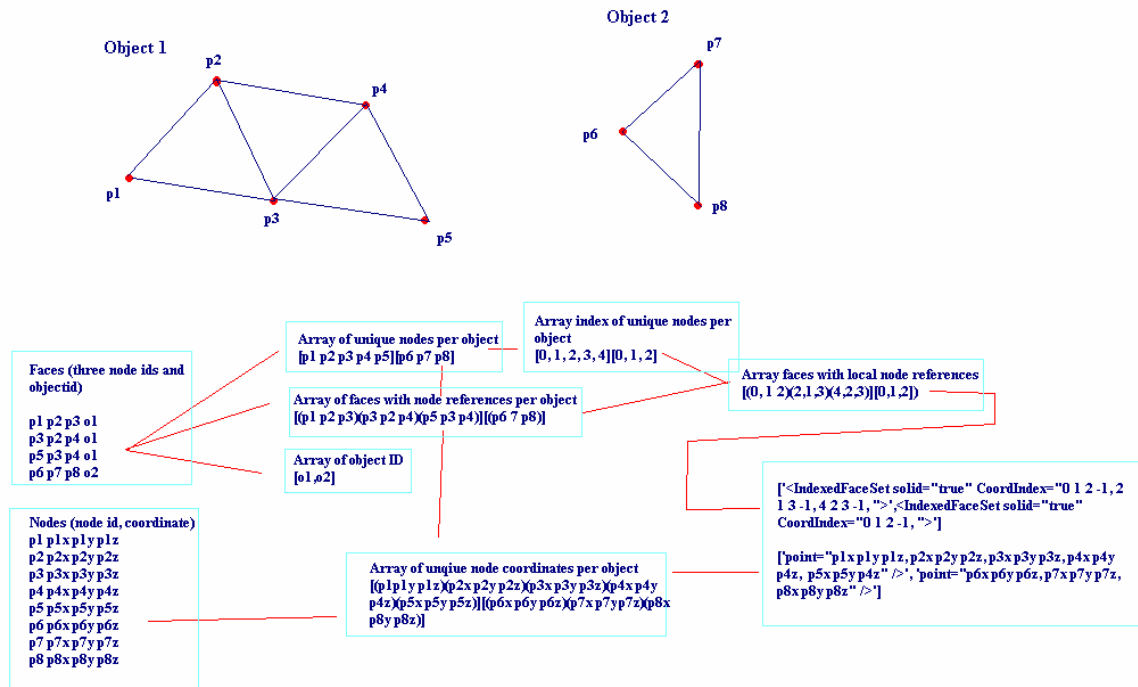


Figure 6.16: The Original / Faster algorithm in a schema.

In the ‘Joinquery’ algorithm all the steps described above as executed in the ‘Original’ and ‘Faster’ algorithm are carried out in one loop. The array of faces / triangles is being looped. For each new face, the object ID is filtered out. When the object ID changes an ‘if’ block is executed (Appendix 6.11). When the ‘if’ block is executed, an item is added to the IndexedFaceSet array and to the CoordlistStringArray array and their start values are initialized.

Because there is a new object, a new item is added to the array of node coordinates per object, and to the array of unique node per object. The object ID is added to the array of unique object IDs (Appendix 6.12).

In the next step of the ‘Joinquery’ algorithm, the node IDs are read from the array with triangles. If the list of unique nodes IDs of the current object is still empty, all three nodes of the triangle are added to the list of unique nodes IDs per object. Then the coordinates of each triangle are read from the array with triangles and added to the CoordlistString array and to the array of node coordinates per object. If the list of unique nodes IDs of the current object is not empty, the node ID and the coordinates of the node are only added to list of unique node IDs, the CoordlistStringArray, and the array of node coordinates per object, if the node ID is not yet in the unique list. Finally, the local indexes of the node ID in the list of unique node IDs per object are being looked up. The script is in Appendix 6.13.

In the last step of the ‘Joinquery’ algorithm script the IndexedFaceSet array is filled. The CoordList is already filled in the previous steps. In Appendix 6.14 it can be sought how it is done for the different output formats. Figure 6.17 schematizes the ‘Joinquery’ algorithm.

## Joinquery algorithm

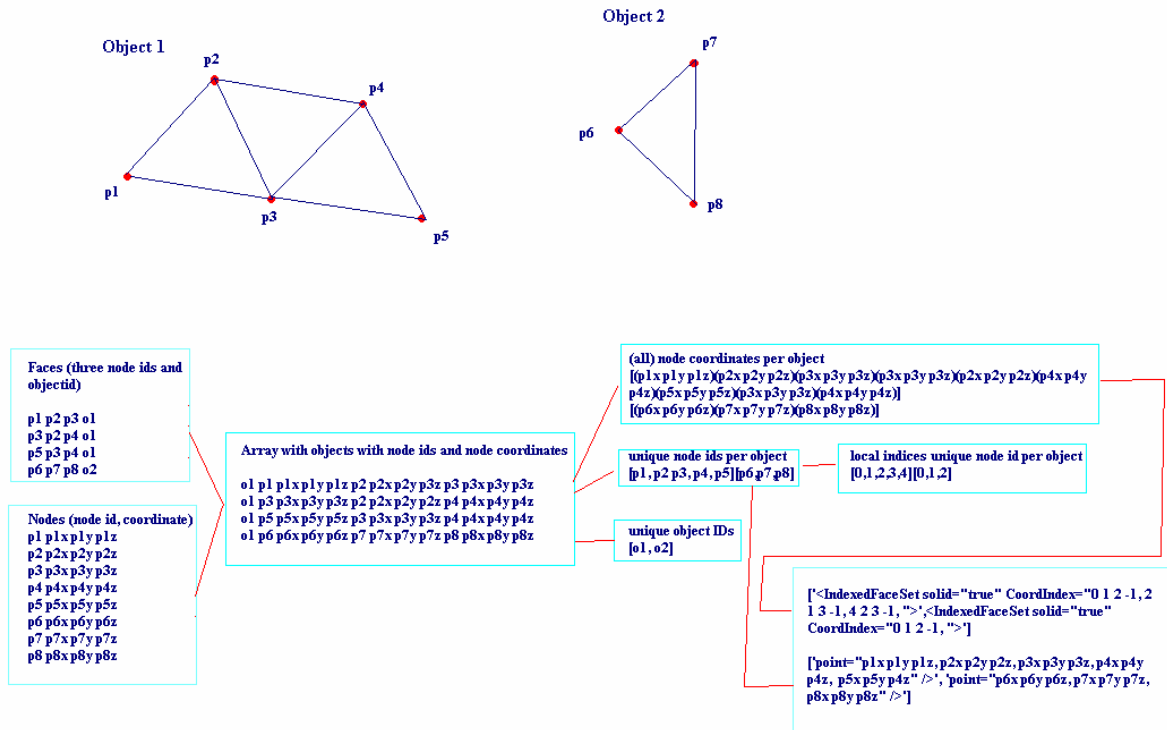


Figure 6.17: The Joinquery algorithm in a schema.

In the next step, the viewpoints per object are created. The viewpoint is pointed to the object and the location is away from the middle point of the object about four times the depth (z-axis going to horizon) and one time height of the object. The orientation is automatically calculated by using the viewpoint location and the middle of the object the viewer is looking to. The script is in Appendix 6.15. For the sample with three objects with id 2, 3, and 5, a sample of the viewpoint array is given in text insertment 6.7.

```
[ (18.969382278797255, 4.2285714285714286, 222.23809523809524),
(17.750287999999998, 14.5625, 113.8125), (18.000575999999999, 2.0,
113.875) ]
```

Text insertment 6.7: Viewpoint string

In the next step, the XML data is created. The first part of the XML contains general information as namespaces and metadata. In addition, a background with a sky and ground can be created. For each object a unique shape is created where the object ID forms part of the object description. The topology and geometry of each object are stored within the tag of each object. Viewpoints are added to each object so from the view can be jumped to a view of any of the objects. Secondly, for each object an anchor node is created so when clicking on the object there will be made a jump from the object to the corresponding record in the database. Lastly, each object will have an appearance node giving it a unique colour and material. The colour is read from the table containing the object's attribute information (Script and 3D data sample (X3D) in Appendix 6.16). In each visualization language, the geometry is stored somewhat different and some of the components, for example viewpoints, colour, and anchors, may be or may be not supported. However, the same initial steps are used for every XML based language to derive the geometry.

**For retrieving data from the database and transforming it into the XML based language CityGML, the following solution is used:**

CityGML makes use of a linear ring. A linear ring of a triangle will contain point p1, p2, and p3 and then again point p1 to close the ring.

From this ring, the GML position list is created. The GML position list is created of the IndexedFaceSetlist per object variable and the array of unique points per object as in the previous subparagraph. Each triangle exist of twelve components namely the first, second and third x, y and z coordinate and again the first coordinate, times the number of points in a triangle (three), is twelve.

By combining the IndexedFaceSetlist and the unique coordinate list per object, it creates the position list for making the linear rings. The transformations prior to this operation are identical to the ones with X3D. The script for creating the GML position list can be found in Appendix 6.9. Text insertment 6.8 gives a sample of one tag in the GML position list.

```
<gml: posList srsDimension="3"> 14 25 2 18.001438 35 2 14 29.9975260 8 14  
25 2</gml: posList>
```

Text insertment 6.8: GML poslist string

For each triangle, a unique surface member is created in the GML, for each unique object a CityObjectMember is created in the CityGML. The script for generating the GML output is put in Appendix 6.17, as well a sample of the GML output.

**For retrieving data from the database and transform it into the XML based language KML, the following solution is used:**

The storage of faces in KML is identical to the storage of faces in CityGML.

Individual faces or triangles are separated by a <Polygon> tag; while a Place Mark tag separates individual objects. In GML the x, y and z component of the coordinates are separated with a space and the coordinate pairs are separated with a space while in KML the x, y and z component of the coordinates are separated with a comma while the coordinates are separated with a space. In KML, the mid point of the object is used to place a placemark point. When this point is clicked, a balloon pops up. This balloon displays a link, which corresponds to the object's record in the database. The part of the script to generate the KML as well a sample of the output KML is put in Appendix 5.18: the prior transformation calculations are identical to the ones for X3D.

#### 6.4.4 2D Data transformation

**For retrieving data from the database and transforming it into the XML based language SVG, the following solution is used:**

The user clicks a submit button in the client. A Python script reads the geometrical data from the database, transforms it to 2D SVG or flat X3D, and returns it to the client.

The initial part of the creation of a SVG file is the same as the script for creation of an X3D or KML file. Coordinates are put in a SVG coordinate position list. Unlike in linear rings the first coordinate is not again repeated at the end of the data. Secondly, only the x and z coordinates are stored to get a 2D view of the data. A translation parameter is added to position the SVG image on the right place in the screen. The script to generate the SVG coordinate position-list is put in Appendix 6.9.

Text insertment 6.9 displays a tag for one polygon or triangle in SVG.

```
[['<polygon points="0.0,0.0 0.0,10.0 20.5999837603,5.0" style="fill:#FFFF00;stroke:#FFFF00;stroke-width:0.25" transform="translate(5 0)" />', '<next polygon>'] [...next object...]
```

Text insertment 6.9: SVG polygon tag

The script to generate SVG data is displayed in Appendix 6.19. As with other XML based languages, the header contains metadata and reference to SVG namespaces. Within the SVG tag the geometry, the size, version, and extent of the SVG is set. An extent of 0 0 370 380 is set to fit the SVG in the 2D data frame provided. To have the SVG cover the whole frame, it is scaled. To attach links to objects, “xlink” is used in combination with an <a> or <g> tag, which groups triangles into one object. Appendix 6.19 contains a sample of the output SVG.

#### 6.4.5 Attribute data transformation

Transforming attribute data from the database into a (HTML) table is done as follows:

The non-spatial attributes of the objects are stored in a separate table in the database. First, the whole table is read from the database and read into an array in Python using PyODBC. If the ‘Joinquery’ algorithm is used only the records are loaded related to the visualized objects. The column names are stored and the data are stored in separate variable. For each row, the values are written within a <tr> tag within the HTML. Within the <tr> tag <td> tags are used to separate the values of for each column. The column names are made bold with the <strong> tag.

Text insertment 6.10 displays the HTML code for the example table 6.4.

<b>Animal</b>	<b>Weight</b>	<b>Colour</b>
Elephant	900	gray
Giraffe	1500	yellow and black

Table 6.4: ‘Animal data’ table

```
<table width="100%" height="100%" bgcolor="#E9FFE6">
  <tr>
    <td><strong>Animal</strong></td>
    <td><strong>Weight</strong></td>
    <td><strong>Color</strong></td>
  </tr>
  <tr>
    <td>Elephant</td>
    <td>900</td>
    <td>gray</td>
  </tr>
  <tr>
    <td>Giraffe</td>
    <td>1400</td>
    <td>yellow and black</td>
  </tr>
</table>
```

Text insertment 6.10: HTML code for sample table

#### 6.4.6 Feature identification

Feature identification is the identification of geographical objects. In a GIS system, features can be clicked and the features attribute information or metadata will be shown to the user. This data will at least contain the feature's unique ID. Feature identification is similar with querying the attribute information from one object in the data (as done in 6.5.5).

When the user clicks an object in the 2D or 3D visualization a Python script is called with the ID of that object as argument. The script queries the TEN attribute data and returns the corresponding record as a HTML table in the dataview of the web application (script and HTML sample in Appendix 6.21 and 6.22). If the house in figure 6.18 is clicked, the attribute information in table 6.5 is displayed.

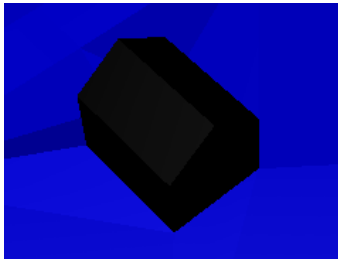


Figure 6.18: House (from prototype website)

OBJECTID	OBJECTNAME	OBJECTOWNER	OBJECTHEIGHT	COLOUR	TETTRANSPARENCY
2	Terrain	Government	50	#755A51	0.00

Table 6.5: Attribute data of terrain feature

When doing feature identification, only the data will be shown of the object clicked. This means one row of data is shown to the user. This is achieved by adopting the query. For example “Select \* from ObjectInfo” will become “Select \* from ObjectInfo where ObjectID = 2”. The selected object can be passed as argument to the function that displays the attribute data. The X3D object node holds an anchor element with a hyperlink with the selected object included and a target frame where to output the data. It makes uses of the principle of HTTP argument passing which is explained in Appendix 6.24.

Hyperlinks can be added as well in X3D, KML, CityGML, and SVG (Appendix 6.25). The most useful implementation of hyperlink is in X3D and SVG where the results are returned within the table view of the prototype application window. In Google Earth, the attribute data is shown within a separate frame in the table. In CityGML, the hyperlink has to be copied to a new web browser window to be displayed.

#### 6.4.7 SQL query

**Querying based on attribute data is explained next:**

Feature selection based on attributes should be possible. For example, select all the houses that are higher than 10 meters, or select all the objects where the landuse is "grass", or select all houses of which the street name contains “John F Kennedy.” These selections are done in the form of an SQL query. A query for selecting the houses that are higher than 10 meters is done like this in SQL: “Select \* from houses where house\_height > 10” (\* means all columns are returned).

The selection should appear in the 2D or 3D visualization, for example by colouring it yellow. For now, queries can be done on the attribute data on not on the geometrical data.

In the SQL query box on the bottom of the prototype page, an input field is presented to the user with a submit button. When the submit button is pressed, a function that reads in the query is called. A sample query is filled in already so that the user knows at least the name of the table that is to be queried. The attributes that is queried can be looked up in the Attribute data view. In an improved version of the prototype the user should be able to select the table names, attributes and values from a listbox as can be done in a standalone GIS (for example ArcView, ArcGIS). The script for generation of the new form is in Appendix 6.26.

The query is read out with JavaScript from the form field and written to a variable. Alternatively, this can be done with CGI or a mod\_python module, which is similar to CGI called FieldStorage.

The queries are picked up from the HTML form field, loaded into a Python variable, and executed on the database using the module PyODBC, which uses ODBC to access an Oracle database with Python and the resulting IDs from the query are used as input for the data visualization.

The query results are visualized in three different frames. In the frame that contains the 3D visualization, the 2D visualization, and the tabular data. In each of these frames, the selected object will be coloured yellow, while the unselected objects retain the original colour.

The visualization process is split up in different parts:

First hyperlinks are created pointing to each of the frames. These hyperlinks are composed of a server path component and a reference to the scripts for the 3D, 2D, and tabular visualization. In addition, the links are supplied with extra arguments, which indicate which objects are selected and not selected. For example when objects with ID 2 and 3 are selected and objects with ID 5 and 6 are not selected they are formatted like this layerselect = "2\_3\_" and layeraselect = "5\_6\_" These strings are formatted using a function called postquery (Appendix 6.27). The input is the query given by the user and the various parameters that are set for visualization. The output is a link hyperlinking to the 3D View, 2D View and Table View with query results.

In table 6.6 is displayed how the attribute information is built up.

ObjectID	Objectname	Objectowner	Objectheight	Colour	Tettransparency
0	Air	Government	50	#0000FF	0.90
3	House	Government	10	#FF1111	0.00
2	Terrain	Pietersen	50	#755A51	0.00

Table 6.6: Attribute data of small test dataset

In an improved prototype, (attribute) data and visualization information should be uncoupled. Technologies as the Styled Layer Descriptor (SLD) which adds appearance to a web service can be used (Open Geospatial Consortium, 2007 [3]).

When the query "Select \* from OBJECTINFO where OBJECTHEIGHT = 50" is made on the attribute information in table 5.5 then the output of the postquery function (if executed on the table, 3D view, or 2D View in order) is given by Appendix 6.28.

Secondly, with JavaScript a link can target to multiple frames at once. This cannot be done with HTML alone. Within the SQL query window a new page is generated with the query that was submitted and a link that will visualize the query within the 3D frame, 2D frame and within the tabular data. The script to generate the new page is displayed in Appendix 6.26.

To get KML or CityGML as output a JavaScript function can change the output-link as is done in Appendix 6.28 as well.

The selection will be coloured yellow in the target specified; as displayed by figure 6.19.

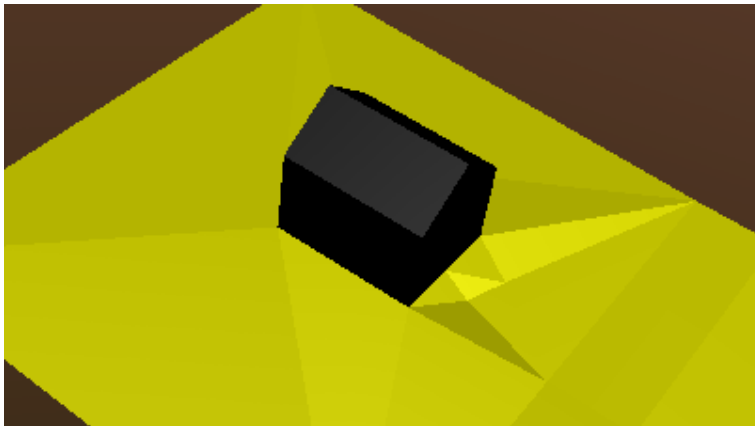


Figure 6.19: Small dataset with selection of earth tets (from prototype website)

With JavaScript, a part of the visualization in X3D or SVG or HTML can be coloured yellow without having to reload the whole file. The visualization is thus accessed dynamically. In Appendix 6.2 is put how to access the 3D visualization in X3D, the 2D visualization in SVG and the table in HTML dynamically. In addition, a sample is given how to colour an X3D node yellow with JavaScript.

#### 6.4.8 Colour

For assigning colours to object is found the following solution:

Colours of different objects in the dataset are in the object attribute table (table 6.7).

ObjectID	Objectname	Objectowner	Objectheight	Colour	Tettransparency
0	Air	Government	50	#0000FF	0.90
3	House	Government	10	#FF1111	0.00
2	Terrain	Pietersen	50	#755A51	0.00

Table 6.7: Attribute data of small test dataset [2]

Colours are stored using a hexadecimal value. The first two characters indicate the red component, the second two characters the blue component, and the last two characters the green component. A value of 00 stands for 0% or colour value zero and the value FF stands for 100% or the colour value 255. As it is hard to code colours by hand this way, KML colour converter (Sgrillo, 2007) can help to find the right colour and its hexadecimal code.

The colours are read from the database before the process 3D or 2D visualization of the data is calculated. First, a query is executed on the ObjectInfo table using PyODBC. In the next step, the object colour is assigned to the object.



This is a bit different for KML, X3D or CityGML because the tags for colour are somewhat different in each output language. A sample is given for X3D (script and output) in Appendix 6.30 and 6.31.

#### 6.4.9 Transparency

The transparency determines the transparency of the object. The transparency can vary between zero which means solid / fully visible and one, which means invisible. By setting it to one, invisible objects, which often exist in tetrahedra networks, can be simulated. The object is still there but is now made invisible. Transparent object still show tooltip information and are still clickable if a procedure is connected to them.

Transparency of different objects is stored in the attribute information (table 6.8) using an integer value between zero and one. A value 1.00 will mean the object is visible; if the value is 0.00 the object is invisible.

Colour	Tettransparency
#0000FF	0.90
#FF1111	0.00
#755A51	0.00

Table 6.8: Transparency and colour attributes of small test dataset

The transparency is read out of the table during the process a 3D or 2D visualization of the data is calculated. First, a query is executed on the ObjectInfo table using PyODBC.

In the next step, the object transparency is assigned to the object. This is also bit different for KML, X3D or CityGML because the tags for transparency are somewhat different in each output language. A sample is given for X3D (script and output) in Appendix 6.30 and Appendix 6.31. In Appendix 6.32 is put how to code transparency in X3D, CityGML, and KML.

#### 6.4.10 Viewpoints

**For establishing interaction between the 2D map and the 3D visualization, the following solution is presented;**

In X3D viewpoints can be added to the 3D visualization. A viewpoint is just an x, y, z coordinate where the viewer is standing and is looking in some direction. A viewpoint can be added to each object automatically. For example a viewpoint 50m away in the x direction and 10m away in the y-direction can be added. The orientation of the viewpoint is calculated in such a way that it is pointed towards the object. The 2D map can be used to link to these viewpoints. In this case, if there is clicked on an object in the 2D map, there is being zoomed to the corresponding object in the 3D visualization, which makes it difficult to get lost. The viewpoint is created by first calculating the midpoint of each object, then the viewpoint will be placed on the same x coordinate as the object, the y coordinate will be away four times the depth of the object from the midpoint, while the z coordinate will be placed one time the height of the object above the object. The script to calculate the viewpoints can be found in Appendix 6.15. Within X3D the viewpoints are stored separately from the objects. Viewpoints and objects not related but can be given a definition name with similar ID, for example viewpoint\_23 and object\_23.

Viewpoints are placed between the background and transform / shape tags. In Appendix 6.33 is displayed what the viewpoints can look like.

A link can be made to a viewpoint by making a hyperlink to the X3D file and adding #viewpoint\_description. For example if a link is to be made in SVG to the viewpoint with the object ID 2 it is coded as text insertment 6.11.

```
<a xlink:href="http://localhost/sitedesing/displayTEN.py#layer_2"
target="3DView_Contents">
```

Text insertment 6.11: Jump to a viewpoint with an URL

The viewpoints can also be accessed under the right mouse button within the X3D file. KML and CityGML files do not use viewpoints.

In text insertment 6.12, the code is given how a viewpoint in the 3D visualization in X3D can be accessed with JavaScript

```
loc.setNodeEventIn("Viewpoint_2", "set_bind", "TRUE");
```

Text insertment 6.12: Jump to a viewpoint with JavaScript

With HTML can be jumped to a viewpoint once and then it does not work anymore. With JavaScript a jump to a new viewpoint can always be made.

#### 6.4.11 Navigation

Navigation is a technique that is part of the plug-in functionality, which the user has to master to navigate around the data. This subparagraph gives instruction how to navigate in the 3D data. Instructions are given how to navigate in X3D data in a web browser. In Appendix 6.34 is described how to navigate in KML data in Google Earth, and how to navigate in CityGML data in the LandXplorer application.

With the examine option in BsContact (for X3D) it is easy to rotate the objects along all axes. When the object is seen from the side and the mouse moved to the left or the right the objects is rotated around it's z axis, when the mouse is moved down or up the object is rotated along it's x or y axis dependent on how it is rotated along the z-axis. For the viewers point the object is rotated along the x-axis. The x-axis is the axis that goes from the left to the right horizontally. It is not rotated around the y-axis, which is the axis that goes from the front to the back or to the horizon. When holding the control key, the object can be moved along the z-axis, x-axis or in between.

With the walking option in BsContact there can be walked on the x, y plane using the arrow keys. With the gliding option, the user can move along the viewer x-axis or the z-axis. If the control key is down the viewer can rotate his view around the z-axis

Flying is almost the same as gliding but now the control button has to be pressed to move along the x-axis and when control is not down the user rotates his view around the z-axis.

With the turning option on the view can be rotated around the z-axis and the viewer's x-axis. When the control key is hold down there is moved along the axes instead of rotation.

If the gaming mode is used, there will be instructions which keys to use. Movements can be done using keyboard keys and with the mouse, the view can be rotated, as in a 3D game like Doom.

Finally, with the jumping option, a movement is made up and down if the mouse is dragged up or down, or a rotation is performed around the viewer's z-axis if the mouse is dragged left or right.

## 6.5 Non-implemented techniques

### 6.5.1 Introduction

In this paragraph five techniques are described which are not implemented in the prototype. These are textures (subparagraph 6.5.2), Level of Detail (subparagraph 6.5.3), data compression (subparagraph 6.5.4), limitation of freedom constraints (subparagraph 6.5.5), and graphical projection (subparagraph 6.5.6). A short motivation is given why the techniques are not implemented.

### 6.5.2 Textures

Textures are images, which are pasted on objects or terrain to make them look more realistic. There are no textures available for the TEN dataset. In figure 6.20 is given a sample where textures are added to the dataset of the house in X3D. Implementation of textures in X3D, KML, and CityGML is extensively discussed in Appendix 6.35.



Figure 6.20: Textured X3D model

### 6.5.3 Level of detail

One of the problems that are faced in visualizing in 3D is the change of the ‘picture’ when the user zooms in or out. When looking at a bigger picture fewer details should be shown, then when looked at it from a closer distance. For example, a block of houses can be displayed as a block or as individual houses with their windows, doors etcetera. This is also called the Level of Detail (LoD) (Romkema, 2006). Level of detail is not implemented in the storage of TEN in the database, and it is a very hard task to calculate different images at different LoD of the data. It would require human decisions; it can never be done fully automatically. In Appendix 6.36 there is more information how to implement Level of Detail in X3D, KML, and CityGML

In figure 6.21 is a sample of the TEN X3D file with LoD of the house on the rough terrain. The representation switches from 3D to 2D when zoomed out at a distance from the zero point greater than 150.



Figure 6.21: Sample of Level of Detail implementation in X3D

#### 6.5.4 Compression

For compression of the XML data compression techniques as developed by (Eppinger et al, 2005) can be used. The methods offer ways to compress triangulated surfaces and thus are especially suited for compressing a 'TEN view' or a 3D TIN. There has been no research into specific compression techniques for TEN. These compression techniques have not been implemented, because the research is more focused on functionality than on speed, and compression scripts are not easily available.

#### 6.5.5 Limitation of freedom constraints

Limitation of freedom constraints has not been added to the view. Extensive help on how to navigate has been added. It has not been researched whether limiting freedom constraint - for example, you can only walk forwards, turn, and left - is possible at all. In Google Earth, the viewpoint cannot be moved under the surface.

#### 6.5.6 Graphical projection

A perspective projection is used in all 3D visualizations. If required, the projection can be altered to orthographic in X3D from the BsContact options.

#### 6.5.7 Streaming

It would be interesting if large datasets could be built up bit by bit in front of the eyes of the user. Mod\_python does not support streaming. It can only return a stream of valid data at once. If the X3D is not valid it will not be displayed to the user. Alternatively the data can be built up with JavaScript and the Scene Access Interface. In this way geometries can be added to the scene one by one. In theory IndexedFaceSet can be added to the dataset one by one. There seem to be no commands to add individual components to the IndexedFaceSet. For example it would be even nicer if in each update of the scene a new triangle is added. It is not possible just to append text to the 3D output too as the plugin always wants valid X3D.

Another problem is sending over data from Python to JavaScript (or vice versa), it is not possible with the current solutions. A JavaScript can format a link to a Python script (with arguments input in a form) and execute it in the browser. A question can be made whether streaming is really required.

Maybe the user will be as satisfied with a counter which indicates the percentage of the dataset that has been loaded. Again this kind of functionality is not easy or impossible to realize with `mod_python`.

Another interesting option would be to implement the streaming in the plugin or viewer taking away the effort to write a script to build up the dataset bit by bit.

## ***7. Evaluation and conclusion***

### **7.1 Discussion / evaluation**

#### **7.1.1 CityGML vs. X3D vs. KML**

The makers of CityGML consider CityGML as the container of the geographic data, while languages as KML and X3D are considered the container of graphical features. CityGML allows data to be stored in a more complex hierarchy than KML and X3D allow to. For example, a landscape can exist of rough land, grassland and city terrain, city terrain can exist of city blocks, city blocks exist of houses and streets, houses exist of rooms, rooms exist of wall, doors, windows, and furniture etc. X3D can only group graphical elements under one group though also groups can be grouped under a group and the different groups at different hierarchies can be given its own definition. The advantage of CityGML is that these definitions are laid down in an (UML) structure so for example when a viewer application reads there is a tree in the data it can lookup the appropriate colours immediately.

On the other side, the big advantage of X3D is, that it can be rendered in a web browser because there are plugins available to do that. A similar plugin might though be developed to display CityGML in a web browser in the future. Furthermore the way tetrahedral data is stored in a database is quite simple, in fact objects are stored with reference to tetrahedra or triangles and the triangles and tetrahedra are stored with a reference to points. Such a simple way of storing data is more close to the way data is stored in X3D (or KML) than how it is stored in CityGML. This makes translation from database to visualization and back easier.

If assumed the attribute data connected to the TEN structure is only connected at object level when TEN data are translated to CityGML individual parts of objects may have to be manually coded, in the example of house the walls, roofs and windows may get a separate code in the CityGML as CityGML provides more detail. A simple way of storing data as in X3D makes some analysis also more easy, for example if a buffer of 100 meters around a borehole is made and all objects inside it are selected, when using CityGML there should be indicated which classes should be selected for the buffer (though this also can be an advantage).

The advantages of KML are that it is very widely used, lots more people have Google Earth installed than an X3D plugin. The disadvantage is that it is not visualized within the web page of the prototype, but it is launched in a separate standalone application. It could be rendered in a web browser within a Google Maps interface, but then it is in 2D. In the future it may be possible to render 3D KML in a web browser too though.

While CityGML is the most suitable language for detailed visualization and attribute analysis, the current TEN test data provided does not have a complex hierarchy as CityGML provides. Because X3D currently allows display of TEN data in a web browser and it can be dynamically modified for example to support attribute selection or geographical analysis, it is the best choice to visualize TEN data for now.

#### **7.1.2 Way of storing TEN in database**

Penninga has chosen to store tetrahedrons and nodes in the database using normal Oracle data types. (Constrained) triangles are derived from the tetrahedra in a view. Tetrahedrons are stored using concatenated strings, for triangles the string exists of point one, point two, point three and an object ID. For tetrahedrons, the string exists of point one, point two, point three, point four and an object ID. Coordinates (x, y, and z coordinate) are each listed in a separate column.

Advantage to store data this way is easy access and compactness however there are some drawbacks too:

- 1 Triangular data is derived from tetrahedral data in a view and access to a view is very slow with ODBC
- 2 When a conversion is made for visualization, the string must be unpacked each time in four components, and the zeros have to be removed

Alternatively, data can be stored in the OGC Simple Feature Types. When storing faces as polygon now only one table is used giving the polygon with its coordinates. To reduce storage space, the so-called areaFeatures that are an aggregate of triangular features with the same ID, can be stored as the multipolygon Simple Feature type. Advantages of the Simple Feature Types are that it is still a compact way of storing, some spatial analysis can be carried out on them easily like buffer or get bounding box and the access is faster than to native data types as string and integer.

As a third possibility, data can be stored in the polyhedron type, which is new in Oracle 11. The polyhedron type features can be derived from the 3D TIN because its features have a closed surface. In future research it has to be tested how fast the latter two options are and whether they work.

As concluded in paragraph 2.8 the best possibility to store the data in is to use 2D / 2.5D topological data types. Advantage is that this way of storing is very compact (no data redundancy), topology is supported, and the data is easily validated. However, a volume datatype is now not available so in attributes should be recorded which object each triangle belongs too. For the future it might be interesting to implement 3D topological TEN based datatypes in the database.

### **7.2.3 Derive visualization from TEN, 3D TIN, or ‘TEN view’**

The visualization can either be derived from a TEN, from the 3D TIN or from the ‘TEN view’. When a TEN is used, the triangles of the object’s surface as well as the triangles of the internal tetrahedronization are derived. In a 3D TIN, only the triangles of the object’s surface are derived. When a ‘TEN view’ is used, only the triangles which are visible for the user are derived from the TEN, by example triangles that mark the boundary between buildings and air and between air and the ground; the so called constrained triangles. The 3D TIN or ‘TEN view’ require fewer triangles than a TEN so it takes less time to visualize them. It is important that the tables of the 3D TIN or ‘TEN view’ are stored as a physical table and not just as view (that means as a derived table), otherwise access time will slow down dramatically. However, this might be a technical issue, which can be solved. The preference goes out to use the surface triangles of the 3D TIN or the constrained triangles of the ‘TEN view’ in advance, because they require fewer triangles than a TEN and only useful information is visualized. An advantage of the 3D TIN over the ‘TEN view’ is that the surface of its features is closed and watertight; therefore, they can be stored as polyhedron. Visualizing data straight from the TEN is only interesting when the user wants to view how the internal tetrahedronization of objects is done. Unfortunately, a 3D TIN is currently not derived in the database. Penninga F has not yet implemented this feature; therefore data from the TEN table or ‘TEN view’ table is used.

### **7.2.4 Choice of prototype components / architecture**

The current architecture is mainly written in Python. Mod\_python is used to let Python scripts run on the server. Results can be returned to the client with mod\_python too. JavaScript is used to read out forms on the client side and to adapt the 2D and 3D visualization dynamically. JavaScript will generate errors more quickly than Python for example if a mistake is made with upper and lowercase. JavaScript errors can be debugged with Firefox. An alternative could be to use Java for the prototype as has been done in many of the related research in paragraph 1.6. With Java, still again JavaScript has to be used to read out the form values or to adapt the visualization dynamically. Another alternative could be to use a Java Applet.

An advantage will be that no exchange is required between two programming languages (JavaScript and Python). All data is input and output now by Java. The disadvantages of using an applet are security issues and it is not sure whether all functionality that has been offered in the prototype made for this thesis can be integrated in such an applet.

Another alternative, which has not been researched, is to skip the middle layer and to calculate the output visualization in the database (with PL/SQL in Oracle). A web service is still required to display the visualization on the web.

## 7.2 Conclusion / summary

**The goal of the research “to develop a prototype of a web application for visualization of 3D geographical information in the form of a TEN that is stored in a database” has been successfully realized.**

**The main question is again: “How is a TEN dataset visualized in a web client, and what options are there to enhance the visualization and to add GIS functionality?”** From the main question, a number of sub questions are derived (paragraph 1.3). The sub questions and a summarized answer to them are stated next. **After the answers to the sub questions, an answer is given on the main question on the end of this paragraph.**

1. What architecture is used to store and visualize the data (paragraph 5.2 and 5.3)?

The 3D TEN data is retrieved from the database, transformed to an XML based visualization format by a (mod) Python script on an Apache HTTP web server and is returned to the client. Alternatively, the TEN data can be read from the database using a Web Feature Service (WFS), which outputs GML 2. In this case, the Apache web server is not involved; the WFS is directly accessed from the client.

An implementation, which has not been tested, is to use PL/SQL or the Oracle SDK to produce XML within the database and then have it returned to the client.

2. In which output formats is the data visualized on the client (paragraph 3.3)?

The visualization output is either visualized as X3D, KML, or CityGML all based on XML. X3D is a 3D format to visualize 3D graphics in a web client; it can directly be visualized in the browser if the browser has an X3D plugin and can be dynamically modified.

KML is a graphical format to add 2D and 3D graphics to Google Earth. When the KML file is returned to the client Google Earth is launched and dataset is being viewed in Google Earth. This format is used by the widest user group.

CityGML is a profile on the Geography Markup Language, version 3.1 (GML 3.1). CityGML is used to store or exchange various kind of geographical information. CityGML cannot be picked up by a plugin in the web browser therefore CityGML is returned as XML in the web client. The XML can be saved to the local disk and be opened in a CityGML viewer application for example LandXplorer, or Aristoteless. CityGML allows storing the data itself, secondly it is very good for the visualization of the data.

GML 3 is not used because it is used for data exchange, and not aimed at visualization (for example colour and transparency cannot be added to GML). In addition, GML 3 is not yet accepted by Web Feature Services.



3. How can a connection be made between the database and the web client (paragraph 3.4)?

A connection can be made between the database and the web client using the ODBC protocol. The ODBC protocol makes it possible to access database independent of the database and the programming language. A module for Python, PyODBC is used to access an Oracle database with Python. Geometrical or attribute data can be requested from the database on request of the user. After the request, it is transformed for appropriate display and returned to the client. Alternatively JDBC can be used to access the database via a middle layer with Java or PL/SQL or the Oracle SDK can be used to generate XML from inside the database, the XML is then directly send to the client.

4. What options do X3D and other XML based languages offer for visualization of TEN geo-information (paragraph 3.3)?

Options that are used in every implemented XML based language for visualization are to have different objects grouped under one tag, to have colour and transparency assigned to objects, and to have hyperlinks attached to objects so that when an object is clicked its attribute data is requested.

5. How is data prepared for visualization (paragraph 6.4)?

TEN data is stored in the database with two tables. The first table holds the tetrahedra, for each tetrahedron an object ID and a reference to four different nodes is stored. The second table holds the nodes with a node ID and an x, y and z coordinate. In a 'TEN view' there is a table with constrained triangles, where lists for each triangle an object ID and a reference to three nodes. For visualization in X3D, the data is transformed to a format where for each object all triangles are listed in an IndexedFaceSet and the nodes are stored in a PointSet attached to the IndexedFaceSet. In KML and CityGML, triangles with the same ID are grouped in one xml tag.

6. How is a dataset designed that exploits all the possibilities of a TEN in comparison to a dataset in another datamodel (paragraph 6.3)?

An ideal TEN dataset exploits the possibilities of modeling data in a TEN, which means its features cannot be modeled in other data structures. Such a dataset will have features as geological layer, a tunnel, and a bridge, overhanging cliffs, an air corridor, some houses, and a church. The big test dataset of a suburban neighborhood of Rotterdam only has many different shaped buildings. A new dataset of the TUDelft campus is being developed at the time of writing.

7. How can interaction be established between a visualization of the 3D model on the client and storage of the 3D model in the database on the server (subparagraph 6.4.6)?

Interaction between the visualization and the database can be achieved by inserting anchors or hyperlinks in the visualization, which point to a record in the database.

8. Is it quicker to visualize a TEN, 3D TIN or 'TEN view' model, and what visualization looks better (paragraph 2.5)?

The visualization of a 3D TIN ‘TEN view’ model is much faster than the visualization of a TEN model because it holds a lot fewer triangles, so the number of bytes that has to be transferred is lower. The visualizations do not show differences in outer appearance except when they are made transparent the 3D TIN and ‘TEN view’ visualization looks more realistic. In the ‘TEN view’ is only what the visualized what can be seen by the user, so it takes up even less space than a 3D TIN though they look the same most of the times, for example a shared wall of two houses may be hidden in the ‘TEN view’ and is there in the 3D TIN but cannot be seen. However, when the viewer is placed inside the house the difference can be seen. The view is thus dependent on the location of the viewer.

9. How is dealt with the fully partitioned space so for example how are air tetrahedra hidden (subparagraph 6.2.5)?

It can be chosen to give air tetrahedra a high transparency or to make them invisible. Because they take up much space and are in the way of other objects, they can be totally discluded from the visualization. However, in the case of for example an air quality simulation they may be included in the visualization

10. How to identify a feature or how to query attribute information from the data, how to make a selection on the data based on attribute information. For example, a query on the landuse can be executed: “select all objects where landuse = ‘grass’” (subparagraph 6.4.6 and 6.4.7)?

Identify is done by including hyperlinks in the objects which point to the corresponding record in the attribute table. Attribute queries can be done using an SQL statement. The results can be visualized in the 3D visualization, the 2D visualization, and the tabular data, for example by colouring the selection yellow. The user can compose a query by looking at the sample SQL statement and by viewing the attribute data in the Table View.

11. How are colour and transparency added to the data visualization (subparagraph 6.4.8)?

Colour and transparency for each object are stored in the attribute information. Before creating visualization from the topology and geometry tables in the database, colour and transparency are read from the attribute table in the database. In an improved situation, the colours and transparency are uncoupled form the data and generated by means of a Styled Layer Description (SLD).

The answer to the main question: “**How is a TEN dataset visualized in a web client, and what options are there to enhance the visualization and to add GIS functionality?**” is:

First, a mod-Python script on a server reads data from an Oracle database, and then transforms the data to an XML based format suitable for visualization in a web client. To enhance the visualization colour and transparency can be added to objects, the data model from which the data is visualized can be variable (TEN and ‘TEN view’) and air tetrahedra can be included or discluded from the visualization.

For object identification, hyperlinks can be included in the objects in the visualization output which link to the attribute information. With a SQL query, attribute selection is enabled. The selection can be specified as input parameter to the script, which creates the visualization, or JavaScript can be used to modify the visualization dynamically.

## 7.3 Future research and recommendations

### 7.3.1 Recommendations

- Python scripts that run on an Apache HTTP server with the mod\_python module, client side JavaScript and HTML form a good combination to build a 3D web GIS viewer to view TEN data or other format. Alternatively, as in other prototype, Python can be replaced with Java, which runs a bit faster, but coding is less efficient.
- A dataset should be designed that captures the essence of TEN which means it should exploit the full possibilities of TEN so it should have a tunnel, air corridor, overhanging cliffs etc. A dataset of the TUDelft Campus is being created at the moment.
- For quick calculation of the visualization and outer appearance, the 'TEN view' or constrained triangles, can best be visualized.
- For big datasets air and earth tetrahedra should not be visualized, otherwise loading time will increase dramatically (4-8 hours for a dataset of > 100000 triangles).
- X3D is the best choice to work with for 3D visualization, it can be dynamically modified and is directly visualized in the browser with an X3D plugin.

### 7.3.2 Feature extension

The application can be extended with all kind of features:

- When a selection is made, it should be stored in memory. When the output format is changed, the selection should be taken over by the new output format.
- The user has the possibility to add and remove datasets from the prototype.
- There should be tools in the toolbox like panning, zooming, and zoom all, which the user can use in the 3D visualization and 2D visualization.
- The 3D visualization in X3D via GML 2 should not only work in Internet Explorer but also in Firefox. There could also be added the possibility to go from GML 2 to CityGML or KML. Colour has to be added to the X3D via GML 2 visualization and hyperlinks have to be added to the X3D via GML 2 visualization so that when an object is clicked its attribute data is shown in the data view. The embedded 2D visualization in SVG should not only work in Firefox but also in Internet Explorer.
- When a large dataset is loaded a counter should be displayed which indicate how much percent of the dataset is loaded. Alternatively, the dataset can be built up bit by bit before the user (streaming). This is applicable to the 3D view, 2D view and Data view.
- Analysis queries should be possible such as buffer, near, distance, intersect and lantern. Queries should not only be made possible on the attribute tables but also on the topology and geometry tables.

### 7.3.3 Prototype research development

Recommendations regarding further research on the development of the prototype

- Separate HTML, Python, and JavaScript, so place them in different files.
- Make all scripts well readable. Separate in functions and modules.
- Add a Content Management System, so the outer appearance of the application can be controlled from a separate file, separate visualization from data, for example investigate if the data can be visualized with the help of Styled Layer Descriptors.
- The application could be enabled not just to visualize TEN data but also data in boundary representation, as a lot of 3D data is available in this format. In addition, the application could be enabled to visualize TINs, DEMs, (CSG) or polyhedra data.
- Speed test can be performed to find out which are the bottlenecks in the algorithms.
- Research possibilities of mod\_python and Python web (GUI) interfaces.
- Research security measures that block some functionality when accessing the web application from the internet instead from a localhost and find out if there are workarounds.
- Enable the visualization of nodes, edges and AreaFeatures, LineFeatures and PointFeatures.
- Find out how to communicate data between Python and JavaScript and vice versa.
- Research visualization using Web Terrain Service (WTS) and Web 3D Service (W3DS).

## *References*

- 3DIF, 2006, U3D, Sample Software, (<http://www.3dif.org/>)
- 3DgeoGmbH, 2007, LandXplorer CityGML viewer, (<http://www.3dgeo.de/citygml.aspx>)
- Adobe, Scalable Vector Graphics: SVG – Zone, Adobe, (<http://www.adobe.com/svg/>)
- Allen G, 2007, Graphics Cards: How to choose the best, (<http://www.selfseo.com/story-18897.php>)
- Arens C., Stoter J, and van Oosterom P.J.M., 2004, Modelling 3D spatial object in a geo-DBMS using a 3D primitive, Agile Conference, April 2003, France.
- Dörschlag D. and Drerup J., 2007, Aristoteles, Department of Geoinformation, Institute of Geodesy and Geoinformation, Bonn, (<http://131.220.71.152/index.php/Aristoteles>)
- Baily P., Morcrette C., and Privat M., 1990, Database Visualization in 3D, Center for Educational Computational Computer Initiatives, Massachusetts Institute of Technology, ([http://ceci.mit.edu/projects/database\\_visualization/index.html](http://ceci.mit.edu/projects/database_visualization/index.html))
- Beard D., 2004, Tasmanian Tasgo (3D) model, Department of Industry, Tourism and Resources, Geoscience / Australian Government, (<http://www.ga.gov.au/map/web3d/tasgo/#intro>)
- Bitmanagement Software, 2007 [1], Do IT in 3D! Interactive Realtime 3D Software based on standards VRML, X3D, MPEG4, and Java, (<http://www.bitmanagement.de/>)
- Bitmanagement Software, 2007 [2]. X3D sample, which makes use of tiling, (<http://www.bitmanagement.de/php-bin/ViewVrml.php?url=http://www.bitmanagement.de/demo/geo/index.wrl&fullPage=yes>)
- Bossler J.D., editor, 2002 [1], Manual of geospatial science and technology, Taylor & Francis, London, USA, p432-436
- Bossler J.D., editor, 2002 [2], Manual of geospatial science and technology, Taylor & Francis, p234
- Carosia A., 2007, 3D Geographic Information Systems ,Institute of Geodesy and Photogrammetry, Technical Hochschule Zurich, Switzerland, ([http://www.gis.ethz.ch/Research/gis/gis\\_3dgis.php](http://www.gis.ethz.ch/Research/gis/gis_3dgis.php))
- Carto: net, 2007, SVG tutorial and example site, (<http://www.carto.net/papers/svg/samples/>)
- Champion S., 2001, JavaScript: How Did We Get There, O'Reilly Media, ([http://www.oreillynet.com/pub/a/JavaScript/2001/04/06/js\\_history.html](http://www.oreillynet.com/pub/a/JavaScript/2001/04/06/js_history.html))
- Chirapiwat T., 2003, Visualization of Spatial/Geographic Information with VRML, Published MSc. thesis at Univesity Of Michigan, Ann Harbour, ,in Solstice: An Electronic Journal of Geography and Mathematics, Vol. XV, Issue 1
- CityGML Wiki, 2007, Snowflake Web Feature Services – CityGML, ([http://www.citygmlwiki.org/index.php/Snowflake\\_Web\\_Feature\\_Services](http://www.citygmlwiki.org/index.php/Snowflake_Web_Feature_Services))

- Cox et al., 2006, Xmmml TWiki website,  
(<https://www.seegrid.csiro.au/twiki/bin/view/Xmmml/WebHome>)
- Dinoloket, 2007, 3D Atlas of the deep subsurface, Geological Survey of the Netherlands ,TNO Built environment and Geosciences,  
(<http://dinolks01.nitg.tno.nl/dinoLks/download/maps/doG3d.jsp>)
- Dooling D., 2002, XSQL, SourceForge, (<http://xsql.sourceforge.net/>)
- Epping F. and Coors V., 2005, Compressing 3-dimensional urban models, proceedings 4th Workshop on Dynamic and Multi-dimensional GIS (DMGIS 05), International Archives of Photogrammetry and Remote Sensing, Vol XXXVI, Part 2 / W29, ISSN 1682-1750.
- Ferg S., 2006, Python and Java: Side by side comparison,  
([http://www.ferg.org/projects/python\\_java\\_side-by-side.html](http://www.ferg.org/projects/python_java_side-by-side.html))
- Giger M., 2006, Earthbrowser: KML needs an AJAX style upgrade,  
(<http://blog.earthbrowser.com/2006/09/kml-needs-ajax-style-upgrade.html>)
- Google, 2007, KML Overview,  
(<http://code.google.com/apis/kml/documentation/index.html>)
- Grass Development Team, 2007, Grass GIS – The World Leading Free Software GIS, Fondazione Bruno Kessler, Italy, (<http://grass.itc.it/>)
- Held G., Abdul-Rahman A. and Zlatanova S., 2004, Web 3D GIS for Urban Environment, Proceedings of the International Symposium and Exhibition on Geoinformation 2004 (ISG2004), Kuala Lumpur, Malaysia.
- Homoet M., Kolbe T, and Quadt, 2005, Mini Web 3D Service Client, Instituts für Kartographie und Geoinformation, Universität Bonn, (<http://wmc.ikg.uni-bonn.de/W3DS/>)
- Lieberman J., Syncline inc. and Sonnet J., 2003, OpenGIS Web Terrain Service implementation specification (draft), Ionic Software.
- IST (Information Services and Technology), 2007, Oracle Client Software: Product Overview.  
(<http://itinfo.mit.edu/article.php?id=6349>)
- Kim K., Lee K, Lee H.G., and Ha Y.L., 1998, Virtual 3D GIS's Functionalities Using Java/VRML Environment, Proceedings of the Earth Observation & Geo-Spatial Web and Internet Workshop '98, Salzburg, 1998, Austria.
- Kleehammer M., 2007, PyODBC, A Python database API module for ODBC, Sourceforge,  
(<http://pyodbc.sourceforge.net/docs.html>)
- Kolbe T. H., 2006, CityGML Background and design, Presentation on CityGML at OGC Web Services Testbed phase 4 (OWS-4), Reston, Virginia.
- Kolbe T.H., 2007, CityGML – Exchange and Storage of Virtual 3D City Models,  
(<http://www.citygml.org>)
- Kumke H., Hoegner L., Meng L. and Stilla U., 2006, Enrichment and Multi-purpose Visualization of Building Models with Emphasis on Thermal Infrared Data, a presentation for the 43th meeting of the working Group Automation in Cartography, Leibniz University of Hannover

Kwan, M.P., and Lee, J., 2003, 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, p 93-113.

Lammeren R. van and Hoogerwerf T., 2003, Geo-Virtual reality and Participatory Planning, CGI Report 2003-07, Wageningen University.

Lemburg M.A., 2007, Python Database API specification v2.0, (<http://www.python.org/dev/peps/pep-0249/>)

Lindsey K., 2006, KevLinDev – Geometry – SVG 3D, Kevin Lindsey Software Development, (<http://www.kevlindev.com/geometry/3D/js3d/index.htm>)

Lowe R.G., 2006, Drawbacks of CGI, Webhelpinghand, ([http://www.webhelpinghand.com/CGI\\_drawbacks.htm](http://www.webhelpinghand.com/CGI_drawbacks.htm))

MapInfo, 2007, Mapinfo Spatialware – Architecture, (<http://extranet.mapinfo.com/products/Architecture.cfm?ProductID=1141>)

Map West Virginia Portal, 2007, (<http://www.mapwv.gov/>)

Media Machines, 2007, Media Machines – Virtual worlds, metaverse, Web 3D – tools and services, X3D, (<http://www.mediamachines.com/>)

Microsoft, 2007, WORD: Open Database Connectivity (ODBC), (<http://support.microsoft.com/kb/110093>)

Morcrette C., 1999, VRML Generation tools for visualization of database content in three dimensions, Published MSc. thesis, Massachusetts institute of technology.

Mozilla.org, 2007, Mozilla SVG project, (<http://www.mozilla.org/projects/svg/>)

Mozilla Developer Center, 2007, Ajax: getting started, ([http://developer.mozilla.org/en/docs/AJAX:Getting\\_Started](http://developer.mozilla.org/en/docs/AJAX:Getting_Started))

NSCA development team, 1995, Common Gateway Interface, (<http://hoohoo.ncsa.uiuc.edu/cgi/intro.html>)

Nebiker S., 2003, Support For Visualization and Animation in a Scalable 3D GIS Environment: Motivation, Concepts and Implementation, ISPRS Workshop on Visualization and Animation of Reality-based 3D Models, Engadin, Switzerland.

Ninsawat S., Raghavan V., Masumoto S., Chemin Y. and Nakano H., 2006, Implementing 3D Web-GIS System Using X3D Technology, Presentation for the FOSS conference, Lausanne Switzerland.

Ninsawat S., 2007. Prototype of 3D Web GIS using X3D technology, Osaka City University, ([http://160.193.95.183/appformap3d/client\\_classic.phtml](http://160.193.95.183/appformap3d/client_classic.phtml))

Octaga, 2007, Octaga – Bringing enterprise data to life, Oslo, (<http://www.octaga.com/>)

Oosterom P.J.M. van and Fendel E.M., 2006, Space for geo-information – 3D Topography – project proposal fase 2 RGI – 011, Geo-database Management Centre (GDMC), TUDelft.

Open Geospatial Consortium, 2007 [1], OpenGIS® Geography Markup Language (GML) Implementation Specification, (<http://www.opengis.net/gml/>)

Open Geospatial Consortium, 2007 [2], Simple Feature Access, Part 1: Common Architecture, (<http://www.opengeospatial.org/standards/sfa>)

Open Geospatial Consortium, 2007 [3], Styled Layer Descriptor, (<http://www.opengeospatial.org/standards/sld>)

Oracle, 2007 [1], Oracle 11g, Siebel, PeopleSoft | Oracle, The World's Largest Enterprise Software Company, (<http://www.oracle.com/index.html>)

Oracle, 2007 [2], Oracle XDK v10 homepage, (<http://www.oracle.com/technology/tech/xml/xdk/xdkbeta.html>)

Oracle Technology Network, 2007, Oracle Instant Client, (<http://www.oracle.com/technology/tech/oci/instantclient/index.html>)

Orlinska K, 2005, 3D Geodatabase Design, OTB – TUDelft (assignment for MSc. GIMA module 6).

Panzer Z. and Becker D., 2006, (information sheet on) Pilot 3D der GDI NRW Stufe II, CeGI Center for Geoinformation GmbH, Dortmund und Stadtplanungsamt, Köln.

Penninga F., 2005, 3D Topographic data modeling, why rigidity is preferable to pragmatism, in 'Spatial Information Theory, Cosit'05', Vol. 3693 of Lecture Notes on Computer Science, Springer, p. 409–425.

Penninga F., 2006, Towards 3D topography using a Feature-based integrated TIN/TEN model, 8th AGILE Conference on GIScience, Estoril, May 26-28, 9 p.

Penninga F., Oosterom, P.J.M. van and Kazar M.B., 2006 [1], A Tetrahedronized Irregular Network based approach for 3D topographic data modelling, the 12th International Symposium on Spatial Data Handling (SDH 2006), as part of GICON 2006, July 2006, Vienna, Austria.

Penninga F., and Oosterom P.J.M. van, 2006 [2], GIST 43, Editing features in a TEN based DBMS approach for 3D Topographic Data Modeling, Geo-Database Management Centre (GDMC), TUDelft.

Prins M., 2003, Is GML only for Internet GIS?, published in Directions magazine.

Quack W, Stoter J and Tijssen T, 2006, Topology in spatial DBMSs, The 3rd international symposium on digital earth, Brno, September, 2003.

Rančić D. and Dačić D., 2005, Ginis Web 3D Modeler – A Framework for 3D Terrain Visualization on the Web, Agile conference, 2005.

Refractions Research Inc., 2004, Geoserver website, (<http://www.refractions.net/geoserver/>)

Refractions Research Inc., 2007, PostGIS: home website, (<http://postgis.refractions.net/>)

Romkema M., 2006 Level of detail in 3D geo-information. OTB – TUDelft (assignment for MSc. GIMA).



Rossum G. van, 1999, Python workshop (presentation), Corporation for National Research Initiatives, Virginia.

Russel J., 2007, Tahiti Views: First thoughts about Python vs. PL/SQL, (<http://tahitiviews.blogspot.com/2007/04/first-thoughts-about-python-vs-plsql.html>)

Schöberl J, 2007, Netgen - Automatic mesh generator. Johannes Kepler University, Linz, (<http://www.hp fem.jku.at/netgen/>)

RGI, 2007, RGI-011: 3D Topografie, (<http://www.rgi-otb.nl/3dtopo/index.htm>)

Sgrillo R., 2007, KML Color Converter (freeware), Personal pages of Richard Sgrillo, Brazil, ([http://www.sgrillo.net/kml\\_color/](http://www.sgrillo.net/kml_color/))

Shi W., 2007, Google Earth, Herausforderung und Chance, 21st Informationsveranstaltung Bayerisches Landesamt für Vermessung und Geoinformation, March 2007, München

Si H., 2007, TetGen. A quality tetrahedral mesh generator and three-dimensional Delaunay triangulator, Research Group Numerical Mathematics and Scientific Computing - Feierstrass institute for Applied Analysis and Stochasitcs – Berlin, (<http://tetgen.berlios.de/>)

Sun Microsystems, 2007, Java SE Technologies – Database, (<http://java.sun.com/javase/technologies/database/>)

Snowflake Software, 2007, Products: Go Publisher, Southampton, (<http://www.snowflakesoftware.co.uk/products/gopublisher/index.htm>)

Stoter J. and Oosterom P.J.M. van, 2002, Incorporating 3D Geo-objects in a 2D Geo-DBMS, ACSM-ASPRS 2002 Annual Conference Proceedings, Washington, D.C.

Stoter J. and Zlatanova S., 2003 [1], 3D GIS where are we standing?, Joint Workshop on Spatial, Temporal and Multi-Dimensional Data Modeling and Analysis, 2-3 October, 2003, Quebec city, Canada.

Stoter J. and Zlatanova S., 2003 [2], 3D GIS where are we standing? Geo-Database Management Centre (GDMC), TUDelft.

SVG Foundation, 2007, SVGI: SVG Implementations- SVG Viewers, (<http://www.svgi.org/directory.html?type=1>)

The Apache Software Foundation, 2007 [1], Mod\_python – Apache / Python integration, (<http://www.modpython.org/>)

The Apache Software Foundation., 2007 [2] Welcome!, The Apache HTTP Server Project, (<http://httpd.apache.org/>)

The Oracle FAQ, 2007, PL/SQL, What is PL/SQL and what is it used for, ([http://www.oraFAQ.com/faq/what\\_is\\_pl\\_sql\\_and\\_what\\_is\\_it\\_used\\_for](http://www.oraFAQ.com/faq/what_is_pl_sql_and_what_is_it_used_for))

Parisi T.,2006, About Ajax3D, (<http://www.ajax3d.org/about.html>)

University of Minnesota, 2007, Welcome to MapServer – UMN MapServer, (<http://mapserver.gis.umn.edu/>)

Verbree E., Penninga F. and Zlatanova S., 2005, GIST 35, Data modeling and data structures for 3D topography, Geo-Database Management Centre (GDMC), TUDelft.

<http://www.gdmc.nl/puitions/reports/GIS35.pdf>

Vries, M.E. de and Stoter J.E., 2003, accessing a 3D geo-DBMS using Web technology, In Proceedings of the ISPRS WG II/5, IV/1 and IV/2 joint workshop on 'spatial, temporal and multi-dimensional data modelling and analysis' (p. 1-8), London.

Vries, M.E. de and Zlatanova S, 2004, Interoperability on the web: the case of 3D geo-data, In P Isaias, P Kommers & M McPherson (Eds.), Proceedings of the IADIS international conference e-society (pp. 667-674). Lisbon: IADIS Press.

Wallace M., 2006, 3D Point D – The metaverse and 3d Web, as blogged by Mark Wallace and friends. Item: What is a building? What does it do?

(<http://www.3pointd.com/20060705/what-is-a-building-what-does-it-do/>)

Web 3D Consortium, 2006 [1], ISO/IEC 19775, Information Technology – Computer graphics and image processing – Extensible 3D (X3D) – Part 1: Architecture and base components – 1 Scope, (<http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/>)

Web 3D Consortium, 2006 [2], ISO/IEC 19775, Information Technology – Computer graphics and image processing – Extensible 3D (X3D) – Part 2: Scene Access Interface (SA) – 1 Scope, (<http://www.web3d.org/x3d/specifications/ISO-IEC-19775-3DAbstractSpecification/>)

Web 3D Consortium, 2006 [3], VRML1997 ISO/IEC 14772-1:1997 –1 Scope, Information Technology – Computer graphics and image processing – The Virtual Reality Modeling Language – Part 1: Functional specification and UTF-8 encoding – 1 Scope, (<http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/part1/scope.html>)

Web 3D Consortium, 2007 [1], Basic, Geo spatial,  
<http://www.web3d.org/x3d/content/examples/Basic/GeoSpatial/>

Web 3D Consortium, 2007 [2], Web3D.org: X3D Viewers, Browsers & Plug-ins,  
([http://www.web3d.org/tools/viewers\\_and\\_browsers/](http://www.web3d.org/tools/viewers_and_browsers/))

Wesselingh S.P., 2006, Visualization of 3d geo-information in ArcScene, TNO Built Environment and Geosciences – Geological Survey of the Netherlands (internship report for MSc. GIMA).

Wesselingh S.P. and Chong S.C., 2005, 3D Web Access for Spatial Planning, OTB – TUDelft (assignment for MSc. GIMA, module 6).

W3 Schools, 2007 [1], JavaScript Introduction, ([http://www.w3schools.com/js/js\\_intro.asp](http://www.w3schools.com/js/js_intro.asp))

W3 Schools, 2007 [2], XSLT on the Client, ([http://www.w3schools.com/xsl/xsl\\_client.asp](http://www.w3schools.com/xsl/xsl_client.asp))

W3C, 2005, Document Object Model (DOM), <http://www.w3.org/DOM/>)

Wikipedia EN, 2007 [1], Oracle Spatial, ([http://en.wikipedia.org/wiki/Oracle\\_Spatial](http://en.wikipedia.org/wiki/Oracle_Spatial))

Wikipedia EN, 2007 [2], SQL, (<http://en.wikipedia.org/wiki/SQL>)

Wikipedia EN, 2007 [3], VRML, (<http://en.wikipedia.org/wiki/VRML>)

Zlatanova. S, 2000, 3D GIS for Urban Development – Chapter eight, Published PhD. thesis, ITC, the Netherlands and ICGV, Austria.

Zu C., Tan E.C., Chang T.K.Y., 2004, 3D Terrain visualization for Web GIS, Map Asia 2003, Kuala Lumpur, Malaysia.

# Appendices

## Introduction

The appendices are ordered per chapter, for example appendix 6.10 is the 10th appendix in Chapter six.

## Appendices chapter two

### Appendix 2.1: Topologic modeling with GML 3

---

```
<gml: TopoSolid gml:id="B1">
  <gml:directFace orientation="+" xlink:href="#F1"/>
  <gml:directFace orientation="+" xlink:href="#F2"/>
  ...
</gml: TopoSolid gml:id="B1">
...

<gml: Face gml:id="F1">
  <gml:directedEdge orientation="+" xlink:href="#c"/>
  <gml:directedEdge orientation="-" xlink:href="#b"/>
  <gml:directedEdge orientation="-" xlink:href="#a"/>
</gml:Face>
<gml: Face gml:id="F2">
  ...
</gml:Face>

<gml:Edge gml:id="a">
<gml:directedNode orientation="-" xlink:href="#A"/>
<gml:directedNode orientation="+" xlink:href="#B"/>
</gml:Edge>
<gml:Edge gml:id="b">
..
</gml:Edge>

</gml: Node gml:id="A">
  <gml: pointProperty>
    <gml: Point>
      <gml: coordinates>
        0.0, 0.0, 0.0
      </gml: coordinates>
    </gml: point>
  </gml: pointProperty>
</gml: Node>
<gml: Node gml:id="B">
...
</gml: Node>
```

---

### Appendix 2.2: Speed tests

Calculate big dataset constrained triangles table (not the view), fast algorithm (incl 0 layer, air and ground ):

11 min

Calculate big dataset constrained triangles table (not the view), fast algorithm (excl 0 layer, air and ground):

1:30 min

Calculate big dataset constrained triangles table (not the view), Joinquery algorithm (incl 0 layer, air and ground) min ): 15 min

Calculate big dataset constrained triangles table (not the view), Joinquery algorithm (excl 0 layer, air and ground) 0:45 min  
Calculate big dataset tetrahedra, fast algorithm (incl 0 layer, air and ground): more than 4 hours (aborted)  
Calculate big dataset tetrahedra, fast algorithm (excl 0 layer, air and ground): 5 min  
Calculate big dataset tetrahedra, slow algorithm (incl 0 layer, air and ground): 6.5 hr  
Calculate big dataset tetrahedra, slow algorithm (excl 0 layer, air and ground): 15 hr

## Appendices chapter three

### Appendix 3.1: Additional information about HTML

HTML markup consists of several types including entities, elements, attributes, data-types, and character references.

Elements are the basis structure for HTML, which has two basic properties: attributes and contents. For example, the tag `<h1>Sijmen</h1>` makes up the word Sijmen (contents) as heading 1.

Attributes are written inside the tags. The attributes of an element are name-value pairs, separated by "=", and written within the start label of an element, after the element's name. The value should be enclosed in single or double quotes, although values consisting of certain characters can be left unquoted in HTML. For example `span id='anId' class='aClass' style='color:red;' title='HyperText Markup Language'>HTML</span>`  
HTML also defines several data types for element content, such as script data and stylesheet data, and a plethora of types for attribute values, including IDs, names, Uniform Resource Identifiers, numbers, units of length, languages, media descriptors, colours, character encodings, dates and times, and so on. All of these data types are specializations of character data.

As of version 4.0, HTML defines a set of 252 character entity references and a set of 1,114,050 numeric character references, both of which allow individual characters to be written via simple markup, rather than literally. A literal character and its markup equivalent are considered equivalent and are rendered identical.

### Appendix 3.2

Oracle spatial consists of:

- A schema that prescribes the storage, syntax, and semantics of supported geometric types.
- A spatial indexing system.
- Operators, functions, and procedures for performing area-of-interest queries, spatial join queries and other spatial analysis operations.
- Functions and procedures for utility and tuning operations.
- A topology data model for working with data about nodes, edges, and faces in a topologic model.
- A network data model for representing capabilities or objects, modelled as nodes and links in a network.

A geo-raster feature to store, index, query, analyze, and deliver geo-raster data as raster image and grid and its associated metadata.

## Appendices chapter four

### Appendix 4.1: VRML fragment

```
#VRML V2.0 utf8
```

```
# Red cone

Shape {
  appearance Appearance {
    material Material {
      diffuseColor 1 0 0
    }
  }
  geometry Cone {
    bottomRadius 0.75
    height 1.6
  }
}
```

(Wikipedia EN, 2007 [3])

## Appendices chapter six

### Appendix 6.1: Database connection

To connect with the database in Python with PyODBC the following line is used:

```
cnxn =
pyodbc.connect('DRIVER={driver};SERVER=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=hostname)(PORT=1521))(CONNECT_
DATA=(SERVICE_NAME=servicename)));UID=username;PWD=password')
```

For the small test dataset the following connection line is used:

```
cnxn = pyodbc.connect('DRIVER={Microsoft ODBC for
Oracle};SERVER=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=casagrande.otb.tudelft.nl)(PORT=1521))(CONNECT_DATA=(SER
VICE_NAME=gisbase)));UID=sijmen;PWD=sijmen')
```

For the large test dataset the following connection line is used:

```
cnxn = pyodbc.connect('DRIVER={Microsoft ODBC for
Oracle};SERVER=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=casagrande.otb.tudelft.nl)(PORT=1521))(CONNECT_DATA=(SER
VICE_NAME=gisbase)));UID=devries;PWD=marian')
```

### Appendix 6.2: Original data as triangles with node references and nodes with coordinates

Triangle table

TRIANGLES			
P1	P2	P3	ID
009	023	027	003
010	014	023	003
014	015	016	003
010	014	025	003

Nodes table

NODES			
ID	X	Y	Z
9,	14,	25,	2

10,	14,	35,	2
14,	14,	35,	8
15,	18,	35,	11
16,	22,	35,	8
23,	14,	30.004904,	2
25,	18.001438,	35,	2
27,	14,	29.9975260,	8

### Appendix 6.3: Loading the face data

```

for i in theRange:
  orientation=triangleortetcode_table[i][0]
  l = triangleortetcode_table[i][1:] #[0:-1]
  #print l
  l2 = []
  l2.append([0:itemlength])
  l2.append([itemlength:2*itemlength])
  l2.append([2*itemlength:3*itemlength])
  l2.append([3*itemlength:4*itemlength])
  #print l2
  for i2 in range(len(l2)):
    if l2[i2] == zerostring:
      l2[i2] = 0
    else:
      while l2[i2][0] == '0':
        l2[i2] = l2[i2][1:]
      l2[i2]=int(l2[i2])

  l3 = (l2[3],l2[0],l2[1],l2[2])
  l4 = (l2[3],l2[0],l2[2],l2[1])

  if excludelayers_boolean == 0:
    if orientation == "+": trianglecode_table2.append(l3)
    if orientation == "-": trianglecode_table2.append(l4)

  if excludelayers_boolean == 1:
    if excludelayers_layers.count(l2[3]) > 0: #if layer is in the list of excluded layers
      pass
    else:
      if orientation == "+": trianglecode_table2.append(l3)
      if orientation == "-": trianglecode_table2.append(l4)

```

### Appendix 6.4: Five first faces of the array generated by the Joinquery algorithm.

```

('0', '1', '0.0', '0.0', '0.0', '2', '0.0', '1.0', '10.0', '20', '20.0', '25.0', '25.0')
('0', '1', '0.0', '0.0', '0.0', '20', '20.0', '25.0', '25.0', '8', '40.0', '0.0', '0.0')
('0', '1', '0.0', '0.0', '0.0', '29', '20.5999837603', '0.0', '5.14999594008', '2', '0.0', '1.0', '10.0')
('0', '1', '0.0', '0.0', '0.0', '8', '40.0', '0.0', '0.0', '29', '20.5999837603', '0.0', '5.14999594008')
('0', '10', '14.0', '2.0', '35.0', '14', '14.0', '8.0', '35.0', '25', '18.001438', '2.0', '35.0')

```

### Appendix 6.5: Creating array of unique nodes per object, array of objects ID and array of faces with point references.

```
objectCounter = 0
```

```

IndexedFaceSetlist_per_object = [[]]
IndexedFaceSetlist_per_object_id = []
pointIDlist_per_object = [[]]
theRange = range(len(trianglecode_table2))
#theRange.pop(0)
IndexedFaceSetlist_per_object_id = [[]]
for i in theRange:
    pid = trianglecode_table2[i][0]
    p1 = trianglecode_table2[i][1]
    p2 = trianglecode_table2[i][2]
    p3 = trianglecode_table2[i][3]

    if pointIDlist_per_object[objectCounter].count(p1) == 0:
        pointIDlist_per_object[objectCounter].append(p1)
    if pointIDlist_per_object[objectCounter].count(p2) == 0:
        pointIDlist_per_object[objectCounter].append(p2)
    if pointIDlist_per_object[objectCounter].count(p3) == 0:
        pointIDlist_per_object[objectCounter].append(p3)

    IndexedFaceSetlist_per_object[objectCounter].append((p1,p2,p3))
    IndexedFaceSetlist_per_object_id[objectCounter] = pid
    if not i == theRange[-1]:
        if not trianglecode_table2[i+1][0] == trianglecode_table2[i][0]:
            objectCounter = objectCounter + 1
            IndexedFaceSetlist_per_object.append([])
            pointIDlist_per_object.append([])
            IndexedFaceSetlist_per_object_id.append([])

```

### Appendix 6.6: Calculating indexes to nodes for every object

```

#Calculates local indexes for the indexed faceset
IndexedFaceSetlist_per_object_newindex = IndexedFaceSetlist_per_object
#copy.deepcopy(IndexedFaceSetlist_per_object)
for i in range(len(IndexedFaceSetlist_per_object)):
    pointIDlist_i = pointIDlist_per_object[i] #copy.deepcopy(pointIDlist_per_object[i])
    # pointIDlist_i.sort()
    for i2 in range(len(IndexedFaceSetlist_per_object[i])):
        face_i_i2 = IndexedFaceSetlist_per_object[i][i2]
        p1 = face_i_i2[0]
        p2 = face_i_i2[1]
        p3 = face_i_i2[2]
        q1 = pointIDlist_i.index(p1)
        q2 = pointIDlist_i.index(p2)
        q3 = pointIDlist_i.index(p3)
        IndexedFaceSetlist_per_object_newindex[i][i2] = (q1, q2, q3)

```

### Appendix 6.7: Creating for each object a list with its unique point coordinates

```

'Original' algorithm

pointXYZlist_per_object = pointIDlist_per_object #copy.deepcopy(pointIDlist_per_object)
for i in range(len(pointIDlist_per_object)):
    for i2 in range(len(pointIDlist_per_object[i])):
        for i3 in range(len(point_table2)):
            if pointIDlist_per_object[i][i2] == point_table2[i3][0]:
                pointXYZlist_per_object[i][i2] = (point_table2[i3][1], point_table2[i3][2], point_table2[i3][3])

'Faster' algorithm

pointXYZlist_per_object = []

```



```

for i in theRange:
    if outputdata == "X3D" or outputdata == "FlatX3D":
        Coordliststring_per_object.append('point="')
        theRange2= range(len(pointIDlist_per_object[i]))
        pointXYZlist_per_object.append([])
        for i2 in theRange2:

            facepx = -1+pointIDlist_per_object[i][i2]
            facepx_xyz = (point_table2[facepx][1], point_table2[facepx][2], point_table2[facepx][3])
            if outputdata == "X3D":
                Coordliststring_per_object[i] = Coordliststring_per_object[i] + str(facepx_xyz[0]) + ' ' + str(facepx_xyz[1]) + ' ' +
                    str(facepx_xyz[2]) + ' '
            if outputdata == "FlatX3D":
                Coordliststring_per_object[i] = Coordliststring_per_object[i] + str(facepx_xyz[0]) + ' ' + '0' + ' ' + str(facepx_xyz[2]) +
                ' '
            pointXYZlist_per_object[i].append(facepx_xyz)

```

### Appendix 6.8: Unique point coordinates per object

```

[[ (14, 25, 2), (14, 35, 2),
  (14,35,8), (18,35,11), (22,35,8), (14,30.004904,2), (18.001438,35,2), (14,29.9975260,8)][...next object...]

```

### Appendix 6.9: Deriving the IndexedFaceSetString / KML , GML and SVG positionlist string

```

if outputdata == "X3D":
    IndexedFaceSetstring_per_object=[]
elif outputdata == "KML":
    KMLposliststring_per_object_newindex=[]
elif outputdata == "CityGML":
    GMLposliststring_per_object_newindex=[]
elif outputdata == "SVG":
    SVGposliststring_per_object_newindex=[]
elif outputdata == "FlatX3D":
    IndexedFaceSetstring_per_object=[]
for i in range(len(IndexedFaceSetlist_per_object_newindex)):
    if outputdata == "X3D":
        IndexedFaceSetstring_per_object.append('IndexedFaceSet solid="false" coordIndex="')
    elif outputdata == "KML":
        KMLposliststring_per_object_newindex.append([])
    elif outputdata == "CityGML":
        GMLposliststring_per_object_newindex.append([])
    elif outputdata == "SVG":
        SVGposliststring_per_object_newindex.append([])
    colorindex = colourandtransparencyarray_id.index(IndexedFaceSetlist_per_object_id[i])
    mycolorHTML = colourandtransparencyarray[colorindex][0]
    mytransparencyINT = colourandtransparencyarray[colorindex][1]
    #print "transp: " + str(mytransparencyINT)
    #print str(RGBToHTMLtransparency(mytransparencyINT))
    if layerselectarray.count(IndexedFaceSetlist_per_object_id[i]) > 0:
        mycolorSVG = "#FFFF00"
        #print "sel " + mycolorKML
    else:
        #print mycolorHTML
        mycolorSVG = mycolorHTML
        #print "non sel " + mycolorKML
    elif outputdata == "FlatX3D":
        IndexedFaceSetstring_per_object.append('IndexedFaceSet solid="false" coordIndex="')
    for i2 in range(len(IndexedFaceSetlist_per_object_newindex[i])):
        face_id = IndexedFaceSetlist_per_object_id[i]
        facepx_i2 = IndexedFaceSetlist_per_object[i][i2]
        faceq_i2 = IndexedFaceSetlist_per_object_newindex[i][i2]

```

```

face_p1 = faceq_i_i2[0]
face_p2 = faceq_i_i2[1]
face_p3 = faceq_i_i2[2]
face_q1 = faceq_i_i2[0]
face_q2 = faceq_i_i2[1]
face_q3 = faceq_i_i2[2]
if outputdata == "X3D":
    #q1, q2 and q3 are the local ids for the faces within one indexedfaceset as not all faces are
    included
    IndexedFaceSetstring_per_object[i] = IndexedFaceSetstring_per_object[i] + str(face_q1) + ' ' +
    str(face_q2) + ' ' + str(face_q3) + ' -1, '
elif outputdata == "KML":KMLposliststring_per_object_newindex[i].append('<coordinates>')
    face_p1_xyz = pointXYZlist_per_object[i][face_q1] #face_p1_xyz = point_table2[-1+face_p1]
    face_p2_xyz = pointXYZlist_per_object[i][face_q2] #face_p2_xyz = point_table2[-1+face_p2]
    face_p3_xyz = pointXYZlist_per_object[i][face_q3] #face_p3_xyz = point_table2[-1+face_p3]
    KMLposliststring_per_object_newindex[i][i2] = KMLposliststring_per_object_newindex[i][i2] + ' ' +
    str(face_p1_xyz[0]) + ',' + str(face_p1_xyz[1]) + ',' + str(face_p1_xyz[2])
    KMLposliststring_per_object_newindex[i][i2] = KMLposliststring_per_object_newindex[i][i2] + ' ' +
    str(face_p2_xyz[0]) + ',' + str(face_p2_xyz[1]) + ',' + str(face_p2_xyz[2])
    KMLposliststring_per_object_newindex[i][i2] = KMLposliststring_per_object_newindex[i][i2] + ' ' +
    str(face_p3_xyz[0]) + ',' + str(face_p3_xyz[1]) + ',' + str(face_p3_xyz[2])
    KMLposliststring_per_object_newindex[i][i2] = KMLposliststring_per_object_newindex[i][i2] + ' ' +
    str(face_p1_xyz[0]) + ',' + str(face_p1_xyz[1]) + ',' + str(face_p1_xyz[2]) # ring is closed with repeat 1st
    coordinate
    KMLposliststring_per_object_newindex[i][i2] = KMLposliststring_per_object_newindex[i][i2] +
    '</coordinates>'

elif outputdata == "CityGML":

    GMLposliststring_per_object_newindex[i].append('<gml:posList srsDimension="3">')
    face_p1_xyz = pointXYZlist_per_object[i][face_q1] #face_p1_xyz = point_table2[-1+face_p1]
    face_p2_xyz = pointXYZlist_per_object[i][face_q2] #face_p2_xyz = point_table2[-1+face_p2]
    face_p3_xyz = pointXYZlist_per_object[i][face_q3] #face_p3_xyz = point_table2[-1+face_p3]
    GMLposliststring_per_object_newindex[i][i2] = GMLposliststring_per_object_newindex[i][i2] + ' ' +
    str(face_p1_xyz[0]) + ' ' + str(face_p1_xyz[1]) + ' ' + str(face_p1_xyz[2])
    GMLposliststring_per_object_newindex[i][i2] = GMLposliststring_per_object_newindex[i][i2] + ' ' +
    str(face_p2_xyz[0]) + ' ' + str(face_p2_xyz[1]) + ' ' + str(face_p2_xyz[2])
    GMLposliststring_per_object_newindex[i][i2] = GMLposliststring_per_object_newindex[i][i2] + ' ' +
    str(face_p3_xyz[0]) + ' ' + str(face_p3_xyz[1]) + ' ' + str(face_p3_xyz[2])
    GMLposliststring_per_object_newindex[i][i2] = GMLposliststring_per_object_newindex[i][i2] + ' ' +
    str(face_p1_xyz[0]) + ' ' + str(face_p1_xyz[1]) + ' ' + str(face_p1_xyz[2]) # ring is closed with repeat 1st
    coordinate
    GMLposliststring_per_object_newindex[i][i2] = GMLposliststring_per_object_newindex[i][i2] +
    '</gml:posList>'

elif outputdata == "SVG":

    if xlinktarget == "Table":
        SVGposliststring_per_object_newindex[i].append('<polygon onclick="" points=""')
    if xlinktarget == "3DView":
        SVGposliststring_per_object_newindex[i].append('<polygon onclick="setViewpoint(' + str(face_id)
        + ')" points=""')
    face_p1_xyz = pointXYZlist_per_object[i][face_q1] #face_p1_xyz = point_table2[-1+face_p1]
    face_p2_xyz = pointXYZlist_per_object[i][face_q2] #face_p2_xyz = point_table2[-1+face_p2]
    face_p3_xyz = pointXYZlist_per_object[i][face_q3] #face_p3_xyz = point_table2[-1+face_p3]
    SVGposliststring_per_object_newindex[i][i2] = SVGposliststring_per_object_newindex[i][i2] +
    str(face_p1_xyz[0]) + ',' + str(face_p1_xyz[2]) + ' '
    SVGposliststring_per_object_newindex[i][i2] = SVGposliststring_per_object_newindex[i][i2] +
    str(face_p2_xyz[0]) + ',' + str(face_p2_xyz[2]) + ' '
    SVGposliststring_per_object_newindex[i][i2] = SVGposliststring_per_object_newindex[i][i2] +
    str(face_p3_xyz[0]) + ',' + str(face_p3_xyz[2])
    SVGposliststring_per_object_newindex[i][i2] = SVGposliststring_per_object_newindex[i][i2] + ""
    =style="fill:' + mycolorSVG + ';stroke:' + mycolorSVG + ';stroke-width:0.0;opacity:' + str(1-
    mytransparencyINT) + '" transform="translate(0 0)" id="SVGSubElement' + str(face_id) + '_' + str(i2) +

```

```

"/>

elif outputdata == "FlatX3D":

    IndexedFaceSetstring_per_object[i] = IndexedFaceSetstring_per_object[i] + str(face_q1) + ' ' +
    str(face_q2) + ' ' + str(face_q3) + ' -1, '

if outputdata == "X3D":
    IndexedFaceSetstring_per_object[i] = '<' + IndexedFaceSetstring_per_object[i] + '>'
elif outputdata == "KML":
    pass
elif outputdata == "CityGML":
    pass
elif outputdata == "SVG":
    pass
elif outputdata == "FlatX3D":
    IndexedFaceSetstring_per_object[i] = '<' + IndexedFaceSetstring_per_object[i] + '>'

```

### Appendix 6.10: Deriving the coordinate list string

```

#Format lines of coordlist
Coordliststring_per_object = []
for i in range(len(pointXYZlist_per_object)):
    start_text = 'point='
    Coordliststring_per_object.append(start_text)
    for i2 in range(len(pointXYZlist_per_object[i])):
        myitemxyz = pointXYZlist_per_object[i][i2]
        myitemx = myitemxyz[0]
        myitemy = myitemxyz[1]
        myitemz = myitemxyz[2]
        Coordliststring_per_object[i] = Coordliststring_per_object[i] + str(myitemx) + ' ' + str(myitemy) + ' ' +
        str(myitemz) + ', '

Coordliststring_per_object[i] = Coordliststring_per_object[i] + "" />

```

### Appendix 6.11: Loop the facearray and monitor change of object ID

```

for i2 in theRange:

    i3 = i3 + 1

    facex_OBJECTID = facearray[i2][0]
    if (i2 == 0):
        facex_OBJECTIDprev = -1
    else:
        facex_OBJECTIDprev = facearray[i2-1][0]
    if (i2 == len(theRange)-1):
        facex_OBJECTIDnext = -1
    else:
        facex_OBJECTIDnext = facearray[i2+1][0]

    if not (facex_OBJECTID == facex_OBJECTIDprev):

```

### Appendix 6.12: Loop the facearray and monitor change of object ID

```

i = i+1
i3 = 0

```

```
if outputdata == "X3D":
    IndexedFaceSetstring_per_object.append("IndexedFaceSet solid="false" coordIndex="")

#print "check1"
uniquepointidlist_per_object.append([])
#print uniquepointidlist_per_object
Coordliststring_per_object.append("point=")
pointXYZlist_per_object.append([])
IndexedFaceSetlist_per_object_id.append(int(facex_OBJECTID))
```

## Appendix 6.13: Loop the facearray and monitor change of object ID

```
face_id = facearray[i2][0]
face_p1 = facearray[i2][1]
face_p2 = facearray[i2][5]
face_p3 = facearray[i2][9]
face_p1_xyz = (facearray[i2][2], facearray[i2][3], facearray[i2][4])
face_p2_xyz = (facearray[i2][6], facearray[i2][7], facearray[i2][8])
face_p3_xyz = (facearray[i2][10], facearray[i2][11], facearray[i2][12])

if uniquepointidlist_per_object[i] == []:
    #print "not here"
    uniquepointidlist_per_object[i].append(face_p1)
    uniquepointidlist_per_object[i].append(face_p2)
    uniquepointidlist_per_object[i].append(face_p3)
    Coordliststring_per_object[i] = Coordliststring_per_object[i] + str(face_p1_xyz[0]) + ' ' + str(face_p1_xyz[1]) + ' ' +
    str(face_p1_xyz[2]) + ', '
    Coordliststring_per_object[i] = Coordliststring_per_object[i] + str(face_p2_xyz[0]) + ' ' + str(face_p2_xyz[1]) + ' ' +
    str(face_p2_xyz[2]) + ', '
    Coordliststring_per_object[i] = Coordliststring_per_object[i] + str(face_p3_xyz[0]) + ' ' + str(face_p3_xyz[1]) + ' ' +
    str(face_p3_xyz[2]) + ', '
    pointXYZlist_per_object[i].append((float(face_p1_xyz[0]), float(face_p1_xyz[1]), float(face_p1_xyz[2])))
    pointXYZlist_per_object[i].append((float(face_p2_xyz[0]), float(face_p2_xyz[1]), float(face_p2_xyz[2])))
    pointXYZlist_per_object[i].append((float(face_p3_xyz[0]), float(face_p3_xyz[1]), float(face_p3_xyz[2])))
else:
    #print "here"
    #print uniquepointidlist_per_object[i]
    #print face_p1
    #print "..."
    if not (uniquepointidlist_per_object[i].count(face_p1) == 0):
        pass
    else:
        #print "p1 here"
        uniquepointidlist_per_object[i].append(face_p1)
        Coordliststring_per_object[i] = Coordliststring_per_object[i] + str(face_p1_xyz[0]) + ' ' + str(face_p1_xyz[1]) + ' ' +
        str(face_p1_xyz[2]) + ', '
        pointXYZlist_per_object[i].append((float(face_p1_xyz[0]), float(face_p1_xyz[1]), float(face_p1_xyz[2])))
    if not (uniquepointidlist_per_object[i].count(face_p2) == 0):
        pass
    else:
        #print "p2 here"
        uniquepointidlist_per_object[i].append(face_p2)
        Coordliststring_per_object[i] = Coordliststring_per_object[i] + str(face_p2_xyz[0]) + ' ' + str(face_p2_xyz[1]) + ' ' +
        str(face_p2_xyz[2]) + ', '
        pointXYZlist_per_object[i].append((float(face_p2_xyz[0]), float(face_p2_xyz[1]), float(face_p2_xyz[2])))
    if not (uniquepointidlist_per_object[i].count(face_p3) == 0):
        pass
    else:
        #print "p3 here"
        uniquepointidlist_per_object[i].append(face_p3)
        Coordliststring_per_object[i] = Coordliststring_per_object[i] + str(face_p3_xyz[0]) + ' ' + str(face_p3_xyz[1]) + ' ' +
        str(face_p3_xyz[2]) + ', '
        pointXYZlist_per_object[i].append((float(face_p3_xyz[0]), float(face_p3_xyz[1]), float(face_p3_xyz[2])))
face_q1 = uniquepointidlist_per_object[i].index(face_p1)
face_q2 = uniquepointidlist_per_object[i].index(face_p2)
face_q3 = uniquepointidlist_per_object[i].index(face_p3)
```

## Appendix 6.14: Filling up the IndexedFaceSet array

```
if outputdata == "X3D":

    IndexedFaceSetstring_per_object[i] = IndexedFaceSetstring_per_object[i] + str(face_q1) + ' ' + str(face_q2) + ' ' +
    str(face_q3) + ' -1, '

elif outputdata == "KML":

    KMLposliststring_per_object_newindex[i].append('<coordinates>')

    KMLposliststring_per_object_newindex[i][i3] = KMLposliststring_per_object_newindex[i][i3] + ' ' + str(face_p1_xyz[0]) +
    ' ' + str(face_p1_xyz[1]) + ' ' + str(face_p1_xyz[2])
    KMLposliststring_per_object_newindex[i][i3] = KMLposliststring_per_object_newindex[i][i3] + ' ' + str(face_p2_xyz[0]) +
    ' ' + str(face_p2_xyz[1]) + ' ' + str(face_p2_xyz[2])
    KMLposliststring_per_object_newindex[i][i3] = KMLposliststring_per_object_newindex[i][i3] + ' ' + str(face_p3_xyz[0]) +
    ' ' + str(face_p3_xyz[1]) + ' ' + str(face_p3_xyz[2])
    KMLposliststring_per_object_newindex[i][i3] = KMLposliststring_per_object_newindex[i][i3] + ' ' + str(face_p1_xyz[0]) +
    ' ' + str(face_p1_xyz[1]) + ' ' + str(face_p1_xyz[2]) # ring is closed with repeat 1st coordinate
    KMLposliststring_per_object_newindex[i][i3] = KMLposliststring_per_object_newindex[i][i3] + '</coordinates>'

elif outputdata == "CityGML":

    GMLposliststring_per_object_newindex[i].append('<gml:posList srsDimension="3">')
    GMLposliststring_per_object_newindex[i][i3] = GMLposliststring_per_object_newindex[i][i3] + ' ' + str(face_p1_xyz[0]) +
    ' ' + str(face_p1_xyz[1]) + ' ' + str(face_p1_xyz[2])
    GMLposliststring_per_object_newindex[i][i3] = GMLposliststring_per_object_newindex[i][i3] + ' ' + str(face_p2_xyz[0]) +
    ' ' + str(face_p2_xyz[1]) + ' ' + str(face_p2_xyz[2])
    GMLposliststring_per_object_newindex[i][i3] = GMLposliststring_per_object_newindex[i][i3] + ' ' + str(face_p3_xyz[0]) +
    ' ' + str(face_p3_xyz[1]) + ' ' + str(face_p3_xyz[2])
    GMLposliststring_per_object_newindex[i][i3] = GMLposliststring_per_object_newindex[i][i3] + ' ' + str(face_p1_xyz[0]) +
    ' ' + str(face_p1_xyz[1]) + ' ' + str(face_p1_xyz[2]) # ring is closed with repeat 1st coordinate
    GMLposliststring_per_object_newindex[i][i3] = GMLposliststring_per_object_newindex[i][i3] + '</gml:posList>'

elif outputdata == "SVG":

    if xlinktarget == "Table":
        SVGposliststring_per_object_newindex[i].append('<polygon onclick="" points=""')
    if xlinktarget == "3DView":
        SVGposliststring_per_object_newindex[i].append('<polygon onclick="setViewpoint(' + str(face_id) + ')" points=""')

    SVGposliststring_per_object_newindex[i][i3] = SVGposliststring_per_object_newindex[i][i3] + str(face_p1_xyz[0]) + ' ' +
    str(face_p1_xyz[2]) + ' '
    SVGposliststring_per_object_newindex[i][i3] = SVGposliststring_per_object_newindex[i][i3] + str(face_p2_xyz[0]) + ' ' +
    str(face_p2_xyz[2]) + ' '
    SVGposliststring_per_object_newindex[i][i3] = SVGposliststring_per_object_newindex[i][i3] + str(face_p3_xyz[0]) + ' ' +
    str(face_p3_xyz[2])
    SVGposliststring_per_object_newindex[i][i3] = SVGposliststring_per_object_newindex[i][i3] + ' fill="" + mycolorSVG + ""
    stroke="" + mycolorSVG + "" stroke-width="0.0" opacity="" + str(1-mytransparency/INT) + "" transform="translate(0 0)"
    id="SVGSubElement' + str(face_id) + ' - ' + str(i2) + '"/>'

elif outputdata == "FlatX3D":

    #q1, q2 and q3 are the local ids for the faces within one indexedfaceset as not all faces are included

    IndexedFaceSetstring_per_object[i] = IndexedFaceSetstring_per_object[i] + str(face_q1) + ' ' + str(face_q2) + ' ' +
    str(face_q3) + ' -1, '
```

## Appendix 6.15: Calculation of the viewpoints

```
midXYZlist_per_object = []
viewpointXYZlist_per_object = []

xdatasetmidXYZ = 0
ydatasetmidXYZ = 0
zdatasetmidXYZ = 0
xdatasetmin = pointXYZlist_per_object[i][0][0]
xdatasetmax = pointXYZlist_per_object[i][0][0]
ydatasetmin = pointXYZlist_per_object[i][0][1] # height from
ydatasetmax = pointXYZlist_per_object[i][0][1]
zdatasetmin = pointXYZlist_per_object[i][0][2] # distance from
zdatasetmax = pointXYZlist_per_object[i][0][2]
ymaxlist = [] #to determine SVG drawing order

for i in range(len(pointXYZlist_per_object)):
    midXYZlist_per_object.append(i)
    viewpointXYZlist_per_object.append(i)
    numberofcoordinates = len(pointXYZlist_per_object[i])
    xtotal = 0
    ytotal = 0
    ztotal = 0
    xmin = pointXYZlist_per_object[i][0][0]
    xmax = pointXYZlist_per_object[i][0][0]
    ymin = pointXYZlist_per_object[i][0][1] # height from
    ymax = pointXYZlist_per_object[i][0][1]
    zmin = pointXYZlist_per_object[i][0][2] # distance from
    zmax = pointXYZlist_per_object[i][0][2]

    for i2 in range (len(pointXYZlist_per_object[i])):
        xtotal = xtotal + pointXYZlist_per_object[i][i2][0]
        ytotal = ytotal + pointXYZlist_per_object[i][i2][1]
        ztotal = ztotal + pointXYZlist_per_object[i][i2][2]
        if pointXYZlist_per_object[i][i2][0] < xmin:
            xmin = pointXYZlist_per_object[i][i2][0]
        if pointXYZlist_per_object[i][i2][0] > xmax:
            xmax = pointXYZlist_per_object[i][i2][0]
        if pointXYZlist_per_object[i][i2][1] < ymin:
            ymin = pointXYZlist_per_object[i][i2][1]
        if pointXYZlist_per_object[i][i2][1] > ymax:
            ymax = pointXYZlist_per_object[i][i2][1]
        if pointXYZlist_per_object[i][i2][2] < zmin:
            zmin = pointXYZlist_per_object[i][i2][2]
        if pointXYZlist_per_object[i][i2][2] > zmax:
            zmax = pointXYZlist_per_object[i][i2][2]

    ymaxlist.append(ymax)
    xavg = xtotal / numberofcoordinates
    yavg = ytotal / numberofcoordinates
    zavg = ztotal / numberofcoordinates
    xrange2 = abs(xmax-xmin)
    yrange = abs(ymax-ymin)
    zrange = abs(zmax-zmin)

    #print zrange

    midXYZlist_per_object[i] = (xavg, yavg, zavg)
    viewpointXYZlist_per_object[i] =(xavg, yavg + 1 * yrange , zavg + 4 * zrange)

xdatasetmidXYZ = xdatasetmidXYZ + xavg
ydatasetmidXYZ = ydatasetmidXYZ + yavg
```

```

zdatasetmidXYZ = zdatasetmidXYZ + zavg
if xmin < xdatasetmin:
    xdatasetmin = xmin
if xmax > xdatasetmax:
    xdatasetmax = xmax
if ymin < ydatasetmin:
    ydatasetmin = ymin
if ymax > ydatasetmax:
    ydatasetmax = ymax
if zmin < zdatasetmin:
    zdatasetmin = zmin
if zmax > zdatasetmax:
    zdatasetmax = zmax

xdatasetavg = xdatasetmidXYZ / len(midXYZlist_per_object)
ydatasetavg = ydatasetmidXYZ / len(midXYZlist_per_object)
zdatasetavg = zdatasetmidXYZ / len(midXYZlist_per_object)
zdatasetrange = abs(zdatasetmax-zdatasetmin)
ydatasetrange = abs(ydatasetmax-ydatasetmin)

midXYZlist_dataset = (xdatasetavg, ydatasetavg, zdatasetavg)
viewpointXYZlist_dataset =(xdatasetavg, ydatasetavg + 1 * ydatasetrange , zdatasetavg + 4 *
zdatasetrange)

orientationstring = getorientationstring(viewpointXYZlist_dataset,midXYZlist_dataset)

print "Viewpoints calculated"

```

## Appendix 6.16: X3D generation script and sample X3D code

### X3D Generation script

```

outputXML = []
outputXML.append('<?xml version="1.0" encoding="UTF-8"?>')
outputXML.append('<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.1//EN"
"http://www.web3d.org/specifications/x3d-3.1.dtd">')
outputXML.append('<X3D version="3.0" profile="Interactive"
xmlns:fo="http://www.w3.org/1999/XSL/Format" xmlns:gml="http://www.opengis.net/gml"
xmlns:tdn="http://www.gdmc.nl/tdn"
xmlns:x3d="http://www.web3d.org/TaskGroups/x3d/translation/X3dSchemaDraft.dtd"
xmlns="http://www.web3D.org/TaskGroups/x3d/translation/X3dSchemaDraft.dtd"
xmlns:xlink="http://www.w3.org/1999/xlink">')
outputXML.append('<head>')
outputXML.append('  <meta name="filename" content="visualization of tetrahedronized data from
Oracle" />')
outputXML.append('  <meta name="generator" content="Python 25" />')
outputXML.append('  <meta name="translator" content="Oracle TET to X3D (2007)" />')
outputXML.append('</head>')
outputXML.append('')
outputXML.append('<Scene>')
outputXML.append('')
outputXML.append('<Background groundAngle="1.309, 1.571" groundColor="0.1 0.1 0, 0.4 0.25 0.2, 0.6 0.6
0.6" skyColor="0.0 0.2 0.7, 0.0 0.5 1.0, 1.0 1.0 1.0" skyAngle="1.309, 1.571"/>')
outputXML.append('')
#The Default Viewpoint has to be manually adjusted for each dataset (for now)
outputXML.append('<NavigationInfo type="Examine"/>')
#rotation = '0.2 -1.8 -1.8'
#viewpointstring = "<Viewpoint DEF='Default' description='Default' orientation='0.4 0.5 1 0' position='20 20
100'/>"
#outputXML.append(viewpointstring)

#orientationstring = "0 1 -0.25 0"
positionstring = str(viewpointXYZlist_dataset[0])+" "+str(viewpointXYZlist_dataset[1])+"

```



```

"+str(viewpointXYZlist_dataset[2])
#print (str(IndexedFaceSetlist_per_object_id[i]))
descriptionstring = "Default"
viewpointstring = "<Viewpoint DEF=" + descriptionstring + " description=" + descriptionstring + "
orientation=" + orientationstring + " position=" + positionstring + "/>"
outputXML.append(viewpointstring)
#print viewpointXYZlist_per_object
#print IndexedFaceSetlist_per_object_id
rotationstring = str(dataset_rotation[0]) + ' ' + str(dataset_rotation[1]) + ' ' + str(dataset_rotation[2]) + ' ' +
str(math.pi*0.5)
for i in range(len(IndexedFaceSetstring_per_object_newindex)):
#<Viewpoint DEF='ConeView' description='Red Cone (jump true)' orientation='0 -1 0 1' position='-4.6 1.4
1.9'/>

#orientationstring = "0 1 -0.25 0"
positionstring = str(viewpointXYZlist_per_object[i][0])+" "+str(viewpointXYZlist_per_object[i][1])+
"+str(viewpointXYZlist_per_object[i][2])
#print (str(IndexedFaceSetlist_per_object_id[i]))
descriptionstring = "Viewpoint_" + str(IndexedFaceSetlist_per_object_id[i])
viewpointstring = "<Viewpoint DEF=" + descriptionstring + " description=" + descriptionstring + "
orientation=" + orientationstring + " position=" + positionstring + "/>"
outputXML.append(viewpointstring)
for i in range(len(IndexedFaceSetstring_per_object_newindex)):
#
outputXML.append(' <Transform DEF="Layer_' + str(IndexedFaceSetlist_per_object_id[i]) + "
translation="0.0 0.0 0.0" rotation="0 0 0" center="0 0 0">')
#outputXML.append(' <Anchor description="Anchor_' + str(IndexedFaceSetlist_per_object_id[i]) + "' +
'parameter="target=TableView_Contents" url="http://localhost/sitesdesign/sd4/DisplayTENTable.py?layer='
+ str(IndexedFaceSetlist_per_object_id[i]) + "'>')
anchorURL =
'http://localhost/sitesdesign/sd4/DisplayTENTable.py?selectionmode=false&#38;dataset_name=' +
str(dataset_name) + '&#38;layer=' + str(IndexedFaceSetlist_per_object_id[i])
#anchorURL = 'http://localhost/sitesdesign/sd4/DisplayTENTable.py'
#print anchorURL
anchorDES = ' <Anchor description="Anchor_' + str(IndexedFaceSetlist_per_object_id[i]) + "' +
'parameter="target=TableView_Contents" url="' + anchorURL + "'>'
#print anchorDES
outputXML.append(anchorDES)
#print outputXML
outputXML.append(' <Shape>')
outputXML.append(' <Appearance>')
#print IndexedFaceSetlist_per_object_id[i]
#print colourandtransparencyarray_id
colorindex = colourandtransparencyarray_id.index(IndexedFaceSetlist_per_object_id[i])
mycolorHTML = colourandtransparencyarray[colorindex][0]
mycolorRGB = HTMLColorToRGB(mycolorHTML)
mytransparencyINT = colourandtransparencyarray[colorindex][1]

#print "transp: " + str(mytransparencyINT)
if layerselectarray.count(IndexedFaceSetlist_per_object_id[i]) > 0:
ColorR = 255/255.0
ColorG = 255/255.0
ColorB = 0/255.0
#print "selected:" + str(ColorR) + ' ' + str(ColorG) + ' ' + str(ColorB)
else:
ColorR = mycolorRGB[0]/255.0
ColorG = mycolorRGB[1]/255.0
ColorB = mycolorRGB[2]/255.0
#print mycolorRGB
#print "unselected:" + str(ColorR) + ' ' + str(ColorG) + ' ' + str(ColorB)
outputXML.append(' <Material DEF="MA_Material_' +
str(IndexedFaceSetlist_per_object_id[i]) + " diffuseColor=" + str(ColorR) + ' ' + str(ColorG) + ' ' + str(ColorB)
+ " specularColor="0.401 0.401 0.401" emissiveColor="0.0 0.0 0.0">')
outputXML.append(' ambientIntensity="0.167" shininess="0.098"
transparency=" + str(mytransparencyINT) + " />')
outputXML.append(' </Appearance>')

```

```

outputXML.append('          ' + IndexedFaceSetstring_per_object_newindex[i])
outputXML.append('          <Coordinate DEF="coord_Layer_' +
str(IndexedFaceSetlist_per_object_id[i]) + """)
outputXML.append('          ' + Coordliststring_per_object[i])
outputXML.append('          </IndexedFaceSet>')
outputXML.append('        </Shape>')
outputXML.append('      </Anchor>')
outputXML.append('    </Transform>')

```

```
outputXML.append("")
```

```

outputXML.append("")
outputXML.append("")
outputXML.append('</Scene>')
outputXML.append('</X3D>')

```

```

outX3D = ".join(outputXML)
#print outX3D
#req.content_type="text/xml"
req.content_type="model/x3d+xml"
req.write(outX3D)
#return apache.OK (zet er een nul achter)

```

#### Sample X3D Code

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE X3D PUBLIC "ISO//Web3D//DTD X3D 3.1//EN" "http://www.web3d.org/specifications/x3d-3.1.dtd">
<X3D version="3.0" profile="Interactive" xmlns:fo="http://www.w3.org/1999/XSL/Format"
xmlns:gml="http://www.opengis.net/gml" xmlns:tdn="http://www.gdmc.nl/tdn"
xmlns:x3d="http://www.web3d.org/TaskGroups/x3d/translation/X3dSchemaDraft.dtd"
xmlns="http://www.web3d.org/TaskGroups/x3d/translation/X3dSchemaDraft.dtd" xmlns:xlink="http://www.w3.org/1999/xlink">
<head>
  <meta name="filename" content="visualization of tetrahedronized data from Oracle" />
  <meta name="generator" content="Python 25" />
  <meta name="translator" content="Oracle TET to X3D (2007)" />
</head>
<Scene>
  <Background groundAngle="1.309, 1.571" groundColor="0.1 0.1 0, 0.4 0.25 0.2, 0.6 0.6 0.6" skyColor="0.0 0.2 0.7, 0.0 0.5 1.0, 1.0
1.0 1.0" skyAngle="1.309, 1.571"/>
  <NavigationInfo type="Examine"/>
  <Viewpoint DEF='Default' description='Default' orientation='0.222427095815 0.75 0.75 6.28318530718' position='18.4594724891
38.5146052109 225.638915777'/>
  <Viewpoint DEF='Viewpoint_0' description='Viewpoint_0' orientation='0.222427095815 0.75 0.75 6.28318530718'
position='18.6119009285 29.1844027677 224.464781537'/>
  <Viewpoint DEF='Viewpoint_2' description='Viewpoint_2' orientation='0.222427095815 0.75 0.75 6.28318530718'
position='19.0162285389 13.796912865 222.452094732'/>
  <Viewpoint DEF='Viewpoint_3' description='Viewpoint_3' orientation='0.222427095815 0.75 0.75 6.28318530718'
position='17.750288 14.5625 69.9998710625'/>
  <Transform DEF="Layer_0" translation="0.0 0.0 0.0" rotation="0 0 0" center="0 0 0">
    <Anchor description="Anchor_0" parameter="target=TableView_Contents"
url="http://localhost/sitedesign/sd4/DisplayTENTable.py&#63;selectionmode=false&#38;dataset_name=onehouse&#38;layer=0">
      <Shape>
        <Appearance>
          <Material DEF="MA_Material_0" diffuseColor="0.0 0.0 1.0" specularColor="0.401 0.401 0.401"
emissiveColor="0.0 0.0 0.0"
          ambientIntensity="0.167" shininess="0.098" transparency="0.90" />
        </Appearance>
        <IndexedFaceSet solid="false" coordIndex="0 1 2 -1, 0 1 3 -1, 0 4 2 -1, 0 4 3 -1, 5 6 7 -1, 5 6 8 -1, 9 10 11 -1, 9 10
8 -1, 12 13 11 -1, 12 13 14 -1, 12 15 16 -1, 12 14 16 -1, 17 13 18 -1, 17 13 14 -1, 17 18 19 -1, 6 20 10 -1, 6 20 21 -1, 6 10 8 -1, 6 7 19 -
1, 6 19 21 -1, 20 10 21 -1, 10 13 11 -1, 10 13 21 -1, 13 18 21 -1, 18 19 21 -1, 1 22 3 -1, 1 23 2 -1, 1 23 22 -1, 15 16 24 -1, 14 16 24 -1,
23 5 7 -1, 23 22 15 -1, 23 15 24 -1, 23 25 5 -1, 23 25 2 -1, 23 26 7 -1, 23 26 24 -1, 25 5 8 -1, 25 9 8 -1, 25 27 9 -1, 25 27 2 -1, 27 28 9 -

```

```

1, 27 28 2 -1, 28 9 11 -1, 28 12 15 -1, 28 12 11 -1, 28 29 2 -1, 28 29 15 -1, 29 22 15 -1, 29 22 3 -1, 29 4 2 -1, 29 4 3 -1, 26 17 14 -1, 26
17 19 -1, 26 7 19 -1, 26 14 24 -1, ">
    <Coordinate DEF="coord_Layer_0"
        point="0.0 0.0 0.0, 0.0 1.0 10.0, 20.0 25.0 25.0, 20.5999837603 0.0 5.14999594008, 40.0 0.0
0.0, 14.0 2.0 35.0, 14.0 8.0 35.0, 14.0 2.0 30.004904, 18.001438 2.0 35.0, 22.0 2.0 35.0, 22.0 8.0 35.0, 22.0 2.0 29.995671, 22.0 2.0
25.0, 22.0 8.0 25.0, 18.00317 2.0 25.0, 20.0159350211 0.499601624473 16.0, 18.3040053238 1.83200024199 23.4880021779, 14.0
8.0 25.0, 18.0 11.0 25.0, 14.0 8.0 29.997526, 18.0 11.0 35.0, 18.0 11.0 29.999836, 20.0141355833 0.499646610417 10.0, 0.0 1.0
16.0, 15.4183601662 1.70083455301 22.3075109771, 0.0 2.0 50.0, 14.0 2.0 25.0, 40.0 3.0 50.0, 40.0 0.0 16.0, 40.0 0.0 10.0, " />
    </IndexedFaceSet>
</Shape>
</Anchor>
</Transform>

<Transform DEF="Layer_2" translation="0.0 0.0 0.0" rotation="0 0 0" center="0 0 0">
  <Anchor description="Anchor_2" parameter="target=TableView_Contents"
url="http://localhost/sitedesign/sd4/DisplayTENTable.py#63;selectionmode=false#38;dataset_name=onehouse#38;layer=2">
    <Shape>
      <Appearance>
        <Material DEF="MA_Material_2" diffuseColor="0.458823529412 0.352941176471 0.317647058824"
specularColor="0.401 0.401 0.401" emissiveColor="0.0 0.0 0.0"
ambientIntensity="0.167" shininess="0.098" transparency="0.00" />
      </Appearance>
      <IndexedFaceSet solid="false" coordIndex="0 1 2 -1, 0 1 3 -1, 0 4 2 -1, 0 4 3 -1, 5 6 7 -1, 8 9 7 -1, 10 11 12 -1, 10 9
13 -1, 10 13 12 -1, 1 14 3 -1, 1 15 2 -1, 1 15 14 -1, 11 12 16 -1, 6 9 7 -1, 6 9 13 -1, 13 12 16 -1, 15 5 6 -1, 15 14 11 -1, 15 11 16 -1, 15
17 5 -1, 15 17 2 -1, 15 18 6 -1, 15 18 16 -1, 17 5 7 -1, 17 8 7 -1, 17 19 8 -1, 17 19 2 -1, 19 20 8 -1, 19 20 2 -1, 20 8 9 -1, 20 10 11 -1, 20
10 9 -1, 20 21 2 -1, 20 21 11 -1, 21 14 11 -1, 21 14 3 -1, 21 4 2 -1, 21 4 3 -1, 18 6 13 -1, 18 13 16 -1, ">
        <Coordinate DEF="coord_Layer_2"
            point="0.0 0.0 0.0, 0.0 1.0 10.0, 20.0 -10.0 25.0, 20.5999837603 0.0 5.14999594008, 40.0 0.0
0.0, 14.0 2.0 35.0, 14.0 2.0 30.004904, 18.001438 2.0 35.0, 22.0 2.0 35.0, 22.0 2.0 29.995671, 22.0 2.0 25.0, 20.0159350211
0.499601624473 16.0, 18.3040053238 1.83200024199 23.4880021779, 18.00317 2.0 25.0, 20.0141355833 0.499646610417 10.0,
0.0 1.0 16.0, 15.4183601662 1.70083455301 22.3075109771, 0.0 2.0 50.0, 14.0 2.0 25.0, 40.0 3.0 50.0, 40.0 0.0 16.0, 40.0 0.0 10.0,
" />
        </IndexedFaceSet>
    </Shape>
  </Anchor>
</Transform>

<Transform DEF="Layer_3" translation="0.0 0.0 0.0" rotation="0 0 0" center="0 0 0">
  <Anchor description="Anchor_3" parameter="target=TableView_Contents"
url="http://localhost/sitedesign/sd4/DisplayTENTable.py#63;selectionmode=false#38;dataset_name=onehouse#38;layer=3">
    <Shape>
      <Appearance>
        <Material DEF="MA_Material_3" diffuseColor="1.0 0.0666666666667 0.0666666666667"
specularColor="0.401 0.401 0.401" emissiveColor="0.0 0.0 0.0"
ambientIntensity="0.167" shininess="0.098" transparency="0.00" />
      </Appearance>
      <IndexedFaceSet solid="false" coordIndex="0 1 2 -1, 0 1 3 -1, 0 2 3 -1, 4 5 6 -1, 4 5 3 -1, 4 6 3 -1, 7 8 6 -1, 7 8 9 -1,
7 6 9 -1, 10 8 11 -1, 10 8 9 -1, 10 11 12 -1, 1 13 5 -1, 1 13 14 -1, 1 5 3 -1, 1 2 12 -1, 1 12 14 -1, 13 5 14 -1, 5 8 6 -1, 5 8 14 -1, 8 11 14 -
1, 11 12 14 -1, 2 6 3 -1, 2 6 9 -1, 15 10 9 -1, 15 10 12 -1, 15 2 9 -1, 15 2 12 -1, ">
        <Coordinate DEF="coord_Layer_3"
            point="14.0 2.0 35.0, 14.0 8.0 35.0, 14.0 2.0 30.004904, 18.001438 2.0 35.0, 22.0 2.0 35.0,
22.0 8.0 35.0, 22.0 2.0 29.995671, 22.0 2.0 25.0, 22.0 8.0 25.0, 18.00317 2.0 25.0, 14.0 8.0 25.0, 18.0 11.0 25.0, 14.0 8.0 29.997526,
18.0 11.0 35.0, 18.0 11.0 29.999836, 14.0 2.0 25.0, " />
        </IndexedFaceSet>
    </Shape>
  </Anchor>
</Transform>

</Scene>
</X3D>

```

**Appendix 6.17: Script to generate CityGML and CityGML output**

```

outputXML = []
outputXML.append('<?xml version="1.0" encoding="UTF-8"?>')
outputXML.append('<!-- Virtual TEN model of house on terrain, Created by Penninga F., TUDelft 2006-->')
outputXML.append('<!-- Level of Detail 3 -->')
outputXML.append('')
outputXML.append('')
outputXML.append('<CityModel xmlns="http://www.citygml.org/citygml/1/0/0" ')
outputXML.append('xmlns:gml="http://www.opengis.net/gml" ')
outputXML.append('xmlns:xlink="http://www.w3.org/1999/xlink" ')
outputXML.append('xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" ')
outputXML.append('xsi:schemaLocation="http://www.citygml.org/citygml/1/0/0 ')
outputXML.append('http://www.citygml.org/citygml/1/0/0/CityGML.xsd">')
outputXML.append('<gml:name>Virtual TEN model of house on terrain</gml:name>')
outputXML.append('<gml:boundedBy>')
outputXML.append('  <gml:Envelope srsName="EPSG:28992">')
outputXML.append('    <gml:pos srsDimension="3">0.0 0.0 0.0 </gml:pos>')
outputXML.append('    <gml:pos srsDimension="3">33.0 34.0 2.5</gml:pos>')
outputXML.append('  <!-- CRS: Gauss-Krueger projection, 3rd Meridian, Bessel ellipsoid -->')
outputXML.append('  <!-- Please note, that the referenced CRS is 2D; it has to be changed to a 3D CRS in')
outputXML.append('  the future in order to be correct -->')
outputXML.append('  </gml:Envelope>')
outputXML.append('</gml:boundedBy>')
outputXML.append('')
outputXML.append('')
for i in range(len(GMLposliststring_per_object_newindex)):
    outputXML.append('  <cityObjectMember>')
    #myGUID = pythoncom.CreateGuid()
    #myGUID = str(myGUID)[1:-1]
    myGUID = str(random.randint(0,9)) + str(random.randint(0,9)) + str(random.randint(0,9)) +
    str(random.randint(0,9))
    outputXML.append('    <GenericCityObject gml:id="GUID_' + myGUID + '">')

    #Uses GenericCityObject though there are many more classes like roofs, windows, TIN etc. check out
    #spec
    outputXML.append('      <externalReference># <!-- This is a reference to the object database -->')
    #externalURL = 'http://localhost/sitedesign/sd4/DisplayTENTable.py?layer=' +
    str(IndexedFaceSetlist_per_object_id[i])
    externalURL =
    'http://localhost/sitedesign/sd4/DisplayTENTable.py?#63;selectionmode=false&#38;dataset_name=' +
    str(dataset_name) + '&#38;layer=' + str(IndexedFaceSetlist_per_object_id[i])
    outputXML.append('        <informationSystem>' + externalURL + '</informationSystem>')
    outputXML.append('        <externalObject>')
    outputXML.append('          <uri>urn:adv:oid:' + myGUID + '</uri>')
    outputXML.append('        </externalObject>')
    outputXML.append('      </externalReference># <!-- This is a reference to the object database -->')
    #outputXML.append('      <gml:name>Layer' + str(IndexedFaceSetlist_per_object_id[i]) + '</gml:name>')
    outputXML.append('      <stringAttribute name="Name">')
    outputXML.append('        <value>TEN Object ' + str(IndexedFaceSetlist_per_object_id[i]) + '</value>')
    outputXML.append('      </stringAttribute>')
    outputXML.append('      <stringAttribute name="GUID">')
    outputXML.append('        <value>' + myGUID + '</value>')
    outputXML.append('      </stringAttribute>')

    outputXML.append('    </GenericCityObject>')
    #print IndexedFaceSetlist_per_object_id[i]
    colorindex = colourandtransparencyarray_id.index(IndexedFaceSetlist_per_object_id[i])
    mycolorHTML = colourandtransparencyarray[colorindex][0]
    mycolorRGB = HTMLColorToRGB(mycolorHTML)
    mytransparencyINT = colourandtransparencyarray[colorindex][1]
    if layerselectarray.count(IndexedFaceSetlist_per_object_id[i]) > 0:
        ColorR = 255/255.0
        ColorG = 255/255.0
        ColorB = 0/255.0

```

```

else:
    ColorR = mycolorRGB[0]/255.0
    ColorG = mycolorRGB[1]/255.0
    ColorB = mycolorRGB[2]/255.0
#print mycolorHTML

outputXML.append(' <gml:MultiSurface >')
for i2 in range(len( GMLposliststring_per_object_newindex[i])):
    outputXML.append(' <gml:surfaceMember>')
    outputXML.append(' <TexturedSurface orientation="+">')
    outputXML.append(' <gml:baseSurface>')
    outputXML.append(' <gml:Polygon>')
    outputXML.append(' <gml:exterior>')
    outputXML.append(' <gml:LinearRing>')
    outputXML.append(' ' + GMLposliststring_per_object_newindex[i][i2])
    outputXML.append(' </gml:LinearRing>')
    outputXML.append(' </gml:exterior>')
    outputXML.append(' </gml:Polygon>')
    outputXML.append(' </gml:baseSurface>')
    outputXML.append(' <appearance>')
    outputXML.append(' <Material>')
    outputXML.append(' <shininess>0.2</shininess>')
    outputXML.append(' <transparency>' + str(mytransparencyINT) + '</transparency>')
    outputXML.append(' <ambientIntensity>1.0</ambientIntensity>')
    outputXML.append(' <diffuseColor>' + str(ColorR) + ' ' + str(ColorG) + ' ' + str(ColorB) +
'</diffuseColor>')
    outputXML.append(' <emissiveColor>0 0 0</emissiveColor>')

    outputXML.append(' </Material>')
    outputXML.append(' </appearance>')
    outputXML.append(' </TexturedSurface>')
    outputXML.append(' </gml:surfaceMember>')

outputXML.append(' </gml:MultiSurface >')
outputXML.append(' </lod3Geometry>')
outputXML.append(' </GenericCityObject>')
outputXML.append(' </cityObjectMember>')

outputXML.append('</CityModel>')

outCityGML = ".join(outputXML)
#print outCityGML
#exit()

#req.content_type="text/xml"
req.content_type="text/xml; subtype=gml/3.1.1"
req.write(outCityGML)

```

#### CityGML Output

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Virtual TEN model of house on terrain, Created by Penninga F., TUDelft 2006-->
<!-- Level of Detail 3 -->

<CityModel xmlns="http://www.citygml.org/citygml/1/0/0"
xmlns:gml="http://www.opengis.net/gml"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.citygml.org/citygml/1/0/0
http://www.citygml.org/citygml/1/0/0/CityGML.xsd">
<gml:name>Virtual TEN model of house on terrain</gml:name>

```

```

<gml:boundedBy>
  <gml:Envelope srsName="EPSG:28992">
    <gml:pos srsDimension="3">0.0 0.0 0.0 </gml:pos>
    <gml:pos srsDimension="3">33.0 34.0 2.5</gml:pos>
    <!-- CRS: Gauss-Krueger projection, 3rd Meridian, Bessel ellipsoid -->
    <!-- Please note, that the referenced CRS is 2D; it has to be changed to a 3D CRS in the future in order to be correct -->
  </gml:Envelope>
</gml:boundedBy>

<cityObjectMember>
  <GenericCityObject gml:id="GUID_2983">
    <externalReference>

<informationSystem>http://localhost/sitesdesign/sd4/DisplayTENTable.py&#63;selectionmode=false&#38;dataset_name=onehouse
&#38;layer=0</informationSystem>
  <externalObject>
    <uri>urn:adv:oid:2983</uri>
  </externalObject>
</externalReference>
  <stringAttribute name="Name">
    <value>TEN Object 0 </value>
  </stringAttribute>
  <stringAttribute name="GUID">
    <value>2983</value>
  </stringAttribute>
  <lod3Geometry>
  <gml:MultiSurface >
    <gml:surfaceMember>
      <TexturedSurface orientation="+">
        <gml:baseSurface>
          <gml:Polygon>
            <gml:exterior>
              <gml:LinearRing>
                <gml:posList srsDimension="3"> 0.0 0.0 0.0 0.0 1.0 10.0 20.0 25.0 25.0 0.0 0.0 0.0</gml:posList>
              </gml:LinearRing>
            </gml:exterior>
          </gml:Polygon>
        </gml:baseSurface>
        <appearance>
          <Material>
            <shininess>0.2</shininess>
            <transparency>0.90</transparency>
            <diffuseColor>0.0 0.0 1.0</diffuseColor>
          </Material>
        </appearance>
      </TexturedSurface>
    </gml:surfaceMember>
    <gml:surfaceMember>
      ...
    </gml:surfaceMember>
  </gml:MultiSurface >
  ...
  </gml:MultiSurface >
  <gml:MultiSurface >
  </lod3Geometry>
</GenericCityObject>
</CityModel>

```

## Appendix 6.18: Script to generate KML and KML output

Script to generate KML

```

outputXML = []
outputXML.append('<?xml version="1.0" encoding="UTF-8"?>')
outputXML.append('<kml xmlns="http://earth.google.com/kml/2.0">')
outputXML.append(' <Document>')
for i in range(len(KMLposliststring_per_object_newindex)):
    #url = 'http://localhost/sitedesign/sd4/DisplayTENTable.py?layer=' +
    str(IndexedFaceSetlist_per_object_id[i])
    #In anchorURL within KML, HTML codes for ? and & have not to be used (they have to be used in X3D)
    anchorURL =
    'http://localhost/sitedesign/sd4/DisplayTENTable.py?selectionmode=false&dataset_name=' +
    str(dataset_name) + '&layer=' + str(IndexedFaceSetlist_per_object_id[i])
    outputXML.append(' <Placemark>')
    outputXML.append(' <name>Layer ' + str(IndexedFaceSetlist_per_object_id[i]) + '</name>')
    outputXML.append(' <description><![CDATA[' + anchorURL + ']]></description>')
    outputXML.append(' <Point>')
    coordinatestring =
    str(midXYZlist_per_object[i][0])+','+str(midXYZlist_per_object[i][1])+','+str(midXYZlist_per_object[i][2])
    outputXML.append(' <coordinates>' + coordinatestring + '</coordinates>')
    outputXML.append(' </Point>')
    outputXML.append(' </Placemark>')
    outputXML.append(' <Placemark>')
    outputXML.append(' <description><![CDATA[' + anchorURL + ']]></description>')
    #outputXML.append(' <description>' + 'Object' + str(IndexedFaceSetlist_per_object_id[i]) +
    '</description>')
    outputXML.append(' <name>Layer ' + str(IndexedFaceSetlist_per_object_id[i]) + '</name>')
    outputXML.append(' <LookAt>')
    outputXML.append(' <longitude>8.853193712983327</longitude>')
    outputXML.append(' <latitude>53.10919982492059</latitude>')
    outputXML.append(' <latitude>53.10919982492059</latitude>')
    outputXML.append(' <tilt>50</tilt>')
    outputXML.append(' <heading>230</heading> ')
    outputXML.append(' </LookAt>')
    colorindex = colourandtransparencyarray_id.index(IndexedFaceSetlist_per_object_id[i])
    mycolorHTML = colourandtransparencyarray[colorindex][0]
    mytransparencyINT = colourandtransparencyarray[colorindex][1]
    #print "transp: " + str(mytransparencyINT)
    #print str(RGBToHTMLTransparency(mytransparencyINT))
    if layerselectarray.count(IndexedFaceSetlist_per_object_id[i]) > 0:
        mycolorKML = "FF00FFFF"
        #print "sel " + mycolorKML
    else:
        #print mycolorHTML
        mycolorKML = RGBToHTMLTransparency(mytransparencyINT)[1:3] + mycolorHTML[5:7] +
        mycolorHTML[3:5] + mycolorHTML[1:3]
        #print "nonsel " + mycolorKML
    outputXML.append(' <Style>')
    outputXML.append(' <PolyStyle>')
    #print layerselectarray
    outputXML.append(' <color>' + mycolorKML + '</color>')
    outputXML.append(' </PolyStyle>')
    outputXML.append(' <LineStyle>')
    outputXML.append(' <width>0.1</width>')
    outputXML.append(' <color>' + mycolorKML + '</color>')
    outputXML.append(' </LineStyle>')
    outputXML.append(' </Style>')
    outputXML.append(' <MultiGeometry>')
    #IndexedFaceSetlist_per_object_id[i]
    for i2 in range(len(KMLposliststring_per_object_newindex[i])):
        outputXML.append(' <Polygon>')
        outputXML.append(' <altitudeMode>relativeToGround</altitudeMode>')
        outputXML.append(' <outerBoundaryIs>')
        outputXML.append(' <LinearRing>')
        outputXML.append(' <coordinates>')
        outputXML.append(KMLposliststring_per_object_newindex[i][i2])
        outputXML.append(' </coordinates>')
        outputXML.append(' </LinearRing>')

```

```

outputXML.append('    </outerBoundaryIs>')
outputXML.append('  </Polygon>')

outputXML.append(' </MultiGeometry>')
outputXML.append(' </Placemark>')

outputXML.append(' </Document>')
outputXML.append(' </kml>')

outKML = ''.join(outputXML)
#req.content_type="text/xml"
req.content_type="application/vnd.google-earth.kml+xml"
req.write(outKML)

```

#### KML Output

```

<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
  <Document>
    <Placemark>
      <name>Layer 0</name>

      <description><![CDATA[http://localhost/sitesdesign/sd4/DisplayTENtable.py?selectionmode=false&dataset_name=onehouse&layer=0]]></description>
      <Point>
        <coordinates>5.0002658843,52.0003494969,4.18440276766 </coordinates>
      </Point>
    </Placemark>
  </Document>

  <description><![CDATA[http://localhost/sitesdesign/sd4/DisplayTENtable.py?selectionmode=false&dataset_name=onehouse&layer=0]]></description>
  <name>Layer 0</name>
  <LookAt>
    <longitude>8.853193712983327</longitude>
    <latitude>53.10919982492059</latitude>
    <altitude>53.10919982492059</altitude>
    <tilt>50</tilt>
    <heading>230</heading>
  </LookAt>
  <Style>
    <PolyStyle>
      <color>19FF0000</color>
    </PolyStyle>
    <LineStyle>
      <width>0.1</width>
      <color>19FF0000</color>
    </LineStyle>
  </Style>
  <MultiGeometry>
    <Polygon>
      <altitudeMode>relativeToGround</altitudeMode>
      <outerBoundaryIs>
        <LinearRing>
          <coordinates>
            5.0,52.0,0.0 5.0,52.0001428571,1.0 5.00028571429,52.0003571429,25.0 5.0,52.0,0.0
          </coordinates>
        </LinearRing>
      </outerBoundaryIs>
    </Polygon>
  </MultiGeometry>
  ...

```



```

    </Polygon>
  </Placemark>
</Placemark>
...
</Placemark>
</Document>
</kml>

```

## Appendix 6.19: Script to generate SVG and SVG output

Script to generate SVG

```

xtopleft = xdatasetmin
xbottomright = xdatasetmax

ytopleft = zdatasetmin
ybottomright = zdatasetmax
boxwidth = xbottomright - xtopleft
boxheight = ybottomright - ytopleft
viewboxstring = str(xtopleft) + " " + str(ytopleft) + " " + str(boxwidth) + " " + str(boxheight)

outputSVG = []
outputSVG.append('<?xml version="1.0" standalone="no"?> ')
outputSVG.append('<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN" ')
outputSVG.append('"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">')
outputSVG.append("")
#parameters for transform, viewBox, width and height may have to be adjusted for each dataset
outputSVG.append('<svg id="SVG3D" width="100%" height="100%" version="1.1" viewBox="' + viewboxstring
+ "' ')
#outputSVG.append('<svg width="500%" height="500%" version="1.1" viewBox="0 0 370 380" ')
outputSVG.append('<xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">')
outputSVG.append('<script xlink:href="http://localhost/sitedesign/sd4/setviewpoint.js"
type="text/ecmascript"/>')
#outputSVG.append('<script type="text/JavaScript" src="setviewpoint.js">')
#outputSVG.append('</script>')
#<![CDATA[]]></script>')
outputSVG.append("")
#<xa>
# <loc xlink:href="http://example.com/A.svg"
#  xlink:title="Schematic diagram"/>
# <loc xlink:href="http://example.org/B.svg"
#  xlink:title="Parts list"/>
# <!-- content of the link goes here -->
# <path d="M50,50L100,20L30,100z"/>
#</xa>

#outputSVG.append('<rect fill="#E9FFE6" stroke="#FF0000" stroke-width="0" x="' + str(xtopleft) + "' y="' +
str(ytopleft) + "' width="' + str(boxwidth) + "' height="' + str(boxheight) + "'/>')
#outputSVG.append('<polygon points="' + str(xtopleft) + ',' + str(ytopleft) + ',' + str(xbottomright) + ',' +
str(ytopleft) + ',' + str(xbottomright) + ',' + str(ybottomright) + ',' + str(xtopleft) + ',' + str(ybottomright) + '"
style="fill:#00FF00;stroke:#00FF00;stroke-width:0.0;opacity:1.0" transform="translate(0 0)" id="Backrect"/>')
for i in range(len(SVGposliststring_per_object_newindex)):
#http://www.asciitable.com/
i_ordered = drawingorder[i]
if xlinktarget == 'Table':
xlinkstring1 =
'xlink:href="http://localhost/sitedesign/sd4/DisplayTENTable.py?#38;selectionmode=false&#38;dataset_name='
+ str(dataset_name) + '&#38;layer=' + str(IndexedFaceSetlist_per_object_id[i_ordered]) + '"
target="TableView_Contents"'
elif xlinktarget == '3DView':
xlinkstring1 = 'xlink:href="http://localhost/sitedesign/sd4/DisplayTEN.py?#Layer_' +
str(IndexedFaceSetlist_per_object_id[i_ordered]) + '" target="3DView_Contents"'

```

```

else:
    xlinkstring1 =
'xlink:href="http://localhost/sitesdesign/sd4/DisplayTENTable.py&#63;selectionmode=false&#38;dataset_name='
+ str(dataset_name) + '&#38;layer=' + str(IndexedFaceSetlist_per_object_id[i_ordered]) + "'
    target="TableView_Contents"
    #xlinkstring1 = 'xlink:href="http://localhost/sitesdesign/sd4/DisplayTENTable.py??dataset_name=' +
dataset_name + '&layer=' + str(IndexedFaceSetlist_per_object_id[i]) + "' target="TableView_Contents"'
#xlinkstring2 = 'xlink:href="http://www.google.nl" xlink:target="3DView_Contents" xlink:name="I3"'
nodeID = IndexedFaceSetlist_per_object_id[i_ordered]

if xlinktarget == 'Table':
    outputSVG.append('<a ' + xlinkstring1 + ' id="SVGMainElement' +
str(IndexedFaceSetlist_per_object_id[i_ordered]) + "'>')
elif xlinktarget == '3DView':
    outputSVG.append('<a id="SVGMainElement' + str(IndexedFaceSetlist_per_object_id[i_ordered]) +
"'>')

#For Multiple links from svg, does not seem to work at leas in Firefox
#outputSVG.append('<xa>')
#outputSVG.append('<loc ' + xlinkstring1 + '/>')
#outputSVG.append('<loc ' + xlinkstring2 + '/>')

for i2 in range(len( SVGposliststring_per_object_newindex[i_ordered])):
    outputSVG.append(' ' + SVGposliststring_per_object_newindex[i_ordered][i2])

outputSVG.append('</a>')
#outputSVG.append('<polygon points="' + str(xtopleft) + ',' + str(ytopleft) + ' ' + str(xbottomright) + ',' +
str(ytopleft) + ' ' + str(xbottomright) + ',' + str(ybottomright) + ' ' + str(xtopleft) + ',' + str(ybottomright) + '"
style="fill:#00FF00;stroke:#0000FF;stroke-width:0.0;opacity:0.1" transform="translate(0 0)" id="Backrect"/>')
outputSVG.append('</svg>')

outSVG = ".join(outputSVG)

#req.content_type="text/xml"
req.content_type="image/svg+xml"
req.write(outSVG)

```

#### SVG Output

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">

<svg id="SVG3D" width="100%" height="100%" version="1.1" viewBox="0.0 0.0 40.0 50.0"
xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
<script xlink:href="http://localhost/sitesdesign/sd4/setviewpoint.js" type="text/ecmascript"/>

<a
xlink:href="http://localhost/sitesdesign/sd4/DisplayTENTable.py&#63;selectionmode=false&#38;dataset_name=onehouse&#38;laye
r=2" target="TableView_Contents" id="SVGMainElement2">
  <polygon onclick="" points="0.0,0.0 20.0,25.0 0.0,10.0" style="fill:#755A51;stroke:#755A51;stroke-width:0.0;opacity:1.00"
transform="translate(0 0)" id="SVGSubElement2_0"/>
  <polygon onclick="" points="0.0,0.0 20.0,25.0 20.5999837603,5.14999594008" style="fill:#755A51;stroke:#755A51;stroke-
width:0.0;opacity:1.00" transform="translate(0 0)" id="SVGSubElement2_1"/>
  ....
</a>
<a
xlink:href="http://localhost/sitesdesign/sd4/DisplayTENTable.py&#63;selectionmode=false&#38;dataset_name=onehouse&#38;laye
r=3" target="TableView_Contents" id="SVGMainElement3">
  ...
  ...
</a>
</svg>

```

## Appendix 6.20: Execute query on attribute data

```
cnxn = pyodbc.connect('DRIVER={Microsoft ODBC for
Oracle};SERVER=(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=casagrande.otb.tudelft.nl)(PORT=1521))(CONNECT_DATA=(SER
VICE_NAME=gisbase)));UID=' + dataset_username + ';PWD=' + dataset_password)
cursorob = cnxn.cursor()
cursorob2 = cnxn.cursor()
cursorselect = cursorob.execute(selectquery)
cursorunselect = cursorob2.execute(unselectquery)
```

## Appendix 6.21: Loading data into array

---

```
if uniqueobjectidlist != "na":
    i2 = 1 #Loads attribute data of selected objects
    for row in cursorselect:
        if uniqueobjectidlist.count(row[0]) > 0:
            trianglecode_table_select.append([])
            for i3 in range(len(row)):
                trianglecode_table_select[i2].append(row[i3])
            i2 = i2+1
    else:
        i2 = 1 #Loads attribute data of selected objects
        for row in cursorselect:
            trianglecode_table_select.append([])
            for i3 in range(len(row)):
                trianglecode_table_select[i2].append(row[i3])
            i2 = i2+1

if uniqueobjectidlist != "na":
    i2 = 1 #Loads attribute data of unselected objects
    for row in cursorunselect:
        if uniqueobjectidlist.count(row[0]) > 0:
            trianglecode_table_unselect.append([])
            for i3 in range(len(row)):
                trianglecode_table_unselect[i2].append(row[i3])
            i2 = i2+1
    else:
        i2 = 1 #Loads attribute data of unselected objects
        for row in cursorunselect:
            trianglecode_table_unselect.append([])
            for i3 in range(len(row)):
                trianglecode_table_unselect[i2].append(row[i3])
            i2 = i2+1
```

---

## Appendix 6.22: Transforming data array into HTML table

```
mytable = []
mytable.append('<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">')
mytable.append('<html>')
mytable.append('<head>')
mytable.append('<title>Table View - (Non spatial) attribute data of objects</title>')
mytable.append('<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">')
mytable.append('<style type="text/css">')
mytable.append('.select{background-color: yellow;}')
mytable.append('.unselect{background-color: #E9FFE6;}')
mytable.append('</style>')
mytable.append('<body id="MyTableBody" bgcolor="#E9FFE6">')
mytable.append('<table id="MyTable" width="75%" border="1">')

for i in range(len(trianglecode_table)):
    if i > 0 and selectionmode == 'true':
        IDstring = "row_" + str(trianglecode_table[i][idcolumn])
        IDstring_short = trianglecode_table[i][idcolumn]
        if layerselectarray.count(IDstring_short) > 0:
            mytable.append(' <tr bgcolor="#FFFF33" id ="' + str(IDstring) + "'>')
        else:
            mytable.append(' <tr id ="' + str(IDstring) + "'>')
    else:
        if i > 0:
            IDstring = "row_" + str(trianglecode_table[i][idcolumn])
            mytable.append(' <tr id ="' + str(IDstring) + "'>')
        else:
            mytable.append(' <tr id="rowheader">')
    for i2 in range(len(trianglecode_table[i])):
        cellvalue = trianglecode_table[i][i2]
        if i == 0:
            mystring = str(cellvalue)

            if mystring == "OBJECTID": #if select == true, if row is columnnames
                idcolumn = i2

                #mystring2 = str(mystring[0])+str(str(mystring[1:]).lower())
                mystring2 = mystring
            mytable.append(' <td><strong>' + mystring2 + '</strong></td>')
        else:
            mytable.append(' <td>' + str(cellvalue) + '</td>')

    mytable.append(' </tr>')

mytable.append('</table>')
#mytable.append('<br>' + str(idcolumn) + '<br>') error checks can be removed
#mytable.append('<br>' + selectionmode + '<br>') error checks can be removed
#mytable.append('<br>' + str(layerselectarray) + '<br>') error checks can be removed
mytable.append('</body>')
mytable.append('</html>')

#print mytable

outHTML = ".join(mytable)

#print outHTML
#req.content_type="text/xml"
req.content_type="text/html"
#req.write(layerselectarray)
req.write(outHTML)
#return apache.OK (zet er een nul achter)
```

### Appendix 6.23: sample HTML code of attribute data

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Object data table</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>
<body>
<table width="75%" border="1">
<tr>
<td><strong>Objectid<strong></td>
<td><strong>Objectname<strong></td>
<td><strong>Objectowner<strong></td>
<td><strong>Objectheight<strong></td>
<td><strong>Colour<strong></td>
</tr>
<tr>
<td>5</td>
<td>HouseFloor</td>
<td>Pietersen</td>
<td>50</td>
<td>#FF0000</td>
</tr>
<tr>
<td>3</td>
<td>House</td>
<td>Pietersen</td>
<td>30</td>
<td>#CCCC00</td>
</tr>
<tr>
<td>2</td>
<td>Terrain</td>
<td>Government</td>
<td>50</td>
<td>#22CCFF</td>
</tr>
</table>
</body>
</html>
```

### Appendix 6.24: HTTP argument passing

For example if an elephant has to be looked up in Google the following hyperlink can be put <http://www.google.nl/search?q=elephant>. Multiple arguments are separated with an & char, for example: “<http://www.google.nl/search?hl=nl&q=hoi&btnG=Google+zoeken&meta=>”

The hyperlink for displaying an object’s attribute is formatted like this:  
“[DisplayTENtable.py?selectionmode=false&dataset=onehouse&layer=2](http://www.google.nl/search?hl=nl&q=hoi&btnG=Google+zoeken&meta=DisplayTENtable.py?selectionmode=false&dataset=onehouse&layer=2)”

## Appendix 6.25: Adding a hyperlink to a 3D object (in X3D, KML and CityGML)

In X3D a hyperlink can be added to an object by using an anchor tag as in the following description. hyperlink can be accessed by clicking on the object.

```
<Anchor description="Layer_2" + parameter="target=tabledata.htm"
url="http://localhost/TENtest/DisplayTENtable.py?layer=2">
```

In KML a hyperlink can be added by inserting a description tag after the placemark tag which is formatted like this. The hyperlink cannot be accessed by clicking on the object as in X3D but by accessing the object's legend within the places window or by clicking the link on the balloon attached to the placemark in the middle of the object.)

```
<description><![CDATA['http://localhost/TENtest/DisplayTENtable.py?layer=2']]></description>
```

In CityGML a hyperlink can be added by inserting an <externalReference> tag before the geometry of each object. Within the externalReference tags a tag <informationSystem> is nested which holds the URL. In one of the two CityGML viewer (LandXplorer) this link can be displayed within the object properties. It does not work as a clickable hyperlink on the object as in X3D. The only way to display the link is to edit the text, copy it and paste it to the URL field of an internet browser.)

```
<externalReference>
<informationSystem> http://localhost/TENtest/DisplayTENtable.py?layer=2 </informationSystem>
</externalReference>
```

In SVG a link can be added to an object tag. An object tag is defined like this <a>. A hyperlink can be added to this tag. The hyperlink can be accessed by clicking on the object.

```
<a xlink:href="http://localhost/TENtest/DisplayTENtable.py?layer=2" target="tabledata.htm">
```

## Appendix 6.26: Formatting of the resulting links of the SQL query

```
returnHTML = []
returnHTML.append('<html>')

returnHTML.append('<head>')
returnHTML.append('<script type="text/JavaScript" src="http://localhost/sitedesign/sd4/selectobject.js">')
returnHTML.append('</script>')
returnHTML.append('</head>')

returnHTML.append('<body bgcolor="#E9FFE6">')
returnHTML.append('The submitted query is: '+ myquery)

ThreeDimQuery = unpack (ThreeDimQuery)
TwoDimQuery = unpack (TwoDimQuery)
TableQuery = unpack (TableQuery)

ThreeDimQuery = ThreeDimQuery.replace("_sellayers_", str(objsel[1]))
TwoDimQuery = TwoDimQuery.replace("_sellayers_", str(objsel[1]))
TableQuery = TableQuery.replace("_sellayers_", str(objsel[1]))
ThreeDimQuery = ThreeDimQuery.replace("_unsellayers_", str(objsel[2]))
TwoDimQuery = TwoDimQuery.replace("_unsellayers_", str(objsel[2]))
TableQuery = TableQuery.replace("_unsellayers_", str(objsel[2]))

ahrefline = "JavaScript:void(top.frames['3DView_Contents'].location.href='"+ str(ThreeDimQuery) +
";void(top.frames['2DView_Contents'].location.href='"+ str(TwoDimQuery) +
```

```

";void(top.frames['TableData_Contents'].location.href="" + str(TableQuery) + "";"
ahrefline = 'JavaScript:alert(&quot;hoi&quot;);alert(&quot;hoi&quot;);'
#when void is used link is activated without reloading the contents page of the SQL box, otherwise link will
display in SQLbox
ahrefline = 'JavaScript:void(top.frames["3DView_Contents"].location.href=' + str(ThreeDimQuery) +
');void(top.frames["2DView_Contents"].location.href=' + str(TwoDimQuery) +
');void(top.frames["TableView_Contents"].location.href=' + str(TableQuery) + ')'
returnHTML.append('<p>')
dynamicalquery = '<A HREF=' + 'JavaScript:selectObjects(&quot;' + str(objsel[1]) + '&quot;' + '>Execute
dynamical query</A>'
staticalquery = '<A HREF=' + ahrefline + '>Execute statical query</A>'
returnHTML.append('<pre>')
returnHTML.append(dynamicalquery + ' ' + staticalquery)
returnHTML.append('</pre>')
returnHTML.append('</body>')

returnHTML.append('</html>')

writequery(req, returnHTML)

```

## Appendix 6.27: Explanation to the postquery function

The postquery function has several input parameters these are the query, the dataset name, the output model (TEN, 3DTIN or 'TEN view'), airexclude (false, true), algorithm and bounding box. First the query is and the resulting object IDs are filtered out and put in an array. This array is used in the creation of a table with attributes. Only the attribute records that fall in the selection are displayed.

```

returnHTML = []
returnHTML.append('<html>')

returnHTML.append('<head>')
returnHTML.append('<script type="text/JavaScript" src="http://localhost/sitedesign/sd4/selectobject.js">')
returnHTML.append('</script>')
returnHTML.append('</head>')

returnHTML.append('<body bgcolor="#E9FFE6">')
returnHTML.append('The submitted query is: '+ myquery)

ThreeDimQuery = unpack (ThreeDimQuery)
TwoDimQuery = unpack (TwoDimQuery)
TableQuery = unpack (TableQuery)

ThreeDimQuery = ThreeDimQuery.replace("_sellayers_", str(objsel[1]))
TwoDimQuery = TwoDimQuery.replace("_sellayers_", str(objsel[1]))
TableQuery = TableQuery.replace("_sellayers_", str(objsel[1]))
ThreeDimQuery = ThreeDimQuery.replace("_unsellayers_", str(objsel[2]))
TwoDimQuery = TwoDimQuery.replace("_unsellayers_", str(objsel[2]))
TableQuery = TableQuery.replace("_unsellayers_", str(objsel[2]))

ahrefline = "JavaScript:void(top.frames['3DView_Contents'].location.href="" + str(ThreeDimQuery) +
";void(top.frames['2DView_Contents'].location.href="" + str(TwoDimQuery) +
";void(top.frames['TableData_Contents'].location.href="" + str(TableQuery) + "";"
ahrefline = 'JavaScript:alert(&quot;hoi&quot;);alert(&quot;hoi&quot;);'
#when void is used link is activated without reloading the contents page of the SQL box, otherwise link will
display in SQLbox
ahrefline = 'JavaScript:void(top.frames["3DView_Contents"].location.href=' + str(ThreeDimQuery) +
');void(top.frames["2DView_Contents"].location.href=' + str(TwoDimQuery) +
');void(top.frames["TableView_Contents"].location.href=' + str(TableQuery) + ')'
returnHTML.append('<p>')
dynamicalquery = '<A HREF=' + 'JavaScript:selectObjects(&quot;' + str(objsel[1]) + '&quot;' + '>Execute
dynamical query</A>'

```

```

staticalquery = '<A HREF=' + ahrefline + '>Execute statical query</A>'
returnHTML.append('<pre>')
returnHTML.append(dynamicalquery + ' ' + staticalquery)
returnHTML.append('</pre>')
returnHTML.append('</body>')

returnHTML.append('</html>')

writequery(req, returnHTML)

```

### Appendix 6.28: the output of the postquery function

if executed on the table, 3D view, or 2D View in order:

```

DisplayTENTable.py?selectionmode=true&layerselect=5_2_&layeraselect=3_
DisplayTEN.py?selectionmode=true&dataset_model="X3D"layerselect=0_2_&layeraselect=3_
DisplayTEN.py?selectionmode=true&dataset_model=SVG&layerselect=0_2_&layeraselect=3_

```

If executed on the 3D view for display of KML or CityGML in order:

```

"DisplayTEN.py?selectionmode=true&dataset_model=KML&layerselect=5_2_&layeraselect=3_"
"DisplayTEN.py?selectionmode=true&dataset_model=CityGML&layerselect=5_2_&layeraselect=3_"

```

### Appendix 6.29: Dynamical acces to visualizations with JavaScript

JavaScript commands to access the 3D visualization in X3D, the 2D visualization in SVG and the table in HTML dynamically:

```

loc = top.document.getElementById("3DView_Contents").contentWindow.document.Contact3D
var loc =

top.document.getElementById("2DView_Contents").contentWindow.document.getElementById("Contact2D").getSVGDocument();

var table = top.document.getElementById("TableData_Contents").contentWindow.document.getElementById("MyTable");

```

JavaScript command to colour a node yellow with ID = 2 in X3D:

```

loc.setNodeField("MA_Material_2", "diffuseColor", "1 1 0");

```

### Appendix 6.30: Getting the colour and transparency of the objects

```

querystring = "select OBJECTID, COLOUR, TETTRANSPARENCY FROM OBJECTINFO"
objectcolourobject = cursor.execute(querystring)
colourandtransparencyarray_id = []
colourandtransparencyarray = []
for row in objectcolourobject:
    colourandtransparencyarray_id.append(int(float(row[0])))
    colourandtransparencyarray.append((row[1], (row[2])))
#print colourandtransparencyarray_id, colourandtransparencyarray

```



### Appendix 6.31: Adding color to the X3D visualization

In first the step object color for the object with chosen ID is being looked up.  
Secondly the hexadecimal color is converted to an RGB color.  
Thirdly each color component is recalculated to a number between 0 and 1 for coding it in X3D (a color is property of the material tag which is unique for each object).

Assigning Color and Transparency to X3D output

```
outputXML.append('          <Appearance>')
#print IndexedFaceSetlist_per_object_id[i]
#print colourandtransparencyarray_id
colorindex = colourandtransparencyarray_id.index(IndexedFaceSetlist_per_object_id[i])
mycolorHTML = colourandtransparencyarray[colorindex][0]
mycolorRGB = HTMLColorToRGB(mycolorHTML)
mytransparencyINT = colourandtransparencyarray[colorindex][1]

#print "transp: " + str(mytransparencyINT)
if layerselctarray.count(IndexedFaceSetlist_per_object_id[i]) > 0:
    ColorR = 255/255.0
    ColorG = 255/255.0
    ColorB = 0/255.0
    #print "selected:" + str(ColorR) + ' ' + str(ColorG) + ' ' + str(ColorB)
else:
    ColorR = mycolorRGB[0]/255.0
    ColorG = mycolorRGB[1]/255.0
    ColorB = mycolorRGB[2]/255.0
    #print mycolorRGB
    #print "unselected:" + str(ColorR) + ' ' + str(ColorG) + ' ' + str(ColorB)
outputXML.append('          <Material DEF="MA_Material_' +
str(IndexedFaceSetlist_per_object_id[i]) + '" diffuseColor="' + str(ColorR) + ' ' + str(ColorG) + ' ' + str(ColorB)
+ '" specularColor="0.401 0.401 0.401" emissiveColor="0.0 0.0 0.0"'
outputXML.append('          ambientIntensity="0.167" shininess="0.098"
transparency="' + str(mytransparencyINT) + '" />')
outputXML.append('          </Appearance>')
```

Color is added as diffuse color (which means light is scattered in all directions, while emissive color and specular color have a default value). An X3D object with colors assigned will look like this.

X3D object with colour and transparency

```
<Transform DEF="Layer_0" translation="0.0 0.0 0.0" rotation="0 0 0" center="0 0 0">
  <Anchor description="Anchor_0" parameter="target=TableView_Contents"
url="http://localhost/sitesdesign/sd4/DisplayTENTable.py&#63;selectionmode=false&#38;dataset_name=onehouse&#38;layer=0">
    <Shape>
      <Appearance>
        <Material DEF="MA_Material_0" diffuseColor="0.0 0.0 1.0" specularColor="0.401 0.401 0.401"
emissiveColor="0.0 0.0 0.0"
          ambientIntensity="0.167" shininess="0.098" transparency="0.90" />
      </Appearance>
      <IndexedFaceSet solid="false" coordIndex="0 1 2 -1, 0 1 3 -1, 0 4 2 -1, 0 4 3 -1, 5 6 7 -1, 5 6 8 -1, 9 10 11 -1, 9 10
8 -1, 12 13 11 -1, 12 13 14 -1, 12 15 16 -1, 12 14 16 -1, 17 13 18 -1, 17 13 14 -1, 17 18 19 -1, 6 20 10 -1, 6 10 8 -1, 6 7 19 -
1, 6 19 21 -1, 20 10 21 -1, 10 13 11 -1, 10 13 21 -1, 13 18 21 -1, 18 19 21 -1, 1 22 3 -1, 1 23 2 -1, 1 23 22 -1, 15 16 24 -1, 14 16 24 -1,
23 5 7 -1, 23 22 15 -1, 23 15 24 -1, 23 25 5 -1, 23 25 2 -1, 23 26 7 -1, 23 26 24 -1, 25 5 8 -1, 25 9 8 -1, 25 27 9 -1, 25 27 2 -1, 27 28 9 -
1, 27 28 2 -1, 28 9 11 -1, 28 12 15 -1, 28 12 11 -1, 28 29 2 -1, 28 29 15 -1, 29 22 15 -1, 29 22 3 -1, 29 4 2 -1, 29 4 3 -1, 26 17 14 -1, 26
17 19 -1, 26 7 19 -1, 26 14 24 -1, ">
      <Coordinate DEF="coord_Layer_0"
        point="0.0 0.0 0.0, 0.0 1.0 10.0, 20.0 25.0 25.0, 20.5999837603 0.0 5.14999594008, 40.0 0.0
0.0, 14.0 2.0 35.0, 14.0 8.0 35.0, 14.0 2.0 30.004904, 18.001438 2.0 35.0, 22.0 2.0 35.0, 22.0 8.0 35.0, 22.0 2.0 29.995671, 22.0 2.0
25.0, 22.0 8.0 25.0, 18.00317 2.0 25.0, 20.0159350211 0.499601624473 16.0, 18.3040053238 1.83200024199 23.4880021779, 14.0
```

```

8.0 25.0, 18.0 11.0 25.0, 14.0 8.0 29.997526, 18.0 11.0 35.0, 18.0 11.0 29.999836, 20.0141355833 0.499646610417 10.0, 0.0 1.0
16.0, 15.4183601662 1.70083455301 22.3075109771, 0.0 2.0 50.0, 14.0 2.0 25.0, 40.0 3.0 50.0, 40.0 0.0 16.0, 40.0 0.0 10.0, " />
      </IndexedFaceSet>
    </Shape>
  </Anchor>
</Transform>

```

### Appendix 6.32: Coding of transparency in X3D, CityGML and KML

In X3D the transparency is coded within the material tag:

```

<Material DEF="MA_Material_2" ambientIntensity="0.167" diffuseColor="1 0 0" emissiveColor="0.0 0.0 0.0"
shininess="0.098" specularColor="0.401 0.401 0.401" transparency="0.5"/>

```

In CityGML transparency is coded with a value between 0 and 1:

```

<transparency>0.90</transparency>

```

In KML transparency is coded in the first two characters of the colour tag using a hexadecimal value:

```

<color>FF22CCFF</color> (first FF means 100% transparency, rest is the RGB colour code)

```

### Appendix 6.33: Coding of viewpoints in X3D

```

<Viewpoint DEF='Default' description='Default' rotation = '0.2 -1.8 -1.8' orientation='0.4 0.5 1 0' position='20 20 100'/>
<Viewpoint DEF='Layer_2' description='Layer_2' orientation='0 -0.25 1 0' position='18.9693822788 4.22857142857
222.238095238'/>
<Viewpoint DEF='Layer_3' description='Layer_3' orientation='0 -0.25 1 0' position='17.750288 14.5625 113.8125'/>
<Viewpoint DEF='Layer_5' description='Layer_5' orientation='0 -0.25 1 0' position='18.000576 2.0 113.875'/>

```

### Appendix 6.34: Navigation in KML / Google Earth and CityGML / LandXplorer

KML / Google Earth

In Google Earth, the circle rotator button in the topright of the window can be used to rotate around the earth z-axis in the middle of the view. The attached plus min slider can be used to move further away or come closer to the earth. The attached x-x slider can be used to rotate the viewpoint from looking straight down on earth from above till seeing the earth from the side (fig. 5.19). It's not possible to move the view below the subsurface

CityGML / LandXplorer

In the CityGML browser LandXplorer the mouse scroll buttons can be used to zoom in and out on the data in examine mode while dragging up and down or left and right with the mouse ensures rotation of the view of the data around the x and y axis. When the control key is held down there is movement instead of rotation around the axes just as in X3D.

In game mode the mouse can be dragged to the left and the right to rotate around the y-axis while there is moved along the z-axis, if the mouse is dragged up and down. When the alt key is hold down a 3D cubical selection can be made. In fly mode if the mouse is dragged left or right there is rotated around the viewer's y-axis. With the mouse scroll button is moved along the viewer's y-axis.

## Appendix 6.35: Texture implementation in X3D, KML and CityGML

### X3D

In X3D textures can be added to objects by adding an ImageTexture node to the shape node. In the ImageTexture node can be referenced to a jpg picture. To specify texture coordinates a TextureCoordinate node can be added to the IndexedFaceSet node. Finally, the point to render the texture on can be set in the TexCoordIndex parameter of the IndexedFaceSetNode. How to paste different textures on different faces of one object (one Indexed Face Set) still has to be found out.

It does not seem to be possible to create drapes in X3D. You can overlay images over a terrain or wrap them around an object but not overlay an image over the terrain and the objects. Specialized 2.5D software has better possibilities to do this.

### CityGML

In GML textures can be added by adding a simple texture tag to the appearance node

```
<SimpleTexture>
<textureMap>177-Saulenbaunord_AO_9.gif</textureMap>
<textureCoordinates>
0.000000 1.000000 0.000000 1.000000 1.000000 0.000000 1.000000 0.000000 0.000000
</textureCoordinates>
</SimpleTexture>
```

The coordinates specify the part of the plane the image should be rendered upon. Now it is rendered on the plane from texture coordinate 0, 0 to 1, 1 which means it covers the full plane. The coordinates are built up like this x1 y1 x2 y2 x3 y3 x4 y4 x1 y5 (just like a linear ring). As in X3D images can be wrapped (draped) around objects but not over a whole terrain including terrain and objects.

### KML

In KML, 3D objects can be modeled naturally in their own coordinate space and exported as COLLADA™ files, then imported into Google Earth and placed on the Earth's surface. So textures are added within a software application that can create COLLADA files. Within the KML, there is a reference this Collada file. In opposite to X3D and GML, textures cannot be referenced as an image, which are linked to objects. Drapes can be modeled in KML by means of a ground overlay where an image is draped on the earth's terrain (so not on objects!) The xml code of a ground overlay looks like this:

```
<GroundOverlay>
<name>Large-scale overlay on terrain</name>
<description>Overlay shows Mount Etna erupting on July 13th, 2001.</description>
<Icon><href>http://code.google.com/apis/kml/documentation/etna.jpg</href></Icon>
<LatLonBox>
<north>37.91904192681665</north>
<south>37.46543388598137</south>
<east>15.35832653742206</east>
<west>14.60128369746704</west>
<rotation>-0.1556640799496235</rotation>
</LatLonBox>
</GroundOverlay>
```

## Appendix 6.36: Level of detail in X3D, CityGML and KML

### X3D

The changing of the presentation of data in a map can be achieved by saving different presentations. In X3D a scene and groups of shapes can define the structure of the map. A group is not necessary to make a map, but it is easy to give a group of shapes the same information.

The Level of Detail can be defined with a LOD tag around a group of shapes or around a shape. The LOD tag defines at which distance these shapes or groups should be visible. For example, it looks like this: LOD range='40, 80, 600'>. In this example the file has four Level of Details with a transition at 40, 80 and 600 (so distance classes < 40, 40-80, 80-600, > 600). Each level of detail has a unique group tag or transform (/shape) tag. If the coordinates of the data are not centered around 0 but around another point for example (144000, 214000, 0) then the centre parameter for Level of detail should be set to this point.

Different levels of detail in TEN networks would require generalization algorithms on the TEN data. This is a quite complex matter and not really within the scope of the subject so this is not done. It would also require storage of the building model at different LoDs in the database; the current TEN data model is not designed for this.

### CityGML

The CityGML language defines five levels, which are cities, neighbourhoods, blocks, buildings, and building elements. (Wallace, 2006)

In the CityGML language LOD objects are modelled with the tag <lod3multisurface> or <lod2solid> depending on the Level of Detail (1 least, 5 most detail) and the kind of object (surface, multisurface, solid etc.) These tags are placed after the object's description tag for example <opening><window> and before the GML tags containing the geometry (Kolbe, 2007)

### KML

In the KML language, also Level of Detail can be specified to ensure that large dataset are only loaded when there are enough pixels to display the data adequately. When a region takes up a small part of the screen maybe because the user is far away from it or a flat area is being view obliquely the LOD mechanism allow you to specify a dataset with lower resolution to be substituted for the full-resolution data.

The <minLodPixels> and <maxLodPixels > elements allow you to specify an area of the screen (in square pixels). When your data is projected onto the screen, it must occupy an area of the screen that is greater than <minLodPixels> and less than <maxLodPixels> in order to be visible. Once the projected size of the Region goes outside of these limits, it is no longer visible, and the Region becomes inactive (in the special case where you want the data to be active to infinite size, specify -1 (the default) for <maxLodPixels>).