Exporting 3D Geoinformation from Baggis Database to CityGML

Erik Olsson

Master’s of Science Thesis in Geoinformatics
TRITA-GIT EX 09-05

Division of Geoinformatics
Royal Institute of Technology (KTH)
100 44 Stockholm, Sweden

April 2009
Abstract

The objective of this research was to investigate how to export 3D data from an existing database, Baggis. The data was to be exported in CityGML format, a relatively new format used to represent 3-dimensional city models. The study area was the region of Stockholm, but the objective was to write a general plugin that would work for any area in the database.

Digpro AB has developed a Java application called DP/Spatial, which is where the plugin was implemented. The plugin, also written in Java, is run from the client side and the server returns a compressed CityGML file which contains all the buildings and other features in the area.

As the database contained several types of geometries, they were all evaluated to determine whether they should be included or not. The geometry types that were included all have a corresponding class in CityGML. The data was stored in an Oracle Spatial database containing geometries in LoD2, Level-of-Detail 2. The database includes basic building geometries and a terrain model consisting of triangles. All building entities also contain address information, fetched from a database different from the Oracle Spatial database where the building geometries are stored. The plugin fetches the geometries from the database, sorts the geometries and generates the corresponding CityGML. The results are then zipped and sent to the client.

A custom CityGML viewer was also developed by using Direct3D and C#. To evaluate the exported data, three different CityGML viewers were used, LandXplorer, Aristoteles and the custom, more basic CityGML viewer written especially for this research.

The results of the research are a fully functional plugin for Digpro’s software DP/Spatial and a basic but functional CityGML viewer. The conclusion of the research is that the plugin was well implemented in the system, the exports work flawlessly and the CityGML files can be opened in the two available CityGML viewers. The custom CityGML viewer provides basic usability, more than enough to validate the exported data.
Acknowledgements

It is a pleasure to thank all the people who made this thesis possible.

Peter Axelsson, Robert Olofsson and Fredrik Widlert at Digpro AB have all been very supportive and helpful. I am very grateful for all the help I have received.

My supervisor, Professor Yifang Ban, helped me outline the thesis and pointed me in the right direction. Thank you.

Irene Rangel, KTH, helped me out a lot during the last part of the research. Her comments have been invaluable for my work. Muchas gracias.

Mao Bo, thank you for helping me out with the CityGML syntax and thank you for evaluating my work.
Contents

List of Figures

List of Tables

1 Introduction 1
1.1 Rationale for the research ................................. 1
1.2 Research objectives ...................................... 2

2 Literature Review 5
2.1 City Geography Markup Language - The CityGML encoding standard 5
  2.1.1 The Building Model ................................. 7
2.2 Keyhole Markup Language ................................ 14
2.3 CityGML and KML - Comparison .......................... 15
2.4 Geometry Types .......................................... 15
  2.4.1 Linestring ........................................... 15
  2.4.2 Linear Ring .......................................... 16
  2.4.3 Point ................................................ 17
2.5 CityGML Viewers and Visualization Tools ................. 17
  2.5.1 LandXplorer CityGML Viewer ........................ 17
  2.5.2 Aristoteles ......................................... 18
  2.5.3 Direct3D ............................................ 19
  2.5.4 Open Graphics Library (OpenGL) .................... 19
  2.5.5 Direct3D and OpenGL - Comparison ................. 20
2.6 Java Architecture for XML Binding (JAXB) ................. 20
  2.6.1 R-tree .............................................. 22
2.7 Spatial Databases and CityGML .......................... 24
  2.7.1 PostGIS ............................................ 24
  2.7.2 Oracle Spatial ..................................... 24
2.8 Tools for Export and Import of CityGML ................. 26
  2.8.1 CityGML4j ......................................... 26
  2.8.2 3DCityDBv2 .................................... 27

3 Project Database - DP/Spatial Platform description 31
3.1 Baggis .................................................. 31
CONTENTS

3.2 The database ................................................................. 32
3.3 Geometry Types ............................................................. 33
3.4 Buildings ........................................................................ 33
3.5 Height information (Digital Terrain Model) ......................... 33
3.6 Ground texture .................................................................. 34
3.7 Roads ............................................................................... 35
3.8 Addresses ......................................................................... 35

4 Methodology ........................................................................ 37
4.1 Data Pre-Processing ......................................................... 41
4.2 CityGML Viewer ............................................................... 41
    4.2.1 Custom CityGML Viewer ............................................. 41
    4.2.2 XML Parsing ............................................................. 42

5 Results .................................................................................. 45
5.1 Export Results ................................................................. 46
    5.1.1 Building export ......................................................... 46
    5.1.2 Addresses ................................................................. 46
    5.1.3 Addresses in the CityGML Viewer ............................... 47
    5.1.4 Trees ....................................................................... 50
    5.1.5 Road export ............................................................. 50
    5.1.6 Digital Terrain Model ................................................ 50
    5.1.7 Orthophoto ............................................................... 52
    5.1.8 Implementation ........................................................ 52
5.2 The CityGML export ......................................................... 53
5.3 CityGML Viewer ............................................................... 55
    5.3.1 Performance ............................................................ 55
    5.3.2 Meshes .................................................................... 55
5.4 Visualization results ......................................................... 59
    5.4.1 Surface Normals ....................................................... 59

6 Conclusions and Future Research ........................................ 63
6.1 Conclusions ...................................................................... 63
6.2 Should the database be constructed to support CityGML directly? 64
6.3 Future research ............................................................... 65

Bibliography ............................................................................. 67

Appendices ............................................................................... 69
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>LoD0, Digital Terrain Model draped with an orthophoto</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>LoD1. Buildings are represented as geometric representations of volumes</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>LoD2, data over Stockholm displayed in CityGML Viewer</td>
<td>10</td>
</tr>
<tr>
<td>2.4</td>
<td>LoD3. Overhanging roof parts and detailed wall structures</td>
<td>11</td>
</tr>
<tr>
<td>2.5</td>
<td>LoD4. Rooms and furniture</td>
<td>12</td>
</tr>
<tr>
<td>2.6</td>
<td>Modularization of CityGML 1.0.0.</td>
<td>13</td>
</tr>
<tr>
<td>2.7</td>
<td>Linestring</td>
<td>16</td>
</tr>
<tr>
<td>2.8</td>
<td>Linear ring</td>
<td>16</td>
</tr>
<tr>
<td>2.9</td>
<td>LandXplorer CityGML Viewer from Autodesk. The data displayed is the Berlin Pariser Platz from the example datasets available at <a href="http://www.citygml.org">www.citygml.org</a></td>
<td>17</td>
</tr>
<tr>
<td>2.10</td>
<td>Berlin Pariser Platz displayed in Aristoteles. The semantic information is easily accessible by selecting a building in the scene</td>
<td>18</td>
</tr>
<tr>
<td>2.11</td>
<td>Direct3D is situated between the application and the device driver to facilitate the communication between the two</td>
<td>19</td>
</tr>
<tr>
<td>2.12</td>
<td>Outline for marshaling objects into XML</td>
<td>21</td>
</tr>
<tr>
<td>2.13</td>
<td>2D rectangle R-tree</td>
<td>22</td>
</tr>
<tr>
<td>2.14</td>
<td>Oracle Spatial components</td>
<td>25</td>
</tr>
<tr>
<td>2.15</td>
<td>3D City Database import/export tool</td>
<td>29</td>
</tr>
<tr>
<td>3.1</td>
<td>The DP/Spatial application family</td>
<td>31</td>
</tr>
<tr>
<td>3.2</td>
<td>Connection schema</td>
<td>32</td>
</tr>
<tr>
<td>3.3</td>
<td>DP/Spatial</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Geometry components are uniquely identified by OType, OId, CType</td>
<td>34</td>
</tr>
<tr>
<td>3.5</td>
<td>Structure of the Digital Terrain Model</td>
<td>34</td>
</tr>
<tr>
<td>4.1</td>
<td>The outline for the plugin, describing the flow of information</td>
<td>38</td>
</tr>
<tr>
<td>4.2</td>
<td>Convex polygon drawn with the triangle fan method. Polygon faces are all drawn from vertex 0</td>
<td>43</td>
</tr>
<tr>
<td>4.3</td>
<td>When drawing a concave polygon with the triangle fan method, faces will be drawn incorrectly</td>
<td>43</td>
</tr>
<tr>
<td>5.1</td>
<td>The search form</td>
<td>48</td>
</tr>
<tr>
<td>5.2</td>
<td>Address marked by a pin</td>
<td>49</td>
</tr>
</tbody>
</table>
List of Figures

5.3 The squares are divided into triangles ............................................. 50
5.4 DTM displayed in the CityGML Viewer ............................................. 51
5.5 Export progress meter ...................................................................... 52
5.6 A CityGML file containing approximately 90,000 polygons displayed in the CityGML viewer ......................................................... 54
5.7 The relationship between a textured 3-dimensional model and texture coordinates U and V. ................................................................. 56
5.8 Face-vertex mesh ............................................................................. 57
5.9 Relation between surface normal and vertex order ................................. 59
5.10 Culling set to Clockwise, causing some geometry to not display correctly 60
5.11 Culling set to none, which displays the polygon patches correctly. The surface normals may however point in the wrong direction, causing lighting to not display correctly ...................................................... 61
List of Tables

2.1 Accuracy requirements for LoD0-4 ............................................. 8
2.2 Comparison of KML and CityGML ............................................. 15
2.3 The main components of Oracle Spatial ..................................... 25
2.4 Database table for buildings in Oracle Spatial ............................. 28

4.1 Components of the CityGML export plugin ............................... 40
## List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CityGML</td>
<td>City Geography Markup Language</td>
</tr>
<tr>
<td>DBMS</td>
<td>DataBase Management System</td>
</tr>
<tr>
<td>DTM</td>
<td>Digital Terrain Model</td>
</tr>
<tr>
<td>ESRI</td>
<td>Environmental Systems Research Institute</td>
</tr>
<tr>
<td>FME</td>
<td>Feature Manipulation Engine</td>
</tr>
<tr>
<td>GIS</td>
<td>Geographic Information System</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardisation</td>
</tr>
<tr>
<td>JAXB</td>
<td>Java Architecture for XML Binding</td>
</tr>
<tr>
<td>JDBC</td>
<td>Java Database Connectivity</td>
</tr>
<tr>
<td>KML</td>
<td>Keyhole Markup Language</td>
</tr>
<tr>
<td>LoD</td>
<td>Level of Detail</td>
</tr>
<tr>
<td>NIS</td>
<td>Network Information System</td>
</tr>
<tr>
<td>OGC</td>
<td>Open Geospatial Consortium</td>
</tr>
<tr>
<td>OpenGL</td>
<td>Open Graphics Library</td>
</tr>
<tr>
<td>PDF</td>
<td>Portable Document Format</td>
</tr>
<tr>
<td>PR-Tree</td>
<td>Priority R-Tree</td>
</tr>
<tr>
<td>R-Tree</td>
<td>Rectangle Tree</td>
</tr>
<tr>
<td>RAM</td>
<td>Random Access Memory</td>
</tr>
<tr>
<td>RDBMS</td>
<td>Relational DataBase Management System</td>
</tr>
</tbody>
</table>
### List of Tables

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SGI</td>
<td>Silicon Graphics Inc.</td>
</tr>
<tr>
<td>SQL</td>
<td>Structured Query Language</td>
</tr>
<tr>
<td>TIN</td>
<td>Triangular Irregular Network</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>ViSuCity</td>
<td>Visual Sustainable City</td>
</tr>
<tr>
<td>WKB</td>
<td>Well-Known Binary</td>
</tr>
<tr>
<td>WKT</td>
<td>Well-Known Text</td>
</tr>
<tr>
<td>WMS</td>
<td>Web Map Service</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Rationale for the research

There is an increasing need for detailed 3-dimensional city models. City models are used for a number of tasks in city planning, such as noise simulation when planning traffic routes, visualizing the impact of planned buildings on the surrounding area and planning of cell phone networks.

To lower the cost of using 3-dimensional city models, there is a need to develop new applications for 3-dimensional city models, since maintaining and generating 3D city models is an expensive task. Examples of new applications are generation of environmental reports, emergency and catastrophe planning and traffic simulations.

For disaster control, 3-dimensional city models in combination with digital terrain models serve as a basis for flood simulations. When using detailed city models, it is not only possible to simulate the water’s impact on the terrain, it is also possible to calculate the impact on individual buildings.

For engineering applications, a detailed city model gives the engineers the ability to plan, test and evaluate the impact of installations such as water and gas pipes.

Real estate companies may use city models to calculate important parameters such as connectivity, public transport access and local infrastructure to attract new buyers.

Traditional GIS applications only require geometric representations with low level-of-detail, but many newer applications require models containing detailed semantic information as a base for decision-making. Semantic information is important for a variety of different tasks, such as disaster management and city planning. Without semantic information, models can almost exclusively be used for visualization, but with semantic and topological information, the models may be used for analysis tasks, thematic queries and spatial data mining. As 3-dimensional city
models become more useful with the introduction of detailed semantic information, the number of potential users of the 3-dimensional model increases, which generally makes models less expensive.

The 3-dimensional city models of today are often stored as a collection of geometry components in a spatial database, with or without detailed semantic information.

When analyzing large areas, different data sets are often combined. Since many different databases use different table structures, and almost all databases differ regarding data quality and detail level, these data sets may be complicated to merge. This introduces a need for a standardized format for data exchange. When two data sets overlap, it is important to guarantee that one spatial object is not included more than once, which is only possible with detailed semantic information. In the case where one object is represented more than once, it is normally desirable to choose the most detailed version of the object.

CityGML is a relatively new format that aims to standardize the storage and exchange of 3-dimensional city models. The city models may include detailed semantic information which provides the user with important information such as addresses and parcel information. The format may be used as either an in-between format between databases or for visualization purposes.

For this research, an export tool for exporting data from an existing database into CityGML format. The existing data is stored in an Oracle Spatial database at Digpro AB and consists of different geometry types; therefore the data will be converted into CityGML by using a custom export plugin for the Digpro software.

1.2 Research objectives

The objectives for this research are to develop a well-functioning CityGML export tool from the Baggis database, evaluate which of the existing data layers that are to be exported and also to evaluate the use of CityGML when exporting 3D data by reviewing the final result. The tool should be implemented in the existing application DP/Spatial from Digpro.

The Baggis database contains a number of different thematic layers. Every layer has to be considered regarding whether to be included or not in the export. The export will focus on the building models, but when relevant, other geometries will also be included such as roads and terrain information.

This research will also evaluate the possibility of building a 3D database directly adapted to CityGML. Based on these objectives, it is possible to quantify and
1.2. RESEARCH OBJECTIVES

evaluate the final results.
Chapter 2

Literature Review

In this chapter, research involving CityGML, KML and PostGIS is presented along with research involving visualization.

2.1 City Geography Markup Language - The CityGML encoding standard

CityGML is an open data model for storage and exchange of virtual city models, implemented as an application schema for the Geography Markup Language, GML. The objective of CityGML is to standardize the representation of city objects with attributes, entities and relations.

On August 20, 2008, the Open Geospatial Consortium adopted version 1.0.0 of CityGML as an official OGC standard for representing cities in 3 dimensions. The semantic representation in CityGML is powerful and flexible. The difference from GML is that CityGML not only represents the geometries of the city objects, but also represents semantic and thematic properties such as addresses, municipalities and clusters. (Gröger et al., 2005)

The two main parts of the markup language are the thematic model and the geometric model. The geometric model represents all geometrical and topological information of objects in a 3-dimensional city model, while the thematic model includes semantic information and utilizes the geometry model for the representation of objects. The semantic properties allow the user to define how different objects are related to each other. For example, a drawer may be located in a room, which belongs to a house.

Benner et al. (2005) conclude that traditional GIS applications like sightline analysis and flood protection require geometric models with low level-of-detail, and very little semantics. Newer applications regarding 3D city models, such as visualization, have also traditionally required little semantics. For these reasons, the city
models of today contain very little semantic information, and often the semantic information available is represented as attributes. Since 3D models are very expensive, both to generate and to maintain, there is a need to develop more applications for 3D city models, some of which are only possible to realize with semantic models. (Benner et al., 2005)

In a semantic model, all different geometry parts are associated with a semantic meaning such as wall, roof or building. With a semantic model, it is possible to realize new applications for 3D models within fields such as urban management and town planning. Benner et al. (2005) also state that a semantics-driven visualization enhances the model’s usability, while explicit semantic information facilitates the usage of intelligent applications for urban development.

Stadler and Kolbe (2007) describe the importance of coherence in 3D city models. Coherence is described as ‘the quality or state of cohering, namely a logical, orderly, and aesthetically consistent relationship of parts’. Stadler and Kolbe (2007) argue that if semantic and geometric aggregations show the same structure, they will be considered coherent, and only then can they be used in conjunction. This conjunction bears two benefits; that geometrical objects contain information about what they are, and that semantic entities contain information about where they are and what their spatial extents are.

Stadler and Kolbe (2007) also argue that spatial complexity has no relation to the number of geometric primitives, but that it rather denotes the subdivision of geometry into meaningful groups. Spatio-semantic coherence is relevant for successful data integration. Stadler and Kolbe (2007) argue that if there are no semantics, integration can only be carried out on the basis of the geometric shape, and that harmonization of the model cannot rely on so called ‘tie points’ or other connecting features for correlation of geometries. Stadler and Kolbe also describe harmonization of semantically coherent models. The detailed semantics may help to identify objects by their properties or to define tie points for adjustments. For example, by knowing the location of a door, the terrain can be aligned to the door’s lower edge to avoid buildings situated above the terrain.

The semantic model of CityGML complies with the ISO191xx standards family framework for the representation and modeling of geographic features. ISO19109 defines geographic features as abstractions of real world objects. Geographic features may be associated with an arbitrary number of spatial and non-spatial attributes. (Gröger et al., 2005)

Objects that appear more than once at different locations, for example lamp-posts, can be modeled once and used multiple times in the same model. (Czerwinski et al., 2008)
2.1. CITY GEOGRAPHY MARKUP LANGUAGE - THE CITYGML ENCODING STANDARD

2.1.1 The Building Model

The building model is the most detailed of the CityGML concepts, defined by the thematic extension module Building, as described earlier. Buildings may be an aggregation of BuildingParts, which allows the separation of different parts of large buildings. The classes Building and BuildingPart include information about building class, function, usage, roof type, height (as well as individual heights for each of the building floors). Both classes may also reference one or more addresses. (Czerwinski et al., 2008)

According to the CityGML encoding standard, building complexes such as airports or hospitals should be aggregated into CityObjectGroups. The different buildings in the group may then be given individual attributes such as ‘Main Building’. The representation and the semantic structure of buildings are refined from LoD1 to LoD4.

There are five different Levels of Detail (LoD’s) in the CityGML encoding standard. The different LoD’s allow for a more detailed view and a low-poly representation of city models. A CityGML file may include several models of the very same building but in different Levels of Detail, which is useful when different parts of the dataset have different sources and accuracy. (Czerwinski et al., 2008)
## CHAPTER 2. LITERATURE REVIEW

<table>
<thead>
<tr>
<th>Model scale description</th>
<th>LoD0</th>
<th>LoD1</th>
<th>LoD2</th>
<th>LoD3</th>
<th>LoD4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Regional, landscape</td>
<td>City, Region</td>
<td>City districts, projects</td>
<td>Architectural models, landmarks</td>
<td>Architectural models with interior</td>
</tr>
<tr>
<td>Class of accuracy</td>
<td>Lowest</td>
<td>Low</td>
<td>Middle</td>
<td>High</td>
<td>Very high</td>
</tr>
<tr>
<td>Absolute 3D point accuracy (position / height)</td>
<td>Lower than LoD1</td>
<td>5/5m</td>
<td>2/2m</td>
<td>0.5/0.5m</td>
<td>0.2/0.2m</td>
</tr>
<tr>
<td>Building installations</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Representative exterior effects</td>
<td>Real object form</td>
</tr>
<tr>
<td>Roof form / structure</td>
<td>No</td>
<td>Flat</td>
<td>Roof type and orientation</td>
<td>Real object form</td>
<td>Real object form</td>
</tr>
<tr>
<td>Roof overhanging parts</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

**Table 2.1.** Accuracy requirements for LoD0-4. Source: Czerwinski et al. (2008)
2.1. CITY GEOGRAPHY MARKUP LANGUAGE - THE CITYGML ENCODING STANDARD

Figure 2.1. LoD0, Digital Terrain Model draped with an orthophoto

LoD0 contains a 2.5D DTM, which may be draped with an aerial photo or a map. Buildings are not included in this LoD.

Figure 2.2. LoD1. Buildings are represented as geometric representations of volumes.

LoD1 may include the DTM from LoD0 but also includes a block model of the city. LoD1 models do not include any building textures or roof structures. The detail levels are also different regarding accuracy and minimum object sizes. In LoD1, both positional and height accuracy has to be at the most 5 m. Also, every object with a footprint of 6x6m has to be included in the model. Buildings consist
of geometric representations of building volumes.

LoD2 introduces the possibility to separate different surfaces into groups of thematic objects, such as roof, wall or floor. A building in LoD2 may also include installations such as chimneys and stairs (Gröger et al., 2005).

LoD2 also allows for the insertion of vegetation objects. LoD2 requires a positional accuracy of 2 m, while the height accuracy is set to 1 m, while objects with a footprint of 4x4m or larger have to be taken into consideration.
2.1. CITY GEOGRAPHY MARKUP LANGUAGE - THE CITYGML ENCODING STANDARD

Figure 2.4. LoD3. Overhanging roof parts and detailed wall structures

LoD3 includes architectural models with even more detailed wall structures, and roof structures along with building parts such as balconies and doors. These structures may be draped with high-resolution textures. Optionally, buildings in LoD3 may include openings for windows and doors. LoD3 also allows for including detailed vegetation and transportation objects. LoD3 requires an accuracy of 0.5m in both heights and positions and also that objects larger than 2x2m are included.
LoD4 is the most detailed LoD and includes the information from LoD3, combined with interior structures for objects such as interior doors, rooms and furniture. LoD4 requires that both heights and positions have an accuracy of 0.2m. (Gröger et al., 2005)

CityGML allows for referencing objects in external data sets, which is commonly used for referencing buildings to parcels and for other purposes. These external references may be essential to allow for updating the dataset. The references are specified as URI’s, Uniform Resource Identifiers, a generic format for referencing resources on-line.

Gröger et al. (2005) state that the building model is the core of CityGML. The building model may be used to represent thematic and spatial attributes of buildings and parts of buildings in the different LoD’s. CityGML allows for including a Digital Terrain Model, DTM; where the terrain may be specified as a raster or grid, a TIN, as break lines or points. (Gröger et al., 2005) Break lines represent discontinuities in the terrain, while points are a simple set of 3D points. The four different terrain types may be combined in the same data set in a number of ways. A terrain region may also be partially replaced by higher resolution terrain information.

Gröger et al. (2005) suggest that a coarse grid representing a terrain surface may be partially replaced by a detailed, high-accuracy TIN, separated by the extents of the 2D footprint polygons. Since version 1.0.0, the CityGML standard has been

Figure 2.5. LoD4. Rooms and furniture
2.1. CITY GEOGRAPHY MARKUP LANGUAGE - THE CITYGML ENCODING STANDARD

partitioned into modules as shown in figure 2.6. The vertical modules include the definitions for the different thematic models, for example building, city furniture and land use. The horizontal modules represent structures, which may be applied to all thematic modules. (Kolbe, 2008)

![Diagram of CityGML modules](image)

**Figure 2.6.** Modularization of CityGML 1.0.0. The vertical modules contain semantic modeling for different thematic areas. Source: Kolbe (2008)

CityGML is restricted to non-curved geometries as support for these geometries is rarely included in GIS or spatial data management systems. During the conversion to CityGML, curved surfaces must therefore be approximated by facetted surfaces or patches. (Kolbe, 2008)

**Applications**

CityGML is a standardized format, which in theory makes data exchange easy. Organizations and companies may import and export data in the CityGML format. Buildings may be referenced against an external database for cross-referencing of buildings along with globally unique ID’s. It is possible to use CityGML for a large number of applications.

Gröger et al. (2005) suggest that the exchange of CityGML models is useful for disaster management, to easily answer questions such as which buildings and which floors that are affected by a flood, which escape route to choose etc. The representation of storey heights above ground and under ground makes it possible to determine to which degree different buildings are affected, information which is useful for planning evacuation and damage assessment.
By computing the volume and determining the characteristics of different objects, it is possible to simulate flooding scenarios, fire scenarios and other natural and unnatural disasters.

## 2.2 Keyhole Markup Language

Keyhole Markup Language is a language schema for representing geographic information with XML. KML was originally developed by Keyhole, Inc. to be used with the Google Earth viewer, which at the time was named Keyhole Earth viewer. A KML file may include a variety of different features, for example polygons, images, 3D models and placemarks. For positioning, KML uses 3-dimensional geographic coordinates containing longitude, latitude and altitude.

On April 14th, 2008, KML was officially approved by the Open Geospatial Consortium as an Open Standard, which means that developers have a standard approach for using KML for sharing geographic data. (Open Geospatial Consortium, 2008)

According to the reference documentation, KML can be used to:

- Annotate the earth
- Specify icons and labels to identify locations
- Create different camera positions to define unique views for different features
- Define image overlays to attach to the ground
- Define styles to specify KML feature appearance
- Write HTML descriptions of KML features
- Organize KML features into hierarchies
- Locate and update KML documents
- Define location and orientation of textured 3D objects

(Google Inc, 2008)

The use of KML is well known and commonly used over the world. The Google Earth Community has over 1,100,000 members as of February 2009. It is possible to create overlays for existing Google maps, which may then be shared with other users. The overlays may include detailed 3D model information with textures, and polygons which represent the building’s geometry. People use KML for a variety of purposes; it may be used for simple tasks, such as indicating stops on a trip but also for more advanced purposes. For example, scientists use KML for detailed mapping.
of models and trends such as weather patterns and earthquake activity. Historical patterns, urban growth and other temporal data may be included with a time span, which allows the modeler to specify during which time a specific phenomenon occurs. (Google Inc, 2008)

Geometries are represented in KML similar to CityGML by using LinearRings and LineStrings.

### 2.3 CityGML and KML - Comparison

While KML and CityGML overlap in functionality, there are a number of important differences. Both standards use similar geometry elements for representing real-world features, such as points, and linear rings, but KML lacks the many of the semantic properties included in CityGML.

KML is normally used as a presentation language, with an underlying map created by a WMS service.

<table>
<thead>
<tr>
<th></th>
<th>KML</th>
<th>CityGML</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometry</td>
<td>Basic</td>
<td>Sophisticated</td>
</tr>
<tr>
<td>Georeferencing</td>
<td>Sophisticated</td>
<td>Very sophisticated</td>
</tr>
<tr>
<td>Appearance</td>
<td>Basic</td>
<td>Sophisticated</td>
</tr>
<tr>
<td>Topology</td>
<td>-</td>
<td>Sophisticated</td>
</tr>
<tr>
<td>Semantics</td>
<td>-</td>
<td>Very sophisticated</td>
</tr>
<tr>
<td>Embedding</td>
<td>Very sophisticated</td>
<td>Very sophisticated</td>
</tr>
</tbody>
</table>

*Table 2.2. Comparison of KML and CityGML*

While KML focuses on geometry and appearance, CityGML also includes advanced semantic feature models.

### 2.4 Geometry Types

There are a number of geometry types that are commonly used for representing geometries. In this section, the most common geometry types are presented along with short descriptions.

#### 2.4.1 Linestring

A linestring is defined as a sequence of points representing the line segments connecting them.
2.4.2 Linear Ring

A linestring is defined as a sequence of points representing the line segments connecting them. A linear ring is defined as a linestring, where the first and the last coordinate is the same. To be considered valid, the linear ring must not self-intersect. The vertex order may be both clockwise and counter-clockwise, although some systems for spatial data management require that the vertices are specified in a specific direction. Polygons are generally represented as linear rings.
2.5. CITYGML VIEWERS AND VISUALIZATION TOOLS

2.4.3 Point

Points are generally used to represent height values or landmarks and consist of coordinates in 1, 2 or 3 dimensions.

2.5 CityGML Viewers and Visualization Tools

There are two CityGML-viewers available on the CityGML web site (as of January, 2009), LandXplorer and Aristoteles, both free of charge.

2.5.1 LandXplorer CityGML Viewer

LandXplorer CityGML Viewer is a part of Autodesk LandXplorer Studio Professional, a package of applications for 3D city and landscape models. The program package is a fully interactive application for authoring and visualization of 3-dimensional geospatial data. The application is currently only available for Microsoft Windows.

Figure 2.9. LandXplorer CityGML Viewer from Autodesk. The data displayed is the Berlin Pariser Platz from the example datasets available at www.citygml.org
2.5.2 Aristoteles

Aristoteles is a viewer written in Java, developed by the Institute of Geodesy and Geoinformation in Bonn. The viewer uses the Java3d API for rendering geometries and allows for displaying more than one CityGML file simultaneously. Aristoteles runs under Windows, Linux and MacOSX. Since the geometry annotation in CityGML is based on the GML standard, it is also possible to view any file conforming to the GML 3.x standard. Aristoteles offers good access to the semantic information contained in the CityGML file, as seen in figure 2.10.

Figure 2.10. Berlin Pariser Platz displayed in Aristoteles. The semantic information is easily accessible by selecting a building in the scene.
2.5. CITYGML VIEWERS AND VISUALIZATION TOOLS

2.5.3 Direct3D

Direct3D is a part of Microsoft’s API DirectX and consists of a number of API’s used for different purposes. For this research, the relevant groups are DirectX Graphics and DirectInput. DirectInput is used for interaction with keyboards and other physical controllers.

The graphics component contains three packages:

- DirectDraw, used for drawing 2-dimensional graphics.
- DXGI, used for management of graphics hardware and monitors.
- Direct3D, used to render 3-dimensional graphics, generally for games and visualization applications. Direct3D is also the base for the API’s used on Microsoft’s console systems Xbox and Xbox 360. Direct3D is the API that will be used for this research.

Direct3D provides interfaces to the different functions of the video card such as functions for textures, materials, clipping, lighting and transformations. (Microsoft, 2008)

Before any actual graphics rendering can be performed, a device has to be initialized. The device is responsible for the rendering of the 3D scene and may be initialized with a number of different options such as full screen, windowed mode, resolution and color depth. When the device has been initialized, it is possible to assign resources such as textures, and vertex information to the device.

2.5.4 Open Graphics Library (OpenGL)

OpenGL is a standard specification developed by Silicon Graphics Inc. in 1992. In 2006, the control of OpenGL was given to the industry consortium The Khronos
Group. The specification defines a platform-independent API for writing graphics applications in 2D and 3D.

Unlike Direct3D, OpenGL is an open standard API. OpenGL aims to abstract the interfacing with 3D graphics hardware by giving developers access to a single API.

The software interface consists of hundreds of calls and functions which allow specification of objects and operations for graphics rendering. The basic functionality of the API is to handle geometries such as points, lines and polygons and to convert them into pixels on the screen. Developers have access to a set of commands for defining geometric objects as well as commands for controlling the rendering of the objects into the buffer. An OpenGL program generally initializes a framebuffer, which is where the actual drawing is performed. When the framebuffer has been initialized and allocated into the memory, OpenGL commands can be called. (Segal and Akeley, 2008)

OpenGL only treats the graphics part of applications, no support for keyboard or mouse input is implemented, which makes it necessary to use a secondary API for implementing input features.

2.5.5 Direct3D and OpenGL - Comparison

Unlike Direct3D, OpenGL is an open standard API, which allows developers to write bindings to the API from virtually any programming language. Direct3D is designed to act as an interface to the 3D hardware, while OpenGL is designed to act as a 3D rendering system which may be hardware accelerated. (Segal and Akeley, 2008)

A few fundamental differences exist between Direct3D and OpenGL. The OpenGL implementation handles hardware resources, while Direct3D relies on the application to handle these resources. OpenGL development is thereby simpler and less complex, while it is more complicated to write a well-performing driver for the 3D hardware. Direct3D developers must write code for managing hardware resources, which gives more flexibility when allocating resources in the most efficient way. Direct3D, which is part of the DirectX API comes with API’s for controlling input and sound, which makes it a complete API for development of visualization applications and computer games.

2.6 Java Architecture for XML Binding (JAXB)

Java Architecture for XML Binding, (JAXB) is an API included in the Java EE platform. JAXB allows for easy binding between XML schemas and Java representations, to facilitate the use of XML data in Java applications.
JAXB includes functionality for reading and writing XML instance documents, combined with functions for generation of XML schemas from Java objects.

The binding framework in JAXB defines API's for accessing functions for unmarshaling (reading data into Java content trees), marshalling (writing XML instance documents from Java objects) and validation. Both marshalling and unmarshaling operations have to be preceded by a binding of the XML schema, which basically refers to the generation of Java classes, which represent the XML schema.

All JAXB implementations include a so-called binding compiler for binding XML schemas. The tool reads the schema and outputs a set of interfaces and classes which can be used for implementation. (Sun Microsystems Inc., 2009)

For a schema over a collection of books of different types, the interfaces may have the following structure:

**CollectionType.java.** Represents the unnamed complex type for the `<Collection>` element.
**Collection.java,** represents the `<Collection>` element.
**BookType.java,** represents the BookType complex type.
**ObjectFactory.java.** Contains methods for generation of instances of the interfaces.
(Sun Microsystems Inc., 2009)

The tool also generates a number of classes for implementation of the classes above. When writing XML data, the first task is to bind the schema for the XML document as explained above.

The second task is to build a content tree, which represents the information that is to be written into the XML document. When these two tasks have been successfully performed, the content tree may be marshaled into the XML document. The API does not require the user to know any XML.

![Figure 2.12. Outline for marshaling objects into XML](image)
2.6.1 R-tree

An R-tree is a tree data structure used for spatial indexing of multi-dimensional information. The R-tree splits the area into so called *Minimum Bounding Rectangles, MBR’s*, bounding boxes.

The R-tree contains a number of non-leaf nodes which contain information about child nodes and the bounding box for each child node. When inserting new entries, it is required that the algorithm uses the bounding boxes contained in the nodes to guarantee that geometries are placed together with nearby geometries. The leaf nodes contain information about contained geometries and a method to access these geometries.

![Figure 2.13. 2D rectangle R-tree](image-url)
2.6. JAVA ARCHITECTURE FOR XML BINDING (JAXB)

Searching an R-tree

R-trees are used for different kinds of spatial queries, for example whether a rectangle contains a point. The search starts from the root node. For every child node, the algorithm decides whether the point is contained within this area. If not, the algorithm continues with the next node. If the point is located within the bounds of the child node, the program continues to search this child node in a recursive manner. When the program reaches a leaf node, it searches all the geometries within this leaf node. The matching geometries are added to the set of results. This search method is much faster than searching every single rectangle.
2.7 Spatial Databases and CityGML

2.7.1 PostGIS

PostGIS is an open source spatial database research project developed by Refractions Research Inc., which adds support for spatial data to the database management system PostgreSQL. Among other things, PostGIS includes support for:

- Points, Polygons, Line strings, Multi-points, Multi-Line strings, Multi-polylines and Geometry Collections.
- Functions for determination of interaction between geometries
- Spatial operators for determination of measurements such as area, distance and length.
- Spatial operators for determination of set operations such as union, difference and buffers.
- R-tree spatial indexing

PostGIS can be used as a backend database, and is used by a number of well-known applications such as MapServer, GeoServer and MapGuide. PostGIS includes support for a number of standards, among them the OGC standards Simple Features, Well Known Text and Well Known Binary. It also complies with the OGC SQL functions for managing spatial data.

(Refractions Research, 2009)

2.7.2 Oracle Spatial

Oracle Spatial is a DBMS for managing geographic data, available as an optional component of the Oracle Database. Oracle Spatial includes a database schema named MDSYS. MD refers to multi-dimensional and includes methods for storing, retrieving, querying and updating spatial data in the database.

The data model refers to the storage of spatial data in a database. The user can define tables containing geometries for storing spatial data. Query and Analysis allows the user to query and update the data in the database. The spatial engine includes most of the components from table 2.3. The visualization layer includes the required functionality for visualization of spatial data.
2.7. SPATIAL DATABASES AND CITYGML

- Spatial indexing system
- Topologic data model
- Network data model
- GeoRaster functionality for indexing, storing and querying of GeoRaster data
- Operators and functions for spatial queries
- Schema over the supported geometric data types

**Table 2.3.** The main components of Oracle Spatial

**Figure 2.14.** Oracle Spatial components

Source: Kothuri et al. (2004)
Example Application: 3D Geodatabase Berlin

In 2007, the official 3D city model of Berlin was upgraded to the CityGML specification 0.4.0. The 3D Geodatabase Berlin is stored in an Oracle Spatial 10G R2 database.

Kolbe, König, Nagel and Stadler (2008) explain that apart from general advantages from using a well-known commercial database management system, it also inhibits some important performance characteristics for an efficient implementation. Among the advantages are:

- Oracle Spatial 10G R2 supports spatial data types in 2D, 3D and 4D.
- Version 10G R2 of the Oracle RDBMS offers efficient management of georeferenced grid data, which makes it possible to store whole DTM’s as well as aerial images as homogeneous objects. (Oracle, 2008)
- By employing stored procedures and triggers, it is possible to implement rules for updating objects. (Gröger et al., 2008)

2.8 Tools for Export and Import of CityGML

2.8.1 CityGML4j

During the time of this research, the Institute for Geodesy and Geoinformation Science at the Technische Universität Berlin introduced CityGML4j, a Java library for import/export of CityGML.

CityGML4j relies on the standardized Java Architecture for XML Binding, JAXB. After fetching building information and geometries from a data source such as a database, buildings and other city furniture are instantiated as Java objects. These objects may then be linked to other objects; for example, building components may be attached to a Building object.

This work method facilitates generation of CityGML documents since the user does not need much previous knowledge about the CityGML syntax. When all the objects have been instantiated, the tree of objects is serialized into a CityGML document. This output is commonly referred to as marshaling. (Institute for Geodesy and Geoinformation Science at the Technische Universität Berlin, 2008)

CityGML4j handles the CityGML objects by using an object-oriented pattern called the factory method. The factory method provides a way to create objects without previously specifying the exact classes of the created objects. This is handled by sub-methods for object creation.
2.8. TOOLS FOR EXPORT AND IMPORT OF CITYGML

// set up citygml4j context
CityGMLContext ctx = new CityGMLContext();
CityGMLFactory citygml =
    ctx.createCityGMLFactory(CityGMLModuleVersion.v0_4_0);
GMLFactory gml = ctx.createGMLFactory();

// create a Building instance
Building building = citygml.createBuilding();

// create a gml:Solid for the LoD2 representation of the building
Solid solid = gml.createSolid();
SolidProperty solidProperty = gml.createSolidProperty();
solidProperty.setSolid(solid);
building.setLod2Solid(solidProperty);

// finally, create a CityModel and add the building
CityModel cityModel = citygml.createCityModel();
CityObjectMember cityObjectMember = citygml.createCityObjectMember();
cityObjectMember.setCityObject(building);
cityModel.addCityObjectMember(cityObjectMember);

Source: Institute for Geodesy and Geoinformation Science at the Technische Universität Berlin (2008)

As seen in the source above, objects are attached to one another to build a tree of objects in the RAM. When the user wants to export the objects to file, the tree is simply written by traversing the tree.

2.8.2 3DCityDBv2

A Java-based front-end tool for import and export of 3D data has been developed to provide a graphical interface to the database. The tool is implemented on top of CityGML4j, which is presented in section 2.8.1. Buildings are implemented in the Oracle Spatial database according to table 2.4.

3DCityDBv2 offers functionality such as full support for CityGML 1.0.0, reading and writing of CityGML instance documents and texture support. (Institute for Geodesy and Geoinformation Science at the Technische Universität Berlin, 2008)
The tool was used to implement the 3D Geodatabase Berlin, as described in section 2.7.2

Geometry components such as openings, rooms and surfaces are all implemented in the Oracle Spatial database as SDO_GEOMETRY. All buildings are differentiated by a unique building ID, which allows for identification, manipulation and
### Table 2.4. Database table for buildings in Oracle Spatial. Source: Kolbe et al. (2008)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Length / Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>NUMBER</td>
<td>NOT NULL</td>
</tr>
<tr>
<td>NAME</td>
<td>VARCHAR2</td>
<td>1000</td>
</tr>
<tr>
<td>NAME_CODESPACE</td>
<td>VARCHAR2</td>
<td>4000</td>
</tr>
<tr>
<td>BUILDING_PARENT_ID</td>
<td>NUMBER</td>
<td></td>
</tr>
<tr>
<td>BUILDING_ROOT_ID</td>
<td>NUMBER</td>
<td></td>
</tr>
<tr>
<td>DESCRIPTION</td>
<td>VARCHAR2</td>
<td>4000</td>
</tr>
<tr>
<td>CLASS</td>
<td>VARCHAR2</td>
<td>256</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>VARCHAR2</td>
<td>1000</td>
</tr>
<tr>
<td>USAGE</td>
<td>VARCHAR2</td>
<td>1000</td>
</tr>
<tr>
<td>YEAR_OF_CONSTRUCTION</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>YEAR_OF_DEMOLITION</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>ROOF_TYPE</td>
<td>VARCHAR2</td>
<td>256</td>
</tr>
<tr>
<td>MEASURED_HEIGHT</td>
<td>BINARY_DOUBLE</td>
<td></td>
</tr>
<tr>
<td>STOREYS_ABOVE_GROUND</td>
<td>NUMBER</td>
<td>8</td>
</tr>
<tr>
<td>STOREYS_BELOW_GROUND</td>
<td>NUMBER</td>
<td>8</td>
</tr>
<tr>
<td>STOREY_HEIGHTS_ABOVE_GROUND</td>
<td>VARCHAR2</td>
<td>4000</td>
</tr>
<tr>
<td>STOREY_HEIGHTS_BELOW_GROUND</td>
<td>VARCHAR2</td>
<td>4000</td>
</tr>
<tr>
<td>LoD1_TERRAIN_INTERSECTION</td>
<td>MDSYS.SDO_GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>LoD2_TERRAIN_INTERSECTION</td>
<td>MDSYS.SDO_GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>LoD3_TERRAIN_INTERSECTION</td>
<td>MDSYS.SDO_GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>LoD4_TERRAIN_INTERSECTION</td>
<td>MDSYS.SDO_GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>LoD2_MULTI_CURVE</td>
<td>MDSYS.SDO_GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>LoD3_MULTI_CURVE</td>
<td>MDSYS.SDO_GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>LoD4_MULTI_CURVE</td>
<td>MDSYS.SDO_GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>LoD1_GEOMETRY_ID</td>
<td>NUMBER</td>
<td></td>
</tr>
<tr>
<td>LoD2_GEOMETRY_ID</td>
<td>NUMBER</td>
<td></td>
</tr>
<tr>
<td>LoD3_GEOMETRY_ID</td>
<td>NUMBER</td>
<td></td>
</tr>
<tr>
<td>LoD4_GEOMETRY_ID</td>
<td>NUMBER</td>
<td></td>
</tr>
</tbody>
</table>

Extraction of buildings. All buildings may be included in different Levels of Detail simultaneously, which allows for easy extraction of certain LoD’s for different purposes. The project source shows that a building may be selected with a SQL statement such as:

```sql
select b.ID, b.BUILDING_PARENT_ID
        ... where b.BUILDING_ROOT_ID = ?
```

Source: DBBuilding.java from the 3DCityDBv2 export tool source code, (Institute for Geodesy and Geoinformation Science at the Technische Universität Berlin, 2008)
2.8. TOOLS FOR EXPORT AND IMPORT OF CITYGML

The tool includes a graphical user interface, which allows the user to perform different operations on the database.

The tool allows the user to filter the data to export in a number of ways. The filter may be set to export only specific GML names, CityObjectMembers, all components within a bounding area or only specific feature classes such as building, water body and vegetation. The raster export only considers the bounding box as a constraint. The user can choose whether to export DTM data or a georeferenced image file.

Figure 2.15. 3D City Database import/export tool
It is also possible to include a timestamp for comparison of datasets from different dates. The documentation to the tool provides detailed instructions along with SQL scripts for setting up an Oracle Spatial database directly adapted to the CityGML standard.
Chapter 3

Project Database - DP/Spatial
Platform description

3.1 Baggis

The DP/Spatial application family is built on the DP/Spatial Base Platform. All applications are customized depending on the customer’s needs. DP/Spatial provides the applications with GIS functionality as well as NIS (Network Information System) functionality. (Digpro AB, 2008)

DP/Spatial Application Family

Applications
NIS
GIS
Modules
Platform

Figure 3.1. The DP/Spatial application family. (Digpro AB, 2008)
CHAPTER 3. PROJECT DATABASE - DP/SPATIAL PLATFORM DESCRIPTION

The client may be in the form of a Java applet, a MicroStation client or another external system. Baggis is represented as DP/Cadaster in figure 3.3. The client communicates with a web server providing Java servlets and web services. The web server queries a database running Oracle Spatial through JDBC for geometries and other spatial data.

![Connection schema](image)

Figure 3.2. Connection schema. (Digpro AB, 2008)

3.2 The database

The database includes all common geometry types, such as polygon, curve, line, and point. The export to CityGML focuses on one particular map theme, called Baskarta 3D, Base Map 3D. The base map includes geometries in 3 dimensions over Stockholm and surrounding areas. The buildings are modeled in different levels of detail, some areas such as the city core are relatively well modeled with 10-20 geometries per buildings (in some cases up to 5000 polygons), while some of the outskirts are modeled in a lower level of detail and mainly consists of blocks representing the buildings. The DTM however is well represented by a point grid with a 20m interval between each point in the grid.

Every object in the map is distinguishable by an Object ID and an Object Type, which in combination are unique for every building, i.e. one building may consist of several geometries with the same Object Id and Object Type, but two separate buildings may not share the combination of these attributes. (Digpro AB, 2008)

The different geometries and ground textures may be extracted by using different modules and Java classes, including export functions to a variety of formats such as FME, PDF, and ESRI Shapefile. While these formats are completely different from CityGML, the basic functionality in these functions is similar to the goal of this research.
3.3 Geometry Types

The Base Map 3D contains symbols and polygons. The symbols are mainly used for the representation of trees and other objects while the polygons represent buildings, roads and other city objects such as bridges and monuments.

The different geometry types are divided into different layers and may easily be fetched on a layer-by-layer basis.

3.4 Buildings

Buildings are represented as a set of geometries contained in five different layers. Each layer represents a different type of building geometry such as roof, floor, wall or detail. Since the information is 3-dimensional and the web applet 2-dimensional, it is hard to verify the layer contents without exporting the data. The geometry components do not have any attached information such as building ID, parcel ID or similar, which makes it necessary to perform a second query against the database using Oracle Spatial commands to fetch this information.

3.5 Height information (Digital Terrain Model)

The Base Map includes a Digital Terrain Model with a resolution of 20m, which may be used in the exported CityGML file to represent the relief of the terrain. The terrain model in the database is represented by square polygons containing four
3.6 Ground texture

The CityGML encoding standard allows for including a relief draped by a raster. A high-resolution orthophoto of the exported area may therefore be included. The available orthophoto in the database has a resolution of 15 cm.
3.7 Roads

Roads, bridges and walkways each have their own Object Type, which makes it easy to identify and export these geometry components. Road geometries in 2D are not available for the entire Stockholm area, but functionality for road export should be included in the export tool.

3.8 Addresses

Most buildings have at least one address. In a city, a building generally has more than one address. As mentioned before, a geometry component does not have any attributes attached to it so for every building a database query or similar is necessary. The CityGML format allows for a building element to contain an unlimited number of addresses. The addresses are stored in a separate database with information about address name and X and Y coordinates for the door.
Chapter 4

Methodology

The Base Map does not include every possible type of CityGML object. The objective of the export function is therefore to provide export functionality for the relevant geometry components and other information, while writing the code in such a way that any kind of data support can be added in the future.

Since there is an application available, the Digpro software, the requirement is to implement the export function in the form of a plugin to this application. The plugin must be robust enough to handle large amounts of data and also fast enough to perform the export within a reasonable amount of time.

The outline of the plugin is presented in Figure 4.1
CHAPTER 4. METHODOLOGY

Client side

The client selects the area to be exported. The coordinates for the bounds are sent to the server.

The user is asked to download the file and may now unzip it and open it in a CityGML viewer.

Server side

The Java servlet queries the database for all addresses in the area.

The Java servlet queries the database for all the geometries in the area.

The Java servlet queries the database for an orthophoto of the area.

The geometries are sorted and processed.

The results are written to a CityGML file. The orthophoto is stored as a JPEG image file. The files are zipped into one single file.

Figure 4.1. The outline for the plugin, describing the flow of information.
For simplicity, the plugin will be broken down into eight different java files on the server side, each class representing a different part of the export. This produces a relatively well-structured CityGML file where all the different object types are written to file in groups.

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Export Objects To CityGML</td>
<td>This is the entrance point for the program. The class extends a server class called BaseCall which allows it to be called from the client side. A number of lists for storing geometry components of different types. The types are DTM components, trees, roads and buildings. The program creates a unique filename based on username and time and binds a file output to this file.</td>
</tr>
<tr>
<td>Export Objects To CityGML Helper</td>
<td>The helper contains functions for loading geometries from the database, with hard-coded settings for inclusion and exclusion of certain geometry types. The helper also assists with writing headers for the CityGML file.</td>
</tr>
<tr>
<td>CityGML Export Building</td>
<td>The building export class is called from the main class. First, the class outputs the CityGML associated with a building, such as CityObjectMember and Building. Secondly, it determines which addresses that are to be included and stores them in a Set. When all the addresses have been determined, the program outputs all the geometry information to the file and finally outputs all the associated addresses.</td>
</tr>
<tr>
<td>Address Info</td>
<td>The class AddressInfo contains a structure for storing address information for easy access. All addresses are represented as a text string for the street, a text string for the house number and doubles for storing the X and Y coordinates of the address. The reason for choosing a text string for storing the house number as a string is that some houses also include letters in the house number, for example 23C.</td>
</tr>
</tbody>
</table>
### CHAPTER 4. METHODOLOGY

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>CityGML DTM Export</td>
<td>The main class calls the DTM class to export geometry components associated with the Digital Terrain Model. The class exports all the geometries in one run, as one single CityObject-Member.</td>
</tr>
<tr>
<td>CityGML Trees Export</td>
<td>The Tree class exports all the trees in the exported scene. The trees are all independent objects and may therefore be exported in one run.</td>
</tr>
<tr>
<td>CityGML Roads Export</td>
<td>All roads are exported with the help of the road export class. The roads are represented with polygons like any other geometry in the export.</td>
</tr>
<tr>
<td>CityGML Raster Export</td>
<td>The class for raster export is different from the other classes in the sense that it does not handle any kind of geometries. The class is called from the main servlet class, which passes the coordinates for the area to the raster class. The raster class creates a URL to an in-house web service based on the coordinates for the area. The web service produces an orthophoto in JPEG format, which is stored at the server and included in the final Zip-file.</td>
</tr>
</tbody>
</table>

**Table 4.1.** Components of the CityGML export plugin
4.1 Data Pre-Processing

Before any export takes place, the program must determine which geometry components to export and which area that is to be exported. The easiest way to select an area is to let the user draw a fence on the map in the client side of the application. The client sends a request to the server for a CityGML file over the selected area, independently of which theme the user has activated for the moment. This is a cleaner, better solution than to have the client upload every single geometry component in the view, which is also an alternative. Since there are many different layers and themes in the database, the program does not select every geometry component in the area. Instead it fetches only the relevant geometry components by using lists of wanted and unwanted object types, each list differentiated by an object type called DP_OTYPE, a subtype, DP_SUBTYPE and a component type, DP_CTYPE. In ExportObjectsToCityGMLHelper, lists of wanted and unwanted geometry types are prepared. The reason to create two lists is that the server would otherwise base the selection on zoom level and selected map theme.

```java
Set<GeometryComponentType> wanted =
    new HashSet<GeometryComponentType>();
wanted.add (GeometryComponentType.get (451001, 10, 210105));
```

Two lists of geometry component types are created, wanted and unwanted, which makes it possible to include and exclude certain object types. Objects that pass through these filters are then fetched from the server and loaded into the RAM. When the components have been fetched they are sorted into a number of smaller lists, one for every type of city object. In total there are four lists, one for height information, one for trees, one for roads and one for building geometries.

The lists are processed individually and the corresponding CityGML is written to file. The buildings have to be processed individually to retain the semantic information and also to include addresses.

When the buildings have been processed, the program processes height information, roads and trees. Finally, a ground texture is generated for the area.

4.2 CityGML Viewer

There are two CityGML-viewers available on the CityGML web site (as of January, 2009), LandXPloerer and Aristoteles, both free of charge.

4.2.1 Custom CityGML Viewer

In this research, a CityGML viewer has been developed to visualize the exported files. The viewer was written in C# and uses the DirectX SDK for high-speed 3D computation.
For better performance, the linestrings are read from the file, converted into polygons and put together in a mesh, a collection of objects. This improves performance dramatically since the draw-method only has to be called once every frame instead of once per polygon.

### 4.2.2 XML Parsing

The XML is parsed using C#’s class XmlTextReader. The reader is instantiated with the selected XML file as follows:

```csharp
XmlTextReader reader = new XmlTextReader(xmlFile);
```

The process of reading the file is relatively straightforward. Since this parser is specifically written for parsing CityGML and all objects are previously known, it is possible to predefine all elements in the code.

```csharp
while (reader.Read())
{
    if (reader.Name == "CityObjectMember")
        readCityObjectMember(reader);
}
```

The method readCityObjectMember is called with the instance of the reader passed on to be able to continue reading at the same line as the while loop above. The method includes a while loop similar to the loop above, only that it includes a few more paths. For example, when the reader encounters a Building, it calls the method readBuilding. Paths are available for each object to be found within the CityObjectMember. Addresses, attributes and positions are all read by different methods. For example, the building parser method calls the method readAddress and lists of positions, vertices, are read by the method readPosList.

The conversion of linestrings into polygon introduces a new set of problems. For simple, concave polygons, a simple triangle fan may be used for drawing the polygon. When using the triangle fan method on convex polygons, the output will include lines that should not have been drawn.
4.2. CITYGML VIEWER

Figure 4.2. Convex polygon drawn with the triangle fan method. Polygon faces are all drawn from vertex 0.

Figure 4.3. When drawing a concave polygon with the triangle fan method, faces will be drawn incorrectly.
Chapter 5

Results

The development of the export function has been relatively spared from any major problems. The questions that proved harder to solve were all directly related to the CityGML standard, regarding syntax and appearance. Since CityGML is still a relatively uncommon format there is not much more information to use for CityGML syntax than the information available at http://www.citygml.org.

The existing viewers, LandXplorer and Aristoteles, although quite sophisticated, sometimes produce different results when viewing the very same CityGML file. This has been a good way to identify errors in the exported CityGML. While one viewer might let slightly incorrect syntax through, the other viewer may not render the scene at all, which leads to the conclusion that something has to be corrected in the export function.
5.1 Export Results

5.1.1 Building export

The building geometry is represented as a list of geometries, linear rings, within a building XML-tag.

```xml
<Building>
  <lod2MultiSurface>
    <gml: MultiSurface>
      <gml: surfaceMember>
        <gml:Polygon>
          <gml: exterior>
            <gml: LinearRing>
              <gml: posList srsDimension="3">
                99940.637 79762.0080000005 43.427
                99940.637 79762.0080000005 23.708
                99940.479 79761.6839999994 23.708
                99940.479 79761.6839999994 43.427
                99940.637 79762.0080000005 43.427
              </gml: posList>
            </gml: LinearRing>
          </gml: exterior>
        </gml:Polygon>
        ... (Tags omitted to save space)
      </gml: surfaceMember>
    </gml: MultiSurface>
  </lod2MultiSurface>
</Building>
```

The export function opens a building tag, iterates over the geometry components and writes out the corresponding CityGML for each and every component. When a new building is reached, the program closes the open building tag and starts over again.

5.1.2 Addresses

Almost every building has at least one address. Since the export mainly treats the city of Stockholm, some buildings are large and may contain 10 or more addresses. At Digpro, there is a database containing addresses and corresponding X and Y coordinates for the Stockholm area. One way to treat the problem of finding addresses would be to query the database for every single geometry component to find out if there are any addresses within this area.
5.1. EXPORT RESULTS

Another method is to query the server for every address within the whole area, about 450,000 addresses. The addresses are put in a priority tree, a variant of an R-tree, which allows the program to quickly determine whether an address is located within the bounds of the geometry component. Since this operation does all the processing within the RAM of the server computer instead of querying an external database for every single address, it is a more efficient solution than to query the database for one address at a time.

For this research, an implementation called PRTree will be used. (Olofsson, 2008)

All the addresses are loaded into the list addressList, which is loaded into the tree.

```java
tree.load(addressList);
```

The program may now query the tree to determine whether an address is located within the bounds of the geometry component, represented as rectangle. When the tree finds a matching address, it is returned as an AddressInfo object.

```java
tree.find(rectangle.getMinY(), rectangle.getMinX(), rectangle.getMaxY(), rectangle.getMaxX()));
```

The addresses are added to a Set to avoid duplicates. Before the building tag is closed, the program iterates through the Set to export the corresponding CityGML.

5.1.3 Addresses in the CityGML Viewer

The CityGML Viewer supports georeferenced addresses. The viewer provides the user with a search form, where addresses may be selected. When double-clicked, the program points the camera to the address, which is also marked by a pin. This is useful for validation of CityGML datasets, although it does require previous knowledge of the area.
Figure 5.1. The search form
5.1. EXPORT RESULTS

Figure 5.2. Address marked by a pin
5.1.4 Trees

Trees are represented as points with attributes representing the tree height. The points are located in two different layers depending on tree type, deciduous or coniferous. The CityGML encoding standard suggests that coniferous trees are represented as PlantClassType 1060 and that deciduous trees are represented as 1070. (Czerwinski et al., 2008)

```java
if (tree.getSubtype() == 10) treeClass = "1060";
if (tree.getSubtype() == 20) treeClass = "1070";
```

In DP/Spatial, subtype 10 represents coniferous trees and subtype 20 deciduous trees. The height information for the tree is available in the geometry component which allows for easy export of the CityGML for the trees with height information and X, Y and Z coordinates.

5.1.5 Road export

Roads are exported as simple geometries, without any metadata. It would have been desirable to include street names and place names, but this was not possible to achieve due to limited address data in the database.

Roads are represented as 3-dimensional geometry components and add a very nice touch to the final result. The roads include height information, which smoothens the roads in the city core, in comparison to the DTM, which is not very smooth when viewed from closer distances.

5.1.6 Digital Terrain Model

The height information is exported in the form of triangle patches. Height information is stored in the database as square polygons with a grid width of 20m. The polygons are divided into triangles according to figure 5.3

![Figure 5.3. The squares are divided into triangles](image)

The generated CityGML describes the terrain as follows:

```xml
<dem:tin>
  <gml:TriangulatedSurface>
    ...
  </gml:TriangulatedSurface>
</dem:tin>
```
5.1. EXPORT RESULTS

As seen in figure 5.4, this method produces a relatively smooth surface. Interpolation in the visualization software can increase the smoothness even more.

Figure 5.4. DTM displayed in the CityGML Viewer
5.1.7 Orthophoto

It is desirable to include an orthophoto in the export to CityGML. It does not provide the user with any additional semantic information, but it dramatically increases the visualization effect. For optimal visualization performance, the orthophotos should be divided into small parts, which allows the viewer to easily read the textures when needed. The textures may later be removed from the video memory when the user navigates to another location. An orthophoto of 10000 * 10000 pixels will be included in the export.

5.1.8 Implementation

The CityGML export tool is implemented in DP/Spatial as a selectable menu item. When called, the user is presented with a progress meter to indicate how many objects that are to be exported. When the export finishes, the user is presented with a ‘Save as..’-dialog.

![Export progress meter](image)

**Figure 5.5.** Export progress meter
The database was not built with CityGML export in mind. This presented a series of problems while coding the CityGML export plugin.

Ideally, the database would contain geometry components sorted by building in a clear and intuitive way, with all metadata easily accessible. In this case, the addresses had to be fetched from a different database, which worked relatively well but slowed down the overall export process.

Parcel numbers were supposed to be included in the export, but had to be left out. There is no direct relation between geometry components and parcel areas in the database, which required that the parcels were identified through Oracle Spatial commands. This did not work very well despite numerous efforts, which was a big disappointment. Visually, it is easy to identify which parcel a building belongs to. Computationally, it presents a greater challenge. Many times building geometries and parcel borders overlap, which causes Oracle Spatial to return more than one parcel number when queried which parcel numbers that intersect with a certain geometry. In other cases, when a building geometry is completely contained within a parcel area, same query returns nothing since no actual interaction takes place. The parcel numbers therefore had to be left out of the export. One solution to this problem would be to write a script to run against the database to create a table of all the parcel numbers and connect them to the geometry components contained in the parcel area.

The export of orthophotos works flawlessly, but does take a certain amount of time since the areas are rather big. It would have been desirable to include the ground texture as a collection of smaller orthophotos, but for visualization purposes this presents a bigger challenge. Polygons should ideally only be draped by one single texture, and since the CityGML viewer drapes the buildings with the orthophoto, this would require that the texture patches overlap to a certain extent to avoid problems with polygons along the borders of each texture patch. For general CityGML export however, where the buildings are to be draped by unique textures, it would be a better solution to identify logical lines for clipping the orthophoto. In this case, there were no building textures available which made it a better decision to drape the whole scene with one single orthophoto. The exported DTM has a resolution of 20m; a higher resolution DTM would have produced an even smoother-looking surface, but the resulting file would quickly become too complex to display without breaking it up into smaller pieces.

In some cases, the triangles which represent the ground surface look smooth and seem to reflect the characteristics of the area. Some city objects, for example roads and other flat surfaces are a bit too bumpy to be used in professional visualization.
Figure 5.6. A CityGML file containing approximately 90,000 polygons displayed in the CityGML viewer.
5.3 CityGML Viewer

5.3.1 Performance

When visualizing 3-dimensional data it is important to know beforehand what the specifications are. For example, if the user is to be able to select different objects, move them around or highlight them, these demands may significantly reduce performance.

For example, if the user is to be able to click and select an object, the process of identifying the object is much simpler if all polygons are drawn individually. However, in this case, it is desirable to access attributes on a per-building level. This is achieved by letting each polygon belong to a group of polygons, which together form an object or a building.

The 2-dimensional mouse cursor position is transformed into a 3-dimensional ray as follows:

```csharp
Vector3 mouseVector0 = new Vector3(x, y, 0);
Vector3 mouseVector1 = new Vector3(x, y, 1);
Vector3 vRayDir = Vector3.Normalize(rEnd - rOrigin);
```

This solution works well, but does require that the program stores every geometry in a separate polygon. To allow for polygon picking on a collection of objects, a mesh, a separate list of information for every object must be kept alongside the mesh. To highlight an object in a mesh, the polygon information has to be removed from the vertex buffer and the display properties changed for the polygon faces kept in this group.

5.3.2 Meshes

One of the biggest optimization improvements possible when writing a 3D viewer is to put all vertices, edges and faces in one common collection, a mesh.
Since many vertices are used in more than one triangle, it is possible to create a table over all the vertices for faster lookup. When rendering a mesh, most 3d graphics cards will automatically determine whether a polygon is within or outside the viewing threshold. If a polygon is determined to be outside of the view, the polygon is excluded from the calculation. Also, if polygons are hidden behind other polygons, the polygon is automatically excluded from the processing which speeds up the rendering.

In the CityGML-viewer, the mesh containing the ground surface is draped with a ground texture, which gives a nice effect. To do this, every vertex must be defined with U and V coordinates which define the position of the vertex relative to the texture position. In this case, only a few lines of code are needed. The process is also known as UV mapping.

\[ U = \frac{(v3.Y - g.ymin)}{g.width} \]
\[ V = 1.0 - \frac{(v3.X - g.xmin)}{g.width} \]

UV coordinates are in many ways similar to a normal map projection in the sense that they show the relation between coordinates in 3D space and in UV space, which is 2-dimensional and ranges from 0.0 to 1.0 in both directions. (Kumke et al., 2007)
5.3. CITYGML VIEWER

The mesh type used in this CityGML viewer is of the type Face-Vertex. A Face-Vertex mesh stores information about vertex positions and which faces use which vertices. (Smith, C., 2006)

<table>
<thead>
<tr>
<th>Faces</th>
</tr>
</thead>
<tbody>
<tr>
<td>f0 v0 v4 v5</td>
</tr>
<tr>
<td>f1 v0 v5 v1</td>
</tr>
<tr>
<td>f2 v1 v5 v6</td>
</tr>
<tr>
<td>f3 v1 v6 v2</td>
</tr>
<tr>
<td>f4 v2 v6 v7</td>
</tr>
<tr>
<td>f5 v2 v7 v3</td>
</tr>
<tr>
<td>f6 v2 v7 v3</td>
</tr>
<tr>
<td>f7 v3 v4 v0</td>
</tr>
<tr>
<td>f8 v8 v5 v4</td>
</tr>
<tr>
<td>f9 v8 v6 v5</td>
</tr>
<tr>
<td>f10 v8 v7 v6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>v0 0,0,0</td>
</tr>
<tr>
<td>v1 1,0,0</td>
</tr>
<tr>
<td>v2 1,1,0</td>
</tr>
<tr>
<td>v3 0,1,0</td>
</tr>
<tr>
<td>v4 0,0,1</td>
</tr>
<tr>
<td>v5 1,0,1</td>
</tr>
<tr>
<td>v6 1,1,1</td>
</tr>
<tr>
<td>v7 0,1,1</td>
</tr>
<tr>
<td>v8 -5,-5,1</td>
</tr>
</tbody>
</table>

\[ f0 f1 \]

\[ f2 f3 f1 \]

\[ f4 f5 f3 \]

\[ f6 f7 f5 \]

\[ f8 f9 f11 f8 \]

\[ f0 f2 f9 f8 \]

\[ f1 f2 f9 f8 \]

\[ f3 f4 f10 f9 \]

\[ f4 f5 f6 f11 f10 \]

\[ f8 f9 f10 f11 \]

This way, the graphics hardware may easily find the vertex positions in the RAM and the computation time is reduced.

Since many vertices are shared by several polygons and therefore located at the same position, the mesh allows for merging these adjacent vertices into one in order to save storage space and speed up computation. Using C# and Direct3D, the mesh creation is done in as follows:

- Polygons are read from the CityGML file and stored in an instance of the class Polygon, written especially for this viewer. The polygon class contains information about the vertex positions. The program also counts the total number of vertices and the number of faces for mesh instantiation.
- An instance of the Direct3D class Mesh is created. The total number of vertices and faces are used for the instantiation.
- All the vertices are stored in an array, in this case named vertexBuffer.
• The indices for each face are also stored in an array. The array length is three times the number of faces, to store the index number for each face vertex. The index array is named indices.

• The indices and the vertices are loaded into the mesh.

• By using the Direct3D functions for optimization, the mesh is optimized for faster processing.

• The mesh is finally loaded into an arraylist named world.

The mesh may now be drawn onto the screen by placing the following code in the method OnPaint, which is called once every frame:

```csharp
foreach (Mesh mesh in world)
{
    for (int i = 0; i < numSubsets; i++)
    {
        device.SetTexture(0, g.texture);
        mesh.DrawSubset(1);
    }
}
```
5.4. Visualization results

When visualizing the data in the CityGML viewer, some characteristics of the data proved complicated to visualize correctly.

5.4.1 Surface Normals

Surface normals are required for correct lighting and shading of surfaces when viewed in 3D. Linestrings from Oracle Spatial are defined counter-clockwise, meaning that polygons viewed from the front will have all vertices in a counter-clockwise order. Oracle (2003)

![Relation between surface normal and vertex order](image)

**Figure 5.9.** Relation between surface normal and vertex order

In theory, this practice works well, but when handling data sets, polygons are often reversed which results in that the graphics hardware will make the wrong-turned polygons invisible from one side. In this research, there were three solutions to this problem, none of which is both clean and fast.
CHAPTER 5. RESULTS

The first solution is to render two identical polygons in the same location by reversing the vertices, resulting in that the polygon is visible from both sides. This doubles the calculation time since every polygon is drawn twice.

The second solution is to change a parameter in the graphics hardware called Culling. Setting the culling flag to clockwise will display all polygons with vertices in clockwise order. Setting it to counter-clockwise will do the reverse. It is also possible to set the culling to none, which displays polygons from both sides. This solution is not very good in the sense that surface normals may point in the wrong direction; which causes the lighting to display incorrectly at certain points.

The third solution is to manually or computationally reverse the vertex order of polygons that are facing in the wrong direction. A very simple algorithm for polygons belonging to a building would be to find the center point of the building and determine the vertex order from that point and verify that all polygons are facing outwards. This however, would not work for ‘dount buildings’ such as the Royal Castle in Stockholm or buildings with a concave footprint, but would work flawlessly for simple buildings. Doing this manually for 3D models would in most cases work well, but would also be very time consuming for larger models.

Figure 5.10. Culling set to Clockwise, causing some geometry to not display correctly
5.4. VISUALIZATION RESULTS

Figure 5.11. Culling set to none, which displays the polygon patches correctly. The surface normals may however point in the wrong direction, causing lighting to not display correctly.
Chapter 6

Conclusions and Future Research

6.1 Conclusions

The advantages of using CityGML are many; it is possible to include a huge amount of semantic information to present the user with information such as ‘room on a floor of a building situated on a street’ etc.

The access to numerous standardized classes and object type codes makes it easy to exchange data between different storage locations without losing information. In fact, the possibility to include large quantities of semantic information is what really makes CityGML such an interesting format.

Performance- and storage-wise, CityGML is far from an ideal format, but the XML standard makes development easy and a simple CityGML parser and viewer may be written in a short period of time, which is not the case for a binary format.

In general, the XML standard is supposed to be easily read by both computers and humans, but an exported CityGML file exceeding only a few kilobytes is hard to evaluate without using a parser. Slightly different kinds of annotation are used in different datasets, which makes it a relatively time-consuming task to write a parser able to handle any kind of dataset. A large amount of different object types are available within the CityGML specification, which requires the parser to be highly flexible and thoroughly tested.

The custom CityGML Viewer proved to be a very successful method to verify the exported data. By using DirectX, large textures (10000 * 10000 pixels) could be displayed at high frame rates. To write a CityGML viewer capable of viewing all possible CityGML features is however a huge task to undertake. This viewer only treats the features exported from the Baggis database, but can be considered to be a successful viewer from a research perspective.

The possibility to include 3-dimensional models and reuse the same model in
more than one location makes it possible to model relatively detailed city models in a short amount of time. For example, traffic lights, trees and other kinds of repeated city objects add a lot of character to a scene. In many cities, traffic lights and lampposts are of the same model, which makes it possible to reuse objects without simplifying or generalizing the reality too much. In the case of this research, trees are represented as coniferous or deciduous in the database, combined with a tree height. The possibility to include random models of the tree type in question adds a lot of eye-candy to the scene.

6.2 Should the database be constructed to support CityGML directly?

To include building textures and semantic information directly in the Oracle Spatial database would make it possible to export an even more detailed and realistic city model, which of course is desirable. The most practical relation to add to the Baggis database would be the parcel numbers, which have proven to be a great challenge to export correctly. Parcel numbers would make it possible to identify buildings when mixing datasets, and would also allow for easy removal of buildings when a version of the building in a higher LoD is available. However, it would be very interesting to build a database directly adapted to the CityGML standard with support for the interior of the buildings with elements such as rooms, stairs, etc.

The possibilities are endless and the problem would be to know where to draw the line regarding what should be included and what should not be included. The possibility to add new information to the database and simultaneously visualize buildings of different LoD’s makes it relatively easy to create material for visualization, such as removal and replacement of buildings for building project plans. 3DCityDBv2 is exactly what a relatively new standard as CityGML needs. To implement a CityGML database from scratch is not a small task considering everything that has to be taken into account. The 3DCityDBv2 SQL scripts for setting up databases are a huge help and may even put CityGML on top of the list of formats to choose from. Another great benefit is compatibility with future versions of CityGML, instead of rewriting all the export functions, it would be possible to simply upgrade the tools to the current version.

In fact, a huge disappointment with this research is that 3DCityDBv2 and CityGML4j was not available at the time of the research. The export plugin is not as well-functioning as hoped and it would have been interesting to study 3DCityDBv2 and CityGML4j.
6.3 Future research

The goal of this research was to investigate how to export the existing data to the CityGML format. While this goal has been reached, it would be interesting to export the existing data directly to the 3DCityDBv2 database. From the database, it would be possible to easily correctly written CityGML.

This however introduces a new problem; when exporting the data set into 3DCityDBv2, it is important to keep both databases updated. Since all the updating takes place in the Baggis database, the synchronization of the two databases would have to be closely monitored to avoid data loss and duplicate data.

Just by looking at the results from this research, it is easy to draw the conclusion that some of the difficulties with the data export are related to the organization of existing data, for example, database tables for parcel numbers were not directly available. By developing and using a best practices manual when gathering and storing data, many of the problems regarding identification of buildings, addresses, different LoD’s and similar would disappear.
Bibliography


T. Kolbe. 2008. Representing and Exchanging 3D City Models with CityGML.

T. Kolbe, G. König, C. Nagel, and A. Stadler. 2008. 3D City Database v2.0 Documentation.


H. Kumke, L. Hoegner, L. Meng, and U. Stilla. 2007. Visualization of Building Models and Factual Data Integrated by CityGML.


Important parts of the CityGML viewer

The CityGML viewer contains a number of different functions, called to parse, prepare and visualize the contents of the CityGML file. The main functions are listed below.

ReadXML

ReadXML is part of the class cityGMLReader, a class used to read and parse CityGML data. The reader is called with pointers to the chosen XML file and also to the different meshes used for visualization, such as world, which contains all the building geometries, and ground, which contains the terrain model. ReadXML reads the CityGML file from the hard drive and processes all the geometries in the file. The geometries are converted into different collections of objects, meshes, to speed up the processing time when visualizing.

OnPaint

OnPaint is located in the main class of the viewer. OnPaint is a so called overridden method, which means that it is called every time Windows repaints the window, which occurs once for every frame.

OnPaint first calls the method device.BeginScene(), a Direct3D call to prepare the graphics hardware for drawing. The method then processes all the meshes and other objects to be painted onto the screen.

When all the objects have been drawn, the OnPaint calls for device.EndScene() and device.Present() to show the rendered frame in the window.


2007


2008


2009


09-005 **Erik Olsson.** Exporting 3D Geoinformation from Baggis Database to CityGML. Supervisors: Peter Axelsson and Yifang Ban. April 2009.