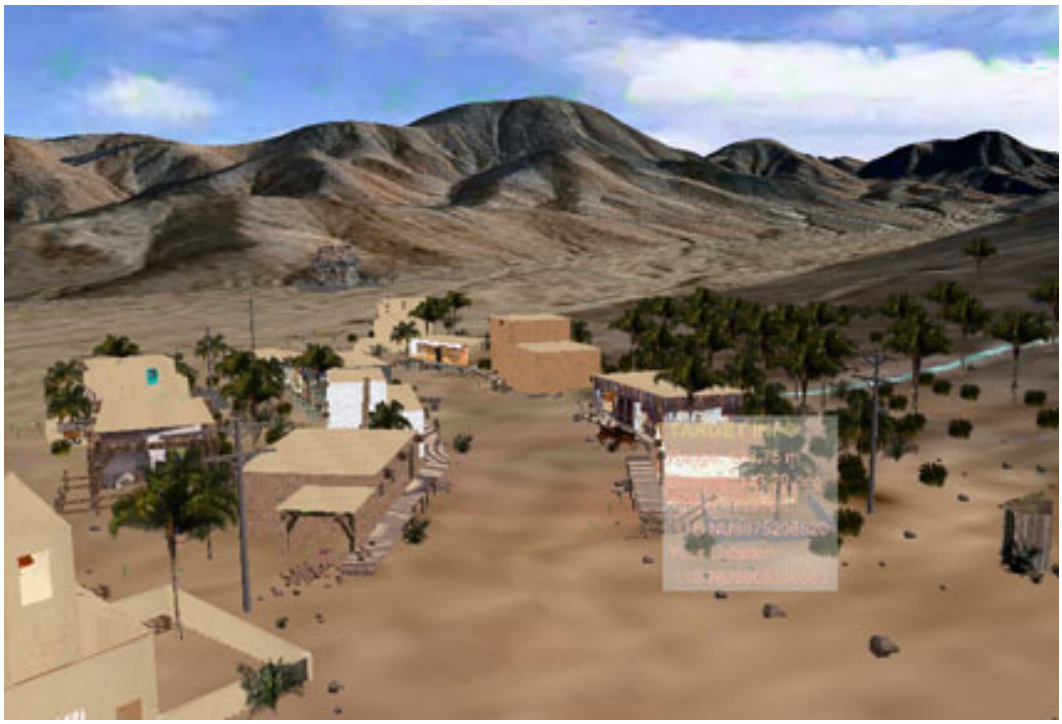
Centre for Geo-Information

Thesis Report GIRS-2005-35

USING OPEN STANDARDS IN 3D VISUALISATION OF GEO-DATA

Using GML, X3D and XML in GIS

Dolf Andringa



18-12-2005

WAGENINGEN UNIVERSITY
WAGENINGEN UR

Table of Contents

Figures

Acknowledgements

First of all I would like to thank all people involved in Open Source projects. Especially the people from the Xj3D project, Apache Xalan project, Apache Xerces project and the people from the Web3D and Open Geospatial Consortia often spend their spare time developing Open Standards and Open Source Software which other like me can use for free for their own projects. Without them, my life would have looked very different probably. In particular I would like to thank Ron Lake from the Open Geospatial Consortium and Allan Hudson from the Web3D Consortium and the Xj3D workgroup for their help. Ron Lake has provided the GML datasets, together with VRML visualisations of these datasets for me to use in my prototype. Allan Hudson has shown a lot of patience in answering my many questions about X3D.
I would also like to thank my supervisors and examiners Ron van Lammeren, John Stuiver and Jandirk Bulens in helping me to do this thesis. This project and my previous education were far from standard projects at our chair group, but they saw my enthusiasm and possibilities of the subject. Last but not least I would like to thank my girlfriend for reading through these 40 pages of boring technical text and Pjotr Prins, who has kicked me to work when things didn't go as they should.

Summary

Currently GIS are mainly used by GIS-specialists. The increase in computer speed and the possibilities of the internet for data transportation and online applications makes it possible for non-GIS-specialists to use GIS. The only remaining restrictions are software restrictions. Dedicated webbased 3D GIS would reduce the large cost and knowledge requirements of many presently used GIS software. In order to enable the rapid, interoperable and affordable development of these dedicated webbased 3D GIS, an open source development environment with several modules providing the necessary functionality and implementing Open Standards (OS), should be developed. This thesis report tries to answer three questions regarding this topic: What basic components should be present in a webbased 3D GIS? Which OS and Open Source Software (OSS) exist to be used in such a web based 3D GIS development environment? Is the 3D visualisation of a 3D dataset possible with such a framework, using existing OS and OSS?
A webbased 3D GIS should mainly provide functionality for the visualisation of geo-data and should provide an interface to analysis functionality like querying the geo-data. Data management issues are limited to being able to use data from different locations and in different formats. This could be achieved by interfacing with WFS, which provides geo-data from different sources in GML format.
The 3D visualisation functionality that should be provided by a webbased 3D GIS can be divided into six main groups: realism, movement, orientation, navigation, explanation and coordinate system. The functionality of these six groups should be natively provided by or possible to implement in the technology used for the visualisation.
GML and X3D are two OS that are very useful for implementing a webbased 3D GIS. They are both based on the XML OS, enabling the use of the XML family standards and many existing XML tools to convert XML based file formats into each other.
X3D provides almost all of the functionality that is needed in a webbased 3D GIS and its internal scripting functionality, the interaction features and Scene Access Interface make it possible to implement the rest. The latter three also allow the real-time manipulation of X3D scenes. Different viewers exist for X3D files, both open source and proprietary programs.
The geospatial component of X3D allows the use of geo-referenced coordinates in different spatial reference systems in X3D scenes. Some work still has to be done in the implementation of this geospatial component in the different viewers though.
The XML based framework for the storage and transport of geo-data: GML, is very flexible and feature rich. It requires application specific implementations of the standard in order to be used. This makes it possible to use GML for many different data models.
A prototype visualisation module was developed that provides basic navigation functionality through a 3D scene. The module implemented the GML and X3D specifications and existing XML tools, specifications and API's to achieve this. This has proven the usefulness of the X3D and GML specifications for webbased 3D GIS.
Several issues were encountered with the development, which were easily solved or will be solved in the near future. The main problem was the lack of full support for the geospatial profile of X3D in X3D viewers. This will be solved in the near future though.
Interesting subjects for further development are the representation of geographic, topologic and thematic properties in the 3D scene, and the addition of modelling functionality.

1. Introduction

The spread of geo-information

The usage of geo-information is rapidly expanding from use by just geo-information specialists to other fields (Altmaier and Kolbe, 2003). Because of the price of soft- and hardware that was needed to manipulate and interpret geo-information, the corresponding knowledge that was needed to do this and the difficulties in distributing geo-information, only geo-information specialists were able to use geo-information regularly. Since hardware is rapidly becoming faster and cheaper, and the internet can be used for the transport of information around the world, software restrictions are the only limits in the widespread use of geo-information.
Until recently most geographic information systems (GIS's) were monolithic, standalone, proprietary systems (Anderson and Moreno-Sanchez, 2003). Programs like the ESRI's ArcGIS family are large, expensive programs that are meant for use by geographic information specialists with the expertise and money to buy and use the systems. Individual policy makers in governmental organisation for example, have neither the expertise and money for such systems nor the needs for all the functionality given by such programs (Peng and Zhang, 2004).
Geo-information is needed more and more in fields of work outside the geo-information science. Policy makers, disaster management organisations, scientists of all walks of specialities, etc. all have need for different kinds of geographic information (Altmaier and Kolbe, 2003). Generally, these people are not geo-information specialists, and therefore rely on others to provide them with the geo-information they need for their work. While this is a good way to divide expertise, for simpler geographic queries and operations or smaller institutions, it would be more efficient if they could access the information themselves. But for these people, the present expensive, knowledge intensive, monolithic systems are not fit for the job.
The growth of the number of Internet connections and speed of the Internet is making it possible to develop smaller, practical, web-based applications that provide only the necessary information and functionality to the end-user without the above mentioned problems (Peng and Zhang, 2004;Boucelma *et al.*, 2002;Anderson and Moreno-Sanchez, 2003). These dedicated GIS programs may provide a solution in the spread of the use of geo-information outside the field of the geo-information specialists and scientists.

2D or 3D?

Traditionally, geo-information has been recorded and presented 2-dimensionally, from the first map drawings by primitive man to satellite imagery. The description and presentation of geo-information in 3D just started some decennia ago. The arrival of fast computers that made it easier to calculate complex transformations and the information potential it provides sparked GIS scientists' interest.
3D geo-information has a higher information potential than 2D geo-information (Altmaier and Kolbe, 2003;Wang, 2005;Kraak, 2003). Because of the third dimension, an extra variable can be included in the visualisation. 3D geo-information seems a lot easier to interpret, especially for non geo-information specialists. All our thinking is in 3D, we see backgrounds behind text on a screen and windows on top of each other, while the screen only has two dimensions (Carman and Welch, 1992;URL: The Mind Project, 2003). 3D visualisation also allows us to display thematic information in a 3D

presentation of it's real geographic locations, making analysis and interpretation easier (Altmaier and Kolbe, 2003;Wang, 2005;Kraak, 2003).
Because of its relative youth and the 2D tradition in geo-information, 3D geo-information has not been widely used, especially not in web-based GIS. The cases it has been used are often specialist proprietary systems. These systems do not provide any facilities to build new web-based dedicated GIS on. To be able to build these 3D web-based dedicated GIS easily, we need some basic interoperable components that provide basic functionalities upon which to build web-based GIS.

Interoperability in relation to Open Standards and Open Source Software

Interoperability has been a problem in GIS software since most GIS software has traditionally been proprietary monolithic software. Vendors used to develop every part of their package themselves from the data storage to the visualisation. In combination with the proprietary nature, which means the source code of the software is kept secret for commercial reasons this created a huge interoperability problem. Writing a software package that is compatible with others was very hard.
Especially file formats are a big problem. Many of the proprietary packages use their own way of storing geo-information. Different (Object) Relational Database Management Systems ((O)RDBMS's) and file formats like respectively Oracle Spatial and the Arcview shapefiles are used by different software vendors. In order to use the information stored in one system in another, one has to convert the information, which is a huge task if it has to be done regularly.

Software vendors have realized this is a big problem and therefore started to provide interfaces to their, mostly 2D, software that can be used in other programs/algorithms. Although this only allows addition to the software on places the vendors allow, it is a big improvement for 2D GIS interoperability. For the same reason, vendors are also starting to support Open Specifications (OS).

The use of OS and Open Software Standards (OSS) can greatly improve interoperability. OS are specifications on how to perform a specific task that are freely available to the public. Examples are HTML, which is a specification on how to store WebPages, and the Web Feature Service (WFS), which is a specification on how to make geo-information from different sources available on the Internet in a transparent way.

OS are formulated by consortia of important organisations and companies in the field of work the specification is intended for. Several consortia are important for the GIS: The World Wide Web Consortium (W3C), responsible for instance for XML (Specification: Bray *et al.*, 2004) and HTML (Specification: Raggett *et al.*, 1999), the Open Geospatial Consortium (OGC), responsible for example for GML (Specification: Cox *et al.*, 2003), WFS (Specification: Vretanos, 2005) and WMS (Specification: Beaujardiere, 2004), and the Web3D Consortium (W3DC), responsible for X3D and VRML (Specification: Web3D Consortium, 2005).

OS allow software developers to make programs that are interoperable through the use of the OS (Anderson and Moreno-Sanchez, 2003). Examples are the HTML OS that allows many programs, techniques and people to use and develop web pages, and the GML OS that allows different programs to make use of the same geo-information datasets. The use of OS in GIS enables people to develop software that is able to communicate with or use other software so both packages can complement each other, instead of having to develop everything within one program. This greatly reduces

development costs and improves interoperability (Anderson and Moreno-Sanchez, 2003).
Open Source Software (OSS) expands the possibilities of OS further. Aside from different programs being interoperable, people are able to contribute to the development of a program and are able to integrate functionality of one program more tightly into another. Software developers can therefore use existing software that already works well and can focus on developing the more specialized parts that do not exist yet (Anderson and Moreno-Sanchez, 2003).

The use of OSS and OS in the 2D GIS world is expanding, even to the proprietary software of the big software vendors, but in the 3D GIS world, little software exists that complies to OS and enables other software packages to build upon them.

Concrete examples of OS useful for web-based 3D GIS are GML, VRML, GeoVRML, X3D and WFS. GML, VRML, GeoVRML and X3D will be discussed more into detail in this report. To sum up their usefulness in this project though: GML is a XML based framework to store geo-data; VRML is a specification for describing 3D scenes and visualising and interacting with them on the internet; GeoVRML is a geospatial extension on VRML to bring GIS functionality to VRML; X3D is the successor of VRML, incorporating the GeoVRML functionality within the specification and also specifying an XML encoding; WFS is a specification for making geo-data from various sources and formats available on the internet transparently. This last specification will only be discussed lightly in this thesis report. It is important nonetheless because it uses GML for geo-data transport and therefore allows web-based GIS that support GML, to use geo-data from many different sources and formats.
This combination of OS is promising in the development of web-based 3D GIS's.

Aim of the thesis project

Because the use of OS and OSS in 3D web-based GIS's is still limited, it is necessary to promote the use of OS to enable the development of interoperable 3D web-based GIS software. To facilitate this, we need to develop a GIS development environment consisting of a set of modules that provide a basic functionality of geo-information reading, visualisation and modification for the web, independent of a specific application or GUI (Graphical User Interface). Such a system should for instance provide the possibility to visualise geo-data in a web-browser, make it possible to visualise effects model or user interaction, etc. Such a system should be developed in a modular way. This means that specific functionalities are developed as separate modules, apart from other functionalities. The advantage of this is that the modification or substitution of one module does not affect other modules (Parnas *et al.*, 1984). One can think for example about a module providing visualisation functions, a module providing spatial modelling functions, etc. Such a development environment could be used by a software developer to be implemented in a specialized GIS application. The software developer only has to develop the specific GUI for the program, and maybe add extra functionality required for the application without having to develop a complete visualisation and modelling program.

This thesis project is aimed at defining the basic parts of a GIS system, and determining which parts are needed in the web-based 3D GIS development environment. The visualisation module of the development environment will be investigated further. What functionalities should it provide and what OS and OSS exist to be used in this module? A prototype visualisation module will then be developed as a test case. A set of 3D GML files will be visualised in 3D in a 3D viewer that can be

integrated into a standard web browser as a plug-in. Some basic navigation functionality like walking and rotation will have to be possible in the 3D scene.

Research questions

This leads me to the following research questions:

1. What basic components should be present in a web based 3D GIS?
2. Which Open Standards and Open Source Software exist to be used in such a web based 3D GIS development environment?
3. Is the 3D visualisation of a 3D dataset possible with such a framework, using existing OS and OSS?

Report outline

This thesis report is divided in the following parts, in order to answer these questions:
Chapter two describes the functionality provided by a basic GIS, and specifies which parts should be provided by a web-based dedicated GIS. This information was gathered from personal knowledge, discussion with experts and literature.
Chapter three describes the different components of a 3D scene, and how a web-based 3D GIS could be made available to the general public. The main source of information in this chapter is literature.
Chapter four describes several OS and OSS tools that could be used in such a web-based GIS. These tools and specifications were mainly investigated by reading their specifications on the internet, reading books, trying them out and asking questions on mailing lists.
Chapter five describes the visualisation module that was developed in order to test the feasibility of using the OS and software described in chapter four in a visualisation module for a web-based GIS as described in chapter three. The knowledge for developing the visualisation module came from my own experience, books, the internet and mailing lists.
The report is finished in chapter six by discussing the results of the thesis and making recommendation on further research and projects.

2. A basic GIS

GIS Functionality

Before describing the functionality and issues involved in a web based 3D GIS, it is important to have clear what functionality exists in general in a GIS. A diagram giving a rough summary of functionality often present in GIS is given in figure 2.1. The 3D visualisation part of this diagram will be discussed in more detail in the next chapter. For now I will discuss in the different parts in general.

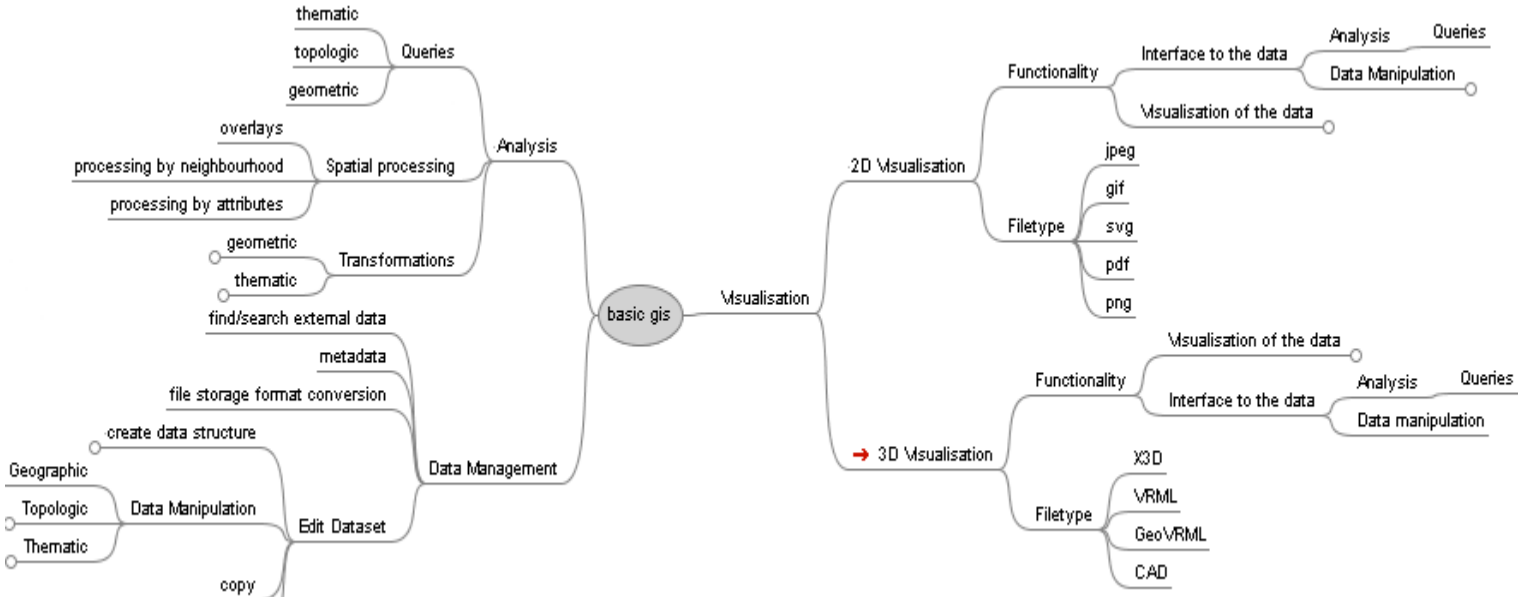


Figure 2.1 The functionality often provided by GIS. A basic GIS is divided in three functional parts: data management, data analysis and data visualization. These parts can be subdivided again into different elements. One could go on endlessly this way. This diagram stops where further specification is not useful anymore at this time.

Data Management

Data management starts with the first collection of the data, either by recording it from real world objects, or by using external data sources. It then involves creating the data structure. The data structure can contain thematic data, geometric data and topologic data. An important choice to be made when creating the data structure is the format in which the data is stored. One possibility is to choose a file based format, which is easily transported to other locations, but which can usually only be used by one person at a time. The other is to choose a database format, with the advantage of storage of the data in one place, where more than one person can access it at the same time. It may be necessary to convert file formats between each other if data from different sources and in a different format should be integrated into the data structure or the data is needed by other applications or users, which need a specific file format.
Another important choice to be made when creating the data structure is the map projection to use for the data. One has to choose a coordinate system with a specific datum and a model for the earth's surface. Data from other sources may be recorded in different map projection types and may therefore have to be converted to the right projection type for our data structure.

Apart from the data itself, metadata should be collected and recorded as well in order for others to interpret the data in the data structure, like the used map projection type. Once the data structure is created, the data can be recorded into the data structure. Data manipulation then involves maintaining the dataset through time by modifying it as necessary.

Analysis

Once the dataset is created, one can start with analysis of the data.
Analysis of the data involves querying the data, processing the data and transforming the data. Querying involves selecting certain objects or properties of objects from the dataset according to certain conditions. You can have thematic queries (select all privately owned houses), topologic queries (select all objects adjacent to this building) and geometric queries (select all objects at coordinates …).
Spatial processing involves combining information to derive new information. This can be done by overlays (overlay a map with soil information with a map with all houses in the same area, so one can determine the soil type each house is built on), processing by neighbourhood and by attributes. Spatial processing can create new data from existing data, which may be stored in the original dataset or in a new dataset, requiring data management again.

Visualisation

Visualisation can provide two functions: actual visualisation of the data and an interface to the data (Kraak, 2003). With pure visualisation of the geo-data, there is a small connection between the geo-data and the user. Only the geo-data the developer of the application chose to visualise directly is visible to the user. The interface to the geo-data gives the user more possibilities to use the geo-data. The user can for instance query the data, transform the data, do spatial processing on the data, and maybe even directly manipulate the data. The interface to the geo-data is therefore an interface to the functionality described in the analysis and data manipulation parts of a basic GIS (figure 2.1).
Further description of the visualisation, especially of 3D visualisation will be done in detail in chapter three. For now the distinction between an interface to the data and a visualisation of the data is enough for the visualisation.

Requirements in a dedicated web based 3D GIS

There are three kinds of GIS users: the data maintainers who setup the database and spatial infrastructure, analysts who use the geo-information to draw conclusions or design alternatives, and the reviewers who evaluate the geo-information. The first group needs the highest level of interaction with the geo-information, while the last group needs the most realistic visualisation (Groetelaers, 2002). The first two groups can be considered GIS specialists, while the last group does not have to be. The last group can contain people like policy makers, citizens, disaster management officials, etc. This group is the group that contains the users of the web based GIS developed with the development environment. The reviewers need immediate access to the geo-data, they should be able to easily orientate themselves within the scene, and should be able to navigate easily (Lammeren van and Hoogerwerf, 2003;Groetelaers, 2002). They should also be able to extract information from the geo-data. They do not need analysis functionality beyond simple queries, and they do not need data manipulation functionality either. These tasks should be carried out by respectively the analysts and data maintainers. It would be very interesting though if models or animations could be

added to the scene, to show the reviewers changes to the 3D scene, triggered by certain actions. These changes could be visualised by making separate scenes or pre-made animations, both made by the analyst, but one could also incorporate models in the scene and provide the reviewer with an interface to these models and view the results of their actions directly in the scene (Lammeren van and Hoogerwerf, 2003). These kinds of functionality make the 3D scene much more complex and are therefore skipped over quickly in this report. The development environment should be built in a proper modular and Open Source way though, to make addition of these and other functionalities possible.
For now the interface to the data should restrict itself to simple queries.

3. The development environment

Client-server model

In general, web applications are deployed in a server-client model, with the server providing the data and some functionality, and the client requesting these data and finishing up the work before displaying the data to the user. On the client side, three levels of clients can be distinguished which follow from the OGC portrayal model (Altmaier and Kolbe, 2003;Specification: Open Geospatial Consortium, 2003) (figure 3.1): thick clients, medium clients and thin clients (figure 3.2, next page). The OGC portrayal model divides the process from selecting the data to displaying the data in four levels: selecting the data, creating the display elements, rendering the display elements and displaying the display elements. The displayed elements are displayed on a computer screen and are therefore always displayed as 2D images. Below the display level, everything can be done in a (virtual) 3D environment. Through this portrayal model, integration of data is possible on different levels. From this model, also the levels of clients follow logically. Thin clients only handle the displaying of the eventual images, like jpeg images or other formats. Medium clients also handle the rendering of the images before displaying them, like a renderer rendering 3D objects from 3D information like a VRML file before displaying them on the screen. A thick client also handles creating the display elements from the data.

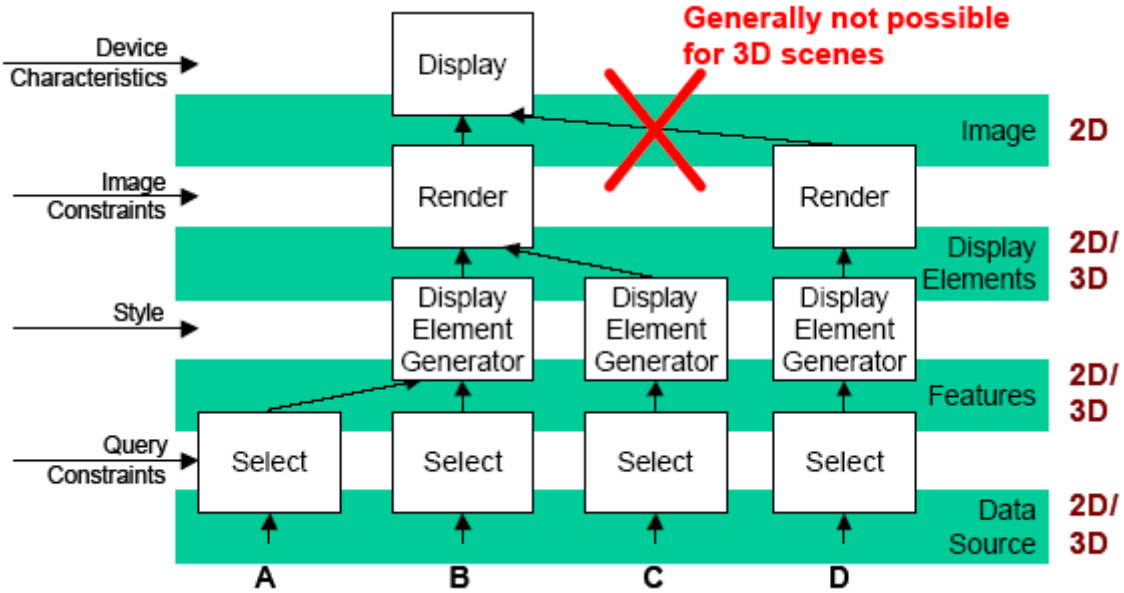


Figure 3.1 The OGC portrayal model. The model illustrates the process from data selection to data visualisation through 4 layers. The model allows for integration of data from different sources at different levels.

A thin client only requires a standard web browser, since it only has to display graphics files. A medium client will need a plug-in to render the 3D elements before displaying them. If a commonly used file format is used for the communication of the display elements, like X3D or VRML, chances are that plug-ins are easily available on the internet. Thick clients will need more software on the client to allow the generation of the display elements from the data. This will generally require stronger client side computers and also involves the installation of extra software on the client. This can be simplified greatly though if a web friendly programming language like java is used. Java applets can be run directly from the web and only require the presence of a java interpreter, that is often already installed on client pc's and can otherwise be downloaded freely. The advantage of a thick client is the high level of control the client has over the data and 3D scene. If more potent analysis and data manipulation facilities are to be incorporated in the development environment, a thick client may be better than a medium client, otherwise a medium client will suffice. A thin client is useless in a 3D environment since it can only display 2D images. In my prototype, I will use a thick client. All programming work will be done on the client anyway, and all data will be present there as well, a thick client will therefore be most practical. Parts of the client can later on be moved to a server environment.

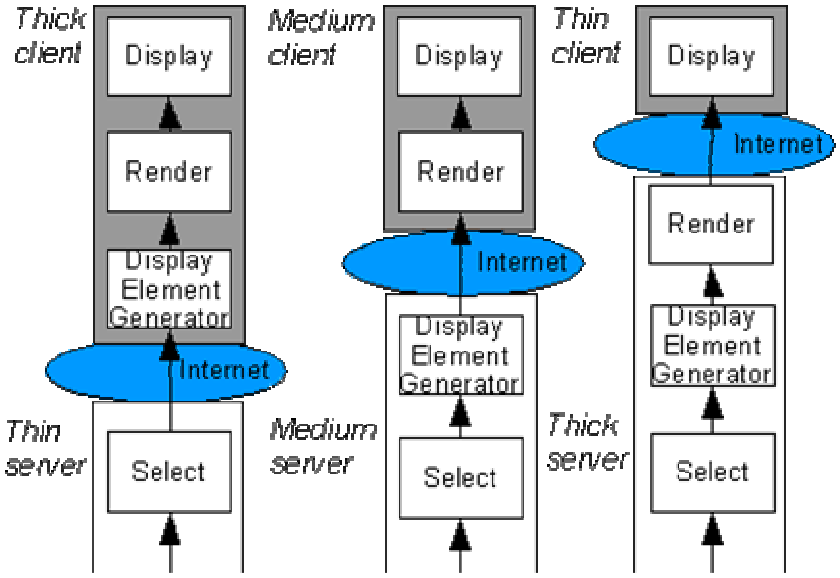


Figure 3.2 The three different levels of servers and clients. The server handles part of the work, and the rest is done by the client. Thick clients get the selected data and create the display elements from them, render them to 3D objects and display them, while thin clients only display the images.

Data selection

The data selection should be possible from many sources transparently. The system should not depend on one proprietary data format. This could be possible by the use of an open standard file format for the communication with the other modules and a modular design of the system. A modular design of the development environment makes it possible to develop multiple data selection modules. If all data selection modules present a common interface to the rest of the development environment, it does not matter what kind of data source is queried as long as the resulting data format is the same. A good first module to develop is a module that queries a Web Feature Service (WFS) (Peng, 2005;Peng and Zhang, 2004;Boucelma *et al.*, 2002;Specification: Beaujardiere, 2004). The WFS provides a common interface for querying geo-data and in turn is able to retrieve the geo-data from many different source formats and locations across the internet. The file format returned by the WFS is an open standard file format: GML (Specification: Cox *et al.*, 2003). By implementing a data selection module that queries web feature services, other modules for data querying may even be unnecessary.

Display element generation

After the data selection, the display elements have to be generated from the data. By the use of a common interface to the data selection modules and a common open standard format for the data returned by the data selection modules, only one display element generator is necessary. The data generated by the display element generator should be a format understood by the renderer. Therefore if the data input understood by the renderer is changed, the display element generator should be changed as well. The interface provided by the display element generator modules should be as uniform as possible, to make it easy to modify one if the data understood by the renderer changes.
What kind of elements the generator should be able to generate depends on the input and output formats it should handle and is therefore described with the renderer.

Renderer

The renderer is implemented in the form of a plug-in in the web browser, which provides both rendering and interaction functionality. The rendering is done from data generated by the display element generator.
The renderer should provide several things. It should enable interaction with the data, it should support several input formats and it should provide a certain look for the 3D scene. These things are summarized in figure 3.3 (see next page).
The interaction can be done on three levels. The renderer can interact with itself. This is the case with normal movement of the viewpoint within the scene. The input data is not changed, the renderer just renders the data differently. Many other interaction functionalities also are interactions within the renderer. The renderer can interact with the display element generator as well. This is the case if essentially the same data is displayed, but the data looks different. The user can for instance decide to give water a different colour, or request the display of objects that were not generated yet, but are present in the data provided by the data selection module.
The last level of interaction is the level with the data itself. The interaction with the data itself can be done within the web based GIS and with the original data sources. Both ways require the data selection modules to provide additional functionality beyond just selecting the data. Data manipulation within the development environment requires the data selection modules to manipulate the data, after it is selected from the original data sources. Interaction with the data from the original data sources requires the data selection modules to manipulate the data in the original sources through queries. Especially this last version of interaction with the data itself may be beyond the scope of this development environment and may be better done by a full fledged GIS. The modification of the data within the data selection module may be very useful for modelling functionality though.
If integrating interaction functionality in the renderer is too complicated, a separate module could be developed as well, which handles the interaction functionality with the display element generator and the data selection modules.
Another factor in choosing a renderer is the input data format(s) it understands. If it understands many input format(s) or it understands a commonly used (open standard) format, changing the renderer may not necessitate changing the display element generator.
A third element in the renderer is how the 3D scene looks and behaves. I will divide this in the following parts: realism, movement, orientation, navigation, explanation and coordinate system (see figure 3.3).

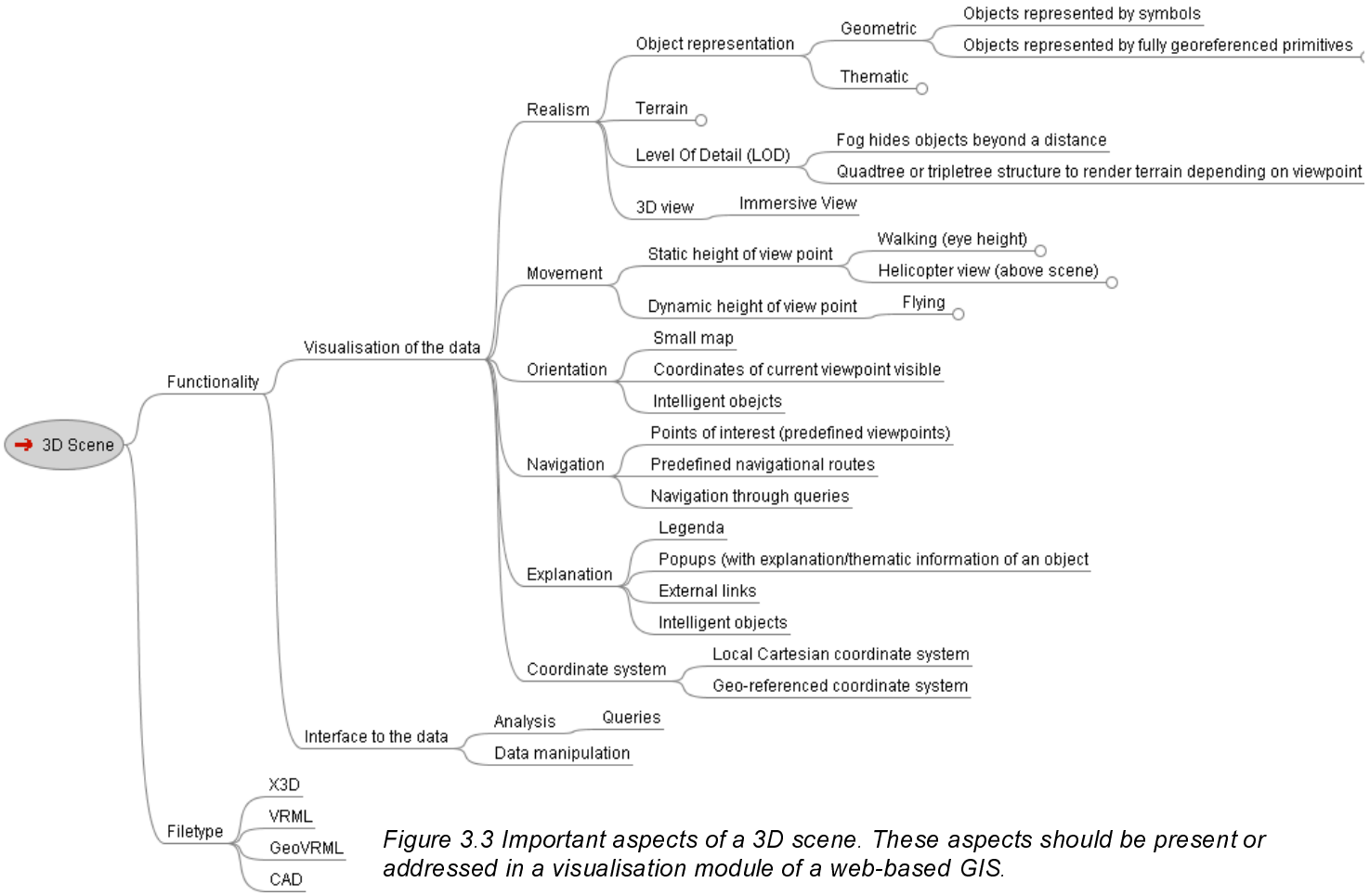


Figure 3.3 Important aspects of a 3D scene. These aspects should be present or addressed in a visualisation module of a web-based GIS.

Realism

The first element of realism is the immersion or type of view. Immersion involves the way users experience themselves in relation to the scene. There are various levels of immersion in Virtual Reality (VR) (Lammeren van and Hoogerwerf, 2003;Groetelaers, 2002). Four levels are interesting for the web (panel 3.1). The head mounted display would be the most realistic level of immersion, especially if the user would be wearing gloves that registered the users' hand movements, and translated them to computer gestures. For now, this level of immersion is not practically useful yet because the helmet and gloves are not widely available. I will therefore use an immersive VR on a computer screen. The level of immersion is not very high since only part of the users' vision is occupied by his view of the virtual world, but the user will have the impression of being part of the virtual world.

Panel 3.1 Six different types of immersion in a virtual world. Modified from Groetelaers (2002).

Types of immersion

Window on a World	*The user looks at a computer screen and has the impression of looking through a window at the real world.*
Video Mapping	*The user looks at a computer screen and sees a projection of himself in the virtual environment because of which he seems to be part of the scene*
Immersive VR	*Immersive VR is a VR-system in which the viewpoint of the user is placed within the virtual environment.*
Head Mounted Display	*The user wears a helmet in which stereo goggles have been incorporated. The virtual world is projected inside the goggles.*

The next element contributing to realism is the way the terrain looks. One can render a flat surface or use a digital elevation model to render the terrain. The terrain can further be displayed with plain colours, or with texture images mapped to the terrain. Mapping texture images can be done randomly or by using geo-referenced images from remote sensing data. Of course using a digital elevation model with images recorded from, for instance, aerial

photography mapped to the right places is the most realistic, but will be most costly in terms of performance.

The next and maybe most important element in realism is the way real world objects are represented in the virtual world. The geographic representation of objects depends on the kind of data used to render the world. If a high scaled dataset is used where cities are point elements, then an entire city will have to be represented by a symbol. If on the other hand a 3D dataset is built from CAD files of a construction site, each building can be rendered from geo-referenced primitives like nodes, lines, surfaces and volumes.
If the dataset is a 2.5D dataset instead of a 3D dataset, the ground surface of objects may be rendered from geo-referenced primitives, but the whole object will just be an extension of the ground surface and therefore may not look very realistic.
Thematic information can also be rendered in the 3D scene. Texture images and colours can be applied to faces to give them a certain look, according to thematic data, and line and point elements can represent thematic data by for instance line width, or point colour or size. These colours, textures and sizes can be applied according to the real world look of object, but also according to other thematic data. Boxes can be placed in a city scene where the height of the box represents the crime level in that area for instance. It greatly depends on the dataset and the purpose of the scene how objects will be rendered. High levels of realism can be reached by using fully geo-referenced 3D objects with image textures from photographs of the real world object, but very abstract scenes can be created as well where objects represent not real world objects but geo-referenced variables. A last factor contributing to the representation of thematic information is sound. Sound can also be used in a scene to add to the thematic representation of objects.

The last factor adding to realism is the level of detail. A scene can be rendered completely the moment it is loaded, but one could also render only parts of the scene according to the location of the viewpoint. In the real world the view of a person is not infinite, so why should all of the scene be loaded at once, if the user can't see it in the real world either. Two elements can be used in adjusting the level of detail: a quad-tree like mechanism and fog. A quad-tree (or triple-tree or similar mechanism) divides a terrain into squares. It the viewpoint comes within a predefined range of a square, the square is divided into 4 new squares and each square is rendered with a new, more detailed texture. This way, not all of a terrain has to be rendered with high resolution textures, but only the part the user can see.
A fog can be used to hide objects beyond a certain distance. Objects will be smaller the farther they are away from the viewpoint, so beyond a certain distance they may be reduced to less then a pixel in size. If a fog is used, these objects are not rendered anymore, saving memory.

Movement

The next element in rendering a 3D scene is the movement inside the scene. Although this is strictly speaking interaction with the scene and not just the visualisation, a 3D scene without movement possibilities is not worth a lot. There are three ways of moving through the scene: walking, flying and a bird's eye view. The bird's eye view and walking both have a static height of the view point. With the bird's eye view this is above the scene and with walking it is at eye height. Flying also allows for the height of the viewpoint to change. Walking is the most realistic way of moving through the scene. The bird's eye view and flying allow for a better overview of the scene though and flying allows for more parts of objects to be visible. Flying gives the most freedom to view from the scene what you want to see. It may be best to give the user the freedom to choose the way to move through the scene.

Orientation

Orientation involves the way the user knows where he is in relation to the rest of the scene. Ones creativity can go a long way in thinking of solutions for orientation, but I will mention three: 2D maps, intelligent objects and a coordinate display. The coordinate display is the simplest form of orientation: Just tell the user the coordinates at which he is located.
Another way to give the user a way to orientate himself is a 2D map. If unobtrusive and clearly not part of the 3D scene, a 2D map can be used to indicate the user's location and may also be used for navigational purposes. It can be used exactly the same way we use maps in the real world, and may therefore be very intuitive in its use. The ability to fly over the scene may reduce its value though.
The third orientation possibility I will discuss is the use of intelligent objects. A well known intelligent object is the, sometimes annoying, paperclip used in Microsoft's Office suite, which tells you things with or without asking for it. If implemented with a less assertive personality though, such an intelligent object can be very useful in a 3D scene, where it can provide orientation, navigation and explanation functionality. As far as orientation goes, it can tell you where you are, maybe in different ways, like which coordinates you are at, which street you are at, or which building you are looking at.

Navigation

Navigation is also an important factor in a 3D scene. One can wander aimlessly through a 3D scene, but usually one wants to extract specific information from the scene. Going straight away to the right locations is very useful then. I will discuss three elements of navigation, although one could think of many more.
The first are points of interest. Points of interest are just predefined viewpoints. One can use the viewer to position the viewpoint directly at a point of interest. This is the easiest way to navigate to specific places within the scene.
The second navigation method is through predefined navigational routes. Through animation or scripting one could predefine routes along which the viewpoint can move. Once started, the viewpoint will automatically move along the navigational route to the end.
The last navigational method is navigation through queries. One could perform a certain query and then highlight the resulting objects, or provide a way to move directly from resulting object to resulting object.
The interface to these navigational methods could be implemented in different ways. Points of interest are for instance already part of some 3D viewers, but an intelligent object or 2D map could also be used to provide the functionality.

Explanation

Explanation involves giving information about an object that is not directly visible. One can think of a legend function that explains the meaning of colours or symbols. But explanation could also mean giving more thematic information to the user than can be displayed in the scene, like the number of occupants in a building, the soil type a certain plot is located on or the zip-code of a plot. There are many ways to provide further information about an object. One is linking to external locations like websites, a second is using popups and a third is using the intelligent object again. Double clicking the object could trigger the intelligent object to provide information about the object for instance.

Coordinate system

The last element in rendering the 3D scene is the method used to position the objects in the scene. One could use a flat surface and a local coordinate system, without any reference to the real world location of the objects, but one could also use geo-referenced coordinates to position the objects, and use a geoid for the earth's surface instead of a flat surface. The latter is the most useful from a GIS point of view, but requires the renderer to know multiple references to the earth's surface and coordinate systems. Not all 3D viewers are able to handle this. The former is possible in all viewers, but requires a transformation of the coordinate system used in the dataset.

Display

The display part is handled by the user's operating system and video adapter. The renderer generates a constant stream of 2D images, which are handled by the operating system and turned into a digital signal, understood by the video adapter. The video adapter further manipulates the signal into a signal understood by the user's display device, which the user looks at. The communication between the renderer and the operating system should be done through a graphics library like OpenGL or DirectX. Which is used, depends on the operating system and the renderer. As long as the renderer uses a graphics library, which is installed on the operating system and the operating system is able to communicate with the display device, no further programming is required.

Wrap up

Most of the functionality discussed in this chapter is usually part of standard 3D visualisation environments defined by OS. If some functionality is not part of the OS itself, it can often be added to it with some additional programming. The OS discussed in the next chapter provide almost all of these elements. The prototype that will be discussed in chapter five will not provide all of this functionality though, since implementing it is outside the scope of a prototype.

4. Open Source and Open Standards

As explained in chapter one, the usage of open source software and open standards can greatly improve interoperability between programs and speed up software development. I therefore propose to base the development environment on open standards and open source software as much as possible. But what open source software and open standards exist for usage in our development environment? This is the question I will answer in this chapter. I will start by explaining the background of markup languages. An important part of the open standards, defined by many different organisations, consists of the definition of markup languages. One very important markup language is the eXtensible Markup Language (XML), which is meant to store arbitrary information. XML is accompanied by other open standards which are defined to interpret and manipulate XML files. This family of XML standards will be discussed next. Then I will continue into the application of this theory in the development environment, focused on the data storage and 3D visualisation parts. For data storage GML (an XML based format for the storage of geo-data) will be discussed, and for 3D visualisation the evolution from VRML to GeoVRML and X3D will be discussed. Linking back to the OGC portrayal model as discussed in chapter 3 (figure 3.1), I will then have covered the features part with the discussion of GML, and the display element part by the discussion of VRML, GeoVRML and X3D. The generation of the images is the last part I will discuss in this chapter, by setting criteria for X3D viewers (renderers) and concluding which viewer is best suited for our purpose.

Markup languages

A markup language combines data and extra information about the data. The extra information, for example about the data's structure or presentation, is expressed using markup, which is intermingled with the actual data (URL: Wikipedia, 2005). The best known example of a markup language is HTML (Specification: Raggett *et al.*, 1999). HTML is a markup language that presents text and other information and describes how the data should be displayed in a web browser. Because markup languages are text based, they are generally easy to interpret for humans. The easy interpretation by humans also makes it possible to write software compatible with a specific markup language without knowing exactly how the language is set up. Just looking at the data, formatted in a certain markup language, enables you to get a basic understanding how the data should be interpreted by software. This greatly facilitates interoperability between software packages.

Binary vs. text based file formats

There are two types of file formats in which data can be stored: binary file formats and text based file formats. Binary file formats are generally smaller in size than text based file formats. This is because text based formats also store extra information like the character encoding used and all white space in the file, while binary formats only store the actual data. Binary formats on the other hand are constructed by specific algorithms, and these algorithms are necessary to extract the information from the file and process the data. Text based formats can be read by anybody without the need for special algorithms to interpret the data. Like with the markup languages, it is easier to write software compatible with a specific file format if it is a text based file format, especially if you are not the author of the file format. If the algorithms used to construct

a binary file format, is not available (any more) the contents of the file is lost. With text-based formats, it is always possible for any person to read the file and therefore extract the data from the file. The greater size of text based formats can be overcome by file compression, essentially converting it to a binary format, but then to a binary format which is commonly used and freely available like zip or rar files. A disadvantage of this method is that the file has to be decompressed before it can be used. An alternative to compressing and decompressing a file is the use of codecs. With codecs, a file is also compressed but can be decompressed on the fly. Only the part of the file that is read at that specific moment is decompressed. The rest of the file is left alone. This is done by many image, audio and video file formats like respectively jpeg, mp3 and divx. A disadvantage of using codecs is that there are many different codecs and the availability of the decompression algorithms of a specific codec may be less widespread than decompression techniques like zip.

XML

XML is an open standard formulated by the World Wide Web Consortium (W3C) which' use is wide spread across the internet. From Wikipedia (URL: Wikipedia, 2001): "The Extensible Markup Language (XML) is a W3C-recommended general-purpose markup language for creating special-purpose markup languages. […] Its primary purpose is to facilitate the sharing of data across different systems, particularly systems connected via the Internet. Languages based on XML […] are defined in a formal way, allowing programs to modify and validate documents in these languages without prior knowledge of their form."

XML is a subset of the SGML specification, a specification for an even more abstract markup language, which I will not discuss further. The big advantage of XML over other text-based file formats is that XML formulates an abstract and general but very strict grammar for encoding information. An XML file consists of elements which can contain child elements and data. XML does not though, like HTML for instance does, define any pre-existing tags with a specific meaning. Specific meanings have to be defined by other specifications, built on top of XML. XML elements consist of a start tag and an end tag, for instance <street> and </street> in the example below. Within one element, child elements and data can be enclosed. An example of an XML can be the following:

```
<street>
        <name>Mainstreet</name>
        <building>
                <type>shop</type>
                <location>
                        (0,0,0);(1,0,0);(1,0,1)(0,0,1); (0,1,0);(1,1,0);(1,1,1);(0,1,1)
                </location>
        </building>
        <building>
                <type>house</type>
                <location>
                        (1,0,0);(2,0,0);(2,0,1)(1,0,1); (1,1,0);(2,1,0);(2,1,1);(1,1,1)
                </location>
        </building>
</street>
```

The street element only contains child elements, as does the building element. The name, type and location elements contain text content (consider the coordinates as a string of text). This gives the XML example a hierarchical structure. All XML files contain a hierarchical element structure. Another property of XML is its strict syntax. An XML file must contain one, and only one top element, here the street element, and child elements must be closed before its parent element is closed. Comparing XML

with HTML again, web browsers often allow HTML elements to be closed after their parent elements and also often allow elements not to be closed by an ending tag at all. This makes it very hard to interpret HTML.

These grammar rules make it possible for all XML-based information to be interpreted and parsed by general XML parsers and interpreters, instead of needing specific parsers and interpreters for each and every text-based format. This paves the way for the use of XML as a markup language for formulating more specific markup languages (Lake *et al.*, 2004;Specification: Bray *et al.*, 2004). It also makes XML quite self documenting, eliminating the need of extensive external documentation in order to be able to read the data. The power of XML therefore really shows when application specific markup languages, based on XML, are formulated and combined with other XML based markup languages and other open standards and software, as I will show next.

The XML Family

Aside from being able to use the same data and general purpose parsers and interpreters for different applications, another advantage of the use of XML-based markup languages is the existence of other open standards formulated by the W3C, which work together with XML: the XML Family (Lake *et al.*, 2004). Examples of such standards are XPath, XLink, XQuery, XSLT, DTD and XSchema.
XPath specifies how to reference elements in XML files. It can be used to point to specific elements in the hierarchical tree of an XML element (Specification: Clark and DeRose, 1999). The reference to an object starts at the top element, known as the root (/) and then references all elements hierarchically. In our example XML file, a location element of a building element could for instance be referenced by /street/building/location.
XLink specifies how to link between XML documents, using the XPath syntax. It enables linking between XML elements on the data level, allowing for the integration of data from different XML sources (Specification: DeRose *et al.*, 2001). In our example, we could include a building element for which the child elements are located in another XML file like this:

```
<street>
        <name>Mainstreet</name>
        <building>
                <type>shop</type>
                <location>
                        (0,0,0);(1,0,0);(1,0,1)(0,0,1); (0,1,0);(1,1,0);(1,1,1);(0,1,1)
                </location>
        </building>
        <building xlink:type=”simple” xlink:href=”otherfile.xml#building1”></building>
</street>
```

The XQuery specification describes a language that can be used to query collections of XML datasets (Specification: Boag *et al.*, 2005). It is semantically similar to SQL and uses XPath for referencing elements within XML datasets.
The XSLT specification describes an XML-based language that can be used to transform XML-based datasets into other datasets. The resulting datasets can, but do not have to be, XML-based datasets or files. The original XML-based dataset is not changed, but a new file or dataset is created. It is often used to transform different XML-based files into each other, or for converting XML-based datasets to HTML or PDF files.

Strictly speaking, no extensive documentation is needed in order to be able to interpret XML-based files. It is useful though, to be able to describe the XML-based markup

language/dataset for metadata purposes. One may for instance wonder what kind of child elements a specific element can contain. Apart from metadata purposes, describing the syntax of the dataset is also important for technical reasons. Looking at our example xml file, one can for instance design a piece of software that extracts all street names from different xml files. In order to be able to do this, the software has to be sure that all xml files it uses, use the same tag for the street name, <name> in this case. If one of the xml file would use a different tag, <streetName> for instance, the application reading the street names will not be able to find the street name in the specific xml file.

To be able to describe and check the syntax of xml files, the Document Type Definition (DTD) and XML Schema Document (XSD) standards have been formulated. DTD is the older of the two, and is a non XML-based markup language which describes an XML dataset. It is quite compact and easier to understand than XSD. XSD is newer and is an XML-based markup language (Lake *et al.*, 2004), eliminating the need of a special parser, like with DTD. XSD is more powerful: it is a more or less object oriented approach to defining XML-based datasets/markup languages and allows for more data types like integers, strings and floating points and restrictions to elements and properties than DTD does. Especially the complex XML-based markup languages are formulated with XSD files. By describing the syntax of a specific XML-based markup language in a DTD or XSD file, one is able to "validate" XML files that implement that markup language. In this case validating means checking if the XML file follows the syntactical rules of the markup language as described in the DTD or XSD.

XML tools

Many tools exist that take implement or make use of the abovementioned standards. A few of these tools will be discussed here.
Using XML data starts with parsing the XML document. Parsing is reading the XML file and representing it in a form that is usable by programs. This can be done in a sequential way, where the parser reads through the XML document from top to bottom and performs some action when certain tags are encountered. It could also be done by reading the document all at once and representing the document in an object oriented way, which could be the Document Object Model specified by the W3C or the JDOM, which is specifically available in Java (Mc Laughlin, 2001). The Apache Xerces parser can implement all three methods of parsing. It is able to read XML documents, validate them according to their DTD or XSD and represent the document in a way it can be used in further programming. It is a very widely used interpreter and is also open source (URL: Apache Xerces Workgroup, 2005). The functionality of Apache Xerces is available through a set of Java or C API's, which can be used from other programs. It is not a stand-alone application since the parsing results in a representation of the document in a programming language. Other parsers are Microsoft's MSXML and Sun Microsystems' Crimson.
For transforming XML documents, several transformers exist that can handle XSLT transformations. The simplest form is a web browser that is able to transform XML documents on the fly to display them in the browser screen. This is especially useful for generating web pages. For transformations that are more complicated, a separate parser should be used. Options are the partially open source Saxon by Saxonica (URL: Kay, 2005), again Microsoft's MSXML and Apache Xalan (URL: Apache Xalan Workgroup, 2005). Of these three, only Apache Xalan is completely open source. It is again a widely used and very good transformer. It comes in two versions, a Java version and a C++ version. The Java version consists of a set of API's that can be used by other programs and a command-line version that can be called with the source and target XML files and the XSLT file to use.

Apart from these tools that are generally used by other programs, there are also graphical programs that facilitate the creation of XML files and the files for the XML family standards like XSD files, DTD files and XSLT files. Two examples of such applications are Stylus Studio XML (URL: DataDirect Technologies, 2005) and Altova XML Spy (URL: Altova, 2005). These applications combine writing XML, XSD, DTD and XSLT files with validating, parsing and transforming them with internal or external parsers and transformers.

The above-mentioned set of standards from the XML family of standards and the accompanying tools are what make XML such a powerful standard (Lake *et al.*, 2004). XML-based datasets combined with the standards from the XML family, make it possible to extract information from all XML-based datasets in a common way, transform XML-based datasets into one another, transform XML-based datasets to other file formats and validate XML-based datasets. It allows us to combine data from different sources into one application, and to present the same data in many different ways. Especially the possibility to convert XML files, written in different XML-based markup languages, into one another is something we will use in our application.

GML

One XML-based markup language I will use is GML. GML is an open framework markup language for encoding geographic information in a plain text file format. The format is an XML format and as of version 3.0 it is completely described in XSD files. GML is used to describe real world objects as features and store them in a file. The objects may be abstract objects like noise zones around an airfield, or concrete objects like buildings and roads.
GML does not pretend to be able to encode all geographic information out-of-the-box. It is a framework. It provides abstract geographic, topologic, thematic, temporal and meta elements, that can be used to specify concrete real-world objects. It does not for instance define a bridge object, but it does provide the “centerLineOf”, width, location, “spans” and metadata elements, that can be used to specify a bridge object.
Because of this, it is necessary to develop an application specific definition of the GML data to be used in an application. Different fields of application (governmental policymaking, landscape planning, utility infrastructure planning and disaster management to name a few) require different geographical information definitions. To a company planning new power lines it may not matter how many inhabitants live in a certain apartment complex but to a fire fighter responding to an emergency it may matter. These differences imply different data models and therefore require the definition of these application specific GML implementations.
The application specific GML definition uses the abstract elements defined in the GML specification and combines these elements into concrete objects. An application specific GML specification can for instance combine coordinate elements, a metadata element and a topological “spans” element from the GML specification with a self-defined “name” element to create a new object called “bridge”.
The application specific dataset is described in a new XSD file. The GML dataset references this application specific XSD file and the XSD files from the GML specification. Apart from using the abstract elements described in the XSD files of the GML specification, an application specific XSD file may also define new elements, like for instance a “name” element for our bridge. To visualise the relationship between XML, GML and the application specific GML implementation, one can see them as subsequent

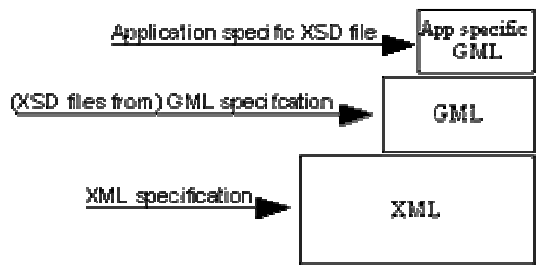


Figure 4.1:The relationship between XML, GML and an application specific implementation of GML, and how these three levels are described.

specialisations of one another, built on top of each other (figure 4.1).

As I already implied, the GML specification consists of multiple XSD files. The specification is divided into functional groups, which are then specified in a specific XSD file. There are for instance two XSD files describing the elementary GML elements, two that specify temporal properties, five that specify geometry and so on. This enables applications to support only subsets of the GML specification. This is an advantage since there are over 20 XSD files that specify all the GML elements. An application that is aimed at a specific audience or data structure can therefore choose to ignore parts of the GML specification that it will not use without compromising portability.

The abstract nature of GML provides room to develop the application specific GML implementations, which makes GML very flexible and useful for many or all data models and GIS applications. To use GML in a webGIS development environment though, the framework will have to be flexible as well in the GML data it is able to use. It will have to be able to read the application specific XSD files and treat the data according to them. Different application specific GML implementations shouldn't necessitate adjustments to the framework.

GML is a very young standard, the first definition was in 1999 but the first version (3.0) that is useful for the actual storage of geo-data instead of only transport between applications was adopted as a standard in 2003. Because of this, there are very few existing application specific GML definitions. In the future however, when more and more people start using GML, general definitions for specific fields may become available, so people won't have to define their own XSD GML specifications. An example is the Dutch topographical service, who will publish their datasets only in GML in the near future (Quak *et al.*, 2002).

As described in the section about text-based file formats, the size of text-based file formats is bigger than that of binary encoded files. This may be a concrete problem with larger GML datasets, describing complete cities in high detail for instance. Compressing the files with standard "all at once" compression algorithms may not solve this since it will add a long waiting time for decompressing the files before they are ready to use. Some binary encoding formats are being developed for XML in general and GML specifically to overcome this problem by allowing on the fly compression and decompression by codecs (Bayardo *et al.*, 2004;Specification: Web3D Consortium, 2005;Specification: Martin and Jano, 1999;URL: Expway, 2005).

VRML

From geo-data storage, I now move on the 3D visualisation standards. The first standard to discuss is VRML. VRML is the Virtual Reality Modelling Language. It is a language that makes it possible to store and render 3D scenes. VRML version 1.0 (specification: Bell *et al.*, 1995) was first defined in 1995 and later superseded by VRML 2.0, which was renamed into VRML97 (Specification: Web3D Consortium, 1997) after the ISO certification process. Since VRML 1.0 is long since out of use, I will only discuss VRML97.
VRML files are plain text files. The file consists of nodes, which have fields, containing the properties of the node or containing child nodes. Three elementary nodes are what generate a basic 3D scene: the shape node, the viewpoint node and the transform node. The shape node defines shapes in the 3D scene. It consists of a geometry field and an appearance field. The appearance field contains properties like colour, lighting, textures, animated textures, etc. The geometry field contains the geometry of the

shape. The geometry can both be based on simple shapes, like a box or cone, or one can define complex objects from points, vertices and faces. The geometry field also contains coordinates for the position of the object, relative to the parent node's origin. The transform node creates this 'parent' origin. The scene has a coordinate system in meters from 0,0,0 with Y being the up direction. The transform node can be the parent of other nodes and defines a new coordinate system in meters, relative to its parent's origin, which can be other transform nodes, or the scene coordinate system. All 'child' objects are positioned relative to the transform node's origin, making it possible to transform many objects by modifying the parent transform node. The transform node also supports rotation and scaling.
Many additional nodes exist, like elevation grids, sound, nodes describing the scene and nodes for user interaction and animation like a timing node and a sensor node.
Apart from nodes and field, additional possibilities give VRML a lot of modelling and animation possibilities. Nodes can contain fields that can provide information to other nodes, take input from other nodes, or both. Routes can be defined to connect these fields between nodes. This way, a sensor node can trigger a timing node, which can modify a transform node for instance. In addition to standard nodes, the routes can also be used to connect script nodes to each other and to normal nodes. Script nodes contain fields for data input and output, and can contain ECMAScript or Java code to process the input data and generate output data. This way, more advanced programming can be embedded in VRML to generate more complex interaction and simulation. To further accommodate the use of programming logic in a scene, the External Authoring Interface (EAI) was formulated. This specifies a programming language independent interface to the VRML scene for external programs. Similar to the script nodes embedded inside the scene, external programs can use the EAI to interact with the VRML scene. Language bindings have been formulated for ECMAScript and Java to the EAI.
VRML also allows the combination of different scenes into one scene. VRML files from different locations on the internet can be combined in this way into one scene. This is very useful for geo-visualisation, where data sources are often distributed across the internet. This way, scenes from different sources can be combined on the VRML level, without the need of having access to the actual geo-data, much like with Web Map Services.
Adding up all the possibilities of VRML, one can conclude that most functionality in the visualisation branch of the diagram in figure 3.3 is present in VRML and therefore makes VRML a good candidate for 3D visualisation on the internet. The internal scripting functionality, the interaction functionality and the EAI enable us to provide the functionality from figure 3.3 that is not included natively, like intelligent objects and 2D maps.
The VRML specification allows complex worlds to be created with much interactivity and modelling functionality. There is a large collection of examples available at the Tecfa website (URL: Tecfa, 2005) and a very nice example is the IrishSpace project (URL: The IrishSpace Project, 1997).

GeoVRML

GeoVRML (Specification: Reddy and Iverson, 2002) is an extension on VRML97. It uses the EAI of VRML97 to add extra functionality to VRML97 for geo visualisation. Four issues in VRML97 were addressed in GeoVRML (Specification: Reddy and Iverson, 2002). I will discuss two of them which involve geo-referencing spatial objects. VRML uses a local Cartesian coordinate system with the origin 0,0,0. This was done because the intended use of VRML97 was to create small virtual world for games and animation. A basic VRML97 scene therefore covers only at most a couple of hundred meters, and has no link to the real world. For geo-visualisation, this link to the real

world is present, and scenes can be as big as whole cities, countries or even our planet. In GIS different coordinate systems exist to reference an objects position relative to the earth, like geocentric (Cartesian relative to the earth's centre), geodetic (relative to the ellipsoid model for the earth's surface, like latitude/longitude) and projective (a coordinate system for a projection of a part of the earth's surface, like UTM). One could transform a dataset to a local Cartesian coordinate system used by VRML, but that would mean loosing the geo-referencing data of the dataset. It would be much better if one could use native coordinates in VRML.
The second problem connected to geo-referencing was the fact that VRML97 could handle single precision floating point numbers for coordinates, which means that coordinates can be accurate to one part in about eight million. For small scale worlds where the scene is not bigger than 1km, and a local Cartesian coordinate system is used, the single precision floating point coordinates can be accurate to 0.1mm. The radius of the earth's surface (under the WGS84 ellipsoid) is 6,378,137m. Using a global scale coordinate system with single precision floating point coordinates would then mean a precision of 0.8m, causing serious problems like jitter.

These two problems were addressed by GeoVRML. In GeoVRML a GeoCoordinate node was added. The GeoCoordinate contains a string field that contains coordinates in a geo-referenced coordinate system, a string field that contains the geo-reference system used for the coordinates, and a field, which can contain a geoorigin node. The geoorigin node is used to create a geographically referenced origin against which all coordinates are referenced. This enables GeoVRML files to directly use geo-referenced coordinates in the scenes. The use of a string filed for the coordinates field of the GeoCoordinate node allows the use of bigger numbers than would be possible with single precision floating point numbers. These extra fields and nodes are interpreted by java classes, which return the necessary information to the VRML97 renderer. The GeoVRML classes actually transform the geo-data on the fly into a local coordinate system with 0,0,0 as the origin. The transformation is done using a java version of the SEDRIS Conversion API. The SEDRIS Conversion API is based on the ISO standard Spatial Reference Model (SRM) and can convert 12 different spatial reference frames losslessly into one another.
With GeoVRML the first possibility arrived to use open standards for visualising geo-data on the web.

X3D

After VRML97, VRML continued to evolve. A lot of functionality was asked for by users that was not part of VRML97, like the GeoVRML extensions. Also, VRML97 has its own file format, which needs a specific parser to interpret and manipulate. People started asking for other data encodings. The arrival of XML and especially the XML family standards and accompanying tools caused a demand for an XML encoding of 3D scenes. The disadvantage of text-based formats, especially their size and processing times caused a demand for a binary encoding. The Web3D Consortium therefore developed X3D, the successor of VRML97.
X3D is really an abstract specification. It defines what kind of objects should exist, what the relationships between objects and the properties of objects should be and how objects should look. It does not describe a single syntax in which to write the functionality though. Together with the abstract X3D specification, other specifications define syntax types for X3D file encodings. There is an XML encoding for X3D, a classic VRML encoding and a binary encoding.
The syntax for each of these encodings is different, but the functionality and behaviour is the same. Authors of X3D scenes are free to choose which encoding they will use for their scenes. This freedom in file encodings allows for great interoperability between

different programs, but it also provides the advantages of the different encoding types to be used in a specific application. The XML encoding adds the power of XML and the XML family to X3D, while the VRML encoding adds the power of existing VRML capable software to X3D and the binary encoding the performance and security advantages (URL: Web3D Consortium, 2005a;URL: Bullard, 2003;URL: Web3D Consortium, 2005b).
Apart from defining different file encodings, the abstract specification of X3D is also different from VRML97. X3D is for instance built up in a modular way. Four different profiles exist that provide different functionalities (Specification: Web3D Consortium, 2005;URL: Web3D Consortium, 2005b): The interchange profile is the basic profile for data communication. It supports geometry, texturing, basic lighting and animation. The interactive profile can be used for simple scene rendering. It supports more advanced lighting and animation, some sensor nodes and advanced timing. The immersive profile enables full 3D scenes with scripting, collision detection, fog and sound. The full profile supports additional advanced functionality like human animation (H-Anim) and GeoSpatial components. The advantage of these profiles is that additional functionality can be added to a profile without other profiles being affected. Advancements in one profile therefore do not have to wait on advancements in other profiles. The addition of new profiles is also possible without the need for changing the existing specification. Domain specific profiles could for instance be developed for X3D quickly.
As I said before, the full profile has a GeoSpatial component. With the GeoSpatial component, the X3D workgroup incorporated the functionality of GeoVRML within X3D. This was done in tight cooperation with the GeoVRML people. The result is that X3D supports double precision floating point coordinates natively, and also natively understands the GeoOrigin node, GeoCoordinate node and other GeoSpatial functionality. This eliminates the need for extra java classes for geo-visualisation.
A last advantage I want to mention about X3D over VRML97 is that the behaviour of X3D elements in browsers is better described than was the case for VRML97. With VRML97 there was a lack of good behavioural description of VRML elements, leaving too much room for the individual browsers to define that behaviour. This resulted in differences between browsers that made it hard to make VRML scenes that were identical in all browsers. In X3D, effort has been put in defining the behaviour of the elements in browser more clearly, making interoperability between browsers easier.

XSLT and the client-server model

As explained before, XSLT is a language that defines how to transform an XML-based dataset into other (XML and non-XML) datasets. With XSLT, it is possible to use the same dataset for different visualisation purposes. One could think of transforming a GML dataset into static HTML pages, SVG files and X3D files. The HTML documents could be used for metadata or reports about the data, while the SVG files could be used to generate 2D maps from the dataset and the X3D files could be used to visualise the objects in 3D. The conversion of the GML dataset into the different formats requires the creation of an XSLT file for each output format. After the XSLT files have been created, the transformation process can be done on different levels in the client-server model. To illustrate this, I'll use the example of the transformation of a GML dataset into X3D, which is subsequently rendered in a viewer. In chapter 2, a simple version of this client-server model has already been discussed. The data selection part of that model is here represented by the generation of the GML file by selection from the underlying dataset. The display elements are here represented in an X3D file, which is generated by an XSLT transformation. The rendering is done by a renderer that is an X3D viewer. This viewer also provides an interface into the scene, and the surrounding browser provides an interface to the webserver.

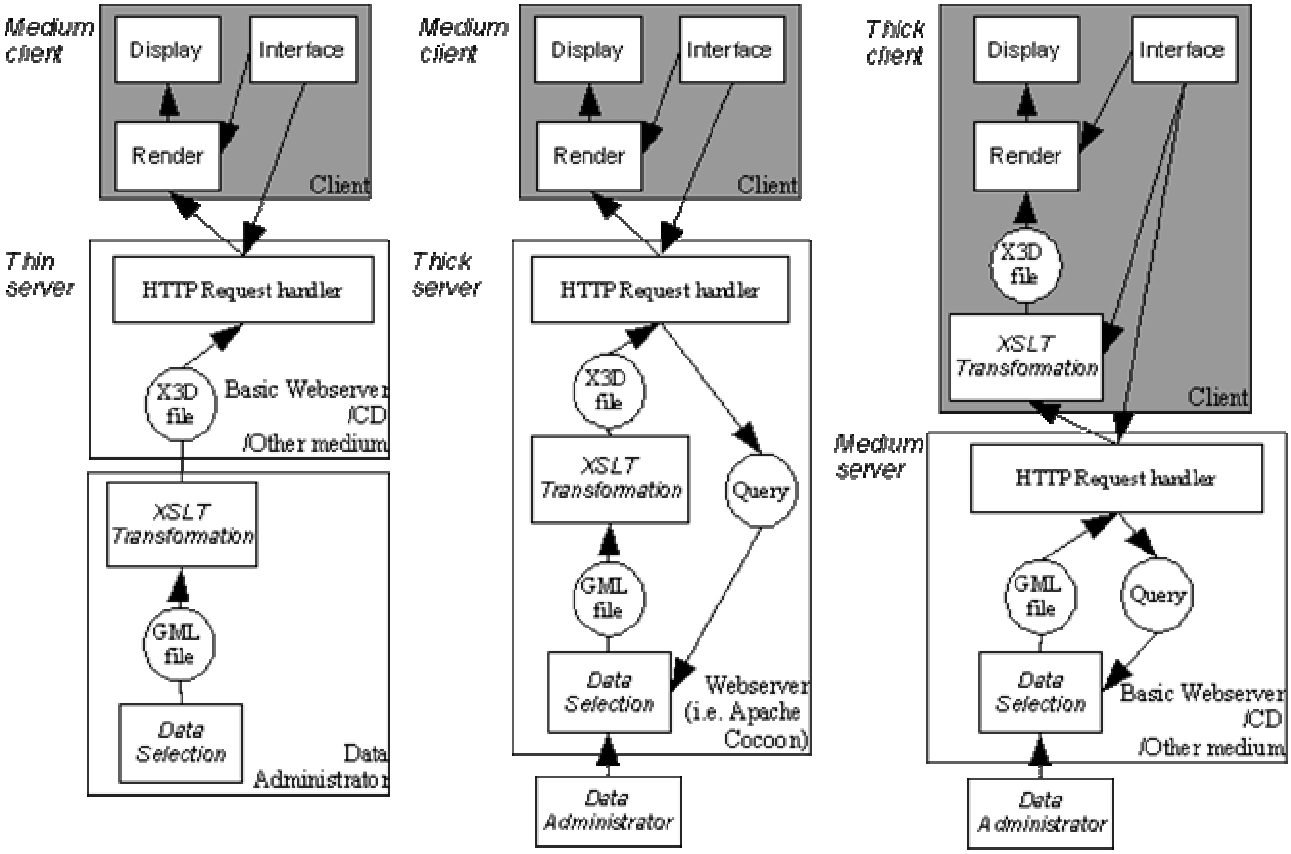


Figure 4.2: The client-server model as described in chapter 3 worked out more detailed for the process of visualizing geo-data through GML and an XSLT transformation in X3D format on the client.

Figure 4.2 illustrates the three models. In the first model the administrator manages the Dataset. The Administrator also handles the display element generation by transforming the data through an XSLT transformation into an X3D file. The X3D files are put on the server, and sent to the client if it requests the file. The client only handles rendering the X3D file and interaction with the X3D scene. This model has the highest performance on the server because the Administrator has to perform some tasks manually (Polys, 2002).
In the second model, the display element generation is done dynamically. The client requests some data, the webserver then selects the correct data from the dataset. The data is sent as a GML file to the display element generator, which generates the X3D file through an XSLT transformation. The webserver returns the X3D file to the client, which handles rendering of and interaction with the X3D scene. This model requires less work from the Administrator, and is more flexible since the display elements are generated interactively. The user could even, through an interface on the server side, influence the display element generation without the need of a thick client. The first model involves medium client and a medium server. The server could even be replaced by a CDROM or the local filesystem. The second model requires a medium client and a thin server. The Apache Cocoon project provides a server that is well suited for this task (URL: The Apache Cocoon Project Management Committee, 2005), but it could also be provided by server-side Java programs, since Java has API's for XSLT, XML handling and XML parsers, as well as bindings for X3D. A disadvantage of this model is the need for a stronger server, since it will require more work by the server.
The third model involves a thin server and a thick client. The webserver only handles the selection of the correct data from the dataset when the client requests specific data. The GML file is sent to the client, which handles the display element generation by an XSLT transformation, rendering of the X3D file and the interaction with the renderer. This allows the client to influence the XSLT transformation, giving the client more control of the representation of the data. The thick client could be implemented with

roughly the same Java application that could be used on the server in model two. In this third model, the webserver could again be replaced by a CDROM or the local file system if the GML file is present locally.

How are these standards related again?

An overview of the relation between the different open standards as described in this chapter is given in figure 4.3. From left to right, the different standards are ordered roughly according to the time they were formulated. From top to bottom, the standards are split into two groups. The top group is a group of standards that was formulated to standardise a way to perform a specific task in order to improve interoperability, resulting in a strict set of rules for the specifications (SGML and XML). The lower group of specifications was specified starting from a specific goal to achieve, resulting in more practical standards with more room for different interpreters to interpret the standards differently and therefore less interoperability between applications (HTML and VRML). Although the development of practically useful specifications took longer in the above approach, this approach now incorporates best of both worlds: well-defined standards with little room for free interpretation but also formulated with a practical goal in mind. Examples of these specifications are X3D and XHTML. GML belongs to this group to a lesser degree since it is still an abstract markup language, which needs further implementation in order to be practically useful. It is not less useful though.

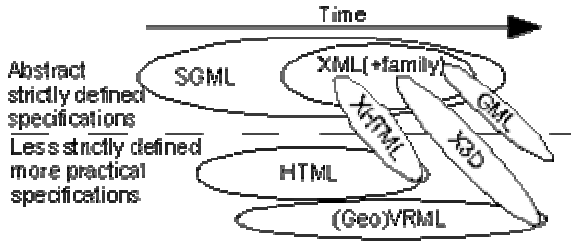


Figure 4.3: A diagram showing the relationship between different markup languages. From right to left the relationship in time of definition and from top to bottom a relationship from abstract but well defined specifications to less strictly defined but practical specifications.

Viewers

A very important part of 3D visualisation with X3D is the X3D viewer. Many browsers exist to view VRML97 and X3D content. Some browsers only support VRML97 content, while others support only X3D and some support both. Most browsers are freely available, at least for non profit or evaluation use, but most are not open source software. To determine which browser will be used in this project, the following viewers were compared according to a fixed set of properties: BS Contact, Blaxxun Contact, Xj3D viewer, Flux Player, Octaga and FreeWRL.

One requirement is the availability of the viewer. Is the browser freely available? Is it open source? The free availability of a browser is very important since the end user has to be able to download and install the viewer easily. It is also a big advantage if the viewer is open source. It would allow everybody who is interested to contribute to the development and would create a more open view into what the viewer supports and how. If some functionality is not yet present, one can contribute to the project by developing the functionality himself.

A second requirement is support for at least the XML encoding of the X3D specification. The classic VRML encoding of X3D and support for VRML97 is also desirable for interoperability purposes.

Another important requirement is the support for the geospatial components of the X3D full profile. These allow objects within the scene to be positioned by their real world

coordinates, preventing loss of data by transforming real world coordinates into a local Cartesian coordinate system.

The last requirement is the functionality it provides for interaction with the scene, like described in chapter three (figure 3.3). A person using the viewer should at least be able to walk and fly through the scene. The ability to click on objects to get more information about the object is also a requirement. Other interactivity elements like the support for sensors to be able to take user input and timed input for animation and modelling is highly desirable and also the support for different viewpoints so the user can jump from one point in the scene to the other is very functional. All these properties are part of the immersive profile of X3D and should therefore not be a problem for an X3D viewer that supports the full profile of X3D.
Because the X3D standard is very young, no viewer officially supports the full X3D profile. BS Contact by Bitmanagement software and the browser of the Xj3D toolkit by the Xj3D workgroup of the Web3D Consortium look like they are furthest with this support though and seem like the best options based on the restrictions.
BS Contact looks like the browser with the most features. A major disadvantage though is that it is only free for evaluation use. An annoying object floats through the scene, asking you if you would like to buy the software. It does support classic VRML and the XML encoding of X3D.
The Xj3D toolkit is much more than an X3D viewer and consists of a large array of Java classes that allow the visualisation and manipulation of X3D scenes. As one example a viewer is included in the toolkit. The toolkit can do much more though; it supports the Scene Access Interface (SAI) defined by the X3D specification, enabling for instance the real-time manipulation of the 3D scene by external programs. The project is an open source project, initiated by the Web3D Consortium. Their support of the geospatial component of X3D does not seem to be complete though, although they are almost there. The fact that the project is supported by the Web3D consortium does seem to guarantee full conformity to the X3D specification when they are finished. Together with the open source nature and the support for the SAI this makes the Xj3D toolkit the most promising environment to use. The interface to the viewer is quite Spartan though, compared to BS Contact and it is quite slow when a 3D scene is loaded. I will try both browsers when testing my application.

5. The Visualisation module

Concept

Part of the development environment as described in chapter three is a visualisation module that is able to display geo-referenced data in 3D on the web. A prototype for this module will be implemented in this study.
A visualisation module would provide the top three levels in the OGC portrayal model as described in figure 3.1. Data selection should be handled by another facility that could be interfaced with through the browser but also through the 3D scene. This will not be handled by the visualisation module though.
Important for this visualisation module is it's extensibility and portability. It should be possible to build extra functionality on top of the module, like data selection, data manipulation and analysis functionality. Communication with other applications and data sources should be possible as well. The open standards described in chapter four should therefore be used to implement the visualisation module. GML could be used as the XML-based format to store the original data. XSLT could then be used to transform the GML dataset into an X3D file, which could then be visualised in an X3D viewer.
The X3D viewer could provide much of the functionality for the 3D scene as described in chapters three (figure 3.3) and four. Features that are not natively supported by the viewer, like intelligent objects, could be created within X3D or through the SAI.
The standards, which will be used for the implementation already provide much of the functionality that is needed to implement the visualisation, and tools and API's exist to use this functionality, as described in chapter four. What has to be developed is the glue between these tools and API's.
The complete visualisation module could be described like in figure 5.1. It starts out

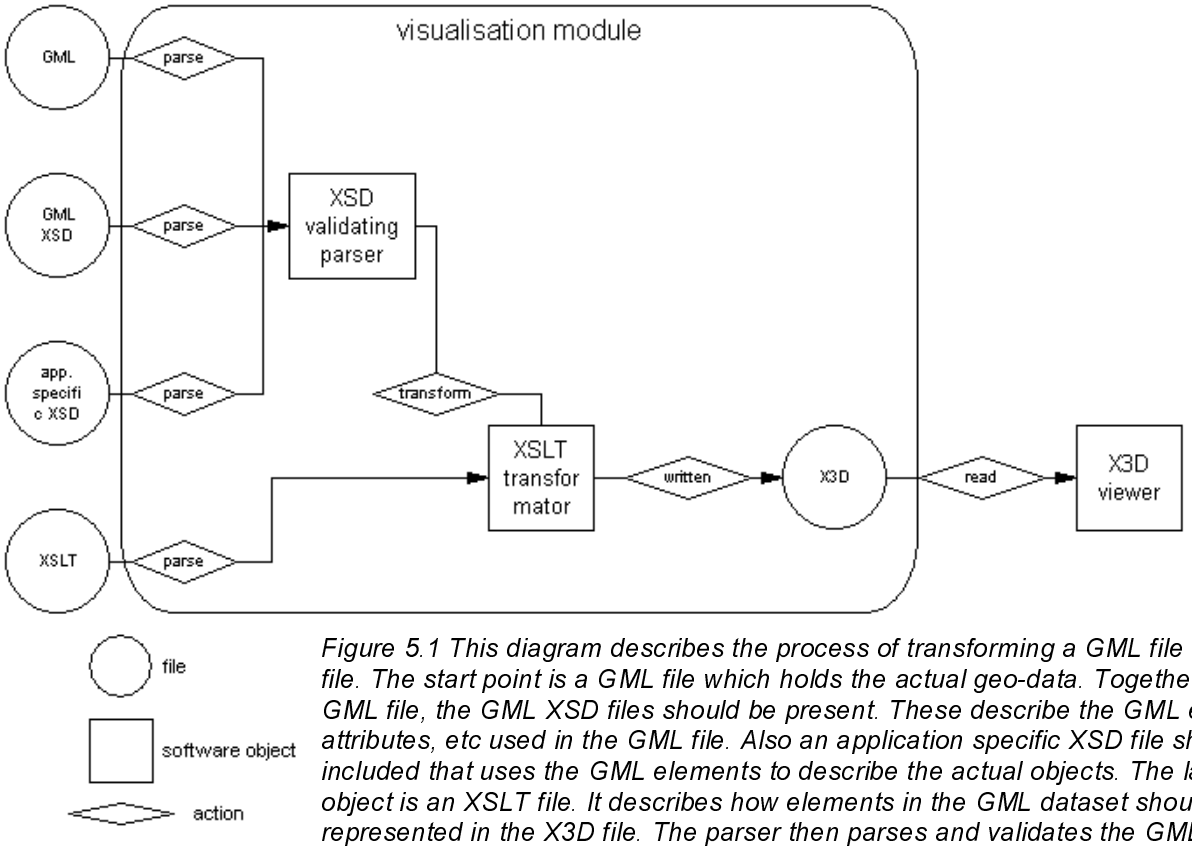


Figure 5.1 This diagram describes the process of transforming a GML file into an X3D file. The start point is a GML file which holds the actual geo-data. Together with the GML file, the GML XSD files should be present. These describe the GML elements, attributes, etc used in the GML file. Also an application specific XSD file should be included that uses the GML elements to describe the actual objects. The last input object is an XSLT file. It describes how elements in the GML dataset should be represented in the X3D file. The parser then parses and validates the GML. This data is used, together with the XSLT file to transform the geo-data into an X3D file, which is then displayed in an X3D viewer.

with a GML file, which contains the data to be visualised. The GML file should conform to the GML specification, contained in the GML XSD files, and also to an application specific implementation of GML, described in the application specific GML file. An XML parser should parse the GML file and validate it according to the various XSD files. The parsed data should be passed on to an XSLT transformer. The transformer should then read an XSLT file which specifies how to represent the GML objects in an X3D file, and should transform the GML data into such an X3D file. The X3D file could then be visualised in an X3D viewer.

Implementation

Input

A set of GML files will be used for the application. The GML files were supplied by Mr Ron Lake, author of "GML: Foundation for the Geo-Web" (Lake *et al.*, 2004) and co-author of the GML specification. The set consists of ten GML (version 3.0) files, all implementing the same application specific XSD file, also included in the set. The files define no real world objects but imaginary scenes.
Together with the GML files, also VRML files are supplied that already contain a 3D representation of the GML file. This enables me to check the result of my work on the VRML files.
Unfortunately the files are not really geo-referenced. This would have been very useful, since it would enable us to test geo-referencing objects in X3D scenes. It is not a major problem though since the possibilities to geo-reference data is present in X3D, and therefore using geo-referenced data will only involve adding extra information to the X3D scene. For now, the data is implemented in a local Cartesian coordinate system.

For the input part, the GML and application specific XSD file have been supplied by Ron Lake. The GML specification XSD files are freely available on the OGC website (URL: Open Geospatial Consortium, 2005), but were also included in the set of file I got from Ron Lake. What remains is the XSL file to specify how certain elements from the GML file should be represented in the X3D file. This has to be written specifically for each application specific XSD file. I therefore wrote this file myself.

Processing

For processing the GML data, the Apache suite of XML processing applications was used. For parsing the files, the Apache Xerces parser was used and for transforming the Apache Xalan transformer. For both programs the Java implementation was used so java could also be used to glue the different parts together. Instead of using native Java though, Python was used with the Jython interpreter that is able to import Java classes into the Python programming language (URL: Jython Project, 2000). This way the efficient Python programming language is used which makes for quick development, with the power of the existing Java API's. The downside is some performance loss since two different programming languages have to be interpreted. For my purpose this is not really a problem. For production environments though, code may have to be written in native Python or Java.
In general, the work of the Python program is as follows: The Python program is called from the command line. Three arguments have to be passed to the program. The first one is the GML file to transform, the second is the XSLT file to use, and the third is the destination X3D file. The Python program first checks if all arguments have been passed. Then it instantiates a Xerces parser through the Java JDOM/SAX API. The parser checks if the GML file is valid according to the XSD files from the GML

specification and also according to the application specific XSD file. It not, it outputs an error; else it will convert the GML file into a memory resident (JDOM) representation. Next an Apache Xalan transformer is instantiated through the Java JAXP API. The XSLT file is passed to it, and then the memory resident representation of the GML file that resulted from parsing, and the file name to output to, are passed to it. It then transforms the memory resident representation into an X3D file. This can be opened with an X3D viewer.
The parsing, validation, and transforming of the GML data only requires very few lines of code. Unfortunately, the transforming can't be done directly from the GML file. This is due to the nature of GML, X3D and XSLT. GML employs a very rich geometric model. Each feature can contain multiple geometric primitives, each in a separate element. To use the example from chapter four again, our building could look like this (I simplified the GML code to make it useful as an example):

```
<building>
        <gml:extentOf>
                <gml:Polygon>
                        <gml:coordinates>
                                0,0,0 1,0,0 1,0,1 0,0,1
                        </gml:coordinates>
                </gml:Polygon>
        </grml:extentOf>
        <gml:multiExtentOf>
                <gml:Polygon>
                        <gml:coordinates>
                                1,0,0 0,0,0 0,1,0 1,1,0
                        </gml:coordinates>
                </gml:Polygon>
                <gml:Polygon>
                        <gml:coordinates>
                                0,0,1 0,0,0 0,1,0 0,1,1
                        </gml:coordinates>
                </gml:Polygon>
                <gml:Polygon>
                        <gml:coordinates>
                                1,0,1 1,0,0 1,1,0 1,1,1
                        </gml:coordinates>
                </gml:Polygon>
                <gml:Polygon>
                        <gml:coordinates>
                                1,0,1 1,0,0 1,1,0 1,1,1
                        </gml:coordinates>
                </gml:Polygon>
        </grml:multiExtentOf>
</building>
```

The building contains an extentOf and a multiExtentOf element. The extentOf element declares a surface: it contains one polygon node. The multiExtentOf element declares a volume. To do this it contains multiple polygons that together form the volume. The volume and surface together declare the building feature. Point, line and more complex primitives can be included in one feature as well.
X3D on the other hand has a far more classical geometric representation of the 3D objects. A representation of the same building in X3D is this:

```
<shape>
        <IndexedFaceSet coordIndex="0 1 2 3 -1 1 0 4 5 -1 3 0 4 6 -1 2 1 5 7 -1 3 1 5 7">
                <coordinate point="0,0,0 1,0,0 1,0,1 0,0,1 0,1,0 1,1,0 0,1,1 1,1,1/>
        </IndexedFaceSet>
</shape>
```

Where we need 31 lines of code to represent this building in GML we only need 5 in X3D. X3D puts all coordinates in one big list, and then references these coordinates by an index number in the coordIndex attribute to construct the faces. Each face is

separated by a -1 in the coordIndex attribute. Whereas the same coordinate can occur more than once inside a GML feature it will not in X3D. In X3D though there is no difference between volumes and faces, except for the number of faces, and a shape can only be one of volume/surface, line or point, but cannot contain all three like in GML. To visualise the GML features therefore a conversion is needed from the GML way of expressing geometry to the X3D way. XSLT is not a procedural programming language though like Java or Python, but a markup language that tells the XSLT transformer how to represent an element in one XML file in another XML file, much like CSS tells the web browser how to represent HTML elements on the screen. Some programming logic can be applied, but changeable variables are not possible. Because of this, XSLT can not handle the transformation of the GML geometry representation into the X3D representation. This therefore had to be done in Python/Java. This is where most of the programming occurred.
After parsing and validating the GML document and transforming it into a memory resident object, this object is processed. Each GML feature is taken, and its geometry read. For each feature a new element is created called “geometry”. In this element the X3D way of expressing geometry is put. If a GML feature defines several types of primitives the first primitive that is defined for the feature in the list from volume/face, line, point is taken. Volume and face are put together since they are the same to X3D: a volume is just a list of faces. When this extra geometry element is added to all the GML features, the memory resident representation is passed to the XSLT transformer and it is transformed into an X3D file. The XSLT file had therefore to be constructed with the new GML representation in mind, not the original GML file. The intermediate GML version can be written to an actual file, but this is not necessary.

Output

The output consists of an X3D file that is displayed by an X3D viewer. Like I said before, the X3D viewer should provide the functionality to interact with the scene. The BS Contact browser provides a user friendly way to navigate through the scene and to interact with it. The objects in the scene can be translated and rotated around the viewpoint, but the viewpoint can also be translated and rotated through the scene. The first is called examining the scene, and the other is called flying, walking, panning, etc. Jumping between different viewpoints is possible, as well as linking to external documents for extra information. The Xj3D browser has less user friendly movement capabilities. All other features are present though.
When using the geospatial component of X3D, the BS Contact viewer reported that the elements belonging to the geospatial component were unknown, while the Xj3D viewer didn’t have this problem. The objects in the scene were hard to find though, since it seemed to put the initial viewpoint at a completely different place or orientation within the scene than the actual objects when the viewpoint was not defined. Defining the viewpoint by using the geospatial component is still a problem since the geoViewpoint element is not completely supported yet. Both browsers therefore have to develop their X3D full profile support further in order to use them in a web based 3D GIS.

Conclusions, problems and restraints

The visualisation of the GML dataset in X3D was successful. Most of the elements in the description of the 3D scene in figure 3.3 was added or could easily be added to the scene. Three problems arose during development that I were solved but may need other approaches in the future.

The first problem is that geo-referencing the objects in the 3D scene was not possible yet since the X3D viewers have no or limited support for the geospatial profile of X3D. This will change in the near future though, so this could be tried again when realistic geo-referenced 3D GML datasets are available and viewers support the geospatial profile fully.

A second problem that arose during the development was the problem of differences in geometry syntax between GML and X3D. This was solved with an intermediate script in Python/Java that reads the geometry from the GML file and stores it in an intermediate GML representation in the geometry syntax of X3D which is then used in the XSLT transformation.

Another problem that arose in developing the visualisation module is about the different coordinate systems used in GIS and computers. Traditionally GIS uses 2D data, where logically the x and y axes are used for positioning. When adding a third dimension, this will then logically be the z-axis. The gaming world based its axes on mathematics. There a 2D coordinate system is used in graphs with x and y axes as respectively the left-to-right and top-to-bottom axes. The z-axis then becomes the front-to-back axis. Since the development of 3D visualisation in computers is mainly done through computer game development, the y-axis has become the up direction in 3D scenes on computers, while in GIS the z-axis is the up direction. Fortunately, the geospatial component of X3D handles this problem. It is possible to set a rotateYUp property to true, so the geo-referenced y and z coordinates are swapped. Since the geospatial component of X3D is not fully supported though, this same functionality had to be added manually. This is possible though by adding a transform element to the X3D scene, which rotates all elements in 90 degrees.
Since these three problems can be solved or will be solved by others, the GML to X3D conversion is a promising environment to develop further in GIS.

6. Conclusions, discussion and recommendations

Three research questions were formulated in the introduction of this report:

1. What basic components should be present in a web based 3D GIS?
2. Which Open Standards and Open Source Software exist to be used in such a web based 3D GIS development environment?
3. Is the 3D visualisation of a 3D dataset possible with such a framework, using existing OS and OSS?

The final conclusions and discussion of these questions will be made in the following three paragraphs, which correspond to the three research questions. I will end with recommendations for further development and further research.

Components of a webbased 3D GIS

Before investigating OS and OSS and starting development, it is important to define the functionality that should be provided by a 3D webbased GIS. Figure 2.1 defines the functionality that is generally provided by a GIS. Three main functional parts can be distinguished: data management, data analysis and data visualisation. Data management is not a major use of a webbased GIS as intended in this study. The only component that is really important is the issue where the data comes from. The WFS is a useful service that is able to make data from many different sources on the internet and in many different formats available in the GML format (Peng, 2005;Peng and Zhang, 2004;Boucelma *et al.*, 2002;Specification: Beaujardiere, 2004). Supporting geo-data stored in GML files therefore allows a webbased GIS to use data from many different sources. Other data management tasks should be done using existing GIS software. Data analysis is more important in webbased dedicated GIS. Especially querying is important. Visualisation, and in this case more specifically 3D visualisation, is the main part of the dedicated webbased GIS that has been investigated in this study. As described in figure 3.3 the 3D scene can provide an interface to analysis functionality of the data and of course, it provides the visualisation functionality of the GIS. Six main themes important for visualisation, as described in figure 3.3, are the realism of the scene, the movement functionality, how users can orientate themselves inside the scene, how users can navigate through the scene, how the data displayed in the scene is explained to the user and how objects are (geo-)referenced inside the scene.
Comparing figure 2.1 and 3.3 with the OGC portrayal model as described in figures 3.1 and 3.2, one can see that especially the renderer is important in a webbased 3D GIS. The renderer is the component that should address the visualisation issues. It can also be an interface to the analysis functionality provided by the GIS, although that can also be programmed in a separate interface. The technology used for the visualisation should therefore natively provide much of the visualisation functionality and the other parts should be easy to develop.

Open standards and open source software

The functionality to be provided by a dedicated webbased 3D GIS can be realised using several open standards and existing open source tools. Based on the XML standard the GML and X3D standards are very promising standards. Since they are XML based standards the possibility exists to use the XML family standards and existing XML tools with them.

The strict grammar rules of GML and the XML family standards like XSLT and XSD enable XML based file formats to be converted into each other easily without loss of data. XML based file formats also improve interoperability since XML files are text-based and the file structure is documented in XSD or DTD files. This enables a software developer to write software compatible with the file format without additional information from the original author of the file.
Apart from useful standards, a lot of (open source) tools and software exists that is able to interpret, manipulate and convert XML based files. Examples are the Apache Cocoon, Apache Xalan and Apache Xerces projects and programs like Stylus XML Studio and Altova's XML Spy.

GML is an XML based open standard that defines a framework for storing geo-data. Apart from being an XML based format another advantage is that it is very flexible, which allows it to be used for many different data models. It is also a very complete specification, defining 2D geometry, 3D geometry, 2D and 3D topology, time elements, metadata elements, etc. It is not a ready for use specification though. When using GML datasets, application specific implementations of the GML specification have to be written in order to be able to describe useful datasets.

X3D is an abstract specification that defines visualisation and interaction elements for 3D scenes. It defines several file encodings, among which is an XML encoding.
Apart from being able to use the XML family standards and XML tools on files encoded in the XML encoding of X3D, another advantage of X3D is its modular setup. This enables the addition of new components to the X3D specification quickly. Some very useful components already exist, like a geospatial component. This enables the use of geo-referenced coordinates directly in X3D scenes, using several different spatial reference systems. It also solves the problem of the different axis definitions in GIS and the computer business.
X3D natively provides much of the functionality as described in figure 3.3. The internal scripting functionality, interaction functionality and SAI enable the addition of parts, which are not natively included like intelligent objects, predefined navigational routes and small maps. This therefore makes X3D very useful as a visualisation standard for webbased 3D GIS.
Several viewers exist to display X3D scenes, some commercial and some open source. The open source Xj3D project is the most promising project, which includes a viewer but also implements the X3D SAI. The BS Contact viewer is the most user friendly and feature rich viewer at this time. Both are working on support for the geospatial component of X3D, but because the X3D specification is relatively young, support for the geospatial component is not ready yet. This will be solved in the near future.

The prototype visualisation module

By developing a prototype for a visualisation module, it has been proven that the approach to visualising 3D geo-data on the web, by using the GML and X3D open standards, is possible. Only a few problems arose and these either were overcome or will be solved when X3D viewers have full support for the geospatial component of X3D. One problem will have to be addressed in further research though. This addresses the difference in syntax between GML and X3D for 3D features. While GML features can have very complex geometries where it is possible for one feature to contain all of the point, line, face and volume primitives at once, in X3D it is only possible for a feature to be either a point, line or face/volume feature (a volume is just a collection of faces in X3D). In GML these primitives are all stored in separate elements

within the feature they belong to, while in X3D each feature can contain only one geometry element defining the complete geometry for the feature. It is impossible to convert these different geometry syntaxes into each other using standard XML tools and standards. This had to be done programmatically therefore. This was not very hard, but since GML files are mostly application specific implementations of the GML specification, the conversion of the geometry developed in this prototype is only useful for this application specific implementation of GML. This restriction can be handled in three different ways.
The first way is to accept this limitation and specify that a requirement for a webbased GIS based on GML and X3D is that it will require an adaptor for each application specific GML implementation. Since the script is quite small, this may not be a problematic requirement.
The second way to handle it is to develop this script more robustly so it will search through the whole GML file to look for all geometry elements and try to determine which feature it belongs to, so all geometry can be combined into the geometry syntax of X3D. This may be quite tricky since the diversity in ways to implement GML is very high, but it may be possible.
The last way is to rewrite the script so it is able to interpret the application specific GML XSD file. This way, the script may be able to determine the structure of the GML file and handle the geometry syntax transformation accordingly. It may even be possible to generate the XSLT file this way, if certain properties that describe the appearance of the features are given to the script. Whether this is possible will have to be researched.

Another issue encountered with the development is with the performance of the module. Text-based file formats are generally larger in size than binary file formats. This may cause higher hardware requirements on computers handling text-based file formats. Since users tend to have quite strong pc's nowadays, it may be worth it to put some of the data processing on the client's computer using a thick client (figure 3.2). A disadvantage is that users will have to install software on their system. Performance could also be increased by using binary formats. For X3D, a binary encoding already exists (Specification: Web3D Consortium, 2005), and for XML in general, investigations are being done in this area (Bayardo *et al.*, 2004;Specification: Martin and Jano, 1999;URL: Expway, 2005). A second issue with performance is with the way this prototype was developed. It was developed in the Python programming language, by making use of the Jython interpreter, which is able to import Java classes into Python by compiling the Python sourcecode to Java bytecode instead of Python bytecode. Because two interpreters (virtual machines) are needed (the Python and the Java interpreters), this is slower than using native Python or Java. It did increase the speed of development though because Python allows for quicker programming and many API's exist in Java for XML handling. For production use, the code may have to be rewritten in native Java or Python, or even C++.

Further research and development

Further development in this area should be focussed at moving the visualisation into a webbased production environment, and fitting it into a specific client-server model (figures 3.2 and 4.2). Performance and portability should also be addressed. With the latter, one should address the issue of different application specific GML implementations, and one should look at other data sources to support, either by interfacing with WFS or by supporting other file formats natively. It would also be interesting to participate in the development of the Xj3D project in order to have influence on the speed and direction of development of the project and be able to incorporate the software in a 3D GIS.

A very interesting subject for further research would be the modelling and data manipulation possibilities of the GML-X3D combination. The SAI of X3D allows the real-time manipulation of the scene. This could be used to visualise the effect of models on a 3D environment in real-time. This could be done by developing a modelling module that interfaces with the SAI and an interface to the module to adjust model parameters. Another possibility is to enable the real-time modification of the 3D scene by using the 3D scene as the interface. It may for instance be possible to add objects to the scene in real-time and visualise the effects of the new object on the scene using a model.

Another interesting subject for further research is the representation of the data in the 3D scene. How should geometric, topologic and thematic data be represented in the scene? Thematic data could be visualised within the scene by using visual properties, but one could also use intelligent objects or links to external sources. Should topologic information be defined explicitly or is our ability to see it for ourselves in the 3D scene enough? And is the use of real 3D data, like with CAD files, realistic or should an effort be made to create a library of realistic 3D symbols that could be used directly in 3D scenes?

Appendix A: Abbreviations and Glossary

Abbreviations

API	Application Interface
EAI	External Authoring Interface
GIS	Geo-Information System or Geo-Information Science
HTML	HyperText Markup Language
GUI	Graphical User Interface
OGC	Open Geospatial Consortium
ORDBMS	Object- Relational Database Management System
OS	Open Specification
OSS	Open Source Software
GIS	Geographic Information System
RDBMS	Relational Database Management System
SAI	Scene Access Interface
SDTS	Spatial Data Transfer System
SQL	Structured Query Language
SVG	Scalable Vector Graphics
VRML	Virtual Reality Modelling Language
W3C	World Wide Web Consortium
W3DC	Web3D Consortium
WFS	Web Feature Service
WMS	Web Map Service
XML	eXtensible Markup Language
XSLT	XML Stylesheet Language Transformation

Glossary

API	An API is a set of functions or classes, written in a certain programming language, written to provide some functionality to other applications. The goal of an API is not to provide a usable program, but to provide functionality to other programs. An example of an API is the Java3D API which defines java classes with which 3D objects can be rendered.
EAI	The VRML97 version of what's called the SAI in X3D. *See SAI.*
GML	This is an OGC standard. GML is an XML based standard to store geo-information.
HTML	The primary file format for webpages.
OGC	The Open Geospatial Consortium. They developed standards like GML, WFS and WMS.
SAI	The Scene Access Interface as defined by the X3D specification. It specifies a platform and programming language independent interface to the X3D scene. This allows for the real-time manipulation of the X3D scene from programs outside the 3D scene. In the X3D specification also Java and ECMA script language bindings have been formulated separately from the SAI specification.

SQL	SQL is a language used by many databases to extract data from the database (URL: Wikipedia, 2005).
SVG	SVG is a W3C standard. SVG is an XML based format to store 2D vector based images. The use of an XML based vector image format makes it possible to zoom in, pan through and interact with the graphic without quality loss.
VRML	A Web3D Consortium standard file format to display and interact with a 3D scene.
W3C	The World Wide Web Consortium. They developed open standards like HTML and XML.
WFS	The Web Feature Service is a web service that transparently provides geo-information from different sources to clients. The client does not know what database or file format the data comes from.
X3D	A Web3D Consortium standard. X3D is an XML based format to store 3D objects for visualisation and interaction.
XLink	The XML Linking Language, or XLink, is an XML markup language used for creating hyperlinks within XML documents. It is a W3C standard (URL: Wikipedia, 2005).
XML	The Extensible Markup Language (XML) is a W3C-recommended general-purpose markup language for creating special-purpose markup languages. Its primary purpose is to facilitate the sharing of data across different systems, particularly systems connected via the Internet. Languages based on XML are defined in a formal way, allowing programs to modify and validate documents in these languages without prior knowledge of their form (URL: Wikipedia, 2005).
XPath	XPath (XML Path Language) is a terse (non-XML) syntax for addressing portions of an XML document. Originally motivated by a desire to provide a common syntax and behavior model between XPointer and XSL, XPath has rapidly been adopted by developers as a small query language (URL: Wikipedia, 2005).
XQuery	Is a W3C standard. It specifies a query language (with some programming language features) that is designed to query collections of XML data (URL: Wikipedia, 2005).
XSLT	XSL Transformations, or XSLT, is an XML-based language used for the transformation of XML documents. The original document is not changed; rather, a new XML document is created based on the content of an existing document. The new document may be serialized (output) by the processor in standard XML syntax or in another format, such as HTML or plain text. XSLT is most often used to convert data between different XML schemas or to convert XML data into web pages or PDF documents (URL: Wikipedia, 2005).

Appendix B: References

Literature

Altmaier, A. and Kolbe, T. H. (2003), *Applications and Solutions for Interoperable 3d Geo-Visualization*, Proceedings of the Photogrammetric Week 2003,

Anderson, G. and Moreno-Sanchez, R. (2003), *Building Web-Based Spatial Information Solutions around Open Specifications and Open Source Software*, Transactions in GIS, **7,** 447-466.

Bayardo, R. J., Josifovski, V., Gruhl, D.*, et al.* (2004) *An evaluation of binary XML encoding optimizations for fast stream based XML processing*, at conference: Thirteenth International World Wide Web Conference Proceedings, WWW2004,

Boucelma, O., Essid, M. and Lacroix, Z. (2002), *A WFS-based mediation system for GIS interoperability*, Proceedings of the ACM Workshop on Advances in Geographic Information Systems, 23-28.

Carman, G. J. and Welch, L. (1992), *Three-dimensional illusory contours and surfaces*, Nature, **360,** 585-587.

Groetelaers, D. A. (2002), *Visuele interactie in 3D-GIS & Virtual Reality*, Geodesia, **44,** 28-31.

Kraak, M.-J. (2003), *Geovisualization illustrated*, ISPRS Journal of Photogrammetry and Remote Sensing, **57,** 390-399.

Lake, R., Burggraf, D. S., Trninic, M.*, et al.* (2004) *Geography Mark-up Language: Foundation for the Geo-Web,* John Wiley & Sons, Chichester, West Sussex, England.

Lammeren van, R. and Hoogerwerf, T. (2003) *Geo-Virtual Reality and participatory planning*, Vol. Centre for Geo-Information, Wageningen University and Research Centre, Wageningen, pp. 61.

Mc Laughlin, B. (2001) *Java & XML,* O'Reilly, Sebastopol, CA, USA.

Parnas, D. L., Clements, P. C. and Weiss, D. M. (1984), *MODULAR STRUCTURE OF COMPLEX SYSTEMS.*, IEEE Transactions on Software Engineering, **SE-11,** 259-266.

Peng, Z.-R. (2005), *A proposed framework for feature-level geospatial data sharing: A case study for transportation network data*, International Journal of Geographical Information Science, **19,** 459-481.

Peng, Z.-R. and Zhang, C. (2004), *The roles of geography markup language (GML), scalable vector graphics (SVG), and Web feature service (WFS) specifications in the development of Internet geographic information systems (GIS)*, Journal of Geographical Systems, **6,** 95-116.

Polys, N. F. (2002) *Stylesheet transformations for interactive visualization: Towards a Web3D chemistry curricula*, at conference: Web3D Symposium Proceedings,

Quak, W., Vries de, M., Thijssen, T.*, et al.* (2002) *GML for exchanging topographic data*, at conference: 5th AGILE Conference on Geographic Information Science, Palma, Spain.

Wang, X. (2005), *Integrating GIS, simulation models, and visualization in traffic impact analysis*, Computers, Environment and Urban Systems, **29,** 471-496.

Specifications

Beaujardiere, J. d. L. (2004), *OpenGIS® Web Feature Service Implementation Specification*, accessed: 19-5-2005, http://portal.opengeospatial.org/files/?artifact_id=5316.

Bell, G., Parisi, A. and Pesce, M. (1995), *The Virtual Reality Modeling Language - Version 1.0 Specification*, accessed: 28-10-2005, http://www.web3d.org/x3d/specifications/vrml/VRML1.0/index.html.

Boag, S., Chamberlin, D., Fernández, M. F.*, et al.* (2005), *XQuery 1.0: An XML Query Language*, accessed: 12-10-2005, last updated: 15-9-2005, http://www.w3.org/TR/xquery/.

Bray, T., Paoli, J., Sperberg-McQueen, C. M.*, et al.* (2004), *Extensible Markup Language (XML) 1.1*, accessed: 21-4-2005, last updated: 15-4-2004, http://www.w3.org/TR/2004/REC-xml11-20040204/.

Clark, J. and DeRose, S. (1999), *XPath*, accessed: 12-10-2005, last updated: 13-3-2000, http://www.w3.org/TR/xpath.

Cox, S., Daisey, P., Lake, R.*, et al.* (2003), *OpenGIS® Geography Markup Language (GML) Implementation Specification*, accessed: 21-4-2005, https://portal.opengeospatial.org/files/?artifact_id=7174.

DeRose, S., Maler, E. and Orchard, D. (2001), *XLink*, accessed: 12-10-2005, last updated: 27-6-2001, http://www.w3.org/TR/xlink/.

Martin, B. and Jano, B. (1999), *WAP Binary XML Content Format*, accessed: 12-10-2005, last updated: 24-6-1999, http://www.w3.org/TR/wbxml/.

Open Geospatial Consortium (2003), *OGC Reference Model*, accessed: 22-9-2005, http://portal.opengeospatial.org/files/?artifact_id=3836.

Raggett, D., Hors Le, A. and Jacobs, I. (1999), *HTML 4.01 Specification*, accessed: 19-5-2005, http://www.w3.org/TR/1999/REC-html401-19991224/.

Reddy, M. and Iverson, L. (2002), *GeoVRML 1.1 Specification*, accessed: 23-10-2005, http://www.geovrml.org/1.1/doc/.

Vretanos, P. A. (2005), *OpenGIS® Web Feature Service Implementation Specification*, accessed: 19-5-2005, https://portal.opengeospatial.org/files/?artifact_id=8339.

Web3D Consortium (1997), *VRML97 Functional specification and VRML97 External Authoring Interface (EAI) International Standard ISO/IEC 14772-1:1997 and ISO/IEC 14772-2:2002 ISO/IEC 14772-1:1997 and ISO/IEC 14772-2:2002*, accessed: 28-102-2005, http://www.web3d.org/x3d/specifications/vrml/ISO-IEC-14772-VRML97/.

Web3D Consortium (2005), *All X3D Specifications and Encodings*, accessed: 28-10-2005, http://www.web3d.org/x3d/specifications/X3DPublicSpecifications.zip.

Websites

Altova (2005), *Altova – XML, Data Management, UML, and Web Services Tools*, accessed: 28-11-2005, http://www.altova.com/.

Apache Xalan Workgroup (2005), *The Apache Xalan Project*, accessed: 28-11-2005, http://xalan.apache.org.

Apache Xerces Workgroup (2005), *The Apache Xerces Project*, accessed: 28-11-2005, http://xerces.apache.org.

Bullard, L. (2003), *Extensible 3D: XML Meets VRML*, accessed: 23-10-2005, http://www.xml.com/lpt/a/2003/08/06/x3d.html.

DataDirect Technologies (2005), *Stylus Studio – XML Editor, XML Data Integration, XML Tools, Web Services and XQuery*, accessed: 28-11-2005, http://www.stylusstudio.com/xml_feature_overview.html.

Expway (2005), *BinXML™*, accessed: 12-10-2005, http://www.expway.com/binxml.php.
Jython Project (2000), *Jython*, accessed: 2-12-2005, last updated: 20-6-2005, http://www.jython.org.
Kay, M. (2005), *The SAXON XSLT and XQuery Processor*, accessed: 28-11-2005, last updated: 24-11-2005, http://saxon.sourceforge.net/.
Open Geospatial Consortium (2005), *Open Geospatial Consortium Schemas*, accessed: 3-12-2005, http://schemas.opengis.net/.
Tecfa (2005), *Tecfa VRML examples*, accessed: 28-10-2005, http://tecfa.unige.ch/guides/vrml/examples/.
The Apache Cocoon Project Management Committee (2005), *The Apache Cocoon Project*, accessed: 24-10-2005, last updated: 14-10-2005, http://cocoon.apache.org/.
The IrishSpace Project (1997), *The Irishspace Project: The Voyage of the Jeanie Johnston II*, accessed: 28-10-2005, last updated: 11-5-2005, http://www.digitalspaceart.com/irish/index.html.
The Mind Project (2003), *Introduction to the Science of Vision*, accessed: 16-12-2005, http://www.mind.ilstu.edu/curriculum2/perception/introvision.html.
Web3D Consortium (2005a), *Why use X3D over VRML 2.0? Here are 10 compelling reasons.*, accessed: 15-4-2005, last updated: 1-2005, http://www.web3d.org/x3d/x3d_vs_vrml.html.
Web3D Consortium (2005b), *X3D Overview*, accessed: 28-10-2005, http://www.web3d.org/x3d/overview.html.
Wikipedia (2001), *The Free Encyclopedia: XML*, accessed: 12-10-2005, last updated: 11-10-2005, http://en.wikipedia.org/wiki/Xml.
Wikipedia (2005), *The Free Encyclopedia: Markup language*, accessed: 29-9-2005, http://en.wikipedia.org/wiki/Markup_language.