Establishing Methods of 3D City Modeling based on Multiple Data Sources

Ludvig Ljungqvist

Master of Science Thesis

Department of Design Sciences
Lund Institute of Technology

in corporation with:

GIS Centre
Lund University, Sweden

ISRN: LUTMDN / TMAT-5064-SE

Lund, 2003

Title:
Establishing Methods of 3D City Modeling based on Multiple Data Sources

Author:
Ludvig Ljungqvist

Supervisors:
Dr. Lars Harrie, Department of Real Estate Sciences, Lund Institute of Technology
Dr. Roy Davies, Department of Design Sciences, Lund Institute of Technology

Examinator:
Dr. Roy Davies, Department of Design Sciences, Lund Institute of Technology

Number of Pages:
102

Keywords:
3D city models, DEM, Photo-realistic Visualization, GIS, 3D-GIS, Texturing,
3D data acquisition, Planning, Virtual Reality

Project Website:
http://www.reflex.lth.se/gis/3Dgis/

TYPESET: Palatino Linotype 11p Heads: Frutiger 45 Light

ISRN LUTMDN / TMAT-5064-SE

Department of Design Sciences
Division of Ergonomics and Aerosol Technology
Lund University

P.O. Box 118
SE-22100 Lund
Sweden

http://www.eat.lth.se
Telephone: +46 46 222 8018
Fax: +46 46 222 44 31

Preface

This Master of Science thesis was produced in a corporation project between the GIS centre and the flexible reality centre at Lund University during the summer and fall of 2003. The thesis is a part of the requirements for a 4.5 year Master of Science degree in Survey Engineering at Lund Institute of Technology (LTH). Supervisors were Dr. Lars Harrie at the GIS centre and Dr. Roy Davies, the flexible reality centre. The work was carried out at the Reality Lab, at Ingvar Kamprad Design Center, Lund.

I would like to thank Roy Davies and Lars Harrie for the technical support and for comments on the report. I would also like to thank Joakim Eriksson at the Flexible Reality centre for his support and contributions to the created model. I want to thank all of the students, working with other masters projects at the Reality Lab, especially Staffan and Björn for comments on the report and Kostas and John for the initial help with the 3D software. The integration between this project and Staffan's car simulator created a wonderful VR experience. Special thanks go to Staffan for that opportunity. Finally, some thanks are directed to Kristina Emgård for being so lovely.

Lund, November 2003

Ludvig Ljungqvist

Abstract

The objective of this Master Thesis is to establish a method for creating computerized 3D city models. The methods are meant to be appropriate for visualization in the planning process. Today, there is an increasing need and interest in 3D city models and many methods and much software is developed. The usage and applications of 3D city models can be found in for example urban planning, tourism, and environmental calculations. The main task accomplished in the project was to investigate and create semi-automatic methods for generation of 3D data from 2D GIS data. The methods were evaluated by modeling Lundagård, a park area in central Lund.

The data material used in the project consisted of 2D polyline GIS layers, 3D elevation points, aerial photographs and a simple 3D representation (box model) extracted from the aerial photographs. Also terrestrial photographs were acquired and used for texturing. The 3D elevation points were used to generate a digital elevation model of the ground. This was achieved by the integration of the 2D polylines and the elevation points within a constraint Delaunay triangulation. The result was a classified multi-textured surface object. The buildings where constructed by enhancing the box model with roof structures, texturing and remodeling of advanced building parts. For an extended visual impression, additional single objects were added. Trees and streetlights were distributed on proper locations obtained from the GIS data. Finally the model was rendered in a daylight environment. It was also exported to a virtual environment.

Together with the created model, the project resulted in proposals of methods to be used in large 3D city models, based on both evaluated and studied methods. It is today possible to create large city models, within the frames of reasonable cost and time, using the knowledge obtained in this project. The main conclusions were that the elevation data has to be detailed for a proper elevation model to be created and that the GIS data could be managed in another way for simplified classification of the elevation model. For easier production of 3D city models, the available data material would have to be complemented with tree age, size and species and the elevations of buildings. Another important conclusion is that today every step of the modeling process, except the terrestrial photographing, could be managed in an almost automatic manner. This thesis finally, addresses the need for further research within the area of updated 3D-GIS databases.

Swedish abstract

Syftet med detta examensarbete är att fastställa en metod för att skapa datoriserade 3D-stadsmodeller, som kan användas för visualisering i planeringsprocessen. Det finns idag ett ökat behov och intresse av 3D-stadsmodeller, och mycket mjukvara och metoder har utvecklats. Tillämpningar och användning av 3D-stadsmodeller finns bl. a. inom fysisk planering, turism och miljöberäkningar. Huvuduppgiften i projektet var att undersöka och skapa halvautomatiska metoder för generering av 3D-data från 2D-GIS-data. Metoderna utvärderades genom modellering av Lundagård, ett parkområde i centrala Lund.

Datamaterialet som användes i projektet bestod av 2D-polylines, 3D-höjdpunkter, flygfoton, och en enkel representation (box-modell) som extraherats ur flygfoton. Dessutom användes markfotografier som togs i området för texturering. 3D-höjdpunkterna användes för framställning av en digital höjdmodell för att representera markytan. Höjdmodellen skapades genom integration av 2D-polylines och höjdpunkterna i en begränsad Delaunay-triangulering. Resultatet var ett klassificerat multi-texturerat ytobjekt. Byggnaderna konstruerades genom vidareutveckling av box-modellen som innefattade takbyggnad, texturering och ommodellering av invecklade byggnadsstrukturer. För att utöka det visuella intrycket lades ytterligare objekt till. Träd och gatlyktor placerades på positioner hämtade från GIS-data. Slutligen, renderades modellen i dagsljus och exporterades även till en virtuell miljö.

Projektet resulterade, tillsammans med den skapade modellen, i förslag på metoder, baserade på både utvärderade och studerade metoder, som kan användas i skapandet av större 3D-stadsmodeller. Med kunskapen som erhållits under detta projekt är det idag möjligt att skapa stora 3D-stadsmodeller till en rimlig kostnad och inom en rimlig tidsram. De huvudsakliga slutsatserna var att höjddata måste vara detaljerad för att en höjdmodell ska kunna skapas och att GIS-data behöver omstruktureras för att förenkla klassificering av höjdmodeller. För vidare förenkling i skapandet av 3D-stadsmodeller behöver det tillgängliga datamaterialet kompletteras med träds ålder, storlek och art och även byggnaders höjder. En annan viktig slutsats är att varje steg i modelleringsprocessen utom markfoton kan göras nästan helt automatiskt. Slutligen, utlyser examensarbetet behov av vidare forskning inom området för beständiga 3D-GIS-databaser.

Table of Contents

PART I – INTRODUCTION

PART II – THEORY

Appendices

Part I
Introduction

1 Introduction

1.1 Background

Virtual Environments (VEs) have several applications and there is an increasing need for modeling reality. Models of city architecture are today used for purposes such as planning, construction and visualization. This Master of Science thesis concentrates on methods used for creation of city models used for urban planning. The area of three dimensional representations of buildings is interdisciplinary between several fields of research. Architects, planners, designers and geographical information-oriented scientists are interested in the usage of models, while researchers within computer graphics and mathematics are involved by interest in graphical rendering, topologic structures and methods of data capturing.

The planning process of the urban environment involves different people from many fields of interest, and VEs can be used as a tool in the planning process. When planning comprehensive plans and detail-plans, several disciplines are included, such as: geographers, economists, environmental scientists, public administrators, etc. Conventional paper-based presentations are limited and drawing conventions isolate participants from each other (Clayden and Szalapaj, 1997). Computerized 3D models within virtual environments are a solution for efficient communication between these planning participants and between planners and public (Li and Lin, 2002). To date, much research has been devoted to the area, but the usage of virtual reality techniques is not wide spread. Some of the major cities in the world are partially modeled but the graphic environments differ considerably in level of detail and amount of information connected to the graphic representations.

Among many other similar institutions, the municipal department of city planning in Lund is interested in different ways of presenting detail-plans and comprehensive plans to the public. Especially, better ways of visualization are required. There is also an interest in to what degree the existing data at the department could be used for 3D modeling (Åkerholm, 2003).

1.2 Objectives

The main objective of this Master of Science thesis is to:

Establish a method for creating computerized 3D city models that is appropriate for visualization in the planning process.

More specifically the aims are to:

- create a minor city model of Lundagård, using available data,
- establish efficient methods to collect accurate data for a 3D city model, and
- find an appropriate method for viewing the 3D city model.

1.3 Limitations

The area of the practical studies in the project is limited to a small urban park area (Lundagård, Lund). Available data to be used in the project is limited to four sources: aerial photographs, a planar 3D solid model of buildings, two-dimensional GIS data and elevation points covering parts of the area (The available data is described in detail in section 7.1). The acquisition of new data is limited to terrestrial photography. The project concentrates on 3D city modeling and thus, the VR applications have not been fully investigated.

1.4 Methodology

The main methods used in this thesis were:

- literature studies of technology and research in the area of 3D city models,
- evaluation of some methods for collecting data,
- evaluation of some methods for manual and automatic modeling, and
- evaluation of methods for displaying the model.

1.5 Report structure

The rest of this document is separated into two major parts; Part II - Theory and Part III - Project.

- Part II – Theory

The theory part (chapters 2-5) presents theory within 3D city models based mainly on literature studies of scientific articles. Chapter 2 is a brief introduction to 3D city models and describes terminology, history and applications. Chapter 3 then presents the data used to create 3D city models; what data is required to construct a model, what data is available to date and how it is collected. Chapter 4 presents some methods of operation of available data and how it could be developed into components of 3D city models. Chapter 5 explores some common data formats and computer software used for 3D modeling.

- Part III – Project

The project part (chapters 6-12) presents practical investigations completed by the author and the results of the task. Chapter 6 introduces the project and Chapter 7 presents the base material available in the study. In Chapter 8 the different methods for creation of the model are described in detail. How the model finally appeared can be found in Chapter 9 together with the proposed method for creating similar, but larger models. Chapter 10 includes the discussions of the project.

1.6 Knowledge entrance requirements

This report is primarily intended for readers with knowledge in geometric data structures and basic data management. A complete understanding of the used terminology requires knowledge in data representations of geometric shapes, multi dimensional space and basic technical terminology. The target audience of the thesis is primarily people with special interests in spatial data (maps or 3D models). For such persons, it may provide valuable guidance to the creation process within 3D city models

Part II
Theory

2 Introduction to 3D city models

This chapter aims at giving the reader a short introduction to 3D city models, history and applications of the subject and an introduction to the concept of level of detail.

2.1 Definitions and terminology

There are various kinds of representations of urban environments. These may be divided into navigable and non navigable city models. Navigable models can be further divided into photo realistically (fig. 2.1) and symbolically represented 3D city models. A 3D city model is a computerized model of a city. The terminology of such models is not fixed. For example, *3D GIS* or *VR GIS* are common examples of expressions which tend to refer to a 3D model or environment where the topological relationships and functionality from a regular 2D GIS are preserved. Other terms like *virtual city*, *cybertown*, *cybercity* or *digital city* are employed on the Internet for a diverse range of information interfaces and content (Dodge *et al.*, 1998). Expressions containing the word *virtual* or *VR*, like *VR city model* or *virtual environment*, often refer to models with a certain degree of usability and requirements of for example interaction. A virtual environment is the interface where the users are able to navigate in a three-dimensional world containing the 3D city model.

Figure 2.1. A photo-realistic 3D city model of parts of Zürich, Switzerland by Cybercity (CyberCity, 1998).

Since this Master of Science thesis treats a wide range of representations, the best suited expression to use is *3D city model* when referring to the model, and *virtual environment* when referring to the entire user interface. As mentioned, the different terms above describe models of varying requirements of functionality and representation. The scope of this theory and project is therefore set on a certain kind of model. The minimum requirements for such models and environments further treated in the thesis are presented below:

The graphic representation of buildings and other objects in the model should be in 2.5D (see explanation below) or 3D. The objects should be created in form of volumes built up by planar faces. A model of the ground should be represented by a surface. Furthermore there should be a possibility to navigate through the model by walking or flying in the virtual city to examine objects from all angles (Dodge *et al.,* 1998). The model should cover an urban area, big enough to create an environment to move around in (larger than architectural models covering a few buildings for visualization of a small area). (Definition by the author)

These requirements exclude some less navigable forms of VR like QuickTime VR (Zhang, 2000) which is an image-based panorama representation. Those kinds of models visualize the area around the camera spot with photorealistic impression but are not navigable. The *feeling of being there* is thus limited by constraining the user to a pre-defined point (Sequerira, 1999). Other approaches that are not to be considered as VR are text and image based representations or icon based 2D pictures flipped to a 3D-perspective (Dodge *et al.,* 1998).

The term *2.5D* is used for describing models where there is only one unique Z-value (elevation value) defined for each pair of XY-coordinates (Sinning-Meister and Gruen, 1996). In 2.5D, spatial objects reach out in three dimensions, like for example buildings on the ground. However, since there is already a z-value defined for each XY-coordinate, a roof extending out from a wall can not be described. In this Master of Science thesis the term 3D is used for both true 3D and 2.5D. The term 2.5D is also used in cases when needed.

2.2 Historic view

The computer development in the second half of the 90's has made rendering of photo-realistic environments possible. 3D city models used in virtual environments came into existence after 1995 when the development in computers, in terms of raw power and graphics hardware performance made it possible to render images in less than a second (Bourdakis, 1997). By 1998 a few models with textured façades where available as screenshots on the Internet, still rendered on powerful machines in research labs. Models available online were seldom compiled using accurate GIS data and generally only covered small parts of the city (Dodge *et al*., 1998). Today many institutions all over the world are working with the development of more advanced 3D city models. For example, data of many major Japanese cities have been acquired automatically, and some parts have been textured with images (Takase *et al.,* 2003). The central parts of Copenhagen is modeled and textured with proper textures. Around 2001 some models were also made in Sweden. For example, parts of Gothenburg, Umeå and the Bottnia railway were represented by models for different purposes. Today's computer power enables rendering and VR applications even on regular workstations, which makes 3D city modeling available to anyone. By the growing interest and availability in this field, ideas are rising about the future applications of 3D city representations.

2.3 Applications of 3D city models

There are an increasing number of applications of 3D city models in many areas (Brenner and Haala, 1999). Even though requirements were stated above, the models still differs much in usage of the environments. For example, virtual reality offers new and exciting opportunities

for users to interact visually and explore 3D geo-data (Huang and Lin, 2002). The city models can, apart from visualization, be used for calculating environmental hazard scenarios or construction solutions. The degree of additional information connected to the model, like for example attributes connected to buildings differs between solutions. The simplest way is to present a static spatial-only model, without the possibility of querying information from objects, which could be successfully used in plain visualization when for example displaying a new building in an existing environment. Some fields of usage need more metadata to satisfy the user. For example, non-visual attribute data like labels, symbols, hints and memos, can be used to give the user more impressions. Two common ways of using 3D city models are presented below: photorealistic and symbolic representations.

(a) (b)

Figure 2.2. A photo-realistic model of Ginza, Japan (a) (Takase et al., 2003) compared to a more symbolic model of Zürich, Switzerland (b) (CyberCity, 1998).

2.3.1 Photorealistic representations

Photorealistic models (fig. 2.2a) are used to give the user a picture of the world as it actually appears. In a photorealistic environment the user may take a virtual city tour to find out where to find attractions. The ability to locate hotels, restaurants and other tourist oriented and additional information may be added for tourism purposes. Tasks like finding the closest way to an item could be better shown and the location of a shop could be queried by its name (Heinonen *et al*., 2000). These kinds of models could also be used for education in historic environments or in military training. Photorealistic models are often used by architects to convince investors and buyers that a product meets their requirements. Patel *et al*., (2002) have shown that VR technology enhances the communication between designers and clients using a strictly visual VR environment. Architectural models do not usually require attributes connected to buildings. When making decisions, the users rely only on the three dimensional image. This makes the degree of photorealism more important in this case. Photorealistic models are also often used in three dimensional computer games.

2.3.2 Environmental and symbolic representations

Instead of visualizing the world photo-realistically, symbolic representation may be used (fig. 2.2b). In symbolic maps groups of features are symbolized by colors or patterns. A model with mapped information can for example be constructed with colors or patterns covering buildings to present statistics (Bourdakis, 1997). In regular 2D GIS systems, symbolic data is often mapped onto the 2D layers showing for example land use, value, crime rate, traffic or pollution.

Symbolic models are often based on GIS databases as the only source. A 3D city model could be stored in a database based on the same symbolic representation as in the GIS systems. To create the third dimension, the object elevation is added as an attribute to each 2D form (building footprint or other multi-cornered object). The third dimension added to the GIS map may in symbolic models not seem to enhance the visual representation compared to a 2D map. Bourdakis (1997) indicates that the benefit of the third dimension is both the freedom of movement and the lack of scale limitations. The freedom of movement enables the user to view the scene from different viewpoints and camera angles, and the scale freedom makes the user able to zoom more dynamically between large and small scales in the map.

Certain types of data fit better than others in this symbolic representation. Dynamic variables like temperature, pollution and traffic are more beneficial when visualizing, and offer a better real-time experience. In some cases the need for visualization disappears. Then the only needed data is the geometric data. This simple way of constructing models is used for simulations of heat and exhausts spreading in big cities. Furthermore, telecommunication companies also use this kind of models to calculate wave propagation in the urban environment (Koer *et al*., 1996). The simulation of such scenarios does not require information about roof slopes or number of windows on a façade, which makes a simple representation suitable. Similarly, 3D models are used in volumetric calculations within mining and oil industry as well as in water and land mass simulations.

2.4 3D city models for planning

Urban planners traditionally use maps printed on paper to explore urban environments, understand urban problems, and design new urban structures. Professionals within architecture and urban structure are trained to read drawings quickly (Bourdakis, 2001). To enable public participation in planning, other methods would be more suitable. Using 3D models in virtual environments will enable access to urban information more quickly (Li and Lin, 2002). It also opens possibilities for manipulation in real-time and can be used collaboratively by various actors to explore different stages of the construction process (Whyte *et al*., 2000).

The use of 3D city models is, as has been stated, wide spread in several areas. The issue of exposing future plans about urban areas and public interaction are the most discussed issues within research in the area. Therefore, this thesis will from now on narrow to treat only photo-realistic models used for planning and visualization of urban areas. Some interactive features concerning non-geometric data will also be treated.

2.5 Basic description of 3D data terminology

3D models are often constructed from two kinds of data: vector structures and 2D raster images. The vector part describes object geometry by triangular surfaces called *faces* or *polygons*. A great number of triangular faces attached to one and other, called *meshes*, usually describe object volumes or outer surfaces. The visualization of the object's materials is created by attaching 2D raster images to cover one or more vector faces respectively (*texturing*). This data structure is described in detail in chapter 3.

2.6 Level of detail

Level of detail (LOD) is a common expression in computer graphics. The term describes the degree of resolution a model reaches compared to reality. For example, a building modeled with balconies and roof windows has greater LOD than a building described as a simple box. There are many reasons why reality is not modeled as it is perceived by the human eye. Two of them are: 1. Reality partly consists of volumes that could not easily be described with triangular surfaces. The amount of planar surfaces needed for modeling for example a balloon to the LOD accepted by the eye is almost infinite. 2. When rendering a photo-realistic model to an image, the computer power needed for rendering increases with the number of faces. To date, most computers are not fast enough to render large truly photo-realistic models of reality at the speed that is required in a VE (30 frames /second). Also to calculate the amount of light the surface of each face reflects to the viewpoint increases the computational effort.

Different groups of LOD are classified by different scientists. Bourdakis (2001) presents a four step classification tested in his 3D city model of Bath 1997:

- **Level 1**. A simple volumetric description of each building with a flat roof for each terrace. Roads, pavements and landscape areas are also added in.
- **Level 2**. Each building is modeled with accurate wall and roof geometry and tagged as a separate object in the model. Trees are also visible
- **Level 3**. Windows, doors, parapets, party walls and free standing walls are added.
- **Level 4**. Architectural details on buildings like chimney pots, balconies are added.

To get as close to photo-realism as possible designers and researchers have to use various tricks or techniques (Bourdakis, 1997). The classification above is only treating the vector structure of the buildings. A usual trick is to acquire terrestrial photographs of planar items such as façades of buildings. The distance from a wall to the window glass is small compared to the whole façade size. Right in front of the façade the image texture appears realistic, but there is an obvious problem when viewing the façade from a more acute angle. Items sticking out from the building appear as paintings on the wall. Though, to a certain degree this method is used successfully without degrading the impression of viewing the 3D scene. For flat sides of objects the use of textures enhances the perceived detail even in the absence of a detailed model (Brenner and Haala, 1999). This way some of the walls of a building may be modeled in level 2, while special objects like in level 4 will be separately modeled and textured. Highly repetitive buildings (like skyscrapers) benefit more from texturing than for example castles since a part of the texture on a skyscraper can be repeated on the wall (*tiling*). In a large urban environment

with rectangular buildings there is a possibility to randomly attach texture maps from a texture library. Well known buildings, called landmarks, are then produced with more detailed geometry and correct texture mapping (Takase *et al*., 2003). This is an effective method to automate texturing, but it may disappoint users well familiar with the city.

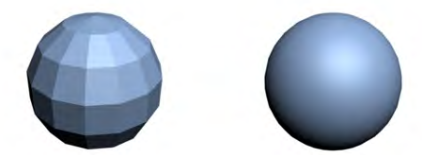


Figure 2.3. Non-shaded compared to shaded sphere.

Other basic and well used tricks involve *shading* and *materials*. For objects where the surface is arched and the whole object is in one color, like for example the pole of traffic signs, there is no reason to use a texture. The object is modeled as a cylinder with a few face segments describing the round pole. The segments are, in the rendering, shaded with colors so that the pole looks circular. This trick is used for all arched objects like balls, cones etc. to avoid an angular appearance (fig. 2.3).

As mentioned, navigating in the 3D city model in a virtual environment raises the requirement for high frame rate. Methods to speed up the rendering are:

- **Culling, clipping and calculation of hidden objects**. – There is no need to render faces in objects that are not visible in the user viewport. Faces in objects outside the view and objects totally behind other faces are not rendered.

- **Reduction of faces –** Objects far away from the user can be rendered with a lower LOD than the ones closer to the user. Some VEs have alternative representations of objects for different LODs automatically alternating when the user approaches the object. Some objects can even be replaced by a single image turning its face against the user while the user moves (billboarding). Some real-time visualization systems even make long distanced buildings disappear to offer online solutions (Beck, 2003).

- **Simplified light calculations –** In most virtual environments objects are given an ambient light similar for all objects. This is one particularly important reason why the virtual environments appear artificial. The problem is partially solved by the computer game industry by pre-calculating the shadows from light sources on textures (radiosity). This works as long as the light source does not move. However, dynamic shadowing is becoming possible on the more advanced 3D graphic cards.

Another factor that has an influence on VE performance is the field of view. Not unlike a change of camera lens the size of the viewport to the VE can be changed. A viewport of 90 degrees gives the user a pleasant experience but increases the need of calculations (Bourdakis, 1997).

2.7 Displaying and using a 3D city model

There are several ways of displaying a 3D city model. One approach is static fully rendered images or movies of the 3D city where the rendering is not limited by time (a computer can be processing one image for hours or days). Effects like shadows, daylight and lightning can in this case increase the visual impression to nearly real images. A movie or picture of planned buildings could this way make a valuable impression on planners. Well rendered movies can constitute a visual attraction but are not open for interaction and freedom of movement for the user. The display of the model therefore often takes place in virtual environments instead. In the virtual environment it is possible to visualize features changing over time and to manipulate the environment while viewing it.

There are three major ways to explore the model in a virtual environment: Walking, flying and examination (Bourdakis, 1997). Walking means moving on the model surface, flying has three dimensional freedom of movement and examination is a view where the entire model is visible in the viewport and the user is free to rotate the object in three dimensions. A 3D city scene can be viewed on a computer screen, projected on a large screen or with head mounted displays (HMDs).

There are also several wearable tools developed for interaction in the virtual environment. Gloves, walking devices and eye tracking devices are hardware that feature more realistic interaction with the VE. Concerning displaying, this thesis constrains to: 1. Rendered images 2. VEs displayed on a computer screen or projected on a screen with possibility of interaction with computer mouse or keyboard. Though, to raise the impression of the third dimension a method for stereo imaging could be used, where the VR environment image is displayed in stereoscopic images. The viewer uses rapidly repeatedly switching (liquid crystal) glasses to alter between two images faster than the eye can register. By also letting the computer switch between two images every second frame, the user experience depth. This method gives the user an illusion of a three dimensional room inside the screen.

It should be mentioned that this kind of exposure may affect the balance sense of the user (Stanney, Kennedy and Kingdon, 2002). Some VEs affects the balance more than others. For navigation in closed VEs (for example buildings), tools that demonstrate the surrounding area (maps, exocentric 3-D views) are recommended if training or exposure time is short, while internal landmarks (for example along a route) are recommended for longer exposure durations (Stanney, Chen & Wedell, 2000). Because of the risk of getting ill, users are recommended to take caution while using VR systems. Also audio is used to enhance the visual scene. Where equipment is available, surround systems can simulate the sounds of features around the user as the user moves in the VE.

3 Geographic data

This chapter describes different kinds of data and data structures that can be used for creating 3D city models and the availability of the data in the society. Also, methods for collecting three-dimensional and two-dimensional geometric data from reality are introduced. The data and methods described are of the kind that may be used as raw material to create 3D city model components (DEMs, objects and textures). Advantages and disadvantages with each collection method are also presented.

3.1 2D / GIS data structures

Two dimensional geographic data is traditionally stored in a GIS or CAD system as points, lines, arcs and areas. The different types of information, such as roads, buildings and trees are stored in separate 2D layers. The layers show interpreted representations of reality like paper maps. Mostly the lines and areas in the map represent a contour image of the part of the object that is closest to the ground. For example, the buildings are represented by the borders of the lowest part of the building i.e. the roof construction is not represented. Some objects, like poles, tree trunks or wells are displayed as single points with only a location (no two dimensional extension). All objects in the 2D map are built up by points and lines or arcs. The locations of these parts and their joint relations are often stored in a relational database (Eklundh and Pilesjö, 1999).

3.2 3D data structures

The geometrical data to be operated in a photo-realistic 3D city model usually includes two types of data: vector data and raster images. A combination of these two is desirable for the possibility to present the variety of objects in urban space (Sinning-Meister and Gruen, 1996). The vector data describes objects like the geometry of buildings by defining their grid structure, while raster images constitute colors and patterns covering the grid surfaces.

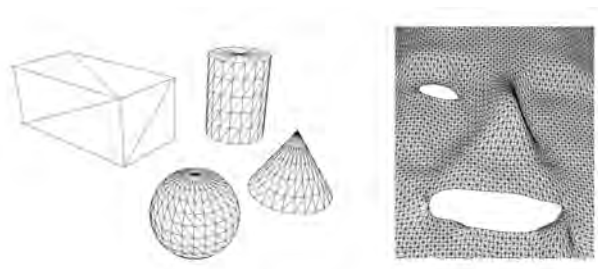


Figure 3.1. Four simple primitives (CSG) and a facet model (boundary structure).

The grid object's structure is either built up by simple primitives combined with each other with union, intersection and subtraction (CSG) or simply built up by triangular surfaces called faces (fig. 3.1). The CSG (Constructive Solid Geometry) structure is often used in CAD and suits well for standard volumetric objects. The CSG primitives are built by a boundary structure but are restricted to specific shapes like box, cylinder and cone. Groups of connected primitives can be

converted to boundary structure by elimination of faces on the inside and merge of connecting faces on the outside of the connected primitives (Brenner and Haala, 1999).

The boundary structure is a triangular irregular network where all triangles are *adjactent* to each other. For example two corners in each triangle are also corners in another triangle. The corners of the triangles are called vertices, and are three dimensional points. Three vertices are linked, together forming a triangular plane called a face. This face's visibility is defined by the plane's normal vector. Hence, the face can only be seen from the side that the normal vector points to. The normal vector is defined by triangles order definition, clockwise or counter-clockwise. From the backside of the mesh the element usually appears invisible. This is not a problem because no object is infinitely thin. The normal vectors of the objects' faces always point out of the objects' volume. The triangle segments always connect to segments of other triangles. This makes all objects closed from all angles. The faces of an object are together called a mesh. The structure described above is used in most 3D applications.

The raster images complement the volumetric objects by adding a texture to it. The images are two dimensional and mostly describe two dimensional surfaces like façades where for example a brick structure is found. There is no reason to build each brick as a vector unit in a grid structure or like a box primitive. The brick structure is therefore replaced by a raster image.

3.3 Differences between 2D and 3D data

As mentioned, in 2D GIS systems the data is sometimes well structured in a fully topological polygon structure where each component is an entity. This approach has many advantages for geometrical calculations. The 3D data structure has traditionally been used without this topological structure. Gruen and Wang (1998) have created a 3D topological structure for describing 3D city model components called V3D. The data structure is created in a relational database and can be converted to the less reachable boundary structure that is more used in 3D applications. A three dimensional fully topological structure would be desirable for geometric analyses in GIS purposes. In visualization purposes this is less important and therefore the feature is not prioritized and reachable in 3D software and 3D data formats. This restriction also makes it harder to connect attributes to individual elements in the model. One big difference is thus that the 2D GIS data is much more compatible with spatial questions and calculations than 3D data.

3.4 Data required for a 3D city model

To build 3D city models, a certain degree of three dimensional geometric data is needed. Much usable data may be fetched from a 2D GIS system, but various elevation data is needed as base material to achieve a somewhat accurate elevation for buildings. If it is a priority to create a model with an elevation model of the ground surface (not only a flat plane describing the earth surface), additional elevation data is required. Also, textural information is needed to complete the photorealistic impression.

The vector structure of a photorealistic 3D city model may be divided into two major parts: a Digital Elevation Model (DEM) and other objects. The DEM which is a model of the earth

surface at the location where the city is built is the most standardized data type in a 3D city model (Gruen and Wang, 1998).

The DEM is usually built by a quadratic network, but a triangular network is to prefer (fig. 3.2). Since the earth's surface originally is irregular it would not be suitable to represent it with a regular grid structure with a quadratic net. The elevation of the surface is therefore sometimes represented by a Triangular Irregular Network (TIN) where vertices are represented at locations where needed (like for example at the top of a hill). An advantage of the TIN is that it can be easily updated by a re-triangulation. How to build a TIN from elevation points is described in section 4.2.

Remaining objects in the model can furthermore be classified into two groups: surface objects (such as roads and waterways) and body objects (such as buildings). These two groups are commonly described with the CSG structure or the boundary structure. Data of these kinds are necessary to describe the volumetric objects separated from the ground and the classification of the different areas in the DEM.

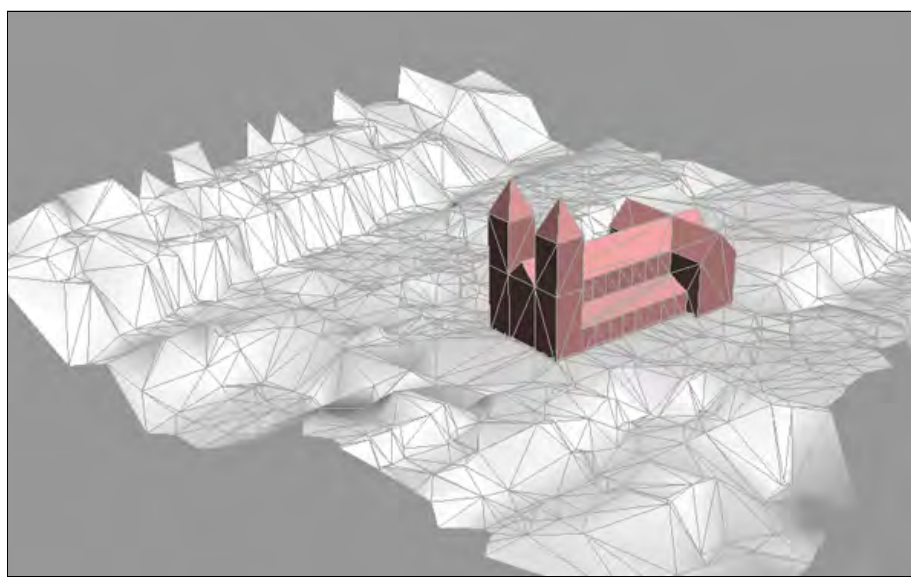


Figure 3.2. A DEM and a body object.

3.5 Available data in Sweden

The availability of base data is in Sweden mostly limited to 2D GIS data and aerial or satellite photographs. The departments of planning in some major cities of Sweden have started collecting 3D building data in form of elevation of roof levels (Åkerholm, 2003). Because of the telecom infrastructure development some geometric data models have also been produced. Elevation data is in GIS, apart from building heights, otherwise in most cases only available on manhole covers and on special occasions where it is needed for projecting land masses. Though, DEMs of population centers have been created.

3.6 Data acquisition methods

3.6.1 Field collection

Field collection here means the traditional methods of acquiring data sections one by one by field work. This is usually done with geodetic systems; a total station and a prism mounted on a stick using two or more reference points in the field. The instruments acquire geometrical points and lines. Traditionally, this method is mostly used for acquiring two dimensional data since that is what is used in regular GIS systems. For this purpose also GPS technology is used by connecting two or more instruments for measuring of 2D data. An advantage with traditional measuring is the high accuracy. Disadvantages are that the method is time-consuming and that three dimensional data of highly elevated objects is hard to reach with the equipment used on a daily basis. For measuring of building heights, other laser instruments are used.

Field collection can also be done with more refined laser instruments. Laser scanning is a method where a laser is used to measure the distance from a laser scanner system to the nearest object. By knowing the location of the laser instrument, the angle and distance to the spots where the laser hits, a point-cloud can be measured. While the laser sweeps over an area in all possible directions a digital camera in the system usually acquires images of all angles. 3D models of large environments require data acquisition from different capture points (Sequerira *et al*., 1999). Laser scanning systems are therefore mounted on moving vehicles such as airplanes for aerial data capture or trucks for terrestrial data capture (Früh and Zakhor, 2001). The laser beam is rotated sideways back and forth to scan the environment continuously while the vehicle is moving. In these kinds of system the location of the vehicle needs to be updated continuously while moving. This problem can partially be solved by using GPS technology for the integration of obtained data in earth bound coordinate systems. The point density of the captured data varies between different systems and factors. One factor is the distance to the capture plane. There is also a need to adjust the desirable density according to the application (Axelsson, 1999). Advantages with laser scanning are the large amount of data obtained and the high accuracy of the captured data. The possibility of image acquisition also opens opportunities for image draping in texturing purposes. The images are also needed for classification of objects (Axelsson, 1999). A disadvantage is that no laser scanning data is captured where surfaces reflects the signal like for example on water and windows (Brenner and Haala, 1999). Besides, the systems for laser scanning are to date too expensive for wide spread use.

3.6.2 Acquisition from photographs

Another form of field collection is the acquisition of photographs. The images are further used for two purposes: texturing and photogrammetry. The process of photogrammetry is resulting in geometric data, and described below. The texturing process is a usage of the acquired data and thus described in section 4.7.

Photography requires a camera to photograph planar surfaces. The photographs are usually acquired orthogonally to the planar surface. The two types of views used in this case are the aerial perspective and the terrestrial perspective. The aerial photos are acquired at high flight elevation to decrease the effect of elevation variations in the landscape and to use a perspective

where for example the walls of the buildings are least visible. It is recommended for further data processing to have flights made during the leaf free season (Wolf, 2000). Also satellites are used for acquisition of aerial images. Terrestrial photos are acquired from ground level since this is one of the perspectives used in the virtual environment. Advantages with capturing raster data is that much data is obtained fast. The disadvantage with collecting terrestrial photos is that it is time-consuming and complicated when trying to capture large surfaces. The main disadvantages with aerial photos is that they are expensive and can only be acquired on days with no clouds. Both methods also have the disadvantage of the central projection which distorts orthogonal lines in the plane. This can be partially rectified by applying a projective transformation (Brenner and Haala, 1999).

Photogrammetry is a technique which enables extraction of 3D vector structures from 2D raster images. To achieve this, two images of the same object from two different angles with known orientation are needed (Bauer *et al*., 2003). Both terrestrial and aerial images can be used for extraction of primitives, DEMs or other polygon representations. There are both automatic and semi-automatic methods for interpretation of the photographs. To date, computerized techniques make it possible to recover dense point-clouds through automatic image orientation and matching (Schindler *et al*., 2003). The automatic methods give rise to data noise and the points obtained differ in level of detail in object structures. Due to the complexity of natural scenes and the lack of performance of image understanding algorithms, the fully automated methods cannot guarantee results stable and reliable enough for practical use (Gruen and Wang, 1998). With semi-automated methods, a human operator creates the point-clouds from measurement on analytical plotters or digital photogrammetric workstations. The points obtained may this way be concentrated to boundaries of man-made objects like e.g. roof corners (Ulm, 2003). An advantage with photogrammetry is that three dimensional data can be captured from easily acquired photographs. A disadvantage is the lack of reliability in automatic methods.

4 Modeling approaches

This chapter describes how the data sets described in the former chapter can be manipulated and integrated with each other to become components in a 3D city model.

4.1 2D to 3D operations

Creation of detailed 3D city models is time consuming and often requires much manual work. The 2D data is an important source to simplify modeling. Usually, spatial data of cities is stored in 2D in a GIS system in separate layers. The layers are divided into for example roads, buildings, trees and are represented by 2D elements like points, line segments and polygons. Since this data, as the only data source, is supplied and updated for large urban areas there is a natural interest in using it as efficiently as possible when building 3D structures (Shilling and Zipf, 2003). The building footprints in the GIS data allow a direct conversion to 3D solids by extruding polygons to a certain height. Each line segment in the 2D map is converted to a rectangular surface usually built up by two triangular faces. It has to be noted that the GIS data is measured on ground level and therefore represents only ground plane information (Brenner and Haala, 1999). The converted 3D solid does therefore not express forms found above ground level. Since this information is not available in the GIS data, the roof height and the shape of the roof in the extruded solid is here a problem. The roof can be approximated to a simple plane orthogonal to all the wall plans in the building. This simplification only requires a single value of building height. The height can be approximated by estimating the height of each floor of the building (if the number of floors can be estimated by field work) or by the average height extracted with photogrammetry (Shilling and Zipf, 2003). If heights can be estimated, this method can also be used for other solid structures like stand alone walls. Though, the shape of roofs and walls is too complex to be extracted and requires 3D data input. Other attributes like largest depth of basement can be used to extrude basements downwards (Ulm, 2003).

The objects in the 2D data that are described by single points are for example trees, manhole covers or streetlights. They can simply be replaced by three dimensional objects on the 2D locations. The elevation of the placement has to be calculated from the DEM, or if the ground is represented by a plane, simply be set to zero. Current elevation placement also applies for buildings.

In contrast to above described objects some surface objects like streets, meadows, railway lines etc have no significant vertical extent (Shilling and Zipf, 2003). The integration of these line segments into the DEM is somewhat more complicated and further discussed in section 4.5. An alternative to the 2D line segments is draping of an ortho-rectified aerial image over the DEM. By applying this technique and converting point and polygon GIS data to 3D, simple models of large urban areas can be rapidly achieved. The level of detail in these kind of models is low (level 1 or 2 on Bourdakis above described scale) and are more suitable for flying exhibition than walking navigation.

4.2 Using point clouds for creation of 3D structures

The results of both laser scanning and photogrammetry are usually large sets of 3D point clouds. The points are used as vertices in calculations to create segments between suitable vertices to constitute triangular faces. When all points are part of triangular faces the result is a digital surface model (DSM) that can be seen as a bivariate function where any line parallel to the acquiring direction penetrates the surface at most once (Park and Kim, 1995). This also means that, on the created surface, none of the normal vectors of any face are pointing away from the capture point. By integrating surfaces from different capture points complete 3D models can be put together. One mathematic method (Delaunay method) for calculating the triangles (triangulation) from points is described.

4.2.1 Delaunay triangulation

A triangular network defined from a large number of points can be set in numerous of combinations. It is necessary to use a method to make the triangles appear balanced over the surface. It is for example often desired not to have thin and sharp triangles in the network. The Delaunay triangulation is often described as the perfect triangulation, since it creates a unique pattern. One of the properties of the triangulation is that the triangles in a Delaunay triangulation are best-possible with respect to regularity. (Assume that a list is made of the minimum angles in each of the constituent triangles in such a way that the list starts with the smallest angle and moves along the sequence to the largest angle. The triangles then have the greatest ordered minimum angle vector.) This means that that the list contains as less long and thin triangles as possible (Worboys, 1995).

4.2.2 Aerial view

From an aerial perspective the captured point clouds are often covering areas with a large amount of buildings and objects. The accuracy of DSM data is highest on faces with less amounts of slope e.g. the data of a flat roof is better than the faces of walls. If the point cloud originates from photogrammetry there is often no data at all concerning vertical elements like walls. A first task, in common to most of these methods, is the separation of objects from the ground surface. This can be achieved by finding the ground surface in the DSM by scanning lines of points one by one (Axelsson, 1999). When the part containing the ground surface is extracted, regions with holes can be interpolated from the existing data to create an exhaustive DEM surface. Extracting objects from the data is hard if the objects are not classified, for example a tree could be similar to a house from an aerial view. By using a 2D ground plane to restrict the extent of buildings the planar regions can be extracted more reliably (Brenner and Haala 1999). Brenner (2001) has shown that it is possible to reconstruct roofs of buildings with a fairly complex structure. This is accomplished by calculating the most likely roof structure of a given roof type in 2D and combining the structure with the automatic generated planar surfaces. This shows the need of integration between different data sets to achieve correct models.

4.2.3 Terrestrial view

From the terrestrial viewpoint, experiments have been carried out considering reconstruction of building walls by Wang *et al*. (2002) and Schindler and Bauer (2003). It is to date possible to extract models of planar walls with windows. The depth values of windows, calculated with

photogrammetry on large buildings, are very noisy. Therefore the façade surface can be approximated by two depth layers: the wall layer and window layer (Wang *et al*., 2002). This method is suitable for highly repetitive buildings with many windows where walls are strictly planar. In other cases, data is often too noisy for accurate construction.

4.3 Multi-spectral image classifications

Buildings and traffic networks are often available from 2D GIS databases but vegetation is usually not represented in detail. By classification of aerial images with algorithms such as the ISODATA algorithm it is possible to discriminate the classes: buildings, trees, roads, grass-covered and shadow out of multi-spectral images. The classification can also be used to automatically determine the position of single trees (Brenner and Haala, 1999). The tree positions are useful complements to the 2D GIS data. Classification could also be helpful in the creation of DEM from aerial data where the DEM can be computed by removing the building and tree elevations using low-pass filtering (Gamba and Houshmand, 2002). It could also facilitate the separation of objects from DSMs (Axelsson, 1999).

4.4 Combining GIS data with elevated 3D structures

It has already been stated that the combination and integration of several data sources is powerful since a high degree of automation can only be reached by integrating different and complementary types of data (Brenner and Haala, 1999). When constructing building objects GIS data may be used to smooth the process of separation of the objects from the DEM.

A problem with combining these two data sources is that they have been generated in different ways and therefore do not represent objects in a comparable way. It is not always easy to assign an object from source (A) to an object in source (B) (Shilling and Zipf, 2003). A building in source (A) may for example be divided into two building structures in source (B). In a GIS, two ground polygons represent two buildings that share the same roof structure. In a set of photogrammetric or laser scanned data the two buildings will appear as one building. The problem can be solved by deciding the most accurate source to restrict the other. This is done by filtering out objects in the less accurate source that are not to be found in the most accurate source.

The following approach is described in Brenner and Haala (1999): the ground plan building polygons are used to restrict the extension of the area which has to be examined, and an algorithm is used to calculate the normal vectors of the roofs' faces. The normal vectors of each point of the faces can be computed by knowing that the lines in the ground plan are perpendicular to the normal vectors of the roof part next to the ground plane lines. Roof regions are created by combining all points with normal vectors compatible to the ground plane line. The algorithm also assumes that each ground plan polygon segment defines one wall and one planar roof region. Hence, a bay or a small tower inside the building will not be reconstructed by the automatic algorithm. It is also needed to be mentioned that special roofs like dome roofs are not possible to reconstruct with this algorithm.

If the roof boundary points are measured manually from stereoscopic aerial images the ground plane data may be used in a different way (Ulm, 2003). The boundary points of the three dimensional building are overlaid on the ground plane. The concerned object points are snapped to the appropriate grid points, which adjust the objects' lines automatically to the direction of the grid lines.

4.5 Integrating non elevated GIS segments into DEM

As mentioned, some objects in the GIS source have no vertical extent. Roads, waterways and railway lines though need to be represented in the DEM. The line segments from the 2D source have to be draped on the DEM without changing their dimensionality in X and Y. To achieve this, the line segments for each time crossing an edge in the DEM need to be introduced by a new 3D point (Shilling and Zipf, 2003). The surface then needs to be retriangulated to include the line segments from the 2D plane. If the DEM is not already created, the segments' breakpoints have to be included in the primary triangulation. All new 3D points then constitute vertices in the created DEM. The elevation of the new 3D points is set by an interpolation with the existing 3D points as interpolation material.

4.6 Interpolation

To create a height model covering the whole area, an elevation interpolation of the 2D data is required. Some interpolation points (p) in the data material do not have an elevation value. The *distance weighted mean value interpolation* is a mathematic interpolation method which is used to create elevations of these points on the basis of the elevation of the known 3D observation points (i). The influence that each observation point has on the treated interpolation point decreases with the distance. All observation points are included in the calculation but observation points far away from the interpolation point have nearly no influence at all on the calculated height. The mathematical formula of this interpolation is described below (Eklundh, 1999):

$$z(x_p) = \frac{\sum_{i=1}^{n} z(x_i) \cdot \frac{1}{d^k}}{\sum_{i=1}^{n} \frac{1}{d^k}}$$

Where:

$z\,(x_p)$	interpolated elevation value
n	number of i-points
$z\,(x_i)$	value for point x_i
d	euclidic distance between x_i and x_p.
k	distance weight

One of the problems with this interpolation is that it becomes clear where the original observation points were located. This is visible by peaks and hollows in the surface. This *bulls-eye effect* (fig. 4.1) appears because the data points have too much influence on the cells in close (Eklundh, 1999).

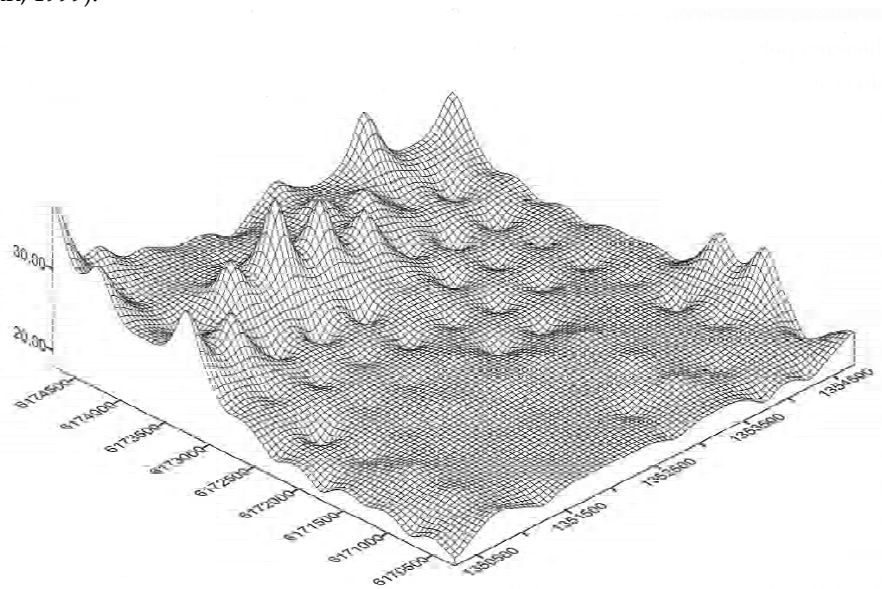


Figure 4.1. The bulls-eye effect in an interpolated surface (Eklundh, 1999).

4.7 Texturing

The goal of the texturing process is to provide a rectified image for each visible building face and roof region. Hence, for each image the corresponding façade polygon has to be selected from the 3D city model. The image has to be correctly positioned, oriented and scaled to represent its associated surface (Brenner and Haala, 1999). Since the structures of walls are planar, the images can be rectified by projective transformation. The vertices in the model that creates the boundary of the façade are then located in the raster image. This way both façades and roofs can be created although roof textures are acquired from aerial photographs. The texturing part is almost always the most time-consuming process when building photorealistic 3D city models (Ranzinger and Gleixner, 1998). Anyway, this process has to be managed in some way because photorealism can only be achieved if terrestrial textures are utilized (Brenner and Haala, 1999).

5 3D computer software and data formats

This chapter introduces some commonly used products and data formats for 3D applications. Three common data formats: DXF, 3DS and VRML are described. Some programs that are used or evaluated in the project are also briefly described: 3Dstudio Max, AutoCAD, OpenSceneGraph, WorldUp and Sitebuilder. Some different data representations and the usage of metadata are also discussed.

5.1 3D data formats

3D data geometry is often represented in a standard way (section 3.2) with vertices, segments and triangular faces. This representation is used in almost all 3D applications and 3D formats, such as in the three formats described below:

DXF, (Data eXchange Format) specifies an ASCII-file that allows the exchange of geometric data between different CAD systems as well as between a CAD system and external programs (MAE157, 2003). In the world of CAD this standard and its younger brother DWG, have been more successful than the attempts to develop formal standards (Whyte *et al*., 2000). The format was originally invented for AutoCAD, but by now all major CAD systems accept it for both import and export of data. DXF handles lines, polygons, text, and parametric curves and surfaces but does not accept solid models in the volume sense. The DXF format is usable for transport of geometric data from a CAD system to for example a GIS system or another 3D modelling program. The 3D structures are usually represented by the entity *3D Face* which defines a surface with four vertices. If a triangular face is desired, the fourth vertex is set similar to the third one (AutoDesk, 2003). The order of the vertices defines the normal vector of the triangular face.

3DS, is the 3Dstudio DOS mesh file format, used by former versions of 3Dstudio. The format is binary and thus, non-readable in a text editor. The format supports features like basic material and colors, single texture maps with amounts, offsets and scales etc. Since it is a well known and much used format many applications also have import and export capability for the 3DS files (3Dstudio Max User Reference, 2002). It is therefore often used for data transport between different software applications.

VRML, (Virtual Reality Modeling Language) is a platform independent standard format for descriptions of 3D environments for the Internet. A VRML browser, which is used as a plug-in to a regular web browser, downloads a VRML-file (a whole model) to the client and enables interaction with the model by flying, walking and examination (Heinonen *et al*,. 2000). Even though it is new, and more efficient formats (for example X3D) are under development, VRML is often chosen for displaying due to the widespread availability of VRML browsers (Shilling and Zipf, 2003). By using the anchor node function in VRML, metadata can be linked to objects or groups of objects using URLs (Chou *et al*., 2000). Objects' properties can be reached by a mouse-click on the object. VRML supports geometric data in form of triangular faces and image textures.

5.2 3D software

On the market of 3D software many programs are often specialized on special tasks. There are 3D modeling and rendering software like 3Dstudio Max or Lightwave that are more specialized on visualizing all kinds of objects. Software like AutoCAD, ArchiCAD or Alias Wavefront is 2D/3D construction applications where focus is set on exact construction representation. In the current century, software for extruding 3D structures from 2D GIS data have started to appear (see section 4.1). One such application was evaluated: Sitebuilder by Multi-Paradigm. A variety of programs are also used for dynamic visualization in VR. WorldUp, EON and OpenSceneGraph are three of them. A brief description of the major software used in the project follows.

5.2.1 AutoCAD

AutoCAD by AutoDesk is the most well known and used CAD program on the market. The application is mostly used for construction development and drawing of construction details. Since it is an excellent tool for drawing 2D and was already early a dominant drawing product, authorities among geography have used it also for map drawing. The multi-layer system creates an efficient way of handling entities in maps. For example different layers are used for buildings, roads or trees. The most used, traditional versions of AutoCAD are more concentrated on drawing than on the spatial data representation, in which way the GIS's handles data. AutoCAD also has support for creating 3D volumes and meshes. The scope is set on volumetric creation of CSG structures. Thus, creation of a building in AutoCAD is often done by combining and cutting boxes or cylinders, instead of building walls with planar faces (mesh structure). AutoCAD is created for exact representations where every detail is measured at millimeter accuracy. (AutoCad, 2003)

5.2.2 Sitebuilder

Sitebuilder by Multi-Paradigm was produced for converting 2D GIS data to 3D models. The software also offers a simulation of a city or terrain model scene. The program is capable of extruding 2D building polygons to 3D boxes and also to creates terrain models from elevation data. There are also automatic functions for placement of single objects like streetlights etc. After the automatic modeling processes the scene is viewed in a VR-environment where the user can fly or walk on the DEM surface. There is also a modeling program by the same company for creation of buildings etc. This software is much used for fast automatic construction of symbolic models of city areas. The software does not have the capability to export the scene to 3DS format, but some advanced VR applications can import scenes directly from sitebuilder. (Sitebuilder, 2003)

5.2.3 3Dstudio Max

3Dstudio Max by Discreet is probably the most used and well known 3D modeling software (fig. 5.1). The software is more concentrated on the mesh system, but solid structures are easily created as well. As solids have been created they are further on treated as mesh objects. The program offers several functions for automatic processing of mesh surfaces. Features like bending, twisting and texturing objects in different ways are just examples of the long list of functions. The *edit mesh* function is a usable resource for editing. The function lets the operator edit the meshes in both face level, segment level and vertex level. Advanced editing of the

structures can be performed by manipulating the mesh objects with these tools. 3Dstudio Max also has powerful functions for lights, rendering and cameras. The advanced rendering and light functions enables realistic rendering of 3D scenes. Short movies can be rendered from the cameras viewpoint by letting a camera sweep through the scene.

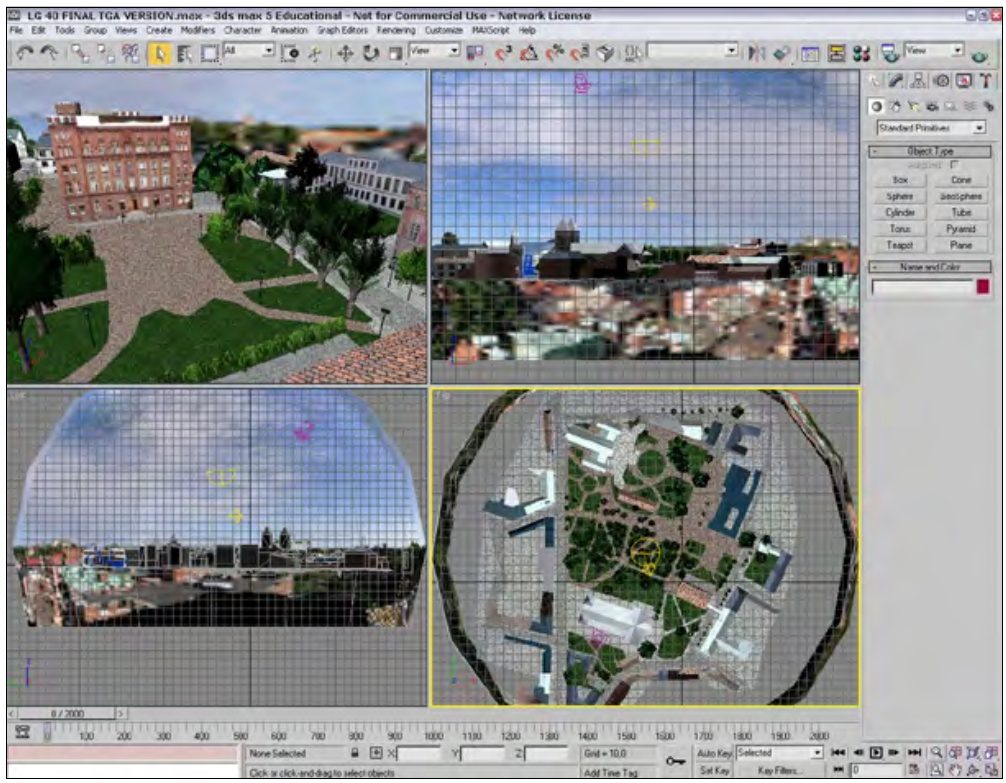


Figure 5.1. A screenshot of the 3Dstudio Max workspace.

5.2.4 WorldUp

WorldUP by Sense8 is a VR visual programming platform that has been developed during several years. The program is primarily used for creating interaction between a user and the virtual scene. This interaction is created by written scripts that are connected to the objects. Objects can for this purpose also be grouped hierarchic in object nodes. The program is able to import 3DS files for integration in the object hierarchy. A feature, not found in all VR software is the stereo visualization possibility. The texture formats supported are jpeg and tga.

5.2.5 OpenSceneGraph

OpenSceneGraph is an open source VR platform created by Don Burns in 1998. The open source project was then adopted by other developers and today it offers a fully functional VR environment written in C++. The interactive VR functions are written directly in C++, and the software does not provide any graphical user interface. OpenSceneGraph also supports stereo visualization and moreover, import of a great number of model and texture formats. There is a special plug-in for 3Dstudio Max for direct export to the OpenSceneGraph format (Openscenegraph, 2003).

Part III
Project

6 Introduction to the project

The project part of this thesis aims to construct a minor 3D city model to study basic methods for creating photorealistic 3D city models. The model is constructed with a given base data material and is limited to collection of new data by terrestrial photography.

The working area to be considered in the project is Lundagård (fig. 6.1), a park in central Lund, Sweden. The reason why this area was chosen is that it is a well known and representative part of the town, with an attractive park and old buildings specific for the town. Some of the buildings are also targets for University events during the year.

Figure 6.1. A photograph of Lundagård by Joakim Eriksson.

A review of the objectives of this project:

Establish a method for creating computerized 3D city models that is appropriate for visualization in the planning process.

Or more specific:

- create a minor city model using mainly available data,
- evaluate methods to collect accurate data for a 3D city model,
- find an appropriate method for viewing the 3D city model.

Note that the concrete results of this project are represented in all chapters from the method chapter and further. The visible result is the created model, but other results are the evaluated methods and the proposed method. Images and movies of the visible result may be downloaded from the project website:
http://www.reflex.lth.se/gis/3Dgis/

The chosen way to collect data and create the city model is described in the method chapter (8). The evaluation of the collection methods used is found in the conclusion chapter (9). Some methods for displaying the model are described in the method chapter (8) and a proposed

method of viewing is described in the proposed method section (9). The user scenarios are described in the discussion chapter (11). Finally, the established method, which is the main objective, is found in chapter 9, of the proposed method.

6.1 Requirement specification

The project area is shown in a map in figure 6.2. The boundary of the walking navigable area is marked in red.

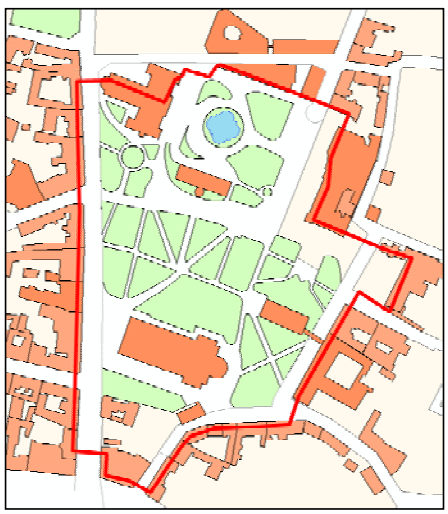


Figure 6.2. A map of Lundagård. The navigable area marked in red.
© Copyright, Lunds kommun.

Due to the given material (described in the next chapter), and the time available for the project, a suitable level of detail for the model had to be chosen. This is partly defined during modeling and estimates the effort in the modeling process. The level is chosen individually for different objects in the 3D scene but according to presented theory the level of detail is most likely corresponding to Bourdakis' level 2 (see theory, section 2.6). The features chosen to be represented are:

- buildings and walls,
- ground structures -grass, cobblestone and gravel paths,
- trees, bushes and streetlights.

The reason that trees, bushes and streetlights were chosen is that all of them are required to get a natural scene and that they are also found in all parts of the area. Other objects like park benches, bus stops and traffic signs were not chosen to be represented in the project model due to time constraints.

Since the area is not big enough for a bypass flight, the model was chosen to be designed primarily for a walking or driving user. The backsides of the buildings in the border part of the model area were therefore not required to be constructed. The model is supposed to be used in both a VE and for high end rendering with advanced light functions in 3Dstudio Max. The lights in the rendered model are based on the light situation in the middle of a summer's day.

7 Material

This chapter presents the data, software and hardware material used in the project. There is also a discussion about what the data can be used for in the project purpose. The section describes the advantages and disadvantages with the available data regarding the project purpose, what the data could be used for and how usable it is.

7.1 Data

The available data of Lundagård to be used is limited to four categories from two sources, except from the terrestrial photographs that were collected for the specific project.

GIS data:

- Two-dimensional data from the primary map established by the local surveying department.
- Elevation information in form of measured points on the ground (point cloud) and elevation curves interpolated from the points (also from primary map).

Photographic related data:

- Aerial photos of Lund, taken in may 2002.
- 3-D solids, derived from the aerial photos, by the National Land Survey of Sweden.

7.1.1 2D data & topographic ground data

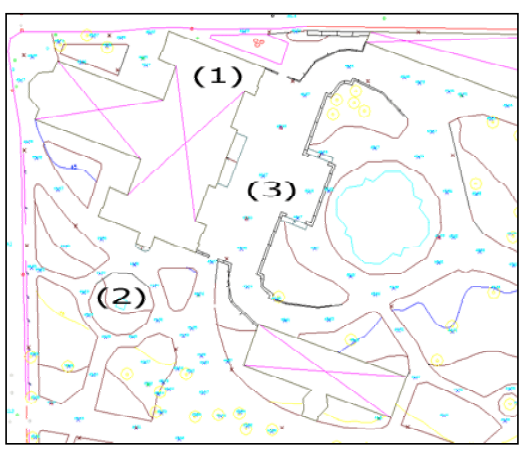


Figure 7.1. GIS layers from the department of city planning, Lund.

The municipal department of city planning in Lund and every other such department in Sweden has the responsibility of maintaining large-scale maps of urban areas. The maps, which contain different layers for measurable items in an urban environment with a few centimeters measurement error, are more than sufficient for making realistic VR city models of urban environments (Bourdakis, 2001). The data was obtained in AutoCAD format (dwg) and consisted of 85 visible layers where as 69 layers contained, interesting information to be used in 3D modeling. The map is situated in a flat coordinate system with local coordinates for Lund. The 2D data layers can be divided into three categories (fig. 7.1):

1. Buildings, walls and elevated ground constructions (stairs, etc.)
2. Surface objects: Road, grass and shrubbery limits on the ground
3. Single objects (Trees, streetlights, manhole covers, etc.)

All layers with single objects (3) give the location of the objects in two dimensions by a marker in the CAD drawing. This marker is represented by the entity in CAD called block. A block is an object formed by many points or lines. For example, a block describing a tree in the drawing is a circle with a much smaller circle inside of it. The centre of the inner circle defines the location of the tree. Blocks are used to give the different entities symbols for the map to be readable. In a block all lines are grouped so that if a line is marked the whole block is marked.

Other layers that contain objects are not represented by blocks (1) & (2). They are simply drawn with polylines or arcs where the objects are divided into several segments of lines. There is no connection between the segments' nodes and the object's properties. This depends on the fact that the data was acquired at different occations and that data was converted from the department's map system *AutoKa* (Åkerholm 2003).

Elevation data is also available in the CAD drawing. The area has been investigated for other purposes; planning of roads and buildings or water calculations and measured data have been saved in the CAD-drawing. The elevation data are represented by single points marked with a cross with a written number on the side of it (3). The number tells the actual height of the cross given in meters above sea level. The number and the cross are stored in two different layers. For some further planning reason elevation curves (fig. 7.2) have been interpolated from the single height points in the area. The curves are represented as arcs with half a meter between the height levels. The curves are not interpolated on ground where buildings and roads are located. The curves are therefore not continuous since roads or buildings are breaking the curves several times.

Another source of elevation data in the area is the height of the manhole covers (fig. 7.3). All manhole cover heights are registered with four decimals and represent the top of the lid of the well. In most cases a group of similar wells, for example day-water wells in a short distance (some meters) of each other are measured with the same height. The wells are mostly located in the middle of roads and in road crossings, where the water supply net is situated.

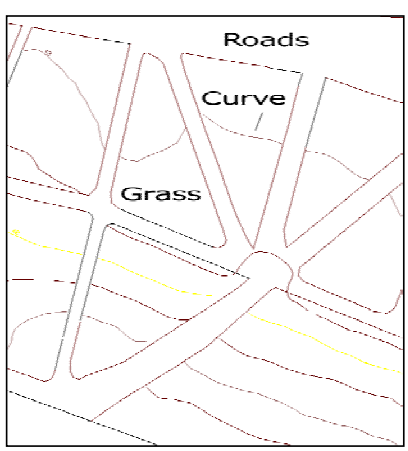


Figure 7.2. GIS layers representing roads, grass and elevation curves.

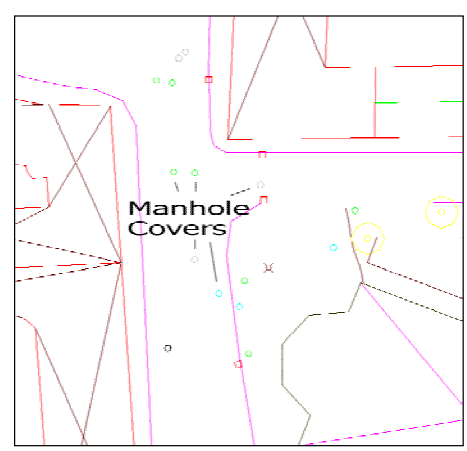


Figure 7.3. GIS layers representing manhole covers etc.

7.1.2 Aerial photos & extracted box model

Digital aerial photos of the area acquired in May 2002 were available to be used in the project. The material is divided into a set of three pictures separated by red, green and blue colors. The images are taken by BlomInfo for the local surveying office in Lund from a height of 2300 meters above ground.

A model of the area with extracted buildings was ordered from the National Land Survey to be used as base material for the model. The result of their extraction is a plane with flat solids representing the buildings in the area in 3DS format (fig. 7.4). The solids are built up in the boundary structure. The bottoms of the buildings are horizontal to a base plane which is flat. This means that the elevations of the buildings are not correct according to the real world. The top of the solids representing the buildings are also flat and horizontal to the base plane although the heights of the roofs vary from building to building. The top plane is created with faces but the bottoms of the buildings are not available. Since all of the buildings in this model are represented with flat roofs many of the buildings appear as boxes. This base model is from now on called the 3D *box model.*

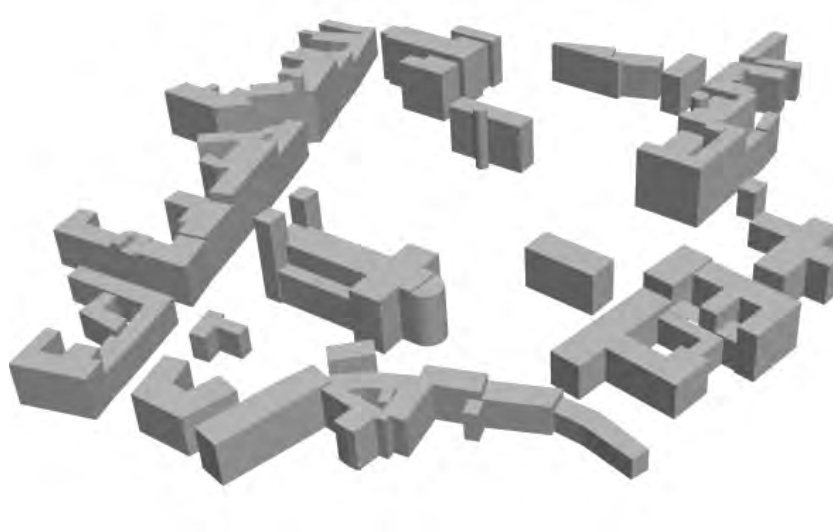


Figure 7.4. A 3D box model, extracted from aerial photographs.

7.1.3 Data quality

The CAD layers containing the outlines of the buildings are measured from the bottom of the buildings where the buildings reach the ground. This means that this data often corresponds to the measurements of the outer walls of the buildings, but does not supply any information about the roof dimensions. A problem with the building polygons is that the polylines are not closed into polygons. A building polygon may be built by several separate polylines. This means that exporting to a topological polygon structure in a GIS requires closure of all polygons. All other layers representing ground buildings or areas suffer from the same

problem. The single points are stored as block points with three coordinates in the CAD system. For trees and lamps the z-coordinate is set to 0. This block system is not used by other applications and therefore this leads to a conversion problem in direct export between a CAD program and a 3D-modeling program.

The elevation points from the CAD data are also structured in CAD blocks with three dimensions given. The fact that the measured points are randomly scattered makes the creation of a correct elevation model harder. Also some southern parts of the project area do not contain any elevation points at all. A regular spread of the points in the area would facilitate the creation of an elevation of the model since interpolations between the points would be more correct. The elevation curves in the CAD data are more or less useless since the lines are frequently interrupted by buildings and road structures. Since the elevation curves also origin from the points, the latter is a better material to use. The elevation points are anyhow usable for creation of a DEM. Due to the fact that aerial photos of the area are acquired during late springtime and that the area is rather leafy, the images are unusable for draping over an elevation model. The chosen level of detail and the resolution of the images are also low in comparison to terrestrial photos. This makes the photos less suitable for draping on the DEM. The photos are though usable to estimate the size of trees in bloom.

The box model of the area is extracted from the aerial photographs taken from 4600 meters above ground. The original purpose of the model was to calculate radiation patterns and the extracted elevation of the boxes are not well corresponding to reality. The building's z-values are therefore several meters higher than expected. Probably, the highest value of the entire roof was chosen to represent the whole roof and in some cases the values seem even higher. Circular parts of buildings are in some cases represented by cylinders and sometimes by boxes. This likely depends on the creator's view of the structures. If a cylindrical building part is built with a rectangular roof the structure is depicted by a rectangular block. A problem with the boxes representing the buildings is that the outer boundaries are corresponding to the roof outlines in contradiction to the CAD data corresponding to the outlines of the walls. Given the chosen level of detail this is not a big problem. The outlines of the walls differ at most 1 meter between the CAD data and the box model. The incorrect elevations appear to be a problem since all elevations have to be adjusted.

7.2 Software and hardware

The software used and evaluated in the project is listed below:

- 3Dstudio Max 5.1 by Discreet Software
- AutoCAD 2000i by Autodesk
- Javaeditor JCreator by Xinox Software
- Shewchuk's Dealunay program
- Sitebuilder by Multi-Paradigm
- Imagemodeler and Sticher by RealViz
- WorldUp by Sense8
- OpenSceneGraph by Don Burns, Robert Osfield and others
- OSGExp – Export from 3Dstudio Max to OSG by Michael Grönager *et al*. UNI-C, Dk

The most used application in the project is 3Dstudio Max. The program was used for manual modeling and rendering of scenes. AutoCAD was used for managing and sorting 2D data before conversion to three dimensional formats. JCreator was used for creating java interfaces creating 3D structures from 2D data. For triangulation of elevation points Shewchuk's Delaunay algorithm (Shewchuk, 1996) was used in the java interface. Sitebuilder was used for evaluation of existing 2D to 3D applications. Also the application Imagemodeler was evaluated. For visualizing in VR both WorldUp and OpenSceneGraph were tested.

The hardware used for modeling and programming was a PC, 1.5 GHz 512 RAM with NVIDIA Geforce 2 Video Device. For acquiring textures a Nikon Coolpix digital camera with 3.2 megapixels was used. A PC, 2 X 2.6 GHz, 2 Gb RAM was used for VR display and rendering.

8 Methods

This chapter describes the methods used to complete the model and also explains why the different methods were chosen. The methods are presented in the order they were used in the project process. Laser scanning and photogrammetry methods are not evaluated.

8.1 Method summary

To give a short structural summary of the main methods used in the project a scheme is presented (fig. 8.1). All methods that are not stated as *automatic* are based on manual editing. The letters A, B and C indicate what knowledge is required for the different steps.

A. Modeling, editing meshes, cloning and moving operations etc. in 3Dstudio Max.
B. Texture editing in Photoshop and texture mapping in 3Dstudio Max.
C. Java implementation and programming of scripts for reading and writing of DXF and VRML formats.

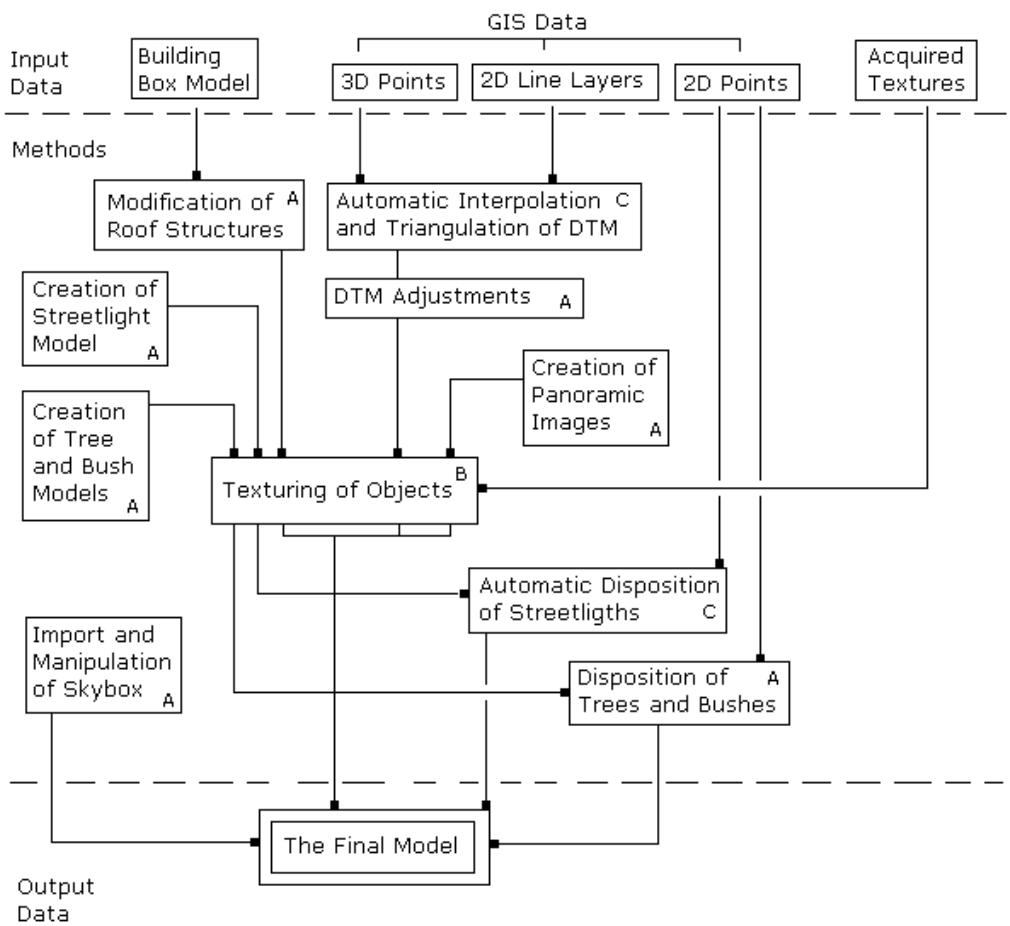


Figure 8.1. A workflow schema of methods used in the project.

An estimation of the time spent and the experienced level of difficulty for each method or method group in the project is presented. The estimated time spent is based on a full knowledge of all methods (if the project would have been restarted with the knowledge obtained by the author in the project). The time variable is given in number of days and the difficulty level from 1 (easy) to 5 (very difficult).

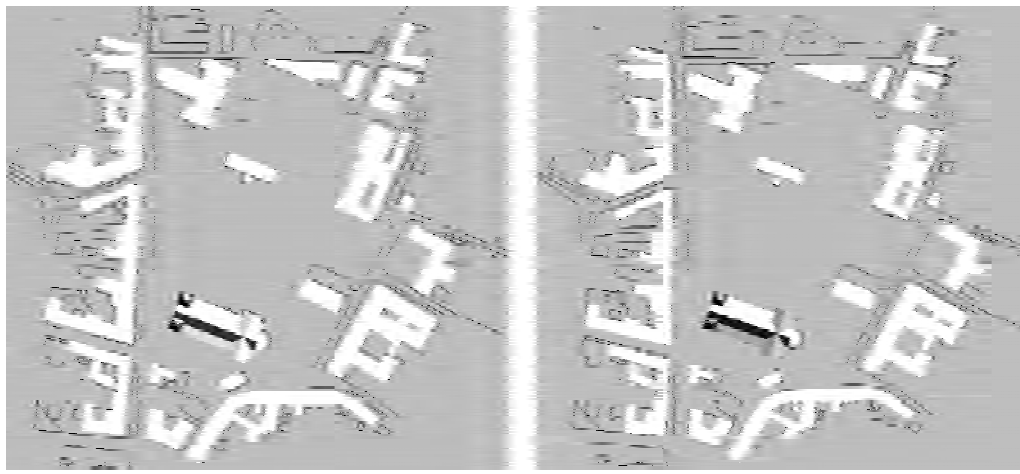
Task	Type	est. Time	Level
Separation of building blocks	A	0.1	1
Transforming 3D box model to 2D data	A	0.1	1
Editing and modeling building models	A	3	3
Building texturing	B	5	4
Creation of elevation model	C	10	5
Creation of tree and bush models	A	1	4
Creation of streetlight models	A	0.3	3
Automatic distribution of streetlights	C	1	4
Manual distribution of trees and bushes	A	1	2
Creation of Panoramic images	A	0.5	3
Importing end editing skybox	A	0.3	2
Final adjustments of merged model	A	0.4	2
Rendering settings	A	1	4
Preparing for exporting to VR	A	1	3
Total estimated time		24,7	

8.2 Separation of building blocks

After importing the 3D box model into 3Dstudio Max the model was examined. The model was united into one object. To be able to treat every building separately in further modeling, the model had to be divided into smaller pieces. All single buildings were therefore made into separate objects and on occasions where several buildings constituted a block, the block was separated from the model.

8.3 Transforming 3D box model to 2D data

In this project, 3Dstudio Max is the application where the different components are gathered and manipulated to a final model. An attempt to adapt the two primary data sources to each other in 3Dstudio Max required a choice of transformation type. The various maps are created in different reference systems with different transformations and projections. The 2D data imported from the CAD system did not correspond correctly to the 3D box model created from the aerial photographs. The two sources placed on top of each other illustrate the difference between the objects' borders.

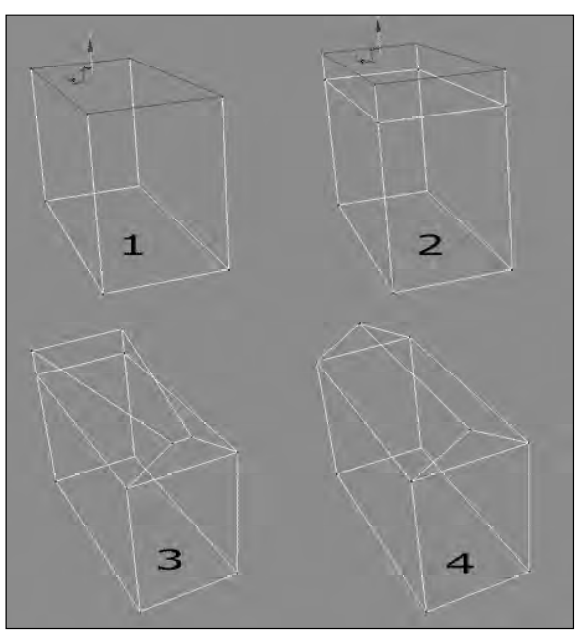


(a) (b)

Figure 8.2. Affine transform (a) and manual approximate transformation (b).

In this situation the problem is usually solved by mathematically calculated transformation adjustment. This requires that both transformation systems are known. A quick manual attempt in 3Dstudio Max showed that a transformation based on only scale and rotation (affine) was not sufficient for a correct adjustment (fig. 8.2a). A more advanced transformation on present data would be difficult. The 2D data is the more accurate data type in two dimensions. The choice was instead to adjust each building block separately to the 2D layer with a manual approximate affine transformation (fig. 8.2b). For single buildings the adjustment was more successful than for bigger blocks. The result was however considered to be suitable for the chosen level of detail.

8.4 Editing and modeling building models



Since all of the buildings in the box model lacked roof structures, a quick solution for creating roofs had to be established. An algorithm for automatic detection of the most likely roof type could be used (Brenner, 2001). However, the roofs vary much in construction, and a manual analytic generalization method was chosen to sort out dormer windows and chimneys. The roofs are created with extensions of elements in the box model. Most of the buildings have one or several parts of saddle roofs. The most basic creation process (only one roof part) is shown in figure 8.3.

Figure 8.3. Roof creation procedure.

For comparison between the building representation in the box model and a manually made model, a new model of the Cathedral of Lund was produced from scratch (fig. 8.4). The model was built in AutoCAD and imported to 3Dstudio Max. It is based on measurements of a wooden model (1:50) available inside the church. The need to create more accurate models of certain buildings sugests that the box model is only sufficient for non-complex buildings.

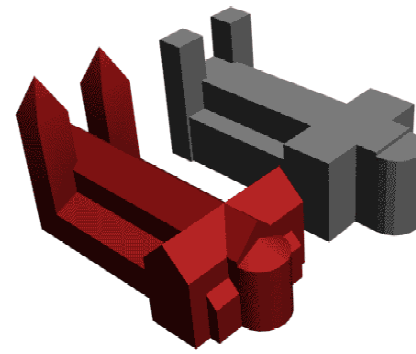


Figure 8.4. Two representations of the Cathedral.

Figure 8.5. A building with stretched out roof.

In the building model created so far, the roof lines are all attached to the building wall lines. In reality the roof is almost always extending out of the building to prevent rain from hitting the façade. To increase realism in the roof structure the roof faces were separated from the rest of the building. The roof was then stretched out in horizontal directions to a realistic looking level (fig. 8.5). The roof is clearly visible from underneath, but from a view orthogonal to the gable the roof will not be visible due to its zero thickness.

Some of the roofs in the area appear flat. In this case the roofs where modeled flat with the façade sticking out of the roofs' boundaries. The method was to extrude the top elements (see step 1-2 in figure 8.3) and delete the horizontal faces on the roof. The borderline between roof and façade then obtains a realistic impression from ground level. The texture on the upper part of the façade is created with an opacity layer to make parts of the top transparent (fig. 8.6). This is further described in the following texturing section.

Figure 8.6. Buildings with flat roofs.

Since the model is created with a somewhat correct DEM and the model area is rather sloped, the buildings' elevations had to be corrected to fit the ground surface (the building blocks from the original 3DS model were created with a plane as ground). To manually lower the buildings into the DEM was the chosen method. A problem was that the textures would have been invisible in some parts because of the slope. The solution to this problem was to create a one meter high building-foot on each building, covered with a brick texture. Instead of lowering the façade into the ground the building foot was put down into the ground surface (fig. 8.7). This method is also commonly used in building construction to enable the lowest base line of the façade to be completely horizontal where the ground is sloped.

Figure 8.7. A building with a tiled building foot, lowered into the DEM.

8.5 Building texturing

8.5.1 Acquisition

For texturing of buildings roof and façade, terrestrial photographies were required. The images were taken in the middle of a cloudy day to prevent shadows on building and roof parts. A digital camera with 3.2 megapixels of resolution was considered sufficient for photographing of the façades. After attempts with various angles of photo direction in relation to façade orientation, the method was set. The texture was easiest to use if the angle was as close to orthogonal as possible, both horizontally and vertically. In cases where the façade was situated far above ground level, there was no possibility of getting a vertically orthogonal angle from the ground. Anyhow, the horizontal angle was preserved. The same method was used for photographing the roofs where the roof could be seen. If the façade was too large to fit in the picture the façade was divided into two or more photos. In some cases where an iterated

structure pattern in the façade was found, only one image of the pattern was acquired for iteration (*tiling*) in the model. In some cases where a whole façade did not fit into one image the image was mirrored along the symmetry line to duplicate its appearance for the whole façade to be represented.

8.5.2 Transformation and correction

The images acquired with the camera are projective representations of the façades on a 2D surface (fig. 8.8a). If both the façades' and the images' orientations and locations are known, the façade in the image can be mathematically rectified to an orthogonal representation. Since this is strenuous and can be manually achieved, the latter method was chosen. The images were manually stretched in Photoshop to become rectangular and orthogonal (fig. 8.8b). The color and brightness of the images were adjusted to correspond to other textures on the same or other buildings. Elements like traffic signs and humans in front of the façade image were removed and replaced by an assumed image of the façade structure behind the removed features. Where an iterated pattern was found, the tiled image had to be adjusted. This was made by placing a duplicate of the image next to the original and adjusting the area where the images bordered each other. The adjustment was necessary for the tiling to be invisible and to make the texture to be realistic. The tiling technique was mostly used on the roof structures but also on some façades.

(a) (b)

Figure 8.8. An acquired image (a) and its adjusted texture representation (b).

8.5.3 Application of textures

3Dstudio Max contains usable functions for application of textures in the model. First a building model is chosen, and then one or more faces of the buildings are chosen. The area of the faces is later shown in an editor where the actual façade image is opened. The area of the faces is further adjusted in scale and rotation to fit the image of the façade. Also the iterated images can be attached to one or many faces in the building model. The amount of times for the pattern to be iterated is free to be chosen in both height and length. It is also necessary to rotate the pattern for it to get the correct orientation in the model.

8.6 Creation of the elevation model

The elevation data available in the project was sufficient for creating a TIN-model of the ground inside the park. Scattered elevation points cover the northern part of the park area and height information on manholes cover the street areas. Unfortunately, the available height data did not cover the whole area. Some additional points were therefore created in the outskirts of the model area. The elevation of these points were gathered from manhole covers. It was necessary to place such points all around the area for the DEM model to extend around the buildings. The DEM was created by triangulation of the elevation points with a 2D Delaunay triangulation program (Shewchuk, 1996). The triangulation was completed in the horizontal plane and the original elevation points were afterwards added to the triangles corners (on the same horizontal location since no points were removed). This operation resulted in a DEM covering the entire area. At this point the usual method is to drape an aerial or satellite photo over the DEM. Since the aerial photo was acquired during the summer and the area is almost covered with trees, this method was not applicable. Besides, the decided level of detail requires a more satisfying method. The 2D ground data specifies three kinds of ground materials: street (cobblestone), walking path (gravel) and grass. As mentioned, the areas are not defined as closed polygons in the CAD data. The method was therefore to treat all line and polyline objects as stand-alone segments and solve the classification problem in 3Dstudio Max when triangles were created. To combine the elevation points with the ground segments and create a triangular network requires an input of point and segment data to the triangulation program. Fortunately, the algorithm is also capable of constraining the triangulation to specified segments (fig. 8.9). I.e. a constrained Delaunay triangulation was performed. The input to and output from the Delaunay triangulation was managed by an interface written in java (appendix B).

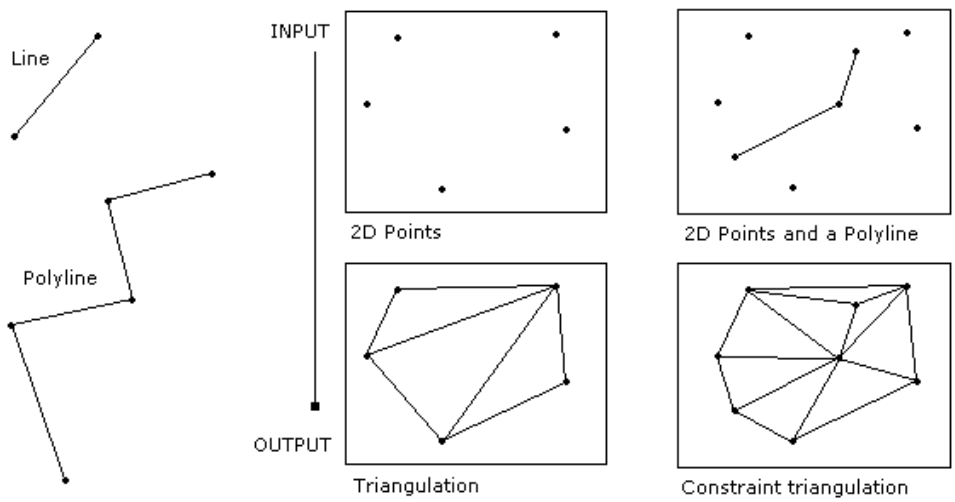


Figure 8.9. Triangulation and constraint triangulation of points and line segments.

8.6.1 A Java interface to the constraint Delaunay triangulation

The java interface, used to input and receive data from the Delaunay program, was originally written by Lars Harrie and Mikael Johansson (Harrie and Johansson, 2003). The input data was required to be put into these specific forms:

1. **inPoints** – An array of 2D points where every pair of indices is an x and y value. I.e. every even index corresponds to an x value and the following odd index to the associated y value.
2. **numPoints** – the total number of points i.e. the number of x and y pairs in **inPoints**
3. **inSegments** – An array of indices where every even index is a startpoint in a segment and the following odd index points to the endpoint in the segment.
4. **numSegments** – the total number of segments i.e. half of the number of **inSegments**

The output data was a number of triangles with corner points at the locations of the input points.

1. **numTriangles** – The number of triangles created by the triangulation
2. **outTriangles** – An array of references to corner points in the **inPoints** array. Every three values in the array is equivalent to the corner points in a triangle.

A restriction with the input data is that if the inPoint list contains any locations more than once, the point will be deleted from the array by the triangulation. The references of the inSegment array are in this case not changed and an error occurs in the triangulation since the inPoints array is reduced and the inSegement array still refers to deleted points. The problem of deleting twin points then has to be handled before the triangulation.

The java interface was further developed into a program for reading 2D and 3D data and creating a surface of the triangulation result. To be able to read the data in ASCII format the CAD file was exported to the DXF format, which is readable by any text editor. The objects to be converted to points and segments where identified in the exported AutoCAD file as *Lines* and *Polylines.* After the triangulation the data was output in another version of the DXF-format, where every triangle was described by three vertices. A detailed description of the input and output data is found in appendix A. A step by step list explains the methods used in the code in a sort of pseudo code. The full java code is to be found in Appendix B.

1. Read elevation points from elevation point file to a list (an ordered linkedlist) of point objects.
2. Read line segments in the 2D DXF file to the list of point objects and list of segment objects. The heights of the points are set to -1. Add the points and segments to the lists only if no twin is found in the list so far.
3. Read polyline segments as in step 2. Special temporal list created with functions for foreseeing the last point of the polyline where no segment is to be created to a new point.
4. Iterate through the list of points to ensure that no twins are found.

5. Convert the lists to the arrays **inPoints** and **inSegments.** Convert the heights of the points to two arrays of heights. One with heights of all points (including -1 heights) and one with only the heights from the elevation points.
6. Interpolate heights on all points where the value is set to -1 with distance weighted mean value interpolation (see theory). The x and y value of all points are also used in the interpolation algorithm.
7. Run the Delaunay Triangulation C-routine with the gathered data.
8. Write 3D faces to DXF file of the obtained triangles.

The triangulation algorithm resulted in a DXF-file readable by 3Dstudio Max.

8.6.2 Manual corrections of the surface

When imported in 3Dstudio Max, the DEM missed some triangles due to DXF reading errors. The number of missing triangles was however small enough to enable manual addition of triangles to make the DEM cover the whole area. An inspection of the surface showed that the required segments from the 2D data were successfully merged into the DEM. The inspection also showed that some interpolated vertices had unrealistic low elevation values (fig. 8.10). This created hollows in the surface, probably caused by the known interpolation problem (bulls-eye effect) described in the theory, section 4.6.

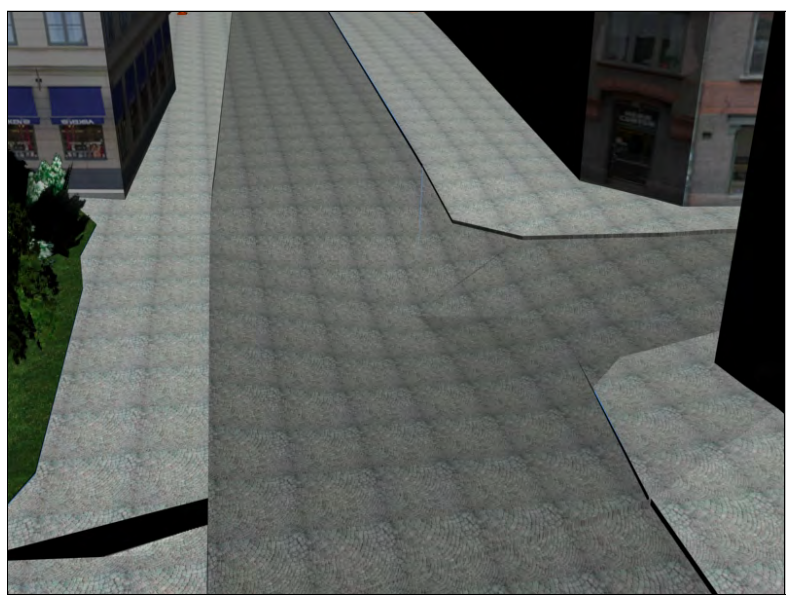


Figure 8.10. Missing triangles and interpolation hollows, probably caused by the bulls-eye effect.

In some of the locations, where errors occurred, the elevation values of the concerned vertices were manually corrected to a level that seemed to correspond with the values of the vertices around. Since the hollows included several vertices where the middle one was the lowest, the corrections where sometimes strenuous. This was mostly due to the display abilities of 3Dstudio Max. The concerned area could not be isolated in specific profile view and the perspective view was insufficient for understanding the vertice's elevations.

8.6.3 Surface classification

The classification of the DEM surface is a division process needed for the texturing of different ground types. The purpose of the constraint Delaunay triangulation was to create faces with line segments at the same locations as the polyline segments. When both layers where put on top of each other in a wire frame view it was clear which triangles belonged to which bounded areas in the GIS data. This made a manual classification of the triangles possible (fig. 8.11). All triangles classified in a special class (for example grass) were detached from the remaining DEM. The classes represented in the GIS data were as follows:

- driving streets,
- gravel paths,
- grass areas,
- pavement and other paved areas.

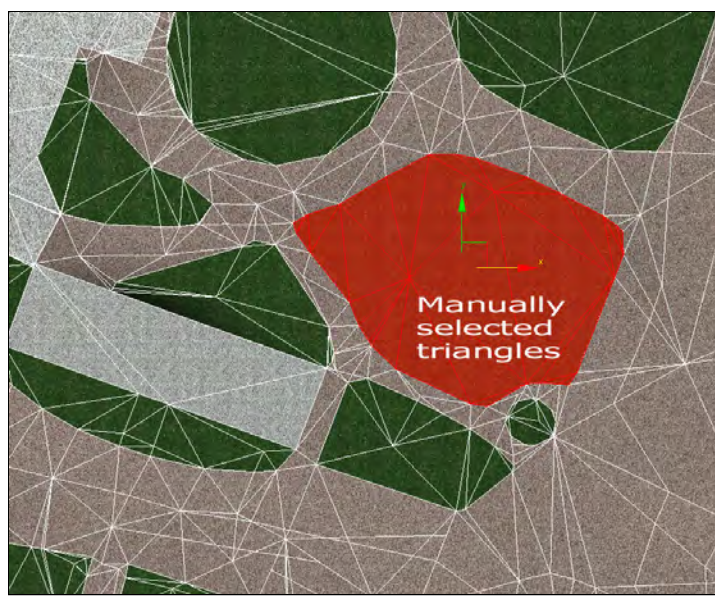


Figure 8.11. Manual selection and classification of faces within different areas.

When all areas were detached to separate objects, the remaining part of the surface was added to the paved areas. The gravel areas were divided into two different objects (there were two kinds of colors of gravel). The separation of the surface into classified objects does not mean that the surfaces appear none-continuous. The division operation is required for simplified ability to apply different textures on different classes.

8.6.4 Surface texturing

For creation of large ground areas the option of unique texture parts for every part of the ground is untenable since the textures would be too large given a high resolution (fig. 8.12). The better technique to use in this case was highly tiled textures. Images of cobblestone were acquired among the other photos in the project and corrected to be tiled. The goal of the tiling is to make the phenomena itself invisible. All the ground textures had to be carefully

neutralized so that there would not be any differences in colors and lighting over the images. Such differences cause visibility of the tiled pattern. The street was textured with the same texture as the pavement but the tiled image for the street was made darker to create a visible difference between the cobblestones in the pavement and the stones in the street.

Figure 8.12. The textured DEM

8.6.5 Surface extrusion

To make the DEM more realistic, some ground classes were extruded to a higher level than others (fig. 8.13). The gravel and the pavement were chosen to remain at the original height while grass and street was manipulated. The grass faces were extruded with about 5 centimeters above the gravel, which made the border between gravel and grass appear stronger. This creates a natural difference since grass is also naturally higher. The street was vice versa extruded downwards so that the street sunk down in the pavement. This also created a natural pavement border known from reality.

Figure 8.13. Extruded street and grass.

8.7 Creation of tree and bush models

Some different methods of tree construction were evaluated. In all these methods the foliage is created by texturing an image of foliage on a plane. An *alpha channel mask* is added to the image to exclude selected parts. These parts then become invisible in the rendering process. The easiest tree construction solution called *billboards* is built by a single-or two planes textured with a profile image of a complete tree (fig. 8.14). If two images are used, the second one is similar to the first and rotated 90 degrees around the tree trunk. This construction appears realistic from an orthogonal view from a distance of 10 meters or more.

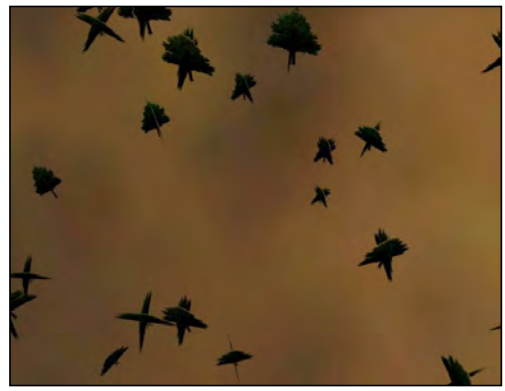


Figure 8.14. Billboard trees from an aerial view.

Figure 8.15. Pear shaped volumetric tree

The billboard technique is used within a VR application where the program turns the plane towards the user, so that the plane will never be seen from the side.

The billboard method is necessary when considering speed of rendering of the Virtual Environment but is only visually correct from ground level. When approaching the tree further it becomes clear that it is not volumetric. From an upper view or a perspective side view the tree looks even more unrealistic when the planar structure becomes obvious (fig. 8.14). To solve the volumetric problem a sphere was placed in the middle of the tree body. The sphere was afterwards deformed to look more like a pear (tree body) and textured with the same texture as the billboard. Four billboards crossed the pear in different angles (fig. 8.15).

This method was visually quite satisfying but the trees still appeared unrealistic from underneath. The fact that all trees were symmetric around the trunk and that the trees had to be homogenous also made the trees appear unrealistic. Furthermore the possibility to create trees in other shapes than the pear shape or overhanging branches was also restricted.

The choice was instead to use a more advanced representation of a tree found in a tutorial on the Internet (Ghost Recon - Modding Tree, 2003). In this method one plane is used for each branch, instead of the whole foliage. This opens the opportunity to create asymmetric trees

with somehow overhanging and sprawling branches. Since every branch is modeled separately, the light is able to beam trough the tree to create realistic shadows on the ground under the tree. The visibility trough the tree also appears more realistic. In this approach the trunk is modeled and textured as a squeezed and deformed cylinder. The branches are afterwards attached to the trunk in all kinds of different angles (fig. 8.16). The bushes were created in a similar way but without any trunks.

Figure 8.16. The final choice of tree and bush representation.

8.7.1 Tree and bush distribution

The tree positions are represented in 2D in the GIS data. Since the trees differ individually in size and shape a random distribution would not have been suitable. The problem is that specific trees in the area have special heights and widths at special locations and that the sorts of trees vary by location. Seven different tree types and three bush types were created. The objects were later cloned and manually distributed at the locations in the 2D data. By comparing with photographs, the most suiting tree type and the tree size could be set individually for each tree. For example, younger and smaller trees where represented by a different model than older and wider trees. Variations in scale and rotation in all three dimensions was required to separate the appearance of the trees cloned from the same model. By varying the tree types, the scene appeared more natural. The bushes were also distributed on locations defined in the 2D GIS data. In this case some areas in the 2D data were defined as bush areas and bushes became randomly scattered there. In one location the bushes were

placed to form a joint hedge. The size of the bushes was varied to make the hedge seem more realistic.

8.8 Creation of streetlight models

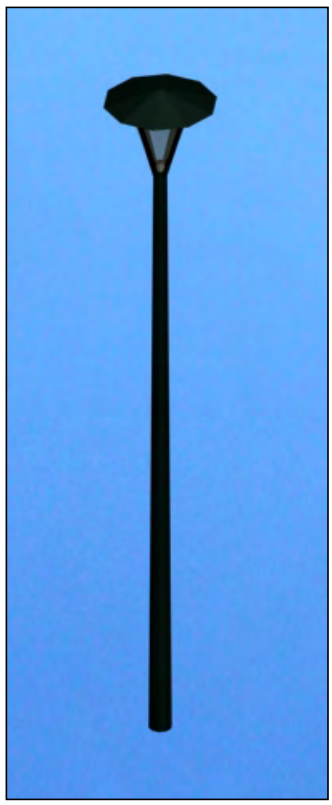


Figure 8.17. A streetlight model with transparent glass.

The area is scattered with a hundred street lights of two different types, one older looking and one more modern. The modern version was chosen to represent both types of lights. In this case it was suitable to use an automatic approach for distribution of the lamps to the GIS data positions because of the similarity between all lamps. Since it is stated in the specification that the model is meant to be created in a daylight situation, there was no need to create light sources in the lamps. The lamps are thus meshed as all other objects in the scene except the light sources. A simple representation of the streetlight was created and exported in the VRML format, because this is easily readable in ASCII (fig. 8.17). A java program was created to read the locations from the 2D CAD-file and distribute the streetlights at these locations. The elevation of the lights was calculated with the same interpolation algorithm (distance weighted mean interpolation) as was used in the elevation model java interface (see theory and section 8.6.1). After the calculations the program wrote a new extended VRML file with all light objects included. The complete program is found in appendix C. The VRML file was imported into 3Dstudio Max for further processing. Since no good solution on how to duplicate the textures of the objects was found the texturing was applied afterwards. The pole was textured in a dark green material texture and the glass bowl in a half transparent white material texture. The application of the material was simplified by a multiple selection of the same part of all poles and application of the materials on these parts on all poles at the same time.

8.9 Panoramic images

As mentioned, in the outer limits of the model area, the buildings are not modeled on the back side. This does not cause any problems since the model is only navigable in its inner part. Though, in the limits of the model area the modeled buildings end abruptly. A method had to be found to create an illusion of a continuing town view outside the model areas' borders. A decision was taken to create large

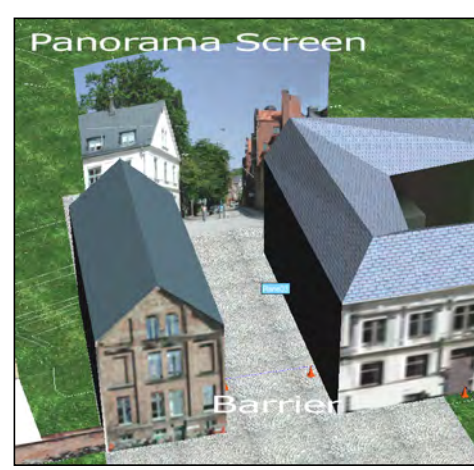


Figure 8.18. Panorama screen in a street.

screens with textured photos to be placed outside the model. In locations where the border was constituted by images of buildings and trees, a panoramic image was placed behind the building models. In locations where the border was a street barrier a panoramic photo of the continuing street was placed behind the barrier (fig. 8.18). The panoramic screens were created by photographs from 3 different points in the street of the model. The images were afterwards stitched together to a panorama visible on an arched screen. The panoramic effect makes the screen appear more realistic from all ground angles within the model limits.

8.10 Skybox

The model would appear unnatural without any illustration of the sky. The easiest way to create a sky is a plain blue color followed by a static sky texture. A color fading sky texture without clouds is an alternative for a cloud free day image. Though, some clouds clearly enhance the impression of the sky. If clouds are meant to be used, the clouded sky has to be textured all around the model. The usage of a single background looks static when rendering movement in the model. To create a sky model covering all visible angles requires texturing on the inside of a volume (sphere, cylinder or box). It is also required to acquire photographs of the sky in all angles. This was done from the roof of one of the higher buildings (AF) on a sunny afternoon in August. The photos were shot with the same digital camera as used before in the project, but this time on a camera tripod. Twelve photos were required to cover the whole circular angle and 24 additional photos were shot in angles pointed more at the sky and the ground. Imported in the computer, the images were treated in a program called *Stitcher* to be placed next to each other with overlapping objects at the corresponding locations in other images. After this calibration process, the program recalculates the textures for texturing on a box, cylinder or sphere. The photography and stitching was completed by Joakim Eriksson at the Department of Design Sciences, Lund Institute of Technology. The textures were then attached to volumes in 3Dstudio Max. The box approach proved to be unsatisfactory because of shadows on different sides of the box. Attachment of a single texture on a sphere turned out to be a much better solution (fig. 8.19).

Figure 8.19. The inside of the skybox

To create a realistic distance to the skyline, the size of the sphere was made just a bit larger than the DEM model. Together with the panoramic images, the skybox creates a natural variant of the continuation of the city. The skybox is adjusted in elevation to make the skyline visible only from a far distance in the model or if the view is set from the air.

8.11 Merging the model parts together

Because of the insufficient material slot function in 3Dstudio Max, and the fact that increasing model sizes decreases working speed, the different components where created in separate files. The buildings together with the panoramic images were created and edited in one file, the DEM and the trees in another. The streetlights were imported as VRML and saved as a separate file. After the unification of the components, some parts had to be corrected. The buildings were lowered to get the right elevation in accordance to the DEM. The panoramic images were adjusted to an elevation that was proper from all viewing distances and angles. Also the streetlights had to be corrected by lowering them together into the ground. Some buildings also had to be moved or resized not to interfere too much with the sidewalks. After the corrections the buildings were however accurate to the GIS data, which is also the more accurate data source.

8.12 Rendering

3Dstudio Max offers a complex rendering environment. Therefore, it was decided that some high-end rendered images and movies of the scene should be created as a part of the result of the final model. Since the skybox was created as an illustration of a sunny day the lighting effects were also required to be set accordingly. At first an ambient skylight was set all over the scene to create a base for all surfaces. This gave a light on all surfaces similar to what would be seen as the normal light situation in an area that is shadowed by an object as for example a building. In additional a directed sunlight was set to light up spotted areas and create shadows on shadowed areas. The parameters were set according to afternoon time in July at the actual location (Skåne, Sweden). Since the north direction was set properly parallel to the north direction in the original 2D data the sun placement and direction was completely natural and accurate to the actual scene. The skybox turned out to be a problem since this object destroyed the light calculations. The chosen and somehow primitive solution was therefore to move the skybox for rendering calculations and then move it back for the rendering. 3Dstudio Max offers two sorts of advanced rendering methods. The light-tracing method recalculates the lights on every single surface of the part of the scene that is visible in the viewport for every image. This gives a realistic image of shadows and lights and is required to be used if objects are moving in the scene during a movie and the shadows are changing. The alternative is the radiosity method where all the shadows are calculated in advance. Since no object was meant to be moving in the rendered movie this method was selected for rendering. Another advantage with this method is that it is much faster than the light-tracing method since one part of the rendering process is already done. 3Dstudio Max also offers the opportunity to create smooth camera movement trough the scene. A camera with wide angle perspective was used to sweep trough all parts of the scene from both the ground and the air. Since the model is optimized to be viewed from a ground perspective, these movies had a better appearance than the ones from the air, especially since the panoramic images were seen as floating in the air from a higher perspective. This view is however interesting to observe since these kinds of magnificent viewpoints can in reality otherwise only be reached with a helicopter.

8.13 Exporting to VR

To create a navigable version of the environment, the model was exported to two different VR software applications. This part of the project was merely a final step to examine the ability to use the model in VR. As mentioned about VR systems, there is a requirement of complexity level for models to be used in navigation. Fortunately, the model was created with this in mind, and thus the expectations of a working VR-model was high.

8.13.1 Export to a WorldUp traffic simulation

The department of Design Sciences at Lund Institute of Technology uses the VR tool called *WorldUp* by the company *Sense8* in various projects. Thus this software was available and was also used for testing navigation in the model. For transferring of the model to WorldUp the 3ds format was used. This process encountered some problems. In some occasions a translation error occurred in some objects. For example a roof was translated to a position far from the model. This was solved by attaching the roof to its building structure. The 3ds format is old and does not support all features like light settings, advanced multi-materials and cameras. It is therefore not preferable.

Another Master of Science project developing parallel with this one, offered an opportunity to create a driving-simulation of the scene. Staffan Stärnevall had created a driving simulation in WorldUp that was integrated with the Lundagård scene (Stärnevall, 2003). To enable this integration a driving path was exported separately with the model. Cars and bicycles then could travel trough the model on this path. By defining the DEM as ground surface it was possible to drive around in all parts of the model. The result of the test proved that the trees were a bit too complex to be rendered in this environment at a decent frame rate. The fact that the trees where quite complex, created with alpha channels and numerous branches, tended to slow the simulation down.

8.13.2 Export to OpenSceneGraph

Similarly, the open source VR environment OpenSceneGraph was trialed. In this case the model was directly exported from 3Dstudio Max to the VR world without passing the 3DS format. In this case, the export was more successful. OpenSceneGraph turned out to handle the model much faster than WorldUp, especially after replacing all jpeg opacity images with the tga format. Neither WorldUp nor OpenSceneGraph had the possibility to adopt the opacity mask system used in 3Dstudio Max. The solution was therefore to use the TGA format where an alpha channel can be included. With very little adjustment, a satisfactory Virtual Environment using OpenSceneGraph was developed. The reduction of the advanced light calculations of course decreased the appearance of the scene. However, a feature that enhanced the visual impression was the stereo viewing capability. This did not work very well with the scene in WorldUp, but was quite impressive in OpenSceneGraph. The final model was viewed on a large screen using a dual processor, 2Gbyte memory PC with the latest nvidia quadro 3000 graphics card. With this system, a satisfactory framerate was obtained. However, due to the size of the textures and complexity of the model, significant optimizations would be required before it would work on an ordinary home-PC.

9 Conclusions

This chapter demonstrates the visible result of the project and the methods proposed for further use. A written summary and images of the work illustrates the appearance of the model. There is also a part about which of the features of reality that were generalized or not represented in the model. The chapter also proposes both evaluated and documented methods for creation of 3D city models. The methodology covers mostly the creation of larger models than the one treated in the project.

9.1 The created model

Both the rendered version and the VR version corresponded to the initial requirements (section 6.1). A short summary of the appearance of the achieved 3D city model is below described in words:

The buildings' façades are represented by rectangular images, constituting windows, doors, walls and items sticking out of the walls. The roofs of the buildings are generalized to few big planes i.e. chimneys, drainpipes and structures sticking out of roofs are sorted out. There is no defined threshold value for the size of the building structures chosen to be modeled. The generalization process is instead varying due to the modeler's sense of feeling. The generalization level of building parts therefore may vary between buildings. Stand alone walls between houses are created by thin rectangular blocks. The ground is represented by a DEM with varying texture depending on ground type. The trees and bushes are modeled as trunks with several branches created of planes. Streetlights are placed at proper locations and panorama screens together with the skybox constitute the outskirts of the model.

9.1.1 Rendered images and screenshots from VR

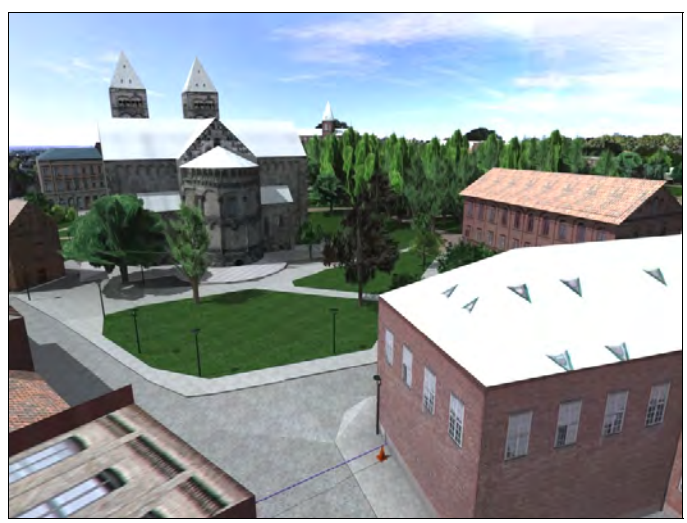


Figure 9.1. Rendered image of the area behind the cathedral.

Figure 9.2. Rendered image of Kyrkogatan.

Figure 9.3. VR screenshot of Kyrkogatan (OSG).

Figure 9.4. Rendered image of the AF building.

Figure 9.5. VR screenshot of the AF building(OSG).

Figure 9.6. Rendered image of the area in front of the cathedral.

More rendered images and movies of the scene are available on the project website.

9.1.2 Model restriction conclusions

A great number of features were chosen not to appear in the model. If this constraint would not have existed, the project could not have been finished within the time schedule. However, some additional features could have fitted into the chosen level of detail. For example, large items like statues, monuments and fountains would have been suitable. Also, traffic signs, outdoor cafés, vehicles and humans could fit.

As mentioned, the buildings have been completed with an individual level of detail depending of the complexity of the building. In some cases where the walls were integrated in the roof by small towers or dormer windows these elements were left out. In many cases these elements where created in a genuine architectural style with many detailed parts. Also the textures of the roofs where generalized by tiling. Chimneys, roof windows and differences in roof material were much generalized to simple tiled roofs of a few types of materials.

The added building feet were, as explained, a highly beneficial measure. In some cases where the texture of the façade already had a foot element the additional foot element appeared unnecessary. This could be solved by further lowering of the building into the DEM, but if the slope was too significant this measure was not satisfactory.

The elevation points used for the DEM were more-or-less randomly spread in the area, and thus there was much difference in the distance between the points. This, in turn, made the triangulation less successful. Another unpleasant matter with the DEM is the accuracy of the elevation in the outskirts of the DEM. Since the elevation data was insufficient in these parts, the points added in the outskirts are also much less accurate than in central parts of the DEM. Also, the integration of 2D elements had influence in this matter. The result was many pointed triangles with one acute angle and two obtuse ones. In combination with the elevation interpolation this gave rise to unwanted effects. When navigating in the VR version of the model it became clear that the pointed triangles became visible, and the surface appeared less smooth.

The algorithm for reading data from the DXF-files was created to read lines and polylines. The 2D data was also, in cases where needed, consisting of arcs. The result of the triangulation does not support arc forms since the triangles are always created by line segments. Some parts of the DEM should, according to reality, be created with circular or arced areas. These were manually approximated and the result is far from close to reality.

Since the trees and bushes are generalized to planar elements there are always angles from where the branches appear unnatural. If one of the branches is viewed from some angles it becomes clear that it is created by planar structures. Another problem with the trees is the limited amount of different trees created. Even if the trees are resized and rotated to become individual, parameters like leaf color and shape restricts the impression of individuality.

9.2 Proposed methods

The methods to be used in the model construction process have to be individually chosen depending on many factors:

1. The amount and form of available and easily reachable data.
2. The chosen level of detail of the intended result.
3. The size of the area to be modeled.
4. The resources in form of competence, time and money.

To propose methods, a possible scenario has to be chosen. The construction of a model of the whole city of Lund in the same level of detail as the Lundagård model could be a possible situation. The data usually available is aerial photos and GIS data, at the same level as used in the project. The technical competence factor is set according to the knowledge obtained in the Lundagård project. The factors of time and money for data acquirement, programming and modeling could be floating variables open for discussion in the following proposals.

The first thing to consider is how a DEM is going to be produced. Since the elevation curves in the GIS data proved unfeasible, the elevations have to be obtained in another way. It is not realistic to expect the elevation data available to be as covering as in Lundagård. However, the manhole covers are often measured by elevation and may be used as a base for the DEM. The water supply net is covering all parts of the built up areas of the town. A problem would be that the manhole covers are only represented on streets, and therefore an interpolation of the surface would probably appear unnatural. An alternative source of elevations is the aerial photos. In cases where streets are visible, for example in wintertime, stereo photographs could be used for manual or automatic image analysis. It would then also be possible to sample data in points that are not situated in the streets like in alleys, gardens and other green areas. The last, but probably the most expensive and efficient method would be to laser-scan the area with a helicopter flying on low heights. Afterwards the ground would have to be separated from other objects, which could be managed by Axelssons approach (Axelsson, 1999) or by a manual method. In areas that are covered with buildings, the DEM would have to be complemented by an interpolation and triangulation to create a bivariate function surface. Otherwise there may be gaps between the surface and buildings. Regardless of method used for collection, a conclusion is that it is desired to have a better spread of data material than the random material used in Lundagård. An automatic approach for classification of the types of ground could be applied to either the aerial photos or the GIS data. A manual approach would be effortful though achievable. The alternative of draping an aerial image over the surface is not an option, with the required level of detail in mind. In addition to this, the aerial photos are often scattered with cars, humans or trees which are not actual parts of the ground.

The second most central entity to represent in the model is of course the buildings. A conclusion is that the GIS data is much more accurate than a box model created from the aerial photos. Thus, the method of creating this level of box models is obsolete. As mentioned, the GIS data only covers the building footprint and thus, there is no representation of the upper parts of the building. The aerial photos could then be used for an extraction of the roof

structure to be put on top of the buildings. Here the approach from Brenner and Haala (1999) could be of much use. This method would automatically calculate the non-complex building's roof from the aerial images and GIS data. The GIS data would then be extruded upwards to the roof. If the elevation of the buildings can not be extracted in an adequate precision, the building elevation data may have to be collected manually with laser instruments.

A problem is still the building façades and roof structures of more complex buildings. Fortunately the amount of such buildings is small in comparison to the amount of non-complex buildings in for example the suburbs. In this case the landmark technique is to prefer, where the most known and specific buildings are created in more detail than others. As shown in the project, when modeling a completely new version of the cathedral, this method is time consuming and should only be applied on important landmarks. However, the vector structure of the buildings is, as known, not the major dilemma in the building creation.

The time consuming part is to acquire textures and attach them to the buildings. When concerning the roofs, the aerial photos are a valuable source for roof texturing. An automatic method for cutting out the roofs from the aerial image with the GIS data as the constraint would probably not be very difficult to program. Certainly, an automatic method would save much time. A requirement for this method to work is a very high resolution of the aerial images. The façade textures are the major problem to solve. It is hard to take sufficient pictures in narrow streets, and there are often obstacles between the façade and the photographer. Moreover, this work is time-consuming and requires a decent weather situation. Since the images are projected through a camera lens there is also a projective distortion. Since a manual transformation did prove to be strenuous, a mathematical method for correcting the images is recommended. Another proposal could be to develop a special camera instrument for the acquirement. The idea would be to let the instrument register its position by a GPS and its angle from a compass. The texture could then be attached to its proper position in the model without the need of manual texturing. Otherwise there does not at the time, seem to exist an easy solution to the texturing issue. The only method to be applied is much manual work. A better technique for texturing of circular objects is needed.

Concerning the single objects like trees and streetlights, the automatic technique developed in the project is of much use. Theoretically, the streetlights distribution algorithm could easily be extended to read all data of the town and place streetlights there. Practically, this would generate more data than 3Dstudio Max could handle smooth on an ordinary computer. This problem is though more computing related, and could be solved by further program development. The elevations of the streetlights would probably also turn out correct depending on the accuracy of the DEM. The trees constitute a greater problem. If a natural appearance is wanted the trees should differ in age, size and type. It is not complicated to create an algorithm for random placement of different trees at the tree locations, but for someone familiar to the real environment, it might appear unnatural. The problem could be solved by collection of the variables of age, type and size of each tree. The position and elevation could then be managed in the same way as the streetlights.

In such a large city model, it would not be necessary to create panorama screens in the outskirts of the model since most of the navigation would probably occur inside the model. However, a skybox is preferred to compose a natural sky, varying with movement in all

positions and also desirable when flying in the model. A much larger representation of the skybox would be needed and this could be managed by additional acquirement of photos from different places in the outskirts of the model.

As mentioned, a large model like this would be too heavy for the standard 3D modeling applications. The model would have to be divided into different areas, where only one area could be treated at the time. This is also beneficial for displaying in Virtual Reality since areas could be loaded only when needed. Another option is to divide the scene by entity, but it would however also have to be divided by area, considering the buildings. A conclusion is also that, for 3Dstudio Max, many objects with many faces are more complex than few objects with the same amount of faces. A trick is to merge objects as much as possible to speed up the processesing. Consideration of other software is also required. There may be other tools that enable the modeling and texturing capacity as 3Dstudio Max that are more suitable to GIS data. Though, it may be hard to find another alternative, when it comes to rendering and light calculations.

For displaying of the model, OpenSceneGraph could be stated as the better alternative, at least in comparison with WorldUp. Especially the direct export feature from 3Dstudio Max, and the fact that the software is free makes it an attractive VR solution. Furthermore, various tricks like billboarding trees and varying levels of detail can be specified in 3Dstudio Max to be used by OpenScenGraph to speed up the rendering process.

To be able to achieve a model of a town of the size of Lund, both collection work and modeling work is required. The collection of the trees' age, size and type would be a huge task, and so would the collection of the building elevations. It may, be necessary to lower the requirement on LOD to save both time and money. Concerning modeling, it is necessary to know much about the software and to have some additional programming skills. The estimation would be that only one person would need the skills obtained in this project, whilst the manual work could be conducted by anyone, after a learning process of some weeks. It would, in this case, be needed to set standards for LOD on buildings and photographing of textures. Finally, the manual processes needed for modeling of a whole town is clearly possible with help from the automatic approaches described.

10 Discussion

This chapter handles some further discussion about modeling and future improvement and 3D-GIS. It also includes some comments about the user experiences of the VR model.

One question to discuss is how the LOD of the buildings is experienced. As mentioned, the subtraction of elements on the buildings led to a decrease of the architectural values. Many detailed buildings became simplified in the façade/roof area. While navigating in the model and observing the rendered movies, there has been no comments about the misrepresentation of building elements. This could be explained by the fact that people usually do not notice the upper parts of the buildings in detail. The elements above our heads are unconsciously seen as an addition to the architectural values, and a removal of these only affects the overall impression. Thus, the observer is not able to point out the missing elements.

Another issue that might affect the experience is the tiling of textures. It may be discussed if this is a technique to use because of the consequence in LOD. If a tiled wall is seen beside a one-image textured wall the difference is clear. The tiled wall looks far more artificial. This, of course has to do with the size and interval of the tiling. However, within at least one distance, the tiling appears unnatural.

Some questions may be raised about the manual vs. automatic methods in the modeling approach chosen. One question is if more parts of the scene could have been automatically handled. The DEM algorithm could for example have sorted grass, gravel and cobblestone by analyzing the GIS data more. As mentioned, the different entities were simply drawn in different layers and thus the areas were not always represented by polygon surfaces. This matter was not further examined, but there may have been a better solution. A less strenuous approach could have been to make a simpler version of the DEM, with fewer polygons, created of a few elevation points. This would not have involved any programming and the process time would have been shorter, but also the LOD would have been lower. Another option would have been to concentrate the development to more programming. This would have resulted in some more refined methods, but a model of this size would not be finished.

It could also be discussed if the Java environment was the right environment to choose. For example MAXscript, which is a scripting language within 3Dstudio Max, could have been used for the automatic placement of the streetlights. The solution of writing VRML files was not optimal since all 3D information was not included in the VRML representation. To distribute the lights directly in 3Dstudio Max would for sure be a better solution. This would have enabled the opportunity to also distribute light sources in the streetlights that would be required for a night scene. Unfortunately, the MAXscript feature was found late in the project and the problem was already solved.

Much of the functionality in the written java programs is also available in other software like Sitebuilder and Microstation. It may be discussed why this software was not used. The main reason is that the programming experience gained in the project is suitable for further development of more advanced automatic techniques. Since 3Dstudio Max was chosen as the

modeling environment, due to its powerful functions, it also seemed suitable to program functions resulting in objects directly imported there.

Some thought about how the created model could be further developed are also worth mentioning. The addition of more details to the model is a natural continuation. To create moving characters like cars, people and animals is an interesting feature. There may be some plans to create a computer game, based on the model, since many powerful game engines have support for 3DS model import. Also, to create a night scene with the 100 streetlights would for sure take the visual impression a step higher.

For smoother navigation, and more effective rendering some VR functions could be added. One feature could be to use different levels of detailed objects on different distances in the scene. For example, a building far away would be represented by a textured box with a low resolution texture. The trees could be managed with billboard technique so that the tree would turn its most natural looking side to the user. Another thought is also to expand the model area to create a model of the whole town center or the whole town.

A question to be answered is which users would have use of models of this type. The model was primarily created for use in planning purposes, but other types of usage can be considered. At the time when this thesis was finished there are planned changes for Lundagård. For example a new tourist center for the cathedral is advertised in an architect competition. The model could be used as base material for import of different building proposals. A VR tour through the area could be a substitute or a complement to paper drawings and other illustrations. There are also plans to reconstruct the *Kulturen* area that is also a part of the model. The new appearance could easily be integrated in the model for later public exposure in a VE. The institution of traffic and road at the department of Technology and Society has showed interest in usage of VEs in driving simulation. The fact that the model is more or less photo-realistic enables a more realistic environment for the driver. Thoughts of creating simulations of environments developed for the physically disabled have also been discussed.

A conclusion, earlier discussed is that both the 2D GIS data and the 3D elevation points have some disadvantages for the purposes of 3D city modeling. It is interesting to discuss how the data could be collected and managed in another way to speed up 3D city modeling. The question has not been fully investigated, but the requirement of topological GIS systems is clear. The system used at the department of city planning in Lund, *AutoKa*, does not support this. The existing data has to be both collected and updated in another system. This could simplify the process of DEM classification. There is also a need for more collection of elevations. Much elevation information is stored in architectural drawings and other paper plans, but the data is not gathered in a covering GIS system. This question needs to be further investigated.

This thesis does not include any user evaluation of the VE created from the model. Some spontaneous feedback has though been uttered by visitors in the Lab, while viewing and navigating in the OpenSceneGraph version. The reactions have been that the model appears beautiful and surprisingly interesting. The fact that many people are familiar to the area also reinforces the impression of *being there*.

An interesting feature for further investigation is whether the data could be converted back to GIS data or not. If changes are made in a VE where planners interact with the buildings' positions, should there not be possible to reconvert the data to the original data source. With future development of 3D-GIS in mind, it may turn out to become vice versa. The 3D model is then the original data source, and 2D maps an extraction of the model.

This project is to be considered only the beginning of more research within this area. Questions about how an updated fully functional 3D-GIS system should work, and how this should be operated via a VR interface is not yet solved. The area of not only small single-purpose models, but functional 3D GIS/VR systems is a buzzword in science at the moment (Grönager, 2003). As mentioned, the area is integrated in many science projects, but a specific project or institution for 3D-GIS does probably not yet exist in Sweden.

11 References

Published:

Axelsson, P. (1999) *Processing of Laser Scanner Data -Algorithms and Applications*, Department of Geodesy and Photogrammetry, Royal Institute of Technology, Stockholm, Sweden.

Bauer, J., Karner, K., Schindler, K., Klaus, A. & Zach, C. (2003) *Segmentation of Building Models from Dense 3D Point-clouds,* Institute for Computer Graphics, Graz Technical University, Austria.

Beck, M. (2003) *Real-Time Visualization of big 3D City Models*, ViewTech AG, Zürich, Switzerland.

Bourdakis, V. (2001) *On Developing Standards for the Creation of VR City Models*, Laboratory of Environmental Communication and Audiovisual Documentation, University of Thessaly, Greece

Bourdakis, V. (1997) *Making Sense of the City*, Centre for advanced Studies in Architecture (CASA), University of Bath, UK

Brenner, C. (2001) *Towards Fully Automatic Generation of City Models*, Inistitute for Photogrammetry, Stuttgart University, Germany

Brenner, C. & Haala, N. (1999) *Rapid Production of Virtual Reality City Models,* Inst. for Photogrammetry, University of Stuttgart.

Brenner, C. & Haala, N. (1999) *Towards Virtual Maps: On the Production of 3D City Models*, Inst. for Photogrammetry, University of Stuttgart, Germany.

Chou, T., Ghung, L., Ku, W. & Lo, W. (2000) *An Implementation of 3D GIS on the web*. GIS Center, Feng Chia University, Taiwan.

Clayden, A. & Szalapaj, P. (1997) *Architecture in Landscape: Integrated CAD Environments for Contextually Situated Design*. Department of Landscape, University of Sheffield, UK.

Dodge, M., Doyle. S., Smith. A. & Fleetwood, S. (1998) Towards the Virtual City: VR & Internet GIS for Urban Planning, Birkbeck College, UK.

Eklundh, L. (1999), Interpolation, chapter 7 in *Geografisk Informations-behandling* (1999) edited by Eklundh, L.

Eklundh, L. & Pilsjö, P. (1999), Rumsliga datastrukturer, chapter 4 in *Geografisk Informations-behandling* (1999) edited by Eklundh, L.

Früh, C. & Zakhor, A. (2001) *Fast 3D Model Generation in Urban Environments*, Video and Image Processing Lab, University of California, Berkley, U.S.

Gamba, P. & Houshmand, B. (2002) *Joint Analysis of SAR, LIDAR and Aerial Imagery for Simultaneous Extraction of Land Cover, DTM and 3D Shape of Builidings*, Department of electronics, University of Pavia, Italy.

Gruen, A. & Wang, X. (1998) *Acquisition and Management of Urban 3-D Data using CyberCity-Modeler*, Institute of Geodesy and Photogrammetry, Swiss Federal Institute of Technology (ETH) Zürich, Switzerland.

Harrie, L. & Johansson, M. (2003*) Real-time data generalization and integration using Java*. Geoforum Perspektiv, Februar, 2003, pp. 29-34

Heinonen, A., Pulkkinen, S. & Rakkolainen, I. (2000) *An Information Database for VRML Cities*, Digital Media Institute, Tampere University of Technology, Finland.

Huang, B. & Lin, H. (2002) *A Java/CGI approach to developing a geographic virtual reality toolkit on the Internet*, Department of Geography and Joint Laboratory for GeoInformation Science, The Chinese University of Hong Kong.

Koer, M., Rehatschek. H. & Gruber, M. (1996) *A Database for a 3D GIS for Urban Environments Supporting Photo-realistic Visualization*, Institute for Computer Graphics, Graz Technical University, Austria.

Li, X. & Lin, H. (2002) *Participatory Comprehensive Plan based on Virtual Geographical Environment*, Joint Laboratory for GeoInformation Science, The Chinese University of Hong Kong, Hong Kong.

Patel, N. K., Campion, S.P. & Fernando T. (2002) *Evaluating the use of Virtual Reality as a Tool for Briefing Clients in Architecture*, Centre of Virtual Environments, University of Salford, UK.

Park, H. and Kwangsoo, K. (1995) *An adaptive Method for Smooth surface Approximation to Scattered 3D Points*, Pohang University of Science and Technology, South Korea.

Ranzinger, M. & Gleixner, G. (1998) *GIS Datasets for 3D Urban Planning*, GRINTECH GesnmbH, Graz, Austria.

Schindler, K. & Bauer, J. (2003) *Detailed Building Reconstruction With Shape Templates*, Institute for Computer Graphics, Graz Technical University, Austria.

Sequerira, V., Ng, K., Wolfart, E., Goncalves, J.G.M. & Hogg, D. (1999) *Automated reconstruction of 3D models from real environments*, European Commission-Joint Research Centre, Ispra VA , Italy & School of Computer Studies, University of Leeds, UK.

Shilling, A. & Zipf, A. (2003) *Generation of VRML City Models for Focus Based Tour Animations, Integration, Modeling and Presentation of Heterogeneous Geo-Data Sources,* Fraunhofer Institute for Computer Graphics, Darmstadt, Germany & European Media Laboratory, EML, Heidelberg, Germany.

Sinning- Meister, M., Gruen, A. & Dan, H. (1996) *3D City Models for CAAD-supported Analysis and Design of Urban Areas*, Institute of Geodesy and Photogrammetry, Swiss Federal Institute of Technology (ETH) Zürich, Switzerland.

Stanney, K. K., Kennedy R. S., & Kingdon K. (2002) Virtual Environment Usage Protocols , chapter 36 in *Handbook of Virtual Environments* (2002) edited by Stanney K. M..

Stanney, K. K., Chen, J., & Wedell, B. (2000) *Navigational metaphor design* (Final Rep. Contract No. N61339-99-C-0098). Orlando, FL: Naval Air Warfare Center Training Systems Division.

Takase, Y., Sho, N., Sone, A. & Shimiya, K. (2003) Automatic Generation of 3D City Models and Related Applications, CAD Center Corporation, Tokyo, Japan.

Ulm, K. (2003) *Improved 3D City Modeling With Cybercity-Modeler using Aerial-,Satellite Imagery and Laserscanner Data*. CC Chair of Photogrammetry, ETH, Zurich, Switzerland.

Wang, X., Totaro, S., Taillandier, F., Hanson, A.R. & Teller, S. (2002) *Recovering Façade Texture and Microstructure from Real-world Images.* Natick, USA., Padua, Italy., Cedex, France., Massachusetts, USA & Cambridge, UK.

Whyte, J., Bouchlaghem, N., Thorpe, A. & McCaffer, R. (2000) *From CAD to virtual reality: modelling approaches, data exchange and interactive 3D building design tools,* Department of Civil and Building Engineering, Loughborough University, Loughborough, Leicestershire, UK.

Wolf, M. (2000) *Photogrammetic Data capture and Calculations for 3D City Models*, Stuttgart, Weinstadt, Germany.

Worboys, M. F (1995) *GIS: A Computing Perspecitve,* Taylor and Francis Ltd (1995). Department of Computer Science, Keele, UK.

Zhang, Z., Tsou, Y. J., & Lin, H. (2000) *GIS for Visual Impact Assessment*, Department of Architecture, Geography, The Chinese University of Hong Kong, China.

Unpublished:

Stärnevall, S. (2003) *Traffic Simulation in Virtual Reality with Possible Application to Rehabilitation.* Master Thesis, Department of Design Sciences, Lund Institute of Technology, Lund, Sweden.

Private Communication:

Åkerholm, T. (2003) ,Department of City Planning, Lund, May 2003

Grönager, M (2003) Danish IT Centre for Education and Research, October 2003

Software references:

Shewchuk, J. R. (1996) Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In First Workshop on Applied Computational Geometry, Philadelphia, Pennsylvania, pp. 124-133.

3Dstudio Max User Reference (2002), 3Dstudio Max 5.1, by Discreet Software.

Internet references:

Openscenegraph (2003) Introduction to the openscengraph project
http://openscenegraph.sourceforge.net/introduction/index.html [2003-11-05]

Sitebuilder (2003) Multigen software center
http://www.multigen.com/support/sc_3dgis.shtml [2003-11-05]

AutoCAD (2003) AutoCAD product specifications 2003
http://www.autodesk.se/adsk/servlet/index?siteID=440386&id=2707944 [2003-11-05]

MAE157 (2003) MAE 157 -- A brief DXF intro
http://haidekker.org/mae157/dxf.html [2003-07-24]

AutoDesk (2003) AutoDesk DXF reference
http://www.autodesk.com/techpubs/autocad/dxf/reference/ [2003-07-24]

Ghost Recon - Modding Tree (2003) Ghost Recon - Modding Tree,Published: 11th July 2003
http://www.ghostrecon.net/html/trees/trees01.htm [2003-10-07]

CyberCity (1998) 3D city models of Zurich by CyberCity
http://www.cybercity.tv/city_zh_e.htm [2003-11-11]

Appendix A

Input and output formats of elevation data

Elevation points

```
AcDbText
 10
77306.793637
 20
75122.60062500001
 30
43.21
```

Line objects

```
AcDbLine
 10
77201.353
 20
75303.99800000001
 30
0.0
 11
77201.06699999999
 21
75297.63800000001
 31
0.0
  0
```

Polyline objects

```
AcDbPolyline
 90
      3
 70
    0
 43
0.0
 10
77163.137
 20
75236.244
 10
77166.69999999999
 20
75235.72999999999
 10
77173.86
 20
```

Input:

The *elevation points* where saved as AutoCAD blocks with the attribute AcDBText for each point. The x, y and z value for each point is following after the 10, 20 and 30 code lines.

The geometry of each *Line* block was beginning with the code AcDBLine and ended with 0. Inside the block the first point of the line was indexed with 10 and 20 for the x and y value and the second point of the line with 11 and 21 for x and y.

As in the line codes, the first point of a *Polyline* starts with 10 and 20 for the x and y value, but in this case the following values are also coded with 10 and 20 for each additional point. The last point is followed by the code 0.

```
3DFACE
  5
7C
  8
0
 10
-236.1373190000013
 20
-135.97299999999814
 30
37.9
 11
-228.50700000001234
 21
-131.1689999999944
 31
39.71107699741894
 12
-229.93400000000838
 22
-129.4719999999943
 32
39.70656651462486
 13
-229.93400000000838
 23
-129.4719999999943
 33
39.70656651462486
  0
```

Output:

For the output of the created triangles, another version of the DXF format was applied. The coding system in this DXF was found to be adoptable to the purpose. To discover how the format was saving triangular faces, a single triangle object was created and exported to a DXF file. The triangle was found in this part of the code:

One face in an object is defined as a four point array. If the fourth point is set similar to the third, the array describes a triangle. By adding more sets of this particular text, the same object contains more triangles. An import of the file to 3Dstudio Max showed one object with lots of triangles.

Appendix B

Triangulation source code

```
import java.io.*;
import java.lang.Double;
import java.util.*;


        // ---------------------------------------------------------------
        // Delauny triangulation interface by Ludvig Ljungqvist 2003
        // Part of a Masters project in 3D-GIS, Lund Institute of Technology
        // The program reads 3D points and 2D GIS layers from DXF files and
        // triangulates a 2D triangluar netork. The elevation of each node in
        // the network is later interpolated with Distance weighted mean value
        // interpolation.
        // ---------------------------------------------------------------


class CDT {

  private native int createTriangles(int numPoints, int numSegments,
                    double inPoints[], int inSegments[], int outTriangles[]);
  public static void main(String args[]) {

        try {

   CDT p = new CDT();

        int numPoints = 0;
        int numSegments = 0;
        int numInHeights = 0;
        int numIPPoints = 0;
        int countPolylines = 0;
        int countPolypoints = 0;
        int countRealPoints = 0;
        int countLines = 0;
        int tempStartPoint = 0;
        int tempEndPoint = 0;
        int vertex1, vertex2, vertex3;
        int count = 0;
        t numTwins = 0;

        LinkedList pointList = new LinkedList();
        LinkedList segmentList = new LinkedList();
        LinkedList tempPointList;

        ListIterator twinIterator;
        ListIterator tempPointIterator;

        Point tempPoint;
        Point tempPoint2;
        Point tempListPoint;
        Point tempNextListPoint;

        double vertexLocation1x, vertexLocation1y, vertexLocation1z;
        double vertexLocation2x, vertexLocation2y, vertexLocation2z;
        double vertexLocation3x, vertexLocation3y, vertexLocation3z;
        double empX;
        double tempY;
        double tempHeight;

        String line = null;
        String logLine = null;
```

```
File outFile = new File("outPutFile.txt");
File logFile = new File("log.txt");
File inFile = new File("merallapfusk.dxf");
File segFile = new File("gbanaokant.dxf");

BufferedWriter writer = new BufferedWriter(new FileWriter(outFile));
BufferedWriter logWriter = new BufferedWriter(new FileWriter(logFile));
LineNumberReader reader = new LineNumberReader(new FileReader(inFile));
BufferedReader segReader = new BufferedReader(new FileReader(segFile));

ReadLine readLine = new ReadLine(segReader);


// ------------------------------------------------------------------
// Reading blocked elevation points from CAD-file to pointList
System.out.println("Reading blocked elevation points from CAD-file to pointList...");
// ------------------------------------------------------------------


line = reader.readLine();
while (line!=null){
        if (line.compareTo("AcDbText")==0){
                line = reader.readLine();
                if(line.compareTo(" 10")==0){
                        tempX = Double.parseDouble(reader.readLine());
                        line = reader.readLine();
                        tempY = Double.parseDouble(reader.readLine());
                        line = reader.readLine();
                        tempHeight = Double.parseDouble(reader.readLine());
                        pointList.add(new Point(tempX, tempY, tempHeight));
                }
                numIPPoints++;
        }
        line=reader.readLine();
}


// ------------------------------------------------------------------
// Reading points and line segments from 2D dxf-file
System.out.println("Reading points and line segments from 2D dxf-file...");
// ------------------------------------------------------------------

line = "none";
while (readLine.ready()){

        // Finding Line segments in DXF-file

        if (line.compareTo("AcDbLine")==0){
                countLines++;
                readLine.readLine();

                line = readLine.readLine();
                if(Double.parseDouble(line)>70000){

                        // Adding startpoint to array if startpoint is not already in the array

                        tempX = Double.parseDouble(line);
                        line = readLine.readLine();
                        tempY = Double.parseDouble(readLine.readLine());

                        tempPoint = searchForTwins(tempX,tempY,pointList);

                        if(tempPoint!=null){
                                tempStartPoint = pointList.indexOf(tempPoint);
                                numTwins++;
                        } else {
                                tempStartPoint = pointList.size();
                                pointList.add(new Point(tempX, tempY, -1));
```

```
                }


                line = readLine.readLine();
                line = readLine.readLine();
                line = readLine.readLine();
                line = readLine.readLine();

                // One more time for endpoint of line

                tempX = Double.parseDouble(line);
                line = readLine.readLine();
                tempY = Double.parseDouble(readLine.readLine());

                tempPoint = searchForTwins(tempX,tempY, pointList);
                if(tempPoint!=null){
                        tempEndPoint = pointList.indexOf(tempPoint);
                        numTwins++;
                } else {
                        tempEndPoint = pointList.size();
                        pointList.add(new Point(tempX, tempY, -1));
                }

                if(!searchForTwinSegments(tempStartPoint, tempEndPoint, segmentList))
                segmentList.add(new Segment(tempStartPoint, tempEndPoint));
        }
}

// Finding PolyLine segments in DXF-file

// Adding PolyLine points to temporal pointList

if (line.compareTo("AcDbPolyline")==0){
        line = readLine.readLine();
        line = readLine.readLine();
        countRealPoints += Double.parseDouble(line);
        countPolylines++;
        tempPointList = new LinkedList();
        while(line.compareTo("  0")!=0){
                line = readLine.readLine();
                if(line.compareTo(" 10")==0){
                        line = readLine.readLine();
                        if(Double.parseDouble(line)>70000){
                                countPolypoints++;
                                tempX = Double.parseDouble(line);
                                line = readLine.readLine();
                                tempY = Double.parseDouble(readLine.readLine());
                                tempPointList.add(new Point(tempX, tempY, -1));
                        }
                }
        }

 // Processing temporal pointList


        tempPointIterator = tempPointList.listIterator(0);
        while(tempPointIterator.hasNext()) {
                tempListPoint = (Point) tempPointIterator.next();

                tempPoint = searchForTwins(tempListPoint.getX(),tempListPoint.getY(),pointList);

                if(tempPointIterator.hasNext()){

                        if(tempPoint!=null){
                                tempStartPoint = pointList.indexOf(tempPoint);
                                numTwins++;
                        } else {
                                tempStartPoint = pointList.size();
                                pointList.add(new Point(tempListPoint.getX(),
```

```
tempListPoint.getY(), -1));
                                        }

                                        tempNextListPoint = (Point) tempPointIterator.next();
                                        tempPoint =
searchForTwins(tempNextListPoint.getX(),tempNextListPoint.getY(),pointList);

                                        if(tempPoint!=null){
                                                tempEndPoint = pointList.indexOf(tempPoint);
                                        } else {
                                                tempEndPoint = pointList.size();
                                        }
                                        if(!searchForTwinSegments(tempStartPoint, tempEndPoint, segmentList))
                                                segmentList.add(new Segment(tempStartPoint, tempEndPoint));
                                        tempPoint = (Point) tempPointIterator.previous();

                                } else {
                                        if(tempPoint==null){
                                                pointList.add(new Point(tempListPoint.getX(),
tempListPoint.getY(), -1));
                                        }
                                }
                        }



                }


                line = readLine.readLine();
        }

        // ---------------------------------------------------------------
        // Searching for twins in LinkedList pointlist
        System.out.println("Searching for twins in LinkedList pointlist...");
        // ---------------------------------------------------------------

        count = 0;
        twinIterator = pointList.listIterator(0);
        while(twinIterator.hasNext()) {
                tempPoint = (Point) twinIterator.next();
                ListIterator twinIterator2 = pointList.listIterator(0);
                while(twinIterator2.hasNext()) {
                        tempPoint2 = (Point) twinIterator2.next();
                        if(tempPoint.getX()==tempPoint2.getX() && tempPoint.getY() == tempPoint2.getY() &&
tempPoint!=tempPoint2) {

                                logLine = "Twin found in point= "+pointList.indexOf(tempPoint)+";";
                                logWriter.write(logLine,0,logLine.length());
                        logWriter.newLine();

                        logLine = "Coordinates=
"+tempPoint.getX()+"and"+tempPoint2.getX()+"&"+tempPoint.getY()+"and"+tempPoint2.getY();
                                logWriter.write(logLine,0,logLine.length());
                        logWriter.newLine();

                        count++;
                        }
                }
        }

        logLine = "Number of twins left in pointList= "+count+";";
        logWriter.write(logLine,0,logLine.length());
        logWriter.newLine();

        System.out.println("Number of Polylines in DXF-file "+countPolylines);
        System.out.println("Number of Points in Polylines in DXF-file "+countPolypoints);
        System.out.println("Number of Accurate Points in Polylines in DXF-file "+countRealPoints);
        System.out.println("Number of Lines in DXF-file "+countLines);
```

```
System.out.println("Number of lines in DXF-file "+readLine.getLines());

// ------------------------------------------------------------------
// Initializing arrays and converting from objectlists to arrays
System.out.println("Initializing arrays and converting from objectlists to arrays...");
// ------------------------------------------------------------------

// Initialize arrays with correct lengths

numPoints = pointList.size();
numSegments = segmentList.size();

double inPoints[] = new double[2*numPoints];
int inSegments[] = new int[2*numSegments];
int outTriangles[] = new int[30*numPoints];
double heights[] = new double[numPoints];
double IPheights[] = new double[numIPPoints/2];

// Converting LinkedList of points to Array for input in Triangulation

int index = 0;
ListIterator pointIterator = pointList.listIterator(0);
while(pointIterator.hasNext()) {
        tempPoint = (Point) pointIterator.next();
        inPoints[index*2] = tempPoint.getX();
        inPoints[index*2+1] = tempPoint.getY();
        heights[index] = tempPoint.getHeight();
        index++;
}

// Copying first part of LinkedList heightpoints to another Array for input in Interpolation

index = 0;
while(index<IPheights.length) {
        IPheights[index] = heights[index];
        index++;
}


// Converting LinkedList of segments to Array for input in Triangulation

index = 0;
Segment tempSegment;
ListIterator segmentIterator = segmentList.listIterator(0);
        while(segmentIterator.hasNext()) {
        tempSegment = (Segment) segmentIterator.next();
        inSegments[index*2] = tempSegment.getStart();
        inSegments[index*2+1] = tempSegment.getEnd();
        index++;
}



// ------------------------------------------------------------------
// Interpolating heights on all points with unknown height
System.out.println("Interpolating heights on all points with unknown height...");
// ------------------------------------------------------------------

index = 0;
while(index<numPoints) {
        if(heights[index]==-1)
                heights[index] = Interpolate(inPoints[index*2], inPoints[index*2+1], inPoints, IPheights);

        index++;
}

// ------------------------------------------------------------------
// Printing Log
System.out.println("Printing Log...");
```

```
// ---------------------------------------------------------------------

// Printing amount of variables to logfile

logLine = "numPoints = "+numPoints+";";
logWriter.write(logLine,0,logLine.length());
logWriter.newLine();

logLine = "numSegments = "+numSegments+";";
logWriter.write(logLine,0,logLine.length());
logWriter.newLine();

logLine = "Number of Twins treated = "+numTwins+";";
logWriter.write(logLine,0,logLine.length());
logWriter.newLine();

logWriter.flush();

// Printing arrays of variables to logfile

for(int a=0;a<inPoints.length;a++){
        logLine = "inPoints["+a+"] = "+inPoints[a]+";";
        logWriter.write(logLine,0,logLine.length());
logWriter.newLine();
}

for(int a=0;a<inSegments.length;a++){
logLine = "inSegments["+a+"] = "+inSegments[a]+";";
        logWriter.write(logLine,0,logLine.length());
logWriter.newLine();
}

for(int a=0;a<heights.length;a++){
        logLine = "Location="+a+" HeightValue="+heights[a];
        logWriter.write(logLine,0,logLine.length());
logWriter.newLine();
}

for(int a=0;a<IPheights.length;a++){
        logLine = "IP Location="+a+" HeightValue="+IPheights[a];
        logWriter.write(logLine,0,logLine.length());
logWriter.newLine();
}

logWriter.flush();

// Count and write twins points in inPoints to logfile

count = 0;
for(int a=0;a<numPoints;a++) {
        for(int i=0;i<numPoints;i++) {
                if (inPoints[a*2] == inPoints[i*2] && inPoints[a*2+1] == inPoints[i*2+1] && a!=i) {
                        logLine = "Dublett i punkt nr = "+a+" "+i;
                        logWriter.write(logLine,0,logLine.length());
                logWriter.newLine();
                        count++;
                }
        }
}

// Checking max value in inSegments

int max=0;
int index2 = 0;
index = 0;
while (index<inSegments.length) {
        if (inSegments[index]>max) {
                max = inSegments[index];
                index2 = index;
```

```
            }
            index++;
}

System.out.println("Maxvalue "+max);
System.out.println("Index "+index2);

//numSegments = 0;
//inSegments = new int[numSegments];

// -----------------------------------------------------------------
// Trianguling
System.out.println("Trianguling...");
// -----------------------------------------------------------------

// Call the c-routine that creates the constrained Delaunay triangles
int numTriangles = p.createTriangles(numPoints, numSegments, inPoints,
inSegments, outTriangles);

System.out.println("Trianguling Done");

System.out.println("Number of input Points = " + numPoints);
System.out.println("Number of input Segments = " + numSegments);
System.out.println("Counted points in inPoints array = " + inPoints.length);
System.out.println("Counted points in inSegments array = " + inSegments.length);

// Print for test purposes only
System.out.println("Number of Triangles Created = " + numTriangles);
System.out.println("Number of outTriangle Points = " + outTriangles.length);

int COUNTERROR=0;
for (int i=0; i<numTriangles; i++){

            vertex1 = outTriangles[3*i+0];
            vertex2 = outTriangles[3*i+1];
            vertex3 = outTriangles[3*i+2];

            //System.out.println("outTriangles[] = "+vertex1+" "+vertex2+" "+vertex3);
            logLine = "outTriangles[] = "+vertex1+" "+vertex2+" "+vertex3+";";
            logWriter.write(logLine,0,logLine.length());
            logWriter.newLine();

            logWriter.flush();

if(vertex1<2740 && vertex2<2740 && vertex3<2470){

writer.write("3DFACE",0,6);
writer.newLine();
writer.write("  5",0,3);
writer.newLine();
writer.write("7C",0,2);
writer.newLine();
writer.write("  8",0,3);
writer.newLine();
writer.write("0",0,1);
writer.newLine();

vertexLocation1x = inPoints[vertex1*2] - 77325.615;
vertexLocation1y = inPoints[vertex1*2+1] - 75088.074;
vertexLocation1z = heights[vertex1];

vertexLocation2x = inPoints[vertex2*2] - 77325.615;
vertexLocation2y = inPoints[vertex2*2+1] - 75088.074;
vertexLocation2z = heights[vertex2];

vertexLocation3x = inPoints[vertex3*2] - 77325.615;
vertexLocation3y = inPoints[vertex3*2+1] - 75088.074;
vertexLocation3z = heights[vertex3];
```

```
//Vertex 1
writer.write(" 10",0,3);
writer.newLine();
//System.out.println(Double.toString(vertexLocation1x));
writer.write(Double.toString(vertexLocation1x),0,Double.toString(vertexLocation1x).length());
writer.newLine();

writer.write(" 20",0,3);
writer.newLine();
//System.out.println(Double.toString(vertexLocation1y));
writer.write(Double.toString(vertexLocation1y),0,Double.toString(vertexLocation1y).length());
writer.newLine();

writer.write(" 30",0,3);
writer.newLine();
//System.out.println(Double.toString(vertexLocation1z));
writer.write(Double.toString(vertexLocation1z),0,Double.toString(vertexLocation1z).length());
writer.newLine();

//Vertex 2
writer.write(" 11",0,3);
writer.newLine();
//System.out.println(Double.toString(vertexLocation2x));
writer.write(Double.toString(vertexLocation2x),0,Double.toString(vertexLocation2x).length());
writer.newLine();

writer.write(" 21",0,3);
writer.newLine();
//System.out.println(Double.toString(vertexLocation2y));
writer.write(Double.toString(vertexLocation2y),0,Double.toString(vertexLocation2y).length());
writer.newLine();

writer.write(" 31",0,3);
writer.newLine();
//System.out.println(Double.toString(vertexLocation1z));
writer.write(Double.toString(vertexLocation2z),0,Double.toString(vertexLocation2z).length());
writer.newLine();

//Vertex 3
writer.write(" 12",0,3);
writer.newLine();
//System.out.println(Double.toString(vertexLocation3x));
writer.write(Double.toString(vertexLocation3x),0,Double.toString(vertexLocation3x).length());
writer.newLine();

writer.write(" 22",0,3);
writer.newLine();
//System.out.println(Double.toString(vertexLocation3y));
writer.write(Double.toString(vertexLocation3y),0,Double.toString(vertexLocation3y).length());
writer.newLine();

writer.write(" 32",0,3);
writer.newLine();
//System.out.println(Double.toString(vertexLocation3z));
writer.write(Double.toString(vertexLocation3z),0,Double.toString(vertexLocation3z).length());
writer.newLine();

//Vertex 4
writer.write(" 13",0,3);
writer.newLine();
//System.out.println(Double.toString(vertexLocation3x));
writer.write(Double.toString(vertexLocation3x),0,Double.toString(vertexLocation3x).length());
writer.newLine();

writer.write(" 23",0,3);
writer.newLine();
//System.out.println(Double.toString(vertexLocation3y));
```

```
        writer.write(Double.toString(vertexLocation3y),0,Double.toString(vertexLocation3y).length());
        writer.newLine();

        writer.write(" 33",0,3);
        writer.newLine();
        //System.out.println(Double.toString(vertexLocation3z));
        writer.write(Double.toString(vertexLocation3z),0,Double.toString(vertexLocation3z).length());
        writer.newLine();

                writer.write("  0",0,3);
                writer.newLine();

        } else {
                System.out.println("Fel "+ COUNTERROR++);
                System.out.println("vertex1: "+vertex1+" vertex2: "+vertex2+" vertex3: "+vertex3);

        }

        }

        writer.flush();
        logWriter.flush();

} catch (Exception e) {e.printStackTrace();}


}
static {
  System.loadLibrary("delaunay");
}


public static double Interpolate(double x, double y, double[] inPoints, double[] IPheights){

        double dist = 0;
        double top = 0;
        double bottom = 0;

        for(int a=0;a<IPheights.length;a++){
                dist = Math.sqrt(Math.pow((x-inPoints[a*2]),2)+Math.pow((y-inPoints[a*2+1]),2));
                top = top + IPheights[a] * (1/Math.pow(dist,2));
                bottom = bottom + (1/Math.pow(dist,2));
        }

        return (top/bottom);
}


public static Point searchForTwins(double x, double y, LinkedList pointList){

        ListIterator pointIterator = pointList.listIterator(0);
        Point tempPoint;
        Point returnPoint = null;

        while(pointIterator.hasNext()){

                tempPoint = (Point) pointIterator.next();
                if (x == tempPoint.getX() && y == tempPoint.getY()) {
                        returnPoint=tempPoint;
                        break;
        } else {

                returnPoint = null;
        }

        }

        return returnPoint;
}
```

```
	public static boolean searchForTwinSegments(int ref1, int ref2, LinkedList segmentList){
		boolean twin = false;
		ListIterator segmentIterator = segmentList.listIterator(0);
		Segment segment;
		while(segmentIterator.hasNext()){
			segment = (Segment) segmentIterator.next();
			if(ref1 == segment.getStart() && ref2 == segment.getEnd() || ref2 == segment.getStart()
&& ref1 == segment.getEnd() || ref1==ref2){
				twin = true;
				break;
			}
		}
		return twin;
	}
}



class Point {
	double x;
	double y;
	double height;
	public Point(double x, double y, double height){
		this.x = x;
		this.y = y;
		this.height = height;
	}
	public double getX(){
		return x;
	}
	public double getY(){
		return y;
	}
	public double getHeight(){
		return height;
	}
}

class Segment {
	int start;
	int end;
	public Segment(int start, int end){
		this.start = start;
		this.end = end;
	}
	public int getStart(){
		return start;
	}
	public int getEnd(){
		return end;
	}
}

class ReadLine {
	long lines = 0;
	String lastLine="";
	long lastLineNumber = 0;
	BufferedReader reader;

	public ReadLine(BufferedReader reader) {
		this.reader=reader;
	}

	public String readLine() {
		lines++;
		try {
			lastLine = reader.readLine();
//			lastLineNumber = reader.getLineNumber() ;
```

```
                        return lastLine;
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return null;
        }
        public boolean ready() {
                try {
                        if (!reader.ready())
                                System.err.println("Last line "+lastLineNumber+":"+lastLine+"\n");
                        return reader.ready();
                } catch (Exception e) {
                        e.printStackTrace();
                }
                return false;
        }
        public long getLines() {
                return lines;
        }
}
```

Appendix C

Object disposition source code

```
import java.io.*;
import java.util.*;
import java.lang.Double;


        // ---------------------------------------------------------------------
        // Object disposition interface by Ludvig Ljungqvist 2003
        // Part of a Masters project in 3D-GIS, Lund Institute of Technology
        // The proram reads the geometry from a VRML-file in ASCII, and
        // 2D points from 2D GIS points in DXF. The VRML object is copied to
        // the locations in the 2D data and printed to a new VRML file.
        // ---------------------------------------------------------------------


class putOutObjects {
  public static void main(String args[]) {

        try {


        int numPoints = 0;
    int count;

    String line;
    String logLine;

    Point tempPoint;

    File heightFile = new File("merallapfusk.dxf");
    File outFile = new File("outObjects.wrl");
    File logFile = new File("log.txt");
    File lightInFile = new File("il2d.dxf");
    File objectInFile = new File("lampobject2.wrl");

        BufferedWriter writer = new BufferedWriter(new FileWriter(outFile));
    BufferedWriter logWriter = new BufferedWriter(new FileWriter(logFile));
    LineNumberReader reader = new LineNumberReader(new FileReader(lightInFile));
    LineNumberReader objectReader = new LineNumberReader(new FileReader(objectInFile));
        LineNumberReader heightReader = new LineNumberReader(new FileReader(heightFile));

        LinkedList heightList = new LinkedList();
        LinkedList lightList = new LinkedList();

        ListIterator heightIterator;
        ListIterator lightIterator;

    double tempX;
    double tempY;
    double tempHeight;

        line = heightReader.readLine();
        while (line!=null){
                if (line.compareTo("AcDbText")==0){
                        line = heightReader.readLine();
                        if(line.compareTo(" 10")==0){
                                tempX = Double.parseDouble(heightReader.readLine());
                                line = heightReader.readLine();
                                tempY = Double.parseDouble(heightReader.readLine());
                                line = heightReader.readLine();
```

```
                                tempHeight = Double.parseDouble(heightReader.readLine());
                                heightList.add(new Point(tempX, tempY, tempHeight));
                        }
                }
                line=heightReader.readLine();
        }

        line = reader.readLine();
        while (line!=null){
                if (line.compareTo("LS")==0){
                        line = reader.readLine();
                        if(line.compareTo(" 10")==0){
                                tempX = Double.parseDouble(reader.readLine());
                                line = reader.readLine();
                                tempY = Double.parseDouble(reader.readLine());
                                lightList.add(new Point(tempX, tempY, -1));
                        }
                }
                line=reader.readLine();
        }



        heightIterator = heightList.listIterator(0);

        while(heightIterator.hasNext()) {

                tempPoint = (Point) heightIterator.next();
                logLine = "Height = "+tempPoint.getX()+" "+tempPoint.getY()+" "+tempPoint.getHeight();
                logWriter.write(logLine,0,logLine.length());
        logWriter.newLine();
        logWriter.flush();

        }

        lightIterator = lightList.listIterator(0);

        while(lightIterator.hasNext()) {

                tempPoint = (Point) lightIterator.next();
                logLine = "Light = "+tempPoint.getX()+" "+tempPoint.getY()+" "+tempPoint.getHeight();
                logWriter.write(logLine,0,logLine.length());
        logWriter.newLine();
        logWriter.flush();
        }
        logWriter.flush();

        lightIterator = lightList.listIterator(0);

        while(lightIterator.hasNext()) {
                tempPoint = (Point) lightIterator.next();
                if(tempPoint.getHeight()==-1)
                        tempPoint.setHeight(Interpolate(tempPoint.getX(),tempPoint.getY(),heightList));

        }

        lightIterator = lightList.listIterator(0);

        while(lightIterator.hasNext()) {

                tempPoint = (Point) lightIterator.next();
                logLine = "Interpolated Light = "+tempPoint.getX()+" "+tempPoint.getY()+"
"+tempPoint.getHeight();
                logWriter.write(logLine,0,logLine.length());
        logWriter.newLine();
        logWriter.flush();
        }
        logWriter.flush();
```

```
        line = "#VRML V2.0 utf8";
    writer.write(line,0,line.length());
    writer.newLine();
        writer.newLine();

    for (int a=0; a<lightList.size(); a++){


                line ="DEF lampo Transform { translation "+(((Point) lightList.get(a)).getX())+" "+((((Point)
lightList.get(a)).getHeight())-40)+" "+(-((Point) lightList.get(a)).getY());
        writer.write(line,0,line.length());
        writer.newLine();

                objectReader = new LineNumberReader(new FileReader(objectInFile));

                line = objectReader.readLine();
                do {
                                writer.write(line,0,line.length());
                        writer.newLine();
                        line = objectReader.readLine();


                } while (line.compareTo("END")!=0);

        }

        writer.flush();
        logWriter.flush();

  } catch (Exception e) {e.printStackTrace();}

  }

  public static double Interpolate(double x, double y, LinkedList inHeights){

        double dist = 0;
        double top = 0;
        double bottom = 0;

        for(int a=0;a<inHeights.size();a++){
                dist = Math.sqrt(Math.pow((x-((Point) inHeights.get(a)).getX()),2)+Math.pow((y-((Point)
inHeights.get(a)).getY()),2));
                top = top + ((Point) inHeights.get(a)).getHeight() * (1/Math.pow(dist,2));
                bottom = bottom + (1/Math.pow(dist,2));
        }

        return (top/bottom);
  }


}

class Point {
        double x;
        double y;
        double height;
        public Point(double x, double y, double height){
                this.x = x;
                this.y = y;
                this.height = height;
        }

        public void setHeight(double height){
                this.height = height;
        }
        public double getX(){
                return (x-77325.615);
```

```
	}
	public double getY(){
		return (y-75088.074);
	}
	public double getHeight(){
		return height;
	}
}
```