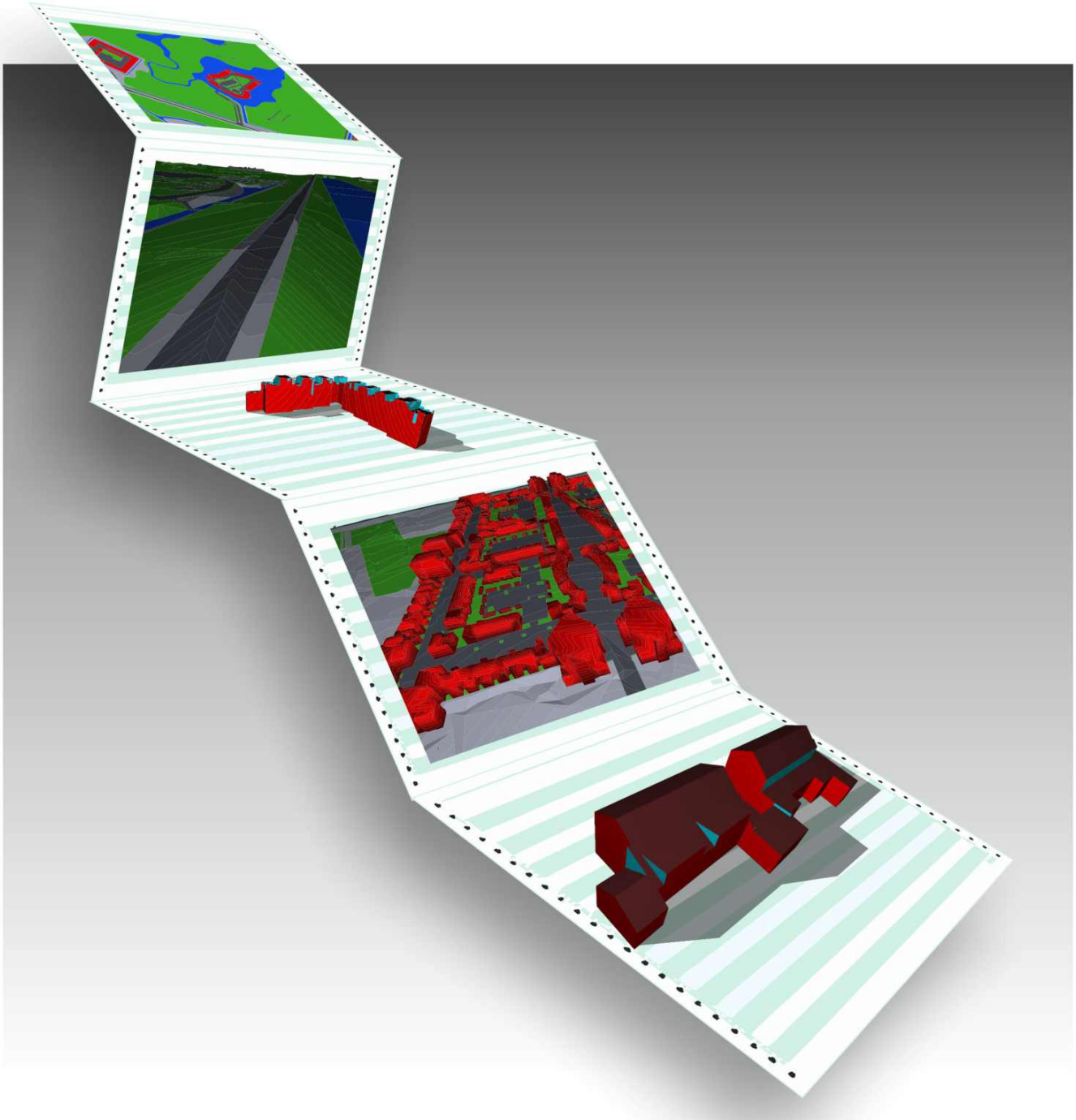
Master Thesis

Automatically Constructing the Third Dimension of IMGeo

Kim van Winden



ICA-3253511
Department of Information and Computing Sciences
Utrecht University
The Netherlands

Supervisors:
Marc van Kreveld (Utrecht University)
Flip van der Valk (Vicrea)
Paul Veerman (Vicrea)

Universiteit Utrecht

vicrea

Abstract

The past year a new Dutch standard for the registration of large scale topography was released, called IMGeo 2.0. This standard includes descriptions and attributes for the majority of objects seen in public spaces. A possible extension to IMGeo 2.0 allows all objects to be modelled in 3D, based on object descriptions and techniques from the CityGML standard. This thesis explores the feasibility of automatically constructing such 3D models by combining 2D geometries and 3D height information. Numerous methods for this type of construction already exist, differing in objects reconstructed, amount of detail, restrictions on construction approach, assumptions on input data and requirements on output data. The approaches created in this thesis aim to construct 3D models in a robust way, accepting sub-optimal results as long as they are a decent and valid representation of reality. Four different construction methods were designed. The first builds large 2.5D terrain models based on point triangulation. These 2.5D surface models form the basis of the actual 3D models. The second and third method construct horizontal roof 3D block models of buildings, with one approach dividing the roof based on point clusters and the other using repeated square subdivisions. The last and most complex method tries to construct 3D building models with full roof shapes using plane and line detection techniques. Degrees of success vary across these methods, with lower success rates the more complex the models become. The constructed terrain model has a number of improvements over a standard triangulation, but terrain generalization is really a field of its own and a lot more improvement can be accomplished. The methods for block models proved to be robust and performed quite well, but can be improved with respect to roof border accuracy. The method that constructs full roof shapes is not robust enough yet, but shows promising possibilities. Overall some creative ways to automatically construct 3D models have been designed, either entirely from scratch or as a modification of existing work, combining techniques from amongst others the fields of geometric algorithms and image processing.

Acknowledgements

It is only when you write the acknowledgements that you realize how many people have actually helped you along during your thesis. Whether directly by reviewing your work or providing new insights, or indirectly by supporting you all the way through, quite a lot of people have become involved in some way. It seems only fair to thank them after a long time of work. So without further ado I would like to thank:

- My supervisors Marc van Kreveld, Flip van der Valk and Paul Veerman for guiding me along and always trying to get me to improve my work.
- Jurre Laven for offering me the opportunity to an interesting graduation project.
- All the colleagues at Vicrea who have helped me out with knowledge and new perspectives during my project, including but not limited to Tom van der Putte, Rob van der Pol, Normen Hamstra and Bjorn Schijff.
- Dean Hintz from Safe Software who helped me out with anything FME related.
- Nik the Brush for designing the front cover and lending me his language.
- And finally of course my family and friends, like Jaqueline, Wilco, Roel and Jeffrey, who were always there for me when I needed them.

Contents

Glossary of terms

AHN2 Current Dutch Height File Version 2
(Actueel Hoogtebestand Nederland 2)

BAG Base Registration for Addresses and Buildings
(Basisregistratie Adressen en Gebouwen)

BGT Standard Registration for Large Scale Topography
(Basisregistratie Grootschalige Topografie)

CAD Computer Aided Design

DTM Digital Terrain Model

ETL Extract Transform Load

FME Feature Manipulation Engine

GBKN Standard Dutch Large Scale Map
(Grootschalige Basiskaart Nederland)

GIS Geographic Information System

GML Geographic Markup Language

IMGeo Information Model Geography
(Informatiemodel Geografie)

LiDAR Light Detection And Ranging

LOD Level Of Detail

NAP Normalized Dutch Water Level
(Normaal Amsterdams Peil)

OGC Open Geospatial Consortium

PDF Portable Document Format

SIG 3D Special Interest Group 3D

TIN Triangulated Irregular Network

XML Extensible Markup Language

Chapter 1

Introduction

In the present world, 3D is a feature gaining in popularity. From movies, TV and games to printers and designs, the addition of a third dimension is an ever more popular feature. The ability to experience depth (or height) like we do in the real world in situations that used to have only two dimensions unlocks a range of new possibilities. Yet with all these possibilities, maps stay flat. In most Geographic Information System (GIS) software the third dimension is absent or included as a single extra value to create something called 2.5D. Now times are slowly changing for this domain as well.

The use of 3D modelling has already entered the world of planning, design and building construction as demonstrated by numerous Computer Aided Design (CAD) programs, so why not store and view geography in three dimensions as well? To allow this in the near future on a national level the new Dutch topography registration IMGeo 2.0 includes a possibility to model and store 3D geography as well, which is explained further in Section 1.1.

This thesis explores the possibilities of the IMGeo 2.0 3D extension, specifically for the local government market that Vicrea serves, and the different methods of automatically generating complete 3D models by combining multiple data sources. The research approach of this thesis is described in Chapter 2. Chapter 3 contains an overview of possible advantages of having access to 3D models and describes 3D applications which would be useful to governments. A description of used data sources and software, and information about existing work on the subject of 3D model generation can be found in Chapter 4. The Chapters 5, 6 and 7 deal with model construction methods for different levels of detail. Finally, a conclusion and further recommendations can be found in Chapter 8.

The rest of this chapter will give a short introduction on the IMGeo 2.0 model, the CityGML model that is used to store and exchange 3D data, the 3D Pilot NL group that works on a national 3D standard and the company Vicrea where this project was executed.

1.1 IMGeo Information Model

In previous years the Dutch large scale topography has been stored in a map containing line features called Standard Dutch Large Scale Map (GBKN). It contains no explicit relations between related objects such as houses and their respective house numbers, storage of semantics is limited and surfaces are only stored implicitly by their bounding line segments. Recent developments have led to the creation of the Standard Registration for Large Scale Topography (BGT) model, a new object-oriented approach to store the Dutch large scale topography that serves as a successor to GBKN. The BGT model is one of many base registrations that together will form a national data service. The BGT specification [Geonovum, 2012a] defines a set of objects with semantic attributes, geometry and topological relations. It also describes the conditions under which objects are allowed to be included, such as a minimum surface area for area objects. These objects, by law, have to be measured in for the entire country and their digital counterparts have to be maintained

by their governing organisations. When the entire country has been measured in the objects should describe the topography of the complete country correctly. As part of the base registration system these objects will also be stored in a national database and will be made available for multiple organisations.

The BGT model is part of Information Model Geography (IMGeo) 2.0[Geonovum, 2012b], from now on just called IMGeo. Of IMGeo only the BGT part is mandatory. This means that all governing organisations have to measure in their objects if they are part of the BGT model. The IMGeo information model further describes optional objects and functionalities. This gives governing organisations a standard for storing information about other objects, some of which are currently already stored but in non-standard ways. These objects could be trees or parts of city furniture.

The optional part also includes the possibility of extending all this 2D information into the third dimension. For this extension the information and data model CityGML is used, which is described in the next section. The information model gives a mapping, where possible, from the different objects described in the IMGeo model to the different classes within CityGML. This allows the modelling of objects in both 2D, 2.5D and 3D.

1.2 CityGML

The CityGML data and information model is an Open Geospatial Consortium (OGC) standard that describes a model for storing semantics and 3D geometries for urban areas. The OGC is an international standards consortium that creates and maintains standards for geospatial data and data exchange. Invented by the German Special Interest Group 3D (SIG 3D), CityGML was originally created as a way to store 3D data. However, over the years it has been considered more and more as an information model describing relationships between objects and specifying attributes to be stored. The data model itself is defined in a number of Extensible Markup Language (XML) schemas. Support for the CityGML file format has been rising gradually and its influence is now increasing outside of Germany as well. Within Germany the standard has been widely adopted.

CityGML defines 5 types of Level Of Detail (LOD), ranging from 0 up to and including 4. Higher levels of detail mean more specific objects and object parts can be included. LOD0 describes a 2.5D terrain model where each object is represented as a polyline, point or surface. On LOD1 a selection of objects is included as block models, where block means that each roof surface is exactly horizontal. LOD2 allows texturing and the inclusion of sloped roofs. LOD3 allows more complex models describing tree skeletons and outside features of houses such as doors and windows. Finally, LOD4 may contain full house models describing even the rooms.

1.3 3D Pilot NL

Under the supervision of Geonovum, an organisation that creates and maintains standards for Dutch geographic data, a working group called 3D Pilot NL has been formed. This group, consisting of representatives from local governments, scientists and geo-focused companies, has been working on creating a standard model to describe 3D data in the Netherlands. It also wants to promote the usage of 3D and educate possible 3D users.

In the first phase of the project the group has looked at the available 3D data, possible usages and existing 3D standards. The final reports show a large amount of available data and use cases. CityGML has been stated as the best option for a national 3D standard. To show the possibilities of the CityGML standard a special XML schema has been created that models the objects from IMGeo as CityGML.

The aim of the second phase has been the education of possible 3D users. Because of the experience from phase one and the actuality of the new model, the creation of 3D IMGeo has been chosen as the focus of the second phase. A website and folder with useful 3D applications has

been created and examples of IMGeo-CityGML files have been provided. Documents have also been written about the problems of maintaining 3D data and advice is given on requirements of and restrictions on 3D IMGeo models. Last but not least, a validator has been written to validate solid volume objects.

There is some overlap between the research in this thesis and the work done by the 3D Pilot NL group. Both clearly revolve around the creation of 3D models, especially for objects that are part of IMGeo. The main differences are that this thesis focuses only on the technical aspects of automatic reconstruction and only existing data sources are used.

1.4 Vicrea

Vicrea is a medium-sized Dutch company based in Amersfoort. The company specialises in geo-IT, delivering software solutions with a geo-spatial aspect. A large focus within the company is on the Dutch local government. For these local governments Vicrea produces a number of software products based on base registrations, for example to facilitate the maintenance of the Base Registration for Addresses and Buildings (BAG). These products provide facilities to maintain, store and access all kinds of data, possibly with geometric components. Most products contain a GIS component which allows registrations to be accessed both from the data side as well as from the geographic side.

One of their latest products is Neuron BGT, a product aimed at maintaining the new IMGeo (and BGT) information models. This software includes the optional objects described in IMGeo. However it does not include the possible extension to 3D.

This thesis project allows Vicrea to keep an eye on current developments of 3D GIS. The inclusion of 3D data and models within their product range could be a unique selling point, open new markets where 3D data is key or offer new kinds of reporting and analysis functions. As Vicrea is historically a 2D focused company the automatic generation of 3D models would allow them to add 3D functionality without gathering new data or having to manually create 3D models.

Chapter 2

Research approach

2.1 Objectives

With the recent rising interests in 3D in general, and the interest of city councils in 3D IMGeo in particular, Vicrea would like to make the step up from 2D to 3D in their products. A simple method of providing access to 3D models is to create them by hand. This is however not part of the work at Vicrea, any type of data conversion is done in an automatic way as much as possible, and the amount of work required to build large 3D models makes the manual approach infeasible. A lot of work has already been done on automatic model construction. With the types of available data and the possibilities and restrictions of 2D IMGeo there is still plenty of opportunity to adapt existing methods or create new methods in order to generate 3D IMGeo successfully. With this in mind, the overall question to be answered in this thesis is:

How feasible is the automatic construction of 3D IMGeo models?

To answer this rather general question a number of different aspects have to be considered, ranging from input data and construction methods to result storage and analysis. To treat all these different aspects the following sub-questions will be answered:

Which uses can 3D IMGeo models have? Different 3D models have different uses. Possible uses will have to be explored, including the type of 3D model required to accomplish such a use. The different models created in this thesis may not fit all uses.

Which data sources are available and can be used for the construction of 3D models? It should be made clear which data can be used for automatic model construction and what properties this data has.

In which ways can different types of models be constructed? Each type of model, and even each level of detail, has a different method of construction. These methods will be described in detail and tested on functionality and result quality

Which limitations come with automatic model construction? The input data and construction methods limit the amount of detail and object types that can be constructed automatically. It should be clear what these limitations are.

These questions will be researched in the rest of this thesis and will be answered in the conclusion in Chapter 8.

2.2 Scope

The field of 3D model construction is a very large one and covering all aspects of it would be infeasible given the time frame for this thesis. To provide a feasible domain the following restrictions are taken into account:

- *Only detail levels 0, 1 and 2 are constructed.*
  Higher levels of detail include structures inside of buildings and these are impossible to be generated automatically from data available in this project. Some researchers such as Benner et al. [2005] have done this type of construction, but this has no relevance for the type of large scale topography that IMGeo represents. Also note that while LOD2 may include textures, no attempt at texturing is included in this thesis.

- *On LOD0, objects constructed are limited to the objects on surface level, with the exception of simple bridges.*
  Available data prohibits the construction of tunnels, bridges and other objects that are not on ground level. The objects included in LOD0 will correspond to the land covering dataset specified by the BGT.

- *On LOD1 and LOD2, object types constructed are limited to Buildings and Other Structures.*
  Buildings and Other Structures (such as garages and sheds) share a general form and can generally be constructed in the same way. The work in his thesis is limited to these object types as the construction of others objects such as trees and tunnels is a field in itself. A number of different objects and their corresponding problems will be mentioned briefly without adding new work.

- *Model construction is only done automatically. No 3D editor will be used, manual influence will be limited to specifying input, output and optional parameters.*
  Manual work mainly includes the original writing of scripts and programs, and in later use the adaptation to different data input and output sources. The data conversion itself will be fully automated from 2D to 3D.

- *Input data is limited to sources that are already available. No new data will be acquired.*
  If extra data would have to be acquired this would severely limit the usefulness of developed methods. With a large number of data sources available on a national level this should be enough for the majority of construction methods.

2.3 Methodology

The first step is to check the specific uses and possibilities of 3D models and the availability of existing data sources. The knowledge gathered in this step is used to create a number of construction methods for different object types and levels of detail. Methods will adapt, or use parts of, existing solutions from the academic or commercial fields. The focus will be on complete and correct methods, which do not necessarily have to be fast. Methods should always return valid results, even if they are sub-optimal.

Evaluation is done by testing the methods on a large number of objects. Results can be evaluated in three different ways. The first is technical validity, whether the resulting data is correct with respect to its file and data format standards. The second is a subjective and visual inspection that checks whether the resulting model resembles the object in reality and whether its looks match human expectations. The last way is to quantify the difference between the input and output data, for example by checking the distance between measured points and created structures. Care should be taken with these kinds of evaluations as a better number does not necessarily mean the model is better. It is possible to over fit a model onto the data. An example of this would be a triangulated surface that incorporates all input points. Its error would be zero, yet no real structure will have been found.

Chapter 3

Possibilities

There are a large number of applications that make use of 3D models. However, the possibilities of 3D models for local governments may not be so obvious. To show why we would want to create 3D topographic models this chapter describes a number of example uses that would fit with the tasks and processes of local governments. At the end of this chapter the requirements of each example will show what kind of uses the models created during this thesis have.

3.1 Visualisation

Spatial awareness for council workers One of the features offered by the BGT is to store data from real-world objects and allow decisions to be made based upon this data. The addition of 3D models to an object offers two distinct improvements. On one hand the object itself will be in 3D, giving a better idea of the object being worked on. On the other hand the area around the object is also in 3D, giving insight to the context of an object. The availability of this extra information might lead to better decisions, or to cost reductions as on-site visits by council workers become less necessary. Bentley has published a white paper by Fredericque and Lapierre [2009] describing the possibilities of 3D GIS for cities which can be used to manage infrastructure.

Better communication towards citizens When the council will be working in an area, communication about this work towards citizens in the area often includes maps of the before and after situations. Next to these maps images of 3D models could be included. If the communication is digital even a 3D Portable Document Format (PDF) file can be included. The 3D models will give citizens a perspective of the situation that matches their own experience. A field study in Germany by Warren-Kretzschmar and Tiedtke [2005] has shown that different forms of visualization in communication is appreciated by citizens and increases their understanding.

3.2 Analysis

Flood risk Floods are not an uncommon risk in the Netherlands. From the west the country is threatened by powerful floods coming from the North Sea which could potentially flood a large part of the country, which happens to be below sea level. From the other side a number of large rivers flood the country every once in a while. There are even areas designated to be flooded on purpose to relieve the water threat. Water height is normally indicated with height above Normalized Dutch Water Level (NAP), the level at which most heights in the Netherlands are based. Unless you are familiar with it, the height above NAP is not that useful to know. With the availability of a 3D terrain model it would be easy to see which areas will be affected by a specific water height. If 3D models of objects are also available it

would even be possible to see how much effect the water has on specific objects, as shown by Mioc [2011]. This information could be useful while planning new areas or while making decisions during flood risks.

Noise pollution In a small country like the Netherlands every new road or railway will pass built-up areas. Inevitably this will lead to problems like noise hindrance. Calculating the effect of a new road or railway on noise levels in the surrounding areas would be a useful feature. Commonly 2D calculations are used for this, but the addition of 3D could lead to more detailed results. By including a 3D model of the surrounding area the influence of 3D objects (like sound walls) can be included in the calculations. It would also be possible to see the difference in noise levels between the first and top floor of an apartment building. As an example, Kurkula and Kuffer [2008] have implemented a noise analysis model on a part of the city Delft in The Netherlands.

Air pollution The environment is a hot topic in the current world and with every new project the effects on the environment have to be considered. Air pollution is one of the possible effects that has to be taken into account. To create accurate air dispersion models for roads the 3D information of for example buildings next to the road have to be included. This way it is possible to include turbulences and measure pollution at different heights, instead of the commonly used method of analysing pollution only for a specific height. With 3D models it is also possible to visualise the air pollution in a more detailed way. Wang et al. [2008] have done this kind of 3D air pollution analysis on a part of The Hague in The Netherlands

3.3 Registration

Sewer models The majority of houses nowadays are connected to the sewer system. Underneath each city exists a vast network of pipelines. Councils already register the locations of sewage pipes and wells. Converting this into a 3D registration will allow for visualization of the entire network. It also allows for analysis of gravity-based sewer flow, possibly with the inclusion of blocked pipelines. While the IMGeo model does not have any objects defined for sewer pipes, adding them to the model is definitely possible. The manholes are already included as optional objects within IMGeo. A good overview of the possibilities of 3D sewer models has been given by Van Capelleveen [1999].

Split buildings The inside of a building in the IMGeo model is not subdivided any further. This means that an apartment building consists of one object and one geometry, independent of its size. It would be useful to allow subdivision of a building into different apartments. This would give a more direct association between address and location. It also allows 3D spatial queries like finding all apartments surrounding a specific apartment. This query in 2D would give too many results as a lot of apartments that are not even close would also be included. A further subdivision of buildings is not included in the IMGeo model. The possibilities and requirements of such a building split in the form of a 3D cadastre have been described by Stoter and Salzmann [2003].

3.4 Model requirements

The three types of uses above have different requirements on their models. Visualisation requires models that look good, analysis requires a correct building structure and registration requires models with a high level of detail. The type of research in this thesis focuses on the second category, for which the models have recognizable roofs and walls and the set of surfaces form a correct volume usable for analysis. These models can be used for visualisation, but no effort is made in this regard and textures are not included. As the maximum level of detail is set to 2 and sewer objects are not supported, the two example uses mentioned under registration are also not a part of this thesis.

Chapter 4

Model generation

The goal of this chapter is to provide a solid base of knowledge on which the construction methods will be built. Section 4.1 defines a number of terms that may not be clear. Section 4.2 shows a small overview of existing work around the subject of 3D model construction. In Section 4.3 the different data sources used are described in some detail. Section 4.4 lists the software used at Vicrea and how they are used for this thesis. Some objects that are not the focus of this thesis, but are worthy of a mention, are listed in Section 4.5 together with some of their related problems and solutions. Finally, Section 4.6 explains in short the functionality and methodology of a number of algorithms used during model construction.

4.1 Definitions

Line Throughout the rest of this thesis the word line will be used to describe a line segment, specified by two end points. If a different type of line is meant, this will be specified. There are two other possible lines. The first is the infinite line, which has a location and a direction, but no end points, as it continues till infinity. The second possibility is the polyline, which consists of one or more line segments, connected to form a string of straight line segments that could describe curves.

Building Whenever the word building or structure is used it is meant to describe a single physical structure. This could be made up of a number of different houses. For example, a row of neighbouring houses will be seen as a single building. Of course, if the method works for the entire building it will also work for a single house within it, but the reverse may not necessarily be true.

Solid A solid is a type of 3D geometry which can be valid under a strict set of rules. By definition a solid describes a volume and consists of a number of faces. All faces must connect exactly without gaps, missing or duplicate vertices, or overlaps. It can also not contain any additional surfaces or intersect itself.

DTM A set of surfaces that describes the 2.5D terrain of an area is often called a Digital Terrain Model (DTM). Such a terrain model does not include objects that are not part of the terrain, such as buildings and trees. A common type for a terrain model is a Triangulated Irregular Network (TIN), in which the entire surface is represented as a set of triangles.

LOD0 Level of detail 0 in 3D IMGeo describes a model in which every object is represented either as a 2.5D surface or as a 3D polyline or point, which represents the 3D footprint of an object. Area objects from the BGT model on terrain height must connect without gaps or overlaps to provide a complete terrain model. Objects above or below terrain height must have their footprints placed at the correct height in the model.

LOD1 Level of detail 1 is the first level with actual 3D models. Eligible objects, such as bridges, buildings and other structures, are represented as a block model. Here, block refers to the fact that any top surface of the model must be horizontal, not that the object is actually constructed out of blocks. These block models must be valid solid geometries.

LOD2 Level of detail 2 allows the inclusion of more objects such as trees and all types of city furniture. The objects mentioned on LOD1 have more detail added to them. The main difference for buildings is that top surfaces no longer have to be horizontal, allowing for more detailed buildings with sloped roofs. Textures can also be included. Once again the building models have to be valid solid geometries. The other objects have no such restrictions.

4.2 Existing work

The field of 3D object reconstruction has existed for some time now, showcasing a large number of problems and different approaches to solutions. Some of this work is focused on the construction of terrain models. Terrain points could be filtered from building and tree points automatically (Rottensteiner and Briese [2002], Bretar et al. [2004]). Others aim for the generalization of terrain models and use road detection to improve this process (Akel et al., 2005).

A different direction is taken by those trying to recognise specific objects or structures in unstructured data. Examples of this are automated detection of tree skeletons in Light Detection And Ranging (LiDAR) data (Binney and Sukhatme [2009]) and automatic construction of TIN models for buildings (Elaksher and Bethel [2002], Alharthy and Bethel [2002] and Rottensteiner et al. [2005]).

A surprisingly large number of people have worked on the combination of 2D and 3D data. Some approaches subdivide the object boundaries to simplify the roof detection steps (Vosselman and Dijkman [2001], Kada and McKinley [2009]), others use this boundary to limit the number of possible planes (Durupt and Taillandier [2006]) or to generate possible roof intersection lines from its skeleton (Brenner [2000]).

This short overview shows that there is not really a single solution to automatic 3D model generation. Each approach has its strengths and weaknesses and no approach is perfect. All of problems within these solutions contain the single problem of detecting a specific structure in unstructured data the way humans could do it.

The combination of 3D and 2D data has been researched extensively, the 2D information used in a multitude of ways, from simply selecting points to limiting search space and providing a starting structure. These ways indicate the differences of data-based and model-based approaches. Data-based approaches try to create a model from the given data while model-based approaches try to fit one of a set of different models on to the data. In general, data-based approaches provide more flexibility but are harder to compute. Methods in this thesis are mostly data-based.

The doctoral work of Oude Elberink [2010] proved to be of great importance for this. His work provided a great starting point, offering a stepwise process for the construction of 3D buildings to which new methods could be added.

4.3 Data sources

To generate 3D models a number of different data sources are used. These sources are or will be available on a national level meaning the methods described here can be used for the entire country of the Netherlands. Besides these sources no other data is required. The rest of this section will describe the data sources in more detail. All these sources are available for the Haverleij area in Den Bosch, the same area used as a test site by the 3D Pilot NL. For that reason it is also the test area for this thesis. An overview of the area can be seen in Appendix A.

4.3.1 BGT

The BGT is the mandatory part of IMGeo, as mentioned in Section 1.1. It consists of 15 different object types, the majority of which have an area geometry. The rest of the objects have polyline geometries, there are no point geometries in the BGT with the exception of label locations.

- Road
- Road support
- Railway
- Bare terrain
- Plant covered terrain
- Water
- Water support
- Building
- Other structure
- Bridge
- Tunnel
- Other construction
- Separator
- Unclassified object
- Functional area

By law, all proprietors of public spaces (e.g. city councils, forestry commission, road maintainers) must deliver a complete and correct dataset of their objects. This is a process spanning multiple years, done in three steps. The first step is an initial delivery of currently available data, which does not have to be correct yet. The second step is to make all the objects from step one correct. The last step is completely and correctly filling the dataset. This entire process is scheduled to be finished in the year 2020, as shown on E-Overheid [2012].

4.3.2 IMGeo

The full IMGeo model allows for the storage of more and more detailed objects. There is no minimum size requirement on objects and similar adjacent objects do not have to be aggregated. The main addition is a number of object types which can be divided into two groups:

City furniture is a group of object types that describe small elements seen on the street which contains objects such as benches, poles and trash cans, the majority of which consist of point geometries.

Other objects are a number of different object types that are only part of IMGeo and can not be categorized as city furniture. These are legislative areas, trees and other separators.

Next to these new object types IMGeo allows further specification of BGT object types. As an example, a plant covered terrain can now specify the types of vegetation (roses, grass, bushes) instead of just the general usage (park). The extra detail of IMGeo allows the objects to be used at different departments of local councils, such as green maintenance or city planning. These departments would currently use their own registration systems.

4.3.3 AHN2

Current Dutch Height File Version 2 (AHN2) is the second version of the national height dataset. Measured by aerial LiDAR flights, AHN2 provides height data for the entire country. Data is available with a point density of six to twelve points per square meter. While the datasets should be land covering, they can contain gaps in practice. Also, there will not be any data points on water surfaces. AHN2 is available in a number of different file formats. The ones used for this thesis are:

Point clouds are large files that contain all the original unfiltered points. Corresponding attributes are number and index of returns, colour and angle.

Unfiltered grids are grid structures that have a single height value for each 0.25 square meter cell, which is the average value for all points within that cell.

Filtered grids are the same type as unfiltered grids with the exception that all cells belonging to non-terrain objects, such as houses and trees, are replaced by nodata values.

The biggest downside of AHN2 is that it is not necessarily up to date. New flights to update the data are few and far between, which can lead to discrepancies between height data and 2D data. Alternatives may be for local councils to do their own LiDAR flights, or to create point clouds from aerial stereo photos which are flown more regularly.

The AHN2 data of the Den Bosch area used for this thesis was provided by Het Waterschapshuis [2012].

4.4 Software

This chapter describes the different software products used by Vicrea, which are also used for the implementation of methods described in this thesis. The 3D component in these products is generally still being worked on. The three products described here take care of storage, visualisation and modification.

4.4.1 Oracle Spatial Database

Oracle offers a database product called Oracle Spatial that supports geometries. It has full support for 3D coordinate systems and 3D geometries in version 11. One downside is that while standard 3D objects such as points are supported in Oracle Locator, specific 3D geometries such as solids and point clouds are only supported in full Oracle Spatial. This is unfortunate as the full Oracle Spatial license is a lot more expensive than an Oracle Locator license.

4.4.2 Cadcorp SIS

Cadcorp SIS is the GIS software that serves as a viewer in most Vicrea products. It has a 3D viewing window in the current version, but 3D support is still somewhat limited. The next version of Cadcorp, version 8, will feature a new and improved 3D viewer and will also support CityGML files. Cadcorp can read from most spatial databases including Oracle Spatial, so 3D data stored in Oracle can be viewed in future Vicrea applications.

4.4.3 Safe Software FME

Safe Software Feature Manipulation Engine (FME) is an Extract Transform Load (ETL) tool with support for geographic data. As Dutch resellers, Vicrea uses and sells this product a lot. It provides an easy user interface to read and write data from a large number of formats, and provides among other things coordinate reprojections and geometric operations to modify data. Functionality is provided in the form of transformers, each with a single specific task which can be combined into complex transformations. These transformers, a description of which is provided by Software [2012], are assumed to work correctly. An example of how a method constructed in FME would look is shown in Figure 4.1.

While FME has no support for the Dutch 3D coordinate reference system, EPSG 7415, it does support 3D geometries as of version 2012. With support for CityGML, point clouds and Oracle databases it makes the processing of data used for this thesis easier. For more complex operations there is the option to process data using python scripts, which increases the flexibility of FME.

4.4.4 Software usage

All of the methods mentioned in this thesis are implemented using FME, either by a combination of standard transformers or by using transformers and Python scripts. FME is used for reading and writing the data, performing simple geometric operations and inspecting the results in their 3D viewer or in the 3D PDF format. Python is used for the more complex and custom algorithms.

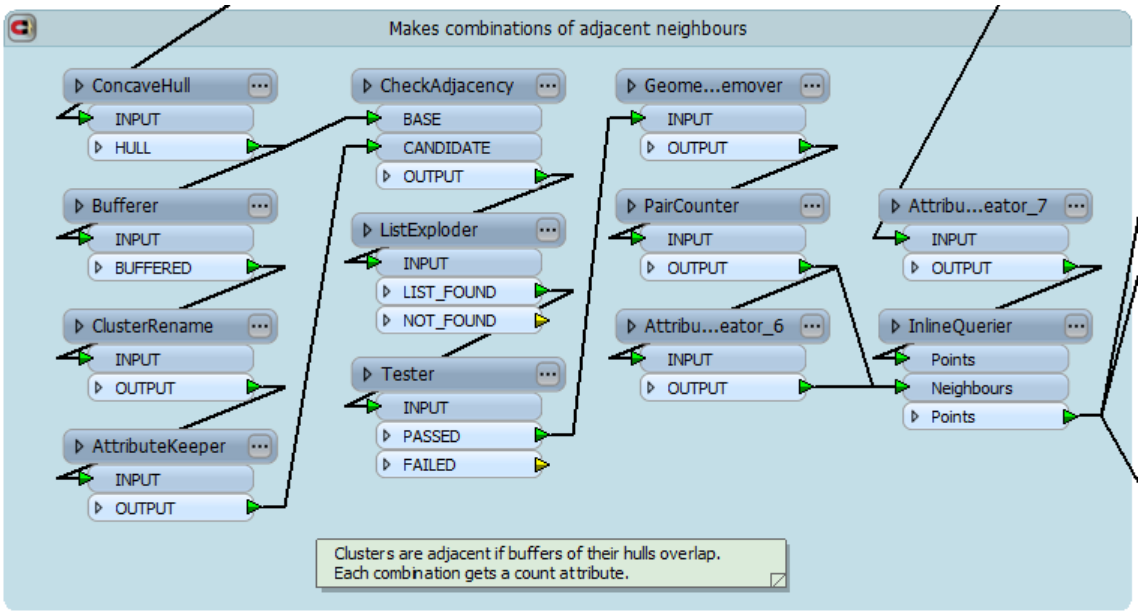


Figure 4.1: Part of an FME Workbench which detects neighbour relations between point sets.

The input and output data of each method is stored in an Oracle Spatial database, with the input data stored in the Vicrea IMGeo table space. Cadcorp SIS is used for data inspection and visualization.

While this software has been used to implement the methods in this thesis, the methods themselves are quite general and should be implementable in any programming or scripting language. The use of this software has only made the implementation of some parts easier.

4.5 Other objects

The work in this thesis is mainly focused on the automatic construction of buildings. However, other objects can be constructed automatically as well. In the next three sections the problems and possible approaches for bridge, tree and city furniture objects will be described in short. The entirely different approaches required for each of these object types, next to the complex approach for buildings, might explain why the construction of a complete 3D model is so complex.

4.5.1 Bridges

Bridges are hard to create for a number of different reasons. Firstly, bridges come in all shapes and sizes, which means that a single construction method for one model will not suffice. Secondly, the height of bridges means that the bridge boundary is no longer an accurate way to select points from height data, as points underneath the bridge may also exist. Lastly, overlapping bridges will cause missing laser data on the bottom bridge, so some form of interpolation will be required.

Oude Elberink [2010] describes a method to construct bridges of the highway type where there is no (or very little) structure above the road surface. His approach is based on a surface-growing algorithm that uses the Hough transform (see Section 4.6) for initial seeds. Surface polygons are extended and combined whenever it is likely they belong to the same road. This method allows bridges to grow underneath each other. These boundaries are triangulated and given thickness to represent a bridge as a solid model. The approach is quite powerful and can be used for a large number of standard road bridges. More complex bridges such as girder and suspension bridges that cover up the road surface can probably only be constructed manually, at least for detail levels higher than 0.

The bridges created for LOD0 are supposed to represent the bottom surface of the bridge. They also connect to the surrounding DTM. If the bottom surface is curved or sloped and is extruded into a horizontal surface that is higher than all the bottom vertices, the result will look more like a building than a bridge. It seems there is not really a good way to model a decent looking bridge in LOD1. Tunnels share this problem, with the added complexity that their entry and exit must form a surface in the DTM model. Simply extruding a bottom surface upwards can and will produce very bad results in that case.

4.5.2 Trees

As pieces of nature, trees do not really work with the concept of straightness. This means modelling them in a vector-based system will always be an approximation, one where a higher quality will require a larger number of separate objects. The large number of different trees also means there is no such thing as a single tree model. Luckily the detection of trees in height data is quite feasible. A decent approach that calculates tree height and crown width is given by Tiede et al. [2005].

Trees can not be represented as block models, so they will not appear in LOD1. In LOD2 they can be included as a single two-object model: a column and a ball. The size of the ball is determined by the crown width, the column height depends on tree height. An example of such a model can be seen in Vosselman [2003]. To model different types of trees a number of different basic models can be used. For example, for coniferous trees the ball can be replaced by a cone.

More detailed tree models belong to LOD3. These are parametrized structures that approximate the branching structure of the tree. Binney and Sukhatme [2009] show a method to estimate these parameters and model the branch structure like a kinematics model where the tree is made up of a large number of tubes connected by joints, where the parameters of each joint specify the orientation relative to the previous tube. The tree is constructed iteratively, starting with the largest branch (the trunk) and gradually adding smaller branches. While these types of parametrized trees look great, their level of detail and construction complexity are probably too much to be used for the large models built in the rest of this thesis.

4.5.3 Furniture elements

City furniture objects such as garbage bins and street lights may be small, but should be included in a full 3D model. Without such objects the model will look empty. Surprisingly little has been written about the detection, construction and inclusion of such objects. These objects can be quite complex, but contain little variety. Within the zone of a single city council the number of different types of garbage bins, street lights and benches will be limited. For example, if a 3D model of each type of bench is available, the only task left is to orient it correctly at the right location. This assumes that the make and model of the furniture is known for each location. If this is not the case a single model could be used for each type of furniture, so a single bench model would be used everywhere.

As for each object the location is known as a point, the last unknown is the orientation. For specific types this can be inferred from the surrounding data. A street light will face the nearest street, a bench is generally orientated towards a road or path. For other objects, like garbage bins, the orientation does not matter much and any choice will do.

As each model is re-used many times it would be more efficient to store each model only once. Within CityGML this has been included using implicit geometries, where a single model is referenced multiple times and transformed using a transformation matrix. While more efficient for CityGML, the use of such a storage method in databases requires an extra conversion step when opening the data in standard viewers such as Cadcorp SIS. It might be an idea to store the original model separately and let each object contain the transformed model and the corresponding transformation matrix. This way the data can be viewed directly from the database and be written away to CityGML files without too much effort.

4.6 Algorithms

A number of standard geometric algorithms will be used in the rest of this thesis, some of which can be quite complex. These will be explained in short in this section. Standard geometric algorithms such as line and plane intersections, distance calculations and buffer creations, while used extensively, will not be explained. A detailed description of general geometric algorithms, including some of those mentioned below, can be found in de Berg et al. [2008].

4.6.1 Delaunay triangulation

The Delaunay triangulation is a method to triangulate a set of points in 2D. It has the property that for each triangle no other points are inside its circumscribed circle. It maximizes the minimum angle over all triangles, resulting in compact triangles. There are numerous methods to construct a Delaunay triangulation such as continuously flipping triangles in a bad triangulation to improve it, or starting with a single triangle and repeatedly adding points until the triangulation is done. The result is a set of triangles that completely covers the input point set. This algorithm can be used as a simple way to create a TIN surface from a set of 3D points.

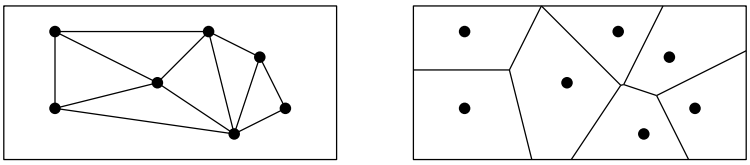


Figure 4.2: A Delaunay triangulation of a point set (left) and its dual Voronoi diagram (right). Note that the edges of the actual Voronoi diagram continue indefinitely, and a section is clipped here for visualization purposes.

The dual form of a Delaunay triangulation is a Voronoi diagram. It is a set of surfaces created from a set of input features in which every surface describes the area closest to a single input feature. In its most basic form the input features are points and all resulting surfaces are convex polygons. The boundary of such a polygon describes a location that is equidistant between two points. The vertices of the Voronoi diagram, where multiple surfaces meet, are equidistant between three or more points. There are a large number of Voronoi variants, including those that work on line and area features and those that consider other points than the closest ones. The Voronoi diagram shows an area of influence and can be used to fill an area based on a number of input geometries. An example of both the Delaunay triangulation and the Voronoi diagram can be seen in Figure 4.2.

4.6.2 Hough transform

Originally coming from the field of Image Processing, the Hough transform is now widely used to detect a large number of parametrized object types in point sets or images. The generalized Hough transform has been described by Duda and Hart [1972]. In its generalized form it can be used to detect a number of parametrized geometries such as infinite lines, planes, circles and cylinders. In its most basic form the Hough transform is used to detect an infinite line in a set of 2D points. The trick of the transform is to map each input point into another coordinate system called Hough space, and stored in a 2D array filled with zeroes at the start, called the accumulator array. An infinite line can be represented by its slope and its distance to the origin. These two parameters form the axes in Hough space. For each point it is not known to which infinite line it could belong, so all possible infinite lines are considered. This means iterating over all possible slope values and calculating the distance parameter given the point coordinates and slope value. Values are mapped to the correct bin in the accumulator array, increasing the value in that bin by 1. This way, each input point increases a bin for each slope value. Once all points have been

entered, high valued bins in Hough space represent line parameters supported by a large number of points. These line parameters can be used to create a number of infinite lines back in the original coordinate system.

4.6.3 Douglas-Peucker simplification

The Douglas and Peucker [1973] algorithm is a method for polyline simplification which reduces the number of vertices on a polyline. The start and end points are always retained and the line in between them forms the starting line for the algorithm. Iteratively, the worst point is added to the current polyline, increasing the number of line segments by one. This addition stops when all points are within a certain distance from the polyline. While the algorithm is meant for polyline simplification, it can also be used to remove extraneous vertices on straight lines using a very small margin.

4.6.4 Dijkstra's algorithm

Dijkstra's shortest path algorithm (Dijkstra [1959]) is a fast and conceptually simple single-source graph search algorithm. It can be used to find the shortest distance from a node in a graph to all other nodes. Its use is limited to graphs without negative edge weights. The algorithm starts with a single source node, with distance zero to itself, and a set of unvisited nodes with distance infinity. Each round the unvisited node with the shortest distance to the source is expanded. All its unvisited neighbours have their distance values updated and the next round starts. Once no more nodes can be visited, each node has the shortest distance value from the source to that node. Simple improvements include stopping once a target node has been reached or searching from both the source and the target node. This algorithm functions as a simple route finder that could be used for navigation systems.

4.6.5 Alpha shapes

An alpha shape is a set of lines representing the shape of a set of points, originally devised by Edelsbrunner et al. [1983]. This can either be a convex or concave hull, depending on the alpha value used during construction. A good way to look at alpha shapes is as the set of edges between every pair of input points that lie on the boundary of an empty circle with radius X. As any circle outside of the point set will be empty, the result will always contain at least an outside hull of the points. If a circle fits entirely inside the point set, the result will be a donut geometry. A very large radius means the part of the boundary usable for a point set approximates a straight infinite line, allowing this method to generate convex hulls as well. Within this thesis, this method is used to create concave hulls.

Chapter 5

Level of detail 0

The most basic form of 3D in CityGML is level of detail 0 (LOD0). Every object described in IMGeo can have a LOD0 representation, which comes in the form of a 2.5D geometry. This could either be a 3D point, polyline or surface, depending on the 2D geometry of an object. Note that whilst the geometry of each object must be 2.5D, the complete model that contains the objects does not have to be. For example, bridges and tunnels can overlap with other objects.

For structure objects like buildings the geometry represents the ground surface. From the definition in the IMGeo model[Geonovum, 2012b], the collection of surfaces on ground level must form a non-overlapping and land covering terrain model. This means that there is no need to store a DTM separately. These LOD0 geometries form the basis upon which all other 3D geometries will be built.

The most basic approach uses a triangulation of height data. Based on the results of the basic approach a number of possible solutions to clearly visible problems in the results are proposed and implemented. The end of the chapter gives a further list of untested improvements.

5.1 Basic approach

This section describes the basic approach for generating a LOD0 model based on a triangulation. First the concept of relative height is explained to show which objects will or will not be included in the result. After this, a short description is given of this kind of terrain triangulation within FME.

5.1.1 Relative height

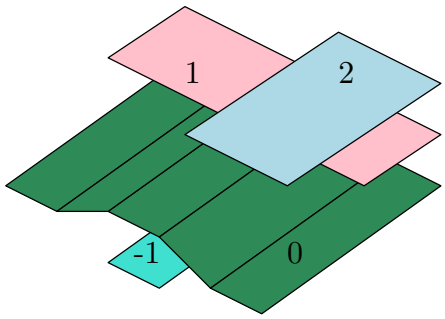


Figure 5.1: A number of objects in 3D with their respective relative height values. It shows the stacking nature of relative height, and the terrain with height 0 that divides the negative underground and positive raised objects.

In the 2D data of IMGeo the only notion of a third dimension is that of relative height. The concept of relative height is specific to IMGeo and gives an indication of object ordering. A relative height of 0 means an object is on ground level. Positive values are above ground level, while negative values are below it. The relative height should not be confused with physical height (i.e. in meters) or a height level. It is possible for two objects with the same relative height to be tens of meters apart in height. Likewise, it is possible for an object with a lower relative height to be higher than an object with a higher relative height. The only thing relative height says is that if two objects overlap, their relative height values indicate their height order. An example of relative height can be seen in Figure 5.1.

A typical example of relative height are bridges, which have a strictly positive relative height. Objects with a relative height unequal to zero should have their 2.5D representation placed on the appropriate height in space. Where two objects with different relative height values meet, their borders should connect without overlap or gaps. The construction of LOD0 in this thesis only includes objects with relative height 0 to form a complete terrain model. The only exception to this are bridge objects as shown in Section 5.2.2.

5.1.2 Triangulation

A good way to model a DTM is by using a TIN geometry. In this network the terrain is modelled as a collection of triangles. It has the benefit of allowing varying complexity. A mostly flat area will use few triangles, a jagged area can be modelled accurately using a lot of triangles. The triangulation for LOD0 is based on the AHN2 filtered grids representing the terrain height. The technique for creating TIN models from raster data is not new. A method to accomplish this was described by Fowler and Little [1979] as early as 1979. The basic principles from that method still apply today. Points are iteratively added to the network until all points lie within a specified maximum height difference to the model. The resulting model will then be split into pieces, forming the geometry of area objects.

Within FME there is basic support for creating TIN models. The general work flow for this is:

1. Make sure the available height data and IMGeo objects match.
2. Add additional points around the heigh data to prevent badly fitted triangles around the edges of the model.
3. Generate a surface model of the height data using the Delaunay triangulation. The Delaunay property guarantees that no triangle, when viewed in 2D, has a fourth point in its circumscribed circle and that the minimum of all angles across all triangles is maximized. In essence, compact triangles are preferred. Some more information about this triangulation can be found in Section 4.6.
4. Drape all features over the model. This sets the z values of all existing vertices. Point objects are now ready.
5. Clip the triangles from the surface model with the previously draped polyline and polygon objects. This sets interior and boundary of these objects to the correct height.
6. Aggregate all the clipped parts belonging to an object in to a single geometry.

The result gives every object with relative height 0 a LOD0 representation. Because all parts are cut from the same triangulation the different objects connect properly without leaving any gaps. The tolerance parameter used for the triangulation should be dependent on the model size and its intended use. For large models a high tolerance is fine to show the overall terrain contours, for small models a low tolerance will include more detail.

5.2 Extension

There are a number of improvements possible for the general construction of TIN models. The improvements described here are specifically aimed towards the IMGeo model and the available datasets. Its aim is to solve three clearly visible errors in the standard method. The first problem

is water moving up and down as part of the terrain. The second problem is the inclusion of bridges, which are not part of the terrain model but will connect to them. The last issue is the inclusion of smooth steep walls in to the model. These problems and their possible solutions are described in the next sections.

5.2.1 Flat water

The basic problem with water areas is that there is no height data available for them. This means that the model for a water area is based on the height data around it. This is especially a problem with steep walls around the water, leading to jagged water surfaces. Yet even with correct height values all around the edge of the water, the water will not appear really smooth as the tolerance of TIN generation allows the height values at the edge to be ignored. A schematic example of this problem is shown in Figure 5.2. Preferably a body of water should be the same height everywhere, ignoring the small height differences in rivers. A solution for this is to pick a single value around the water to serve as water height. The minimum height value can be used for this, if we assume that the water level is the lowest point in the vicinity. A step-by-step method to accomplish this:

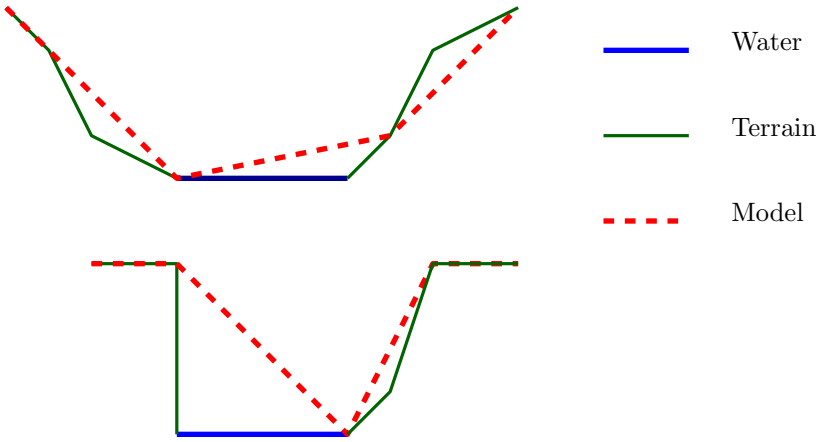


Figure 5.2: A side-view of two terrain parts and their associated faulty models. On top a reasonable model near sloped edges. The bottom shows the large error existing near vertical walls.

1. Aggregate all touching water objects into a single polygon. This guarantees there will not be a vertical wall in the middle of a water surface.
2. Select the minimum height value from a small buffer around the polygon.
3. Delete all height values within the polygon from the height grid. This guarantees a flat geometry.
4. Split the polygon into lines and set their z value to the minimum height.
5. Use the lines as break lines in the triangulation.

A break line is a line that has to be included in the resulting TIN model. This forces the triangulation down to water level. As the break lines go all around the water object and there is no height data left in-between, the inside of the water surface will be flat. The result of this improvement can be seen in Figure 5.3.

5.2.2 Bridges

Even though bridges are not the focus of this thesis, a small attempt has been made on including these in the LOD0 model automatically. The challenge with bridges is in the connection with the underlying terrain model. As the LOD0 representation should be the bottom surface of the bridge, the data from the AHN2 unfiltered grid is not always usable as it represents the top of

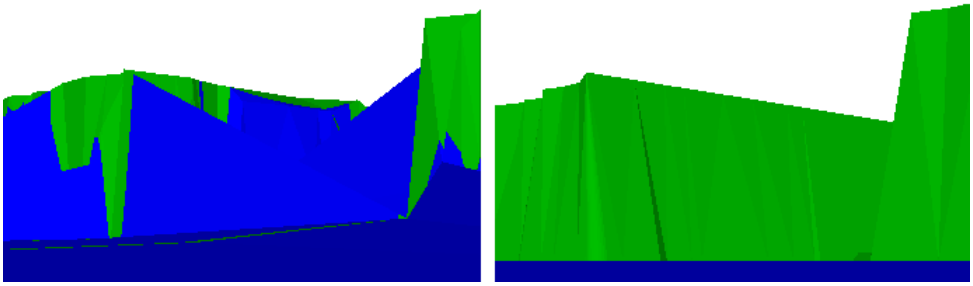


Figure 5.3: Original (left) and improved flat water (right).

the bridge. A better solution is to derive the bridge height from the areas where it connects to the terrain model. By identifying the connecting edges and setting them to the proper height, the rest of the bridge can be interpolated. A possible method to accomplish this:

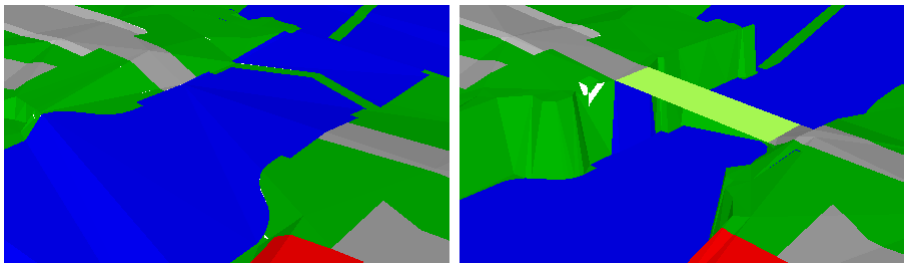


Figure 5.4: Original without bridge (left) and improved with bridge (right). The wall of water on the right is caused by shifting the water break line, as the bridge break line may not be moved.

1. Select all objects with relative height 0 touching a bridge object.
2. Determine the shared lines between the bridge and these objects.
3. Set the height of each line or vertex of a line to the maximum height value in the area around that vertex or line.
4. Use these lines as break lines in the standard triangulation.
5. Create a new surface model using just these lines as input.
6. Create the bridge surface using the same method as described in the standard triangulation, but this time on the newly created bridge model.

This method only works under the assumption that a bridge connects exactly to objects at ground level on at least two sides, the bridge is modelled as a single object from side to side and the surface itself is not curved. As the lines are used in both the bridge surface model and the terrain model, the connection between the two will be correct. The only bridge in the dataset can be seen in Figure 5.4.

Realistically, these assumptions are too strict and do not match with a lot of bridges in reality. It would be safe to say that on LOD0 the inclusion of bridges is also manual work. Even if the bridge is constructed correctly, the overall model in which it is placed can be wrong. Let us consider the following restrictions.

- Object boundaries may not be changed.
- Surfaces should be assigned to the correct object.
- Vertical surfaces are not allowed.

With these restrictions in mind, imagine a bridge that starts on the boundary in between a section of water and a construction. The side of the construction will be a vertical wall between

the section of water and the bridge. To prevent this vertical wall one of the break lines should be moved. However, each of the options will lead to a new problem. If the bottom break line is moved towards the water, either the water boundary is changed or the wall surface is assigned to the wrong object. If we move the top break line the boundary of the bridge will have to change or the bridge will not connect to the construction.

5.2.3 Steep walls

Steep walls suffer from the tolerance allowed in the triangulation. This can make walls appear a lot less steep than they are, as the points that form the top and bottom edge of the wall do not have to be included. Another point is that the rectangular height values from the grid do not correspond well with the ridges in reality, leading to jagged edges which are especially visible near steep walls.

One possible solution is to detect these walls beforehand and make sure they appear correctly. The height grid can be seen as an image, on which techniques from image processing can be used. One of these techniques is called edge detection, which gives each cell a value of steepness depending on its neighbouring cells. There are a lot of different methods, one of which is called a Sobel operator. Using a 3x3 matrix it estimates the gradient in both x and y direction and combines these into a general gradient estimate. Selecting all cells with a value higher than a certain threshold produces a set of points that can be considered edges. A full description of the Sobel operator has been written by Vincent and Folurunso [2009].

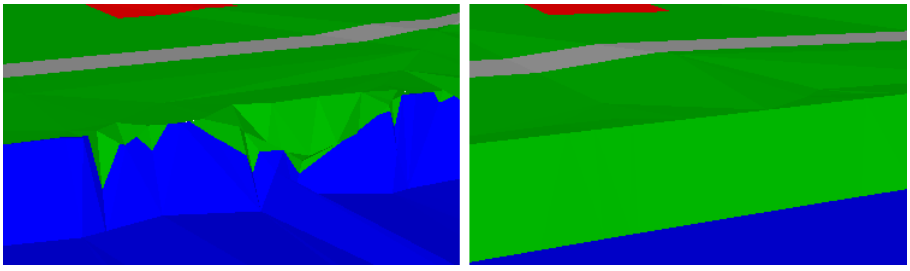


Figure 5.5: Original (left) and improved walls (right).

The next step is to convert these points into usable lines. An operation called the Hough transformation can be used for this, which is described in Section 4.6. It converts each parametrized object into a different parameter space. In this case it converts every point into all possible infinite lines going through that point, representing the infinite line in a different coordinate system using angle and distance to origin. The other coordinate system is called the accumulator array and can be seen as a large counter. Each point will increase a large number of counters in the accumulator array. After adding all points the locations with the highest values represent infinite lines with the largest number of points on or near those lines. Those infinite lines are most likely along steep walls.

The last step is to break each infinite line up into line segments. First all edge points within a certain distance of an infinite line are selected. After this, clusters are formed along the length of that line. Clusters must have points within a specific distance of each other and must also have a minimum amount of supporting points. The first and last point of a cluster determine the segment length. These segments will probably correspond to steep walls in reality. The segments should be duplicated and set to the correct height, either the maximum or minimum height value in the vicinity. These break lines should be moved apart slightly as break lines in the same location do not work in a triangulation. This movement is described in Section 5.2.4. An example of the improved smooth steep walls can be seen in Figure 5.5.

A downside to this approach is that the height grids only have one value for every 50 centimetres in each direction. This means that a detected edge could be spread out over more than a meter, so both steep and less-steep walls could show up the same. Correct selection of parameters such

as gradient threshold and break line movement distance is important to get this approach working as intended.

5.2.4 Break lines

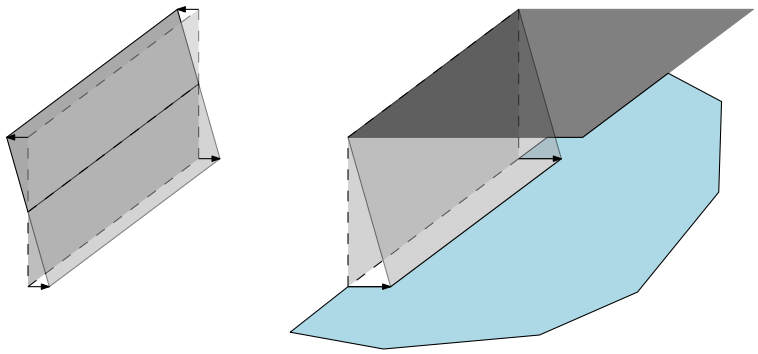


Figure 5.6: The two types of break line movement near a steep wall (left) and underneath a bridge that crosses water (right).

As all these solutions use break lines to force specific parts of the model there is a chance that these break lines are spatially related. If two lines overlap they should be moved apart. This happens in the case of vertical wall break lines mentioned previously, but can also occur when a bridge connects to land exactly on the edge of shore. In this case the break line of the water will be moved, as the bridge break line is also included in the bridge itself. This is the best of two undesirable solutions, as mentioned in Section 5.2.2 An example of these types of break line movements can be seen in Figures 5.6. If somehow two break lines cross in 2D, one of them should be removed as they can not be included in the triangulation and do not really have a proper explanation in reality.

5.3 Results

Comparing the results from the extended triangulation as described in the previous section to the results of the standard triangulation, the total number of triangles went up from 15,000 to 45,000. This is mostly caused by extra height values in water areas, the extra detail forced in by break lines and the large number of vertices included by stroked arcs. The computation time also saw a large increase by about a factor fifteen. Even with several performance improvements left to make the improved version will always be a lot slower to compute than the standard version. Most expensive operations are clipping the large number of triangles into aggregated surfaces and converting the edge grid from the Sobel transformer into single points.

Overall the improved version leads to a more accurate model. Detected walls look a lot more smooth, the water is no longer hilly and some bridges can be included. However, some issues remain. As can be seen in Figure 5.4 the bridge does not connect on the correct height on one side. This is because of a lack of height data on that side of the bridge. This might be improved by also looking at the unfiltered height grid if the filtered grid has no data available. Also visible is the wall underneath the bridge which is part of the water. As the break line from the water is moved inside the water, the surface up towards the bridge break line will count as water. It is hard to imagine a reasonable solution that avoids vertical walls, does not change the 2D object boundaries and leaves the bridge connection intact.

Another issue is caused by the wall break lines, which are assigned to the minimum and maximum value in their area. This makes the break lines very susceptible to outliers, causing walls to be made steeper than they are in reality. A better solution might be to look at the

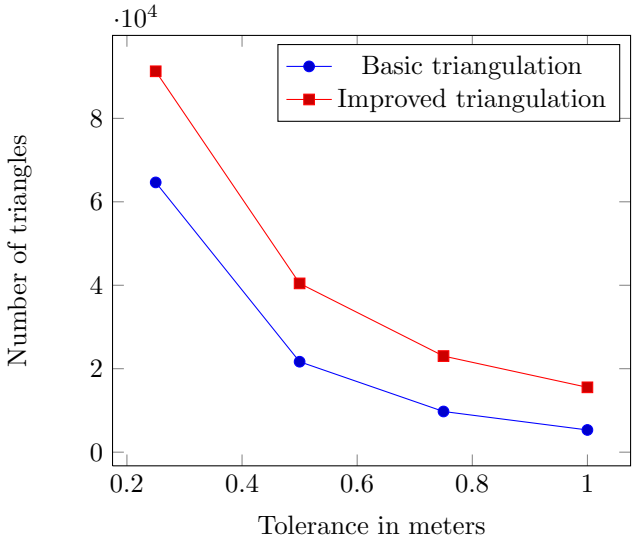


Figure 5.7: The complexity of the terrain model displayed as a function of tolerance.

median values on both the high and the low side. This negates the influence of outliers and also improves support for walls that are not exactly horizontal.

Lastly, the used tolerance value of one meter caused some hills or ridges to appear quite jagged, as points on the edge may or may not be included or the triangulation connects the points in a suboptimal way. A lower tolerance may improve this, but will have a large impact on the performance. Lower tolerance values would be more feasible when used on small datasets.

Further evaluation on the resulting models is quite hard to accomplish, as there is not really a suitable evaluation method. A numerical evaluation based on point height differences seems meaningless, as it is already known the maximum difference is at most one meter. All that is left is technical validity, whether the set of objects form a connected terrain model made up of planar surfaces. As the objects are correctly connected in 2D and all objects are cut from the same triangulation, this is automatically the case.

One thing to consider is the impact the tolerance value and the improvements described in this chapter have on the complexity of the model. A test was performed on a square kilometre area testing four different tolerance values and using both the basic as well as the improved triangulation. The result of this test can be seen in Figure 5.7, using the number of triangles in the model as an indication of model complexity. It clearly shows the effect of the improvements on the model complexity, but also shows the larger effect caused by the decrease of the tolerance value. The extra complexity caused by the added improvements seems to grow the smaller the tolerance value becomes, but this growth is small compared to the complexity added by lowering the tolerance value. With this much effect, the choice of tolerance becomes an important factor, allowing a trade-off between complexity and construction speed on one side and model detail on the other side.

Even with the above mentioned issues still left and a subjective evaluation method, the three improvements have been shown to produce decent results. After generalisation and performance improvement steps this method can be used to create LOD0 models of arbitrary datasets.

5.4 Further improvements

Even though the three solutions mentioned before improve the quality of the resulting model, there are a number of improvements left to add.

- The polygon objects in IMGeo can contain a large number of vertices on a small area. One

reason for this is stroked arcs, where an arc is represented by a large number of lines as part of a polyline. As the edges of water areas are used as break lines any stroked arcs belonging to water objects will come back in the TIN model. This can be seen in Figure 5.8. A simplification step could reduce the number of vertices for each arc before the model is built. This will reduce the complexity of the model.

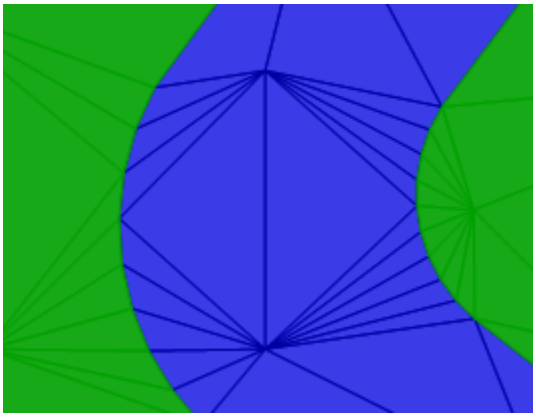


Figure 5.8: Triangulated stroked arcs in a water object.

- While a TIN model allows complexity to vary across the model, the tolerance with which the model is built is constant. The triangulation could be rewritten to determine the complexity of an area and use a lower tolerance in complex locations.This could improve the amount of detail in specific parts of the model.
- The usage of amongst others clustering, the Hough transform and the Sobel operator introduces a large number of parameters into the model creation. Some parameters will work well with default values, others will need to be adapted to a specific model. Preferably these parameters would be removed from the model creation, either by estimating usable values or by circumventing the need for them, making the method easier to use across multiple datasets.
- The Hough transform currently runs over the entire set of edge points. Edge points are assigned to a nearby infinite line with the highest overall support. This may lead to suboptimal segments being created during clustering. An example case can be seen in Figure 5.9. A possible solution is to build clusters of edge points and do a Hough transform for each of these clusters. This will produce a best line for each group of edge points.

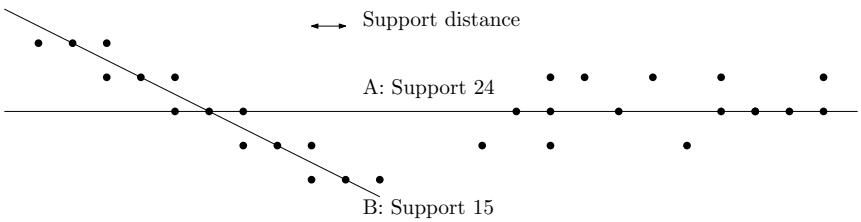


Figure 5.9: Even though line B forms a better fit for the points on the left, a segment will be created along line A as it has a higher support.

- The points used in the surface model are connected using a 2D Delaunay triangulation. While the Delaunay property produces good triangles in 2D, it does not necessarily produce good terrain models. An example of this can be seen in Figure 5.10. Preferably we would

want lines following contours, not crossing directly over them. This could be accomplished by considering contour lines while building the TIN model. However, this would require rewriting the entire triangulation and the standard FME SurfaceModeller transformer could no longer be used.

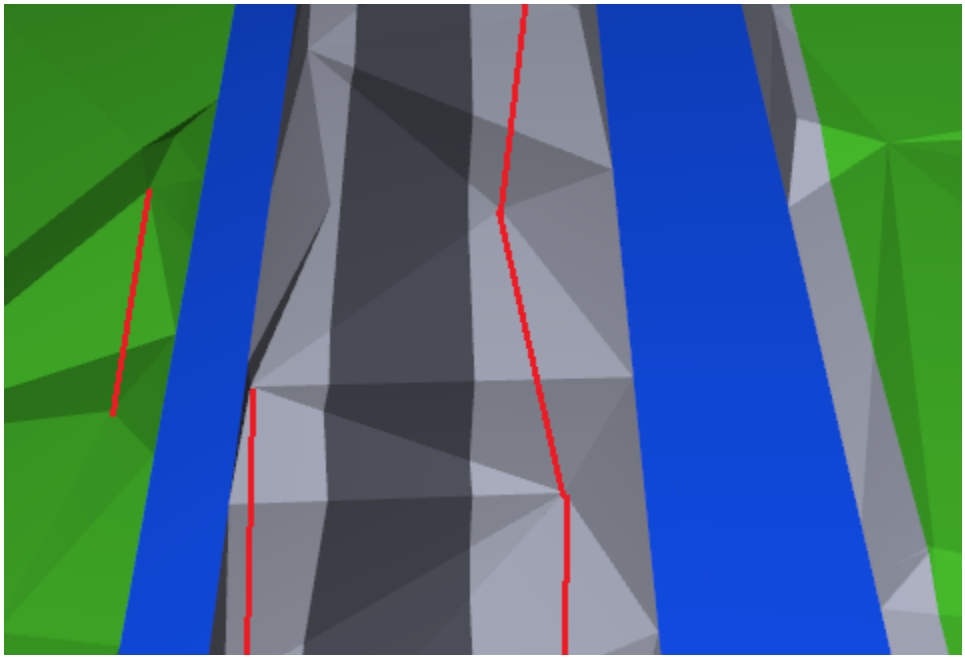


Figure 5.10: A situation in which a different triangle selection could lead to a better result. An example of flipped triangles which would follow the contours better is shown red.

Chapter 6

Level of detail 1

The lowest detail level with true 3D models is level of detail 1 (LOD1). At this level all included objects are represented in block form. This means a solid object with a flat horizontal top. This block must still connect to the surrounding terrain at ground level. The basic way to do this is quite simple. The object footprint created in LOD0 is extruded to a specific height, depending on the height values within the object boundaries. As long as the chosen height is higher than any of the original points in the footprint this will produce a valid solid object. There is one possible improvement that would be quite interesting. This improvement comes from the clear observation that buildings rarely look like a single block. It might be useful to split an object footprint into multiple parts, each of which gets extruded to a different height. For humans this subdivision is easy to make, however it is quite hard to make a computer recognize different building parts. This chapter describes a method that automatically constructs a LOD1 building consisting of multiple different blocks, based on AHN2 data and the LOD0 surfaces created in Chapter 5. The first part of this method subdivides an object boundary in such a way that each resulting horizontal surface corresponds to a part of the roof. The second part uses this subdivision to construct a valid solid model of the object. These two parts will be discussed in the following two sections. At the end of this chapter the results of this method will be evaluated and a number of possible improvements will be given.

6.1 Automatic building subdivision

For small buildings, a single block is enough as representation. For larger or more complex buildings a representation consisting of multiple blocks of different heights would be better suited. To create the separate blocks the building footprint has to be subdivided. Preferably footprints are subdivided in a way such that:

1. Every surface area is larger than some minimum size.
2. Adjacent areas have a noticeable height difference.
3. Each area represents a single part of a building, either a horizontal roof or a connected set of sloped roofs.
4. The combined areas represent the entire object correctly.

The rest of this section describes the steps used to split a building in such a way that these properties are met and that the result is always a subdivision that covers the object surface completely and without overlaps.

6.1.1 Clustering & Voronoi diagrams

The most important part of this approach is to cluster the (down sampled) height grid points in such a way that each cluster is grouped in both horizontal distance and height. To accomplish this

type of hierarchical clustering a graph is created with all the grid points as nodes. An edge is added between two nodes if they are within a specific horizontal and vertical distance. This process can be executed quite fast as the integer values of coordinates can be used perfectly as indices in a spatial index system, reducing the list of possible neighbours immensely. The resulting graphs are explored using a graph search algorithm and all nodes in a single connected graph are added to the same cluster. The boundary of each cluster is then calculated using Alpha shapes, which are described in Section 4.6 and can be used to create concave hulls. Overlapping or touching parts between clusters are removed.

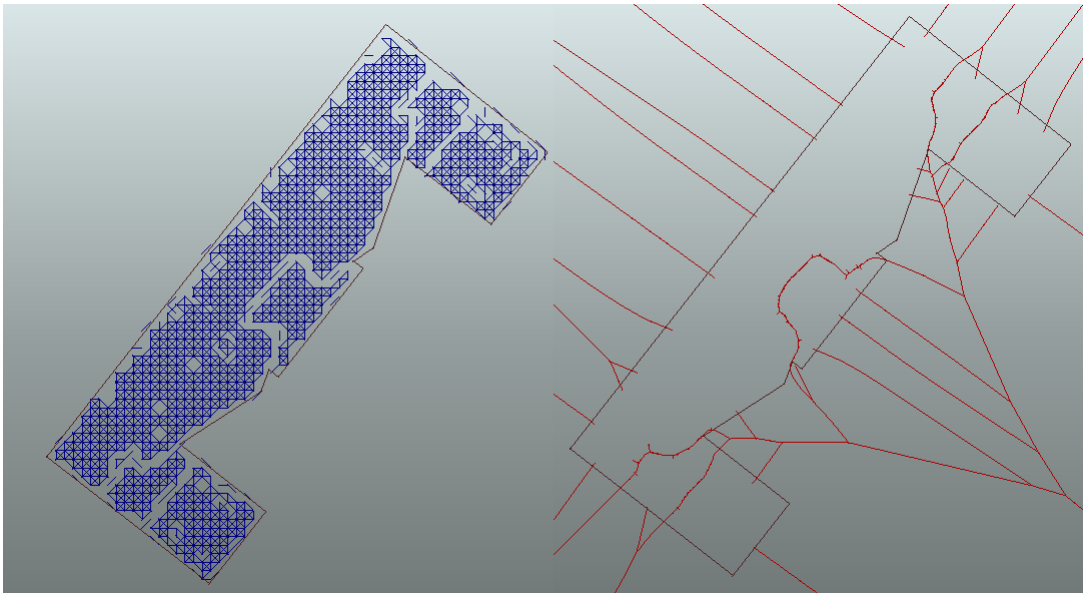


Figure 6.1: Clustered points (left) and the Voronoi diagram created from those points (right).

To subdivide the footprint properly the polyline in-between each adjacent pair of clusters is used. This polyline, called a subdivision line, is created by calculating a Voronoi diagram of all the points on the (densified) cluster boundaries. These points are used instead of the cluster boundaries as FME does not support Voronoi diagrams on line features. All Voronoi lines that are not (partially) inside a cluster are then merged together with the object footprint to create an area subdivision. Two stages of this step are shown in Figure 6.1.

6.1.2 Line improvements

While the previous steps result in a correct subdivision, it is not a very smooth one. Each subdivision line consists of a very large number of line segments. This can be improved using the simplification algorithm by Douglas and Peucker, as described in Section 4.6. The number of vertices in the subdivision line will decrease and in the end the subdivision line will consist of a number of long and straight line segments. Note that the start and end point will not be changed.

Another point of concern is the intersection between boundary and subdivision. The locations at which the subdivision lines intersect the object boundary will not be very accurate, as it is based on a coarse clustering. An assumption is made that if an intersection happens close to a corner in the object boundary, the intersection should probably be at that corner. The subdivision line endpoints are therefore snapped to nearby boundary nodes. The result of this step can be seen in Figure 6.2.

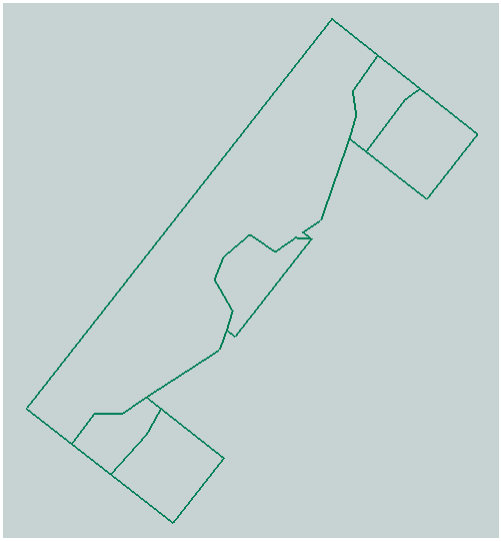


Figure 6.2: Simplified lines created from the Voronoi diagram shown in Figure 6.1.

6.1.3 Area improvements

The resulting subdivision can contain some small area features. These can originate by accident in the previous step and should therefore be removed. But even if they were a correct part of the subdivision, the level of detail associated with such small areas is not consistent with LOD1. Furthermore, it is possible for separate clusters to result in adjacent areas with a similar height level. As we want a noticeable height difference between adjacent blocks, these areas are merged together. Lastly, the snapping of subdivision line endpoints can create areas outside of the object boundary. These are removed.

6.1.4 Roof height

The newly created area features still need to be set to the right height. The height of each individual area is set to the mean value of all height points within its boundary. At the very least each roof section is higher than all of the vertices in its floor surface. At the end of this process each object has a floor surface and one or more horizontal roof surfaces at the correct height that cover the floor surface perfectly.

6.1.5 Model properties

The goal of this approach was to subdivide object footprints in such a way that the properties mentioned at the start of this section hold. The removal of small areas by merging them into larger ones guarantees the first property, minimum area size. By merging adjacent areas with similar heights the second property is met. The method of clustering points by creating graphs between similar points ensures not only that horizontal building parts are clustered but also that sloped roofs become part of the same cluster. This is the third property. The last property is that the resulting subdivision is non-overlapping and covering within the object boundaries. Clipping the geometries with the boundary ensures that the object model can not grow outside of its original footprint. Using lines for subdivision guarantees both that overlaps can not occur in the result and that the entire footprint will be covered. At the very least, when no proper subdivision lines exist, the original footprint is turned into a block model.

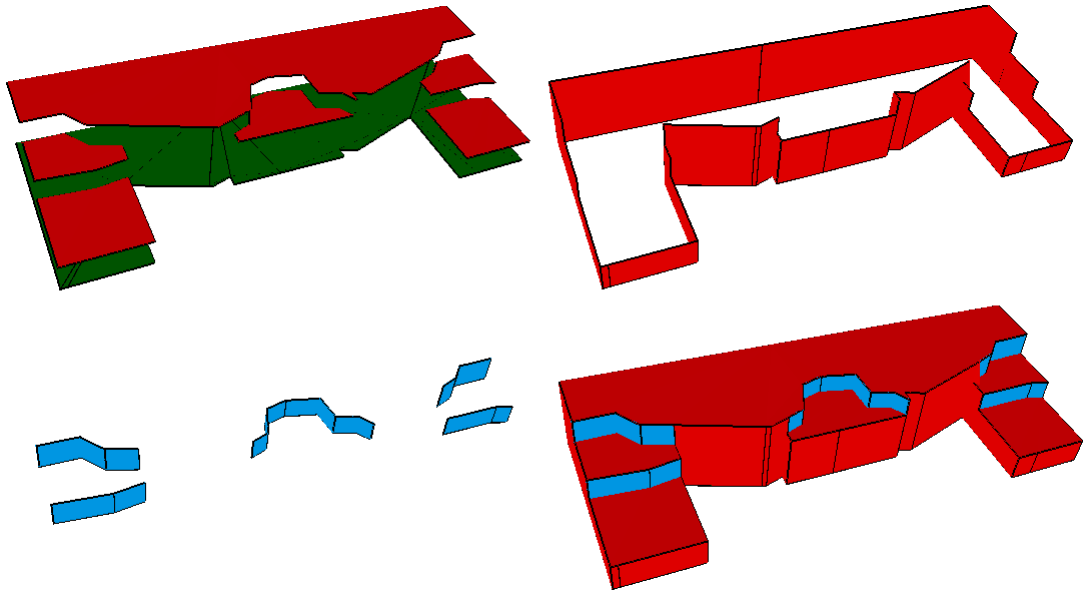


Figure 6.3: An example of the different building parts: roofs and floors (top left), walls (top right), balconies (bottom left) and the combination into a building (bottom right).

6.2 Valid solid construction

The last part of constructing LOD1 models is combining the floor and roof surfaces into valid solids. To accomplish this task, vertical surfaces have to be added that connect the roof surfaces to the floor. The general approach is equal to that of Ledoux and Meijers [2011], where vertices are visited per column, creating valid surfaces that contain additional vertices to connect correctly with adjacent surfaces. Key differences are the sloped floors and multiple roof surfaces for each object. These multiple surfaces mean that there are not only vertical surfaces on the object boundary, but also inside it. These inner surfaces, from now on called balconies, are created separately. An example of the four different surface types can be seen in Figure 6.3.

6.2.1 Walls

A single surface is created for each edge on the floor polygon. All roof vertices and lines above the edge are selected and considered for inclusion in the order at which they lie on the edge. At a specific node column the roof line in the direction of the endpoint of the edge is chosen. This line will be the exit of the node column. To create valid solids, any vertex in the node column between the exit height and the previous height is added to the surface boundary. After these additional vertices the exit line itself is added to the boundary. This process repeats until the end of the edge has been reached. At this point the vertices of the floor line are added and the result is a closed planar surface representing a wall, that incorporates the floor and roof edges.

6.2.2 Balconies

A balcony is the vertical surface between two roof surfaces. All roof edges are compared to each other. Those pairs of edges that overlap entirely form the basis for balcony surfaces. Surface construction happens in the same way as with walls, except that for each surface there are only two node columns and two lines. Once again intermediate points in the column should be added to the surface boundary.

6.3 Results

The method described here was used to create 595 buildings from the Den Bosch dataset, an example of which can be seen in Figure 6.4. The resulting solid objects were validated with the GeometryValidator transformer inside FME. Of the 595 different objects, 589 were valid solids. The errors on the six invalid objects were: dangling faces (4), non-closed solid (1) and non-solid object (1). The majority of errors were caused by suboptimal snapping in the subdivision phase, causing (near) collinear lines and surfaces in the final model. The method is considered a successful way to create valid solid geometries, as valid solids are produced for 99% of the buildings tested.

When using a quantitative evaluation, quite an improvement is visible between a single horizontal roof surface or a set of roof surfaces for each building. Comparing the height of all 129,142 points to the height of the surface containing it, the mean absolute height difference is lowered from 2.07 meters to 1.03 meters, halving the error value. The standard deviation also dropped from 2.06 meters to 1.39 meters. Even though an average error of one meter seems like a lot, it is unlikely that this value can be improved much further. Even if the subdivision was perfect, large errors would still be caused by sections of height data that are too small to receive their own surface, and the practical problem of trying to model non-horizontal objects with horizontal surface.

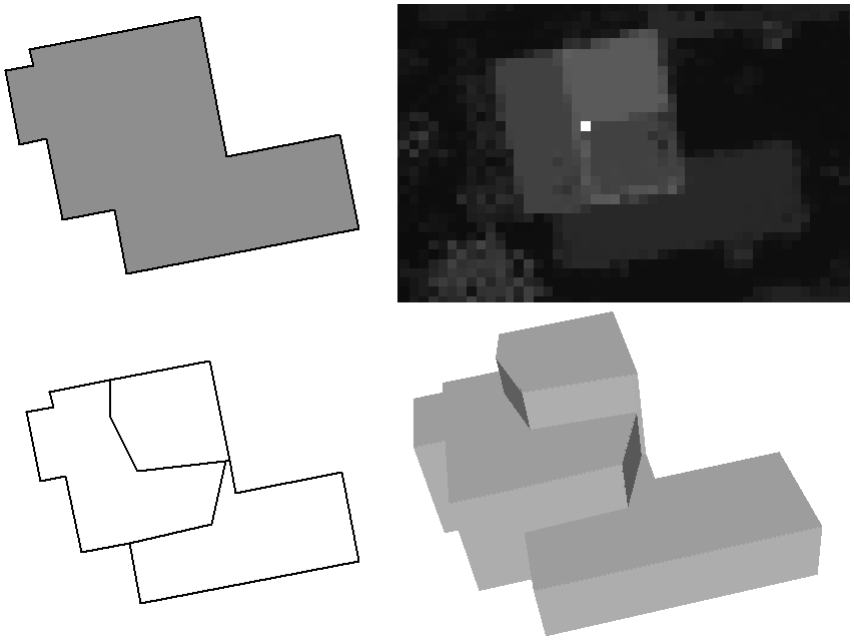


Figure 6.4: An example of input and output.

Considering the semantic correctness of the resulting models it seems that the method is also working as intended. Roof surfaces are created in the right location for areas that have large height differences with respect to their neighbours. Structures do become a lot more recognizable, as can be seen in Figure 6.5. Overall this construction method for LOD1 seems to perform well in all types of evaluation.

That does not mean there are no issues with the resulting model. While snapping to corners has been included in the subdivision process, the selection of the corner which should be snapped to can be improved immensely. In the current method subdivision lines are snapped to closest corners, whether that leads to good models or not. Incorrect results of this selection can include collinear lines and very small building sections. Another issue in the results is that the shape of the resulting roofs does not lead to rectangular shapes where they would be expected. Some of these issues are caused by inaccurate source data, others are caused by the general approach of the

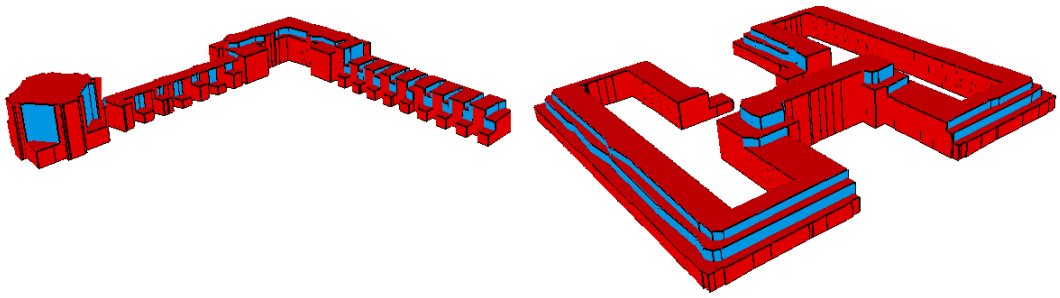


Figure 6.5: Two complex structures made more recognizable by subdivision.

subdivision, which does not aim for rectangular shapes. Examples of these two issues are shown in Figure 6.6. These problems are discussed further in Section 6.4.

Overall the approach for constructing subdivided LOD1 models results in technically correct solid objects. The subdivision process correctly identifies the number of roof surfaces, their location and size, but does not always result in an accurate shape.

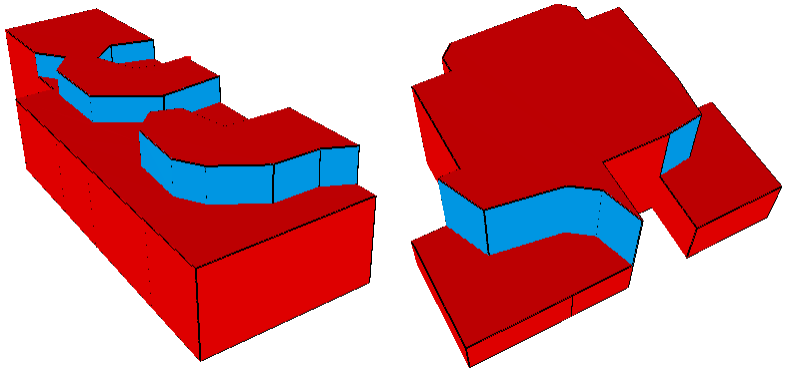


Figure 6.6: Two types of problems: misshaped smooth roof surfaces (left) which should be square, and a subdivision made worse by snapping to the wrong corner (right) when the original subdivision line would have been better.

6.4 Further improvements

There are a number of improvements left to make on this approach for generating LOD1 models. Two of these, improved snapping and straighter shapes, will be discussed in the next two sections. Another point that could be improved is that with this method, sloped and horizontal roofs that meet at some height will all become part of the same roof surface. This is not a large issue, but the approach described here could not make a split between these two types of roofs if required. Of course, if this kind of split does matter, it might be a better idea to look at level of detail 2 (LOD2) instead. Lastly, this approach could also work using AHN2 point clouds instead of grids. This has not been tested and there could be issues with scaling up the amount of data.

6.4.1 Improved snapping

There are two common problems related to snapping, both of which are shown in Figure 6.7. The first problem is when a subdivision line will overlap with or is close to a boundary edge. This respectively leads to self-intersecting faces and dangling faces. This problem can be solved by

paying attention to nearby boundaries while snapping. Instead of snapping to the closest corner point, the end points of the subdivision line should snap to a nearby corner such that no other boundary edge, other than those next to the intersection points, is close to the subdivision line.

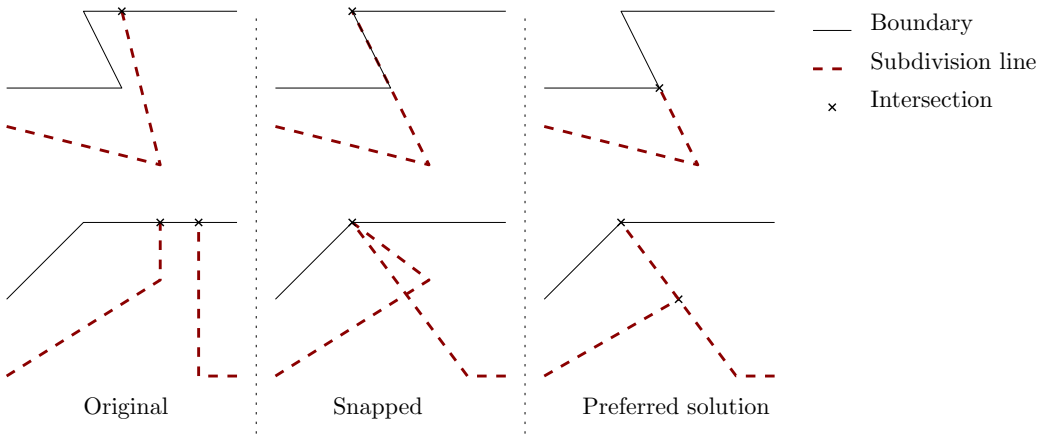


Figure 6.7: Snapping problems and preferred solutions.

The second problem occurs when two (non-intersecting) subdivision lines intersect after snapping to the same corner. This creates an additional surface which, depending on its size, can lead to self-intersecting faces, dangling faces or a correct but wrongly shaped solid. A solution could be to check for intersections while snapping. If such an intersection exists, cut the two lines at that intersection and replace the two line sections between the corner and intersection with a single line.

6.4.2 Shape straightening

This approach for model generation will work for any type of building. However, the majority of buildings fall into a specific category in which most intersections are perpendicular. Instead of perpendicular intersections, this approach generally results in smoother shapes and corners. Part of this is caused by the use of alpha shapes when creating clusters. The size of the structuring element used for alpha shapes prevents right angles on subdivision lines. An alternative to alpha shapes could lead to improved angles. Another cause is missing height data in the source grid or data that is clipped outside as its grid point falls just outside of the boundary.

Other approaches to this problem have included the building footprint skeleton, or to only allow lines along the two main axes of the building. However, these approaches make assumptions based on the building footprint which go against the general nature of the method in this thesis. Instead, a number of post-processing steps could be used:

- Any subdivision lines that are almost perpendicular with respect to their intersecting boundaries should be made perpendicular.
- Any point, or small group of points, on a subdivision line that resemble a right angle within a short distance should be made into a right angle.
- Subdivision lines that are almost parallel to one of the main axes should be made parallel to that axis.

These three steps, together with an alpha shape alternative, would result in squarified shapes that follow the building footprint, while other shapes are still allowed in the model. Note that changing subdivision lines can lead to unexpected issues such as intersections or roof surfaces outside of the boundary.

6.5 Block models

The concept of a block model in CityGML relates to the horizontal top surfaces of each building. However, the thought arose: what if a block model would be constructed from actual blocks? These blocks would have to be created in such a way that each point in the input data has at most some maximum height difference with its corresponding block, a restriction suggested in the final results of the 3D Pilot NL. Inspired by the construction method of quad trees, a method was created that subdivides the object surface into squares. This method consists of two parts, the first of which builds the tree downwards, whilst the second constructs a building on the way back up.

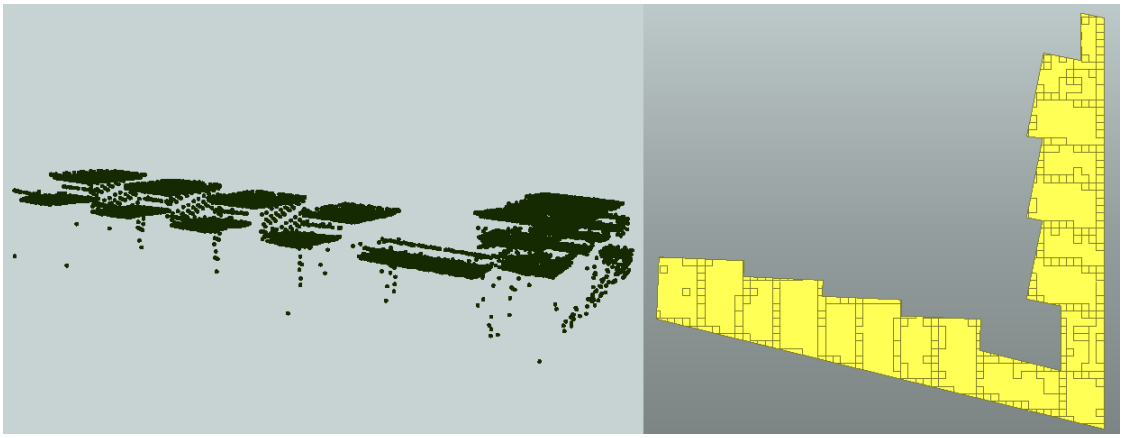


Figure 6.8: An input point set (left) and the corresponding merged squares (right).

In the first part all input points and the corresponding object boundary are rotated in such a way that the main axes of the object boundary correspond with the X- and Y-axis. The required rotation is derived from a minimum area oriented bounding box, created using a rotating callipers approach. This will probably line up the result squares with the main axes of the building. A starting square is created that encompasses the entire object. Recursively, this square is split in to four new squares if there are points inside it that have a height difference larger than the allowed when compared with the average point height inside the square. Square splitting stops if there are less than 2 points inside it or if a specified maximum tree depth has been reached. The height for each square is the average height of all points inside it or the height of its parent if it does not contain any points.

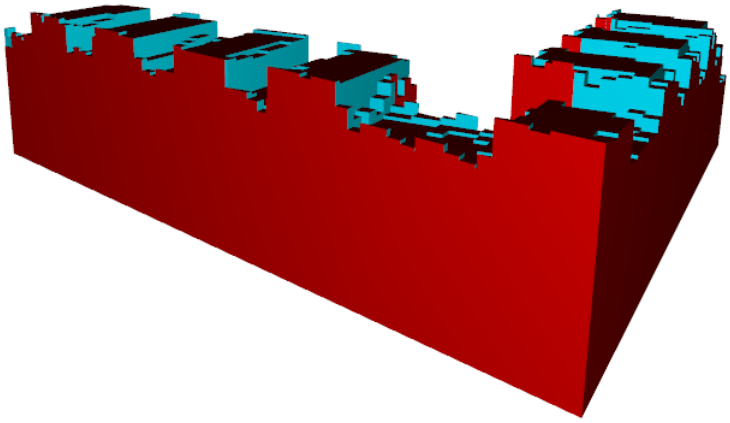


Figure 6.9: A visualisation of a LOD1 block model constructed out of blocks.

In the second part the squares from part one are used to construct a solid object. Each square is mapped to its nearest height level, which could be every 20 centimetres. This provides a real block look and allows the merging of squares that are almost equal in height. Such squares are dissolved into a single non-square surface, as can be seen in Figure 6.8. These squares are used as input roof surfaces for the construction method used in Section 6.2, which adds the wall and balcony surfaces to the floor and roof and produces a solid object. One result of this method is shown in Figure 6.9.

While this method does not find or reconstruct the building structure, it is a reasonably easy way to construct a decent building representation that could be used for volume calculations. The amount of detail can be changed by manipulating the maximum height difference and maximum tree depth.

Chapter 7

Level of detail 2

The highest level of detail discussed here is LOD2. One of the key differences for LOD2 is that structures are no longer limited to block models. The inclusion of proper models for smaller objects such as benches, street lights or trees is also possible on LOD2. As the focus of this thesis is on the construction of buildings only those will be discussed here. A short description of other objects can be found in Section 4.5.

7.1 Method

The shape of the roof starts to play an important role for structures on LOD2. The method of building reconstruction described here follows the work of Oude Elberink [2010], creating buildings with a data-based approach. This approach will be implemented and expanded where necessary. As the process from source data to final structure is complex, it will be explained in a number of different steps in the rest of this section.

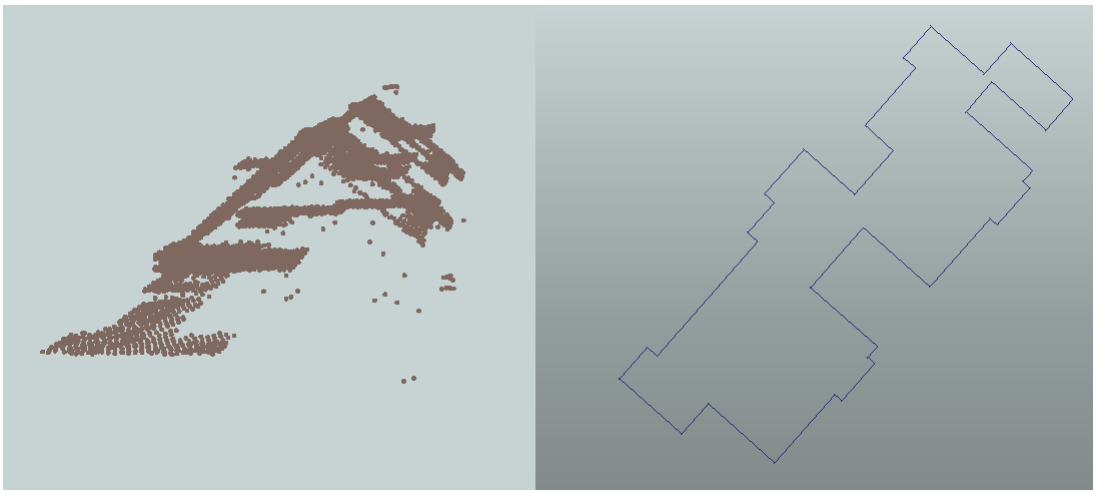


Figure 7.1: An example of starting data: clipped and filtered height points (left) and the object boundary (right).

For the construction of LOD2 structures the AHN2 point clouds are used. For each building only the points within the object boundary are kept. Unlike the approach by Oude Elberink [2010], there can be no structure parts outside of this boundary as the resulting geometry will no longer match with the one stored in IMGeo. An example of the two starting datasets can be seen in Figure 7.1.

7.1.1 Plane detection

The first step is the detection of non-vertical planes in the set of points. Two possibilities for this type of shape extraction are described by Vosselman et al. [2004]. He describes two types of Hough transform, one of which is a standard approach to find parametrized planes and the other is an improved version which requires the availability of normal vectors for each point. In this case a plane is defined by three parameters: the slope in the X- and Y-direction, and the height above the origin point. There are three downsides to a standard Hough transform for plane detection. The first is that to process a single point all possible planes through that point have to be considered. This leads to a large number of required calculations. The second downside is that all these results have to be stored in a 3D accumulator array. To get precise planes this accumulator array would have to be huge. The last downside is that as each point produces a surface through the accumulator array instead of a single point, the extraction of multiple different planes from the same Hough transform can be problematic. The second solution does not have any of these downsides. With a normal vector available for each point the plane through a point is uniquely defined, a point only results in one point in accumulator space. As the three parameters are independent the accumulator array can be also be split into multiple parts, which is more efficient. This solution is a lot better, but sadly, points do not have normal vectors.

The implementation used for this thesis is a version of the Hough transform that uses normal vectors. To overcome the unavailability of normal vectors these are estimated. A TIN structure is built using all the points. The normal vector is calculated for each triangle in the result set. The normal vector for a point is the average vector of all adjacent triangles. Compared to the commonly used method of deriving the normal from a plane fitted to nearby points, as described by Hoppe et al. [1992], this method will produce less accurate normals as it is more susceptible to noise and outliers. However, it does provide a simple and fast way to construct decent point normals for large datasets, allowing control over which normals are or are not allowed. For example, triangles with a steep slope are discarded so that vertices at the edge of a roof will have a normal that fits with its roof section, instead of with its roof section and its neighbouring wall section. Some points will not receive any normals this way and will be discarded. Those points will belong to wall sections, or building sections that are not planar, and will not be used in the remaining part of the construction method. The parameters that describe a plane are also different in this implementation: instead of slope, the angle of the plane in X- and Y-direction is used, and the distance above origin is replaced by the shortest distance between the (local) origin and the plane.

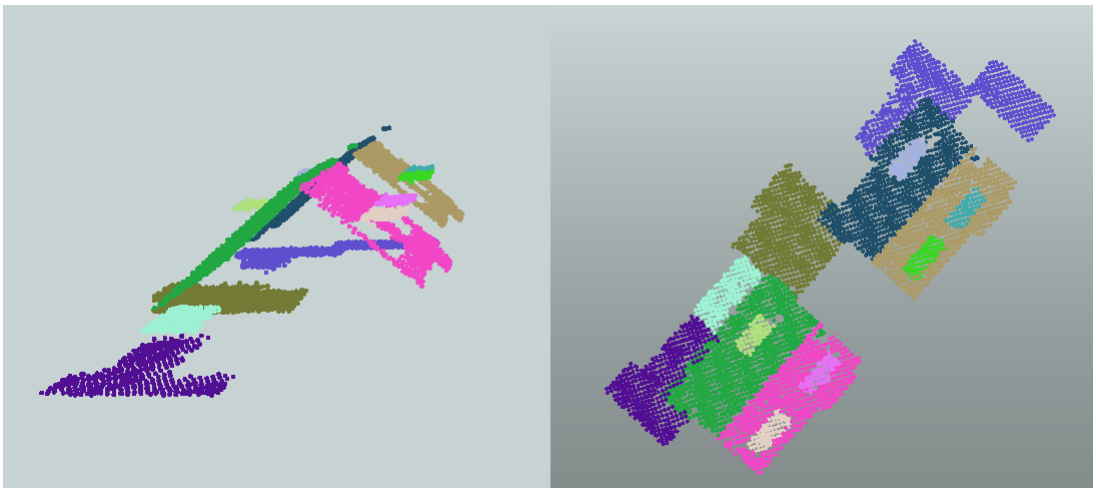


Figure 7.2: A 3D (left) and top view (right) of colour coded clusters on detected planes.

Now that every point has a normal vector, it is easy to create a Hough transform that converts each point into a plane in Hough space, as explained in Section 4.6. Hough space is limited to 80

degrees deviation in each direction with respect to a vertical normal, to prevent the detection of possible (near-)vertical walls. The bin size of the accumulator array is 1 degree for the angles and 10 centimetres for the distance.

From the accumulator array the best plane is chosen. All points belonging to this plane will be marked as used and removed from the rest of the points. Note that in assigning points to a plane a different measure of equality is used than when assigning points to accumulator space. The small step size of the accumulator array is required to get accurate planes. However, as our normal vectors were only an estimation, a lot of points will not be mapped exactly to their plane in accumulator space: the angle or distance could be slightly off. To overcome this issue, points are not assigned to planes during the plane detection, but are assigned to the best fitting plane once all planes have been found.

This implementation of the Hough transform has a lot of similarities with the efficient RANSAC algorithm by Schnabel et al. [2007], which can also be used for primitive shape detection in 2D and 3D. Similarities include the use of point normals, the error measurements, based on distance and angular deviation, and the ability to find planes supported by many points. The main difference is the random sampling used by the RANSAC algorithm, which means that the best supported plane is not guaranteed to be found. In turn, this results in a shorter running time and lower memory requirements when compared to the Hough transform. Both methods will probably perform equally well for the task described here, which could be the reason that both methods co-exist.

Once one plane is found, the unused points are entered in a new accumulator array to find the next plane. This process keeps repeating itself until the number of supporting points for a plane is less than a certain threshold. At this part in the process the majority of points are assigned to a plane, but multiple roof surfaces may be on the same plane. The next step is splitting each set of points on a specific plane into different clusters. Points belong to the same cluster if the distance in between them is less than half a meter. Resulting small clusters are dropped. At the end of this step each cluster of points should represent a single roof surface defined by its corresponding plane. A clustered and colour coded example is shown in Figure 7.2.

The set of all clusters may contain some gaps when compared to the structure boundary. While small gaps are fine, large gaps can cause problems later on in the process. Such a gap could be caused by a non-planar building section. Each large gap is filled with points on a new horizontal plane, the height of which is the average of all original points within the gap. While not entirely accurate, it guarantees that the entire structure boundary is covered in clusters.

7.1.2 Line creation

The next step in the process is to detect the edges in between roof sections. While there are a large number of different roof edges, such as dormers, eaves and gutters, only two types are differentiated here: step-edges and intersections. The difference between these edges is not the function they have in reality but the way to they come to exist in the model. Intersections are edges that are well defined as aprt of the infinite intersection line between two planes. Step-edges are all the other edges, those that can not use the plane information and must be constructed from the point clusters. Step-edges represent vertical steps on the roof surface, such as on the edge between a house and the adjacent garage. Note that edges can only exist between neighbours, where neighbours are defined as pairs point clusters, of which the two aggregated buffers of the point clusters overlap. These neighbour relations can be used later on to define the types of roofs.

Intersections

The method used for finding intersection edges is the same as used by Oude Elberink [2010], although he calls them interior roof edges. Firstly, the infinite intersection line between two neighbouring roof planes is calculated, if it exists. Secondly, this intersection line is only an edge if both point clusters have points near the intersection line at an overlapping segment alongside of

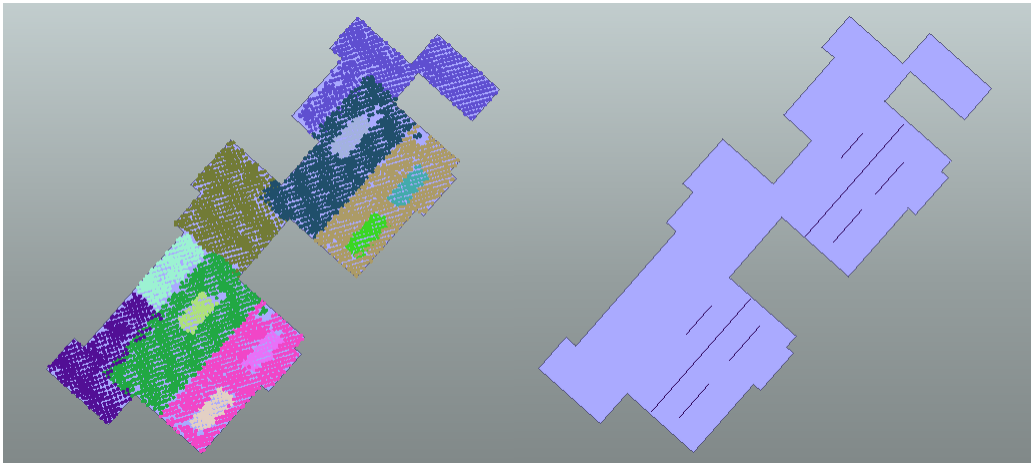


Figure 7.3: A top view of clustered points (left) and detected intersection lines (right) between detected planes.

it. The overlapping line segment is the intersection edge. An example result can be seen in Figure 7.3.

Step-edges

The detection method for step-edges used by Oude Elberink [2010] is quite interesting. For each pair of neighbouring point clusters a TIN structure is created. Step-edges occur around lines that run between points of different clusters, with the horizontal length of the line limited and the difference in height higher than some threshold. Such a line describes a height difference between two adjacent roof surfaces. If there are multiple such lines close together, there is probably a step-edge at that location. The creation of a step-edge from the set of lines is done as described by Rottensteiner and Briese [2003], using the gradient of the lines. However, they already indicate that their method could use some work, especially in the case where the result would not be a single step-edge.

The approach described here is quite similar but differs on some key points. In the first phase the same TIN-based approach is used, however the threshold on height difference is dropped and replaced by the constraint that none of the TIN-lines used for step-edges may be near one of the intersection edges found in the previous step. There are two reasons for this change. Firstly, point clusters created by gap filling in the plane detection step may lead to reasonably similar adjacent point clusters. As their difference in height would be small, no step-edge in between them would be found. Secondly, the goal is not to find all edges that have a height difference, but all non-intersection edges that split two point clusters.

For the creation of the step-edge itself, each line found in the previous step is replaced by its centre point. This set of points represents the rough shape of the step-edge. If the step-edge was a guaranteed straight line through the point set there would be numerous methods to construct that line. However, the set of points may represent multiple lines. To solve this problem the set of points is split into groups that likely represent a single line segment. An aggregated buffer is created around all the points. In a repeating process the longest line between two points that does not intersect that buffer is found. The process repeats until no line long enough can be found. Once a line is found all points near it are discarded for the rest of the process. The lines found are general approximations of the step-edges, but the longest-line method is not very accurate. A second step selects all points in a buffer around each line and uses these points in combination with a standard infinite line fitting algorithm. Here a least-squares fit is used. The length of the final line is determined by the buffer it is created from. While this method is complex and not really suitable for large sets of points, for the majority of buildings this will not present a problem. Either

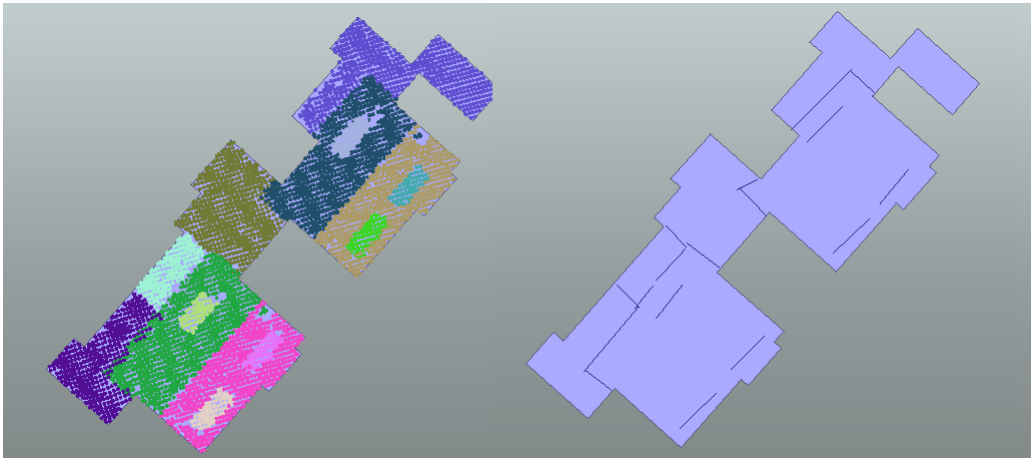


Figure 7.4: A top view of the clustered points (left) and detected step-edge lines (right) between point clusters.

the step-edges of the buildings are small or a large step-edge is split up into multiple sections. Worst-case scenario is a single step-edge running around an entire large building. If this is the case, the step-edge point set can be split up automatically. A result of this process is shown in Figure 7.4.

Overall, this method produces good step-edges. However a good result can not be guaranteed. There may be some points missing in the input data, resulting in a split step-edge. Short step-edges can either be left out because of their length or the ratio of point set width to step-edge length can lead to step-edges with a direction that is off.

7.1.3 Surface creation

The transition from lines that are not topologically connected to correct polygons is always a complex task. An initial attempt to extend, intersect or move nearby lines to form polygons was reasonably successful, but no guarantee could be given on its results. A single missing connection would mean the polygon is not closed, making all the correct connections worthless. The problem is that the line detection method in the previous step is not guaranteed to find all lines and even if it did, a roof section may not always be bounded by straight lines. Round or jagged boundaries may lead to gaps in the set of lines. With this in mind the aim became to create a method that could give some guarantees on its results, even if the set of input lines is not complete. Preferably it should be guaranteed that each input point cluster becomes a closed off roof polygon in the result.

To give such a guarantee each point cluster is considered individually. The output lines must form a complete ring around such a point cluster. The input lines for a single cluster are the step-edges, intersection lines and boundary lines that overlap with a one meter buffer around the point cluster. Given these lines the task is to not only connect these lines in such a way that they form a complete ring, but to do it in such a way that the resulting ring corresponds to the shape of the point cluster.

The problem at this stage is that it is not known which lines should form connections. To overcome this problem some sort of ordering is needed. The start and end point of each line is mapped to the closest point on a line representing the concave hull of the point cluster. A radius of 35 centimetres is used to create this type of concave hull. For use on other datasets this parameter should be turned into a function on average point cloud point density. This concave hull is the jagged shape we are trying to model with straight lines, as can be seen in Figure 7.5. For each line endpoint a measure is stored that contains the distance of its corresponding closest point along the hull. The line endpoints are reconnected in order of this increasing measure. The improvement

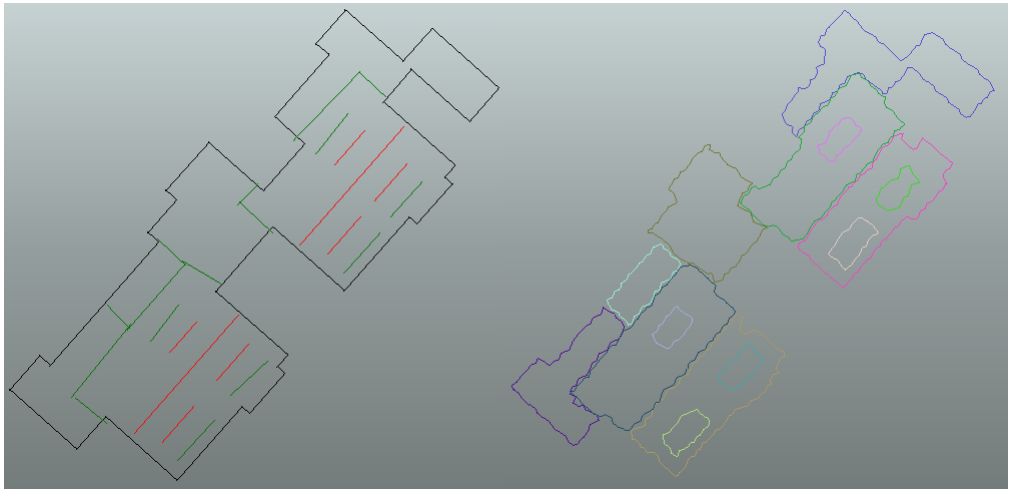


Figure 7.5: Plane creation input: three types of unconnected lines (left) and point cluster hulls (right).

of this approach is twofold: every line is oriented the same way around the hull and the lines can now be ordered by their measure values.

The next step is to consider possible or likely connections between lines. These possibilities are stored in a directed graph, in which each node represents a line and each edge represents a possible connection. An edge is added between two lines if their start- and endpoint measures are less than one meter apart. If a section of the hull is not covered by any line, that section is not counted in the one meter rule, to allow connections over missing lines. To guarantee that each line can be used, lines with no incoming or outgoing edges receive an edge to the line with the closest nearby measure with respect to the unconnected start or end measures.

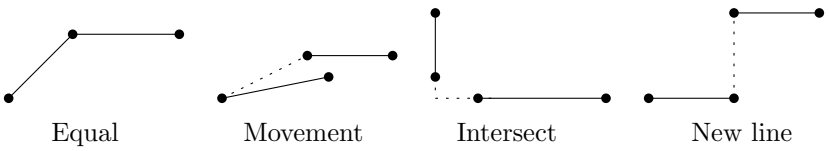


Figure 7.6: The four possible ways to connect two subsequent lines.

To find the quality of each of these possible connections some kind of penalty value is required. At this point we should consider that there are three types of lines: boundaries, intersections and step-edges. During construction each line can also be marked as fixed, meaning its endpoints are connected and may no longer be moved. Boundary lines are fixed by default. These types of lines can be connected in different ways. Four possible connections are considered, which are also shown in Figure 7.6:

Equal If the end point of one line corresponds to the starting point of the next line, they are already connected and no action has to be taken. This connection comes without a penalty.

Movement If the end point of one line and the starting point of the next line are close, one of them could be moved to the location of the other. The penalty depends on the line that is moved. Movement distance along the line and perpendicular distance from the line are considered as penalty, with the latter weighing more heavily than the former. This is only an option if one of the lines is a non-fixed step-edge, as they are the only type that may be moved.

Intersect Two non-parallel lines intersect at some point. By extending (or reducing) each line to the intersection point they become connected. The penalty depends on the distance that

both lines have to grow or shrink to reach the intersection point. All line types may intersect, but lines marked as fixed may not be extended. An intersection on a fixed line can not extend it, it can only be split into two. Distance along a fixed line does not count as penalty.

New line Any pair of lines can be connected by adding a new line in between them. If one of the lines is fixed, a new perpendicular connection from the fixed line can also be considered. The penalty depends on the length of the new line. A new line should only be considered for large gaps, when there is no other option available.

The evaluation of these possible connection types results in one lowest penalty and best connection type for each edge. The best overall connection is then calculated using Dijkstra's shortest path algorithm, explained in Section 4.6. It is assumed that the input line covering most of the hull will be in the result and it will serve as both the source and the target node for the search algorithm. The resulting path through the graph will contain the best way to connect the set of lines to form a complete ring. Now all that is left to do is to actually process the connections as stored in the edges. All lines that form the ring will be marked as fixed in the end.

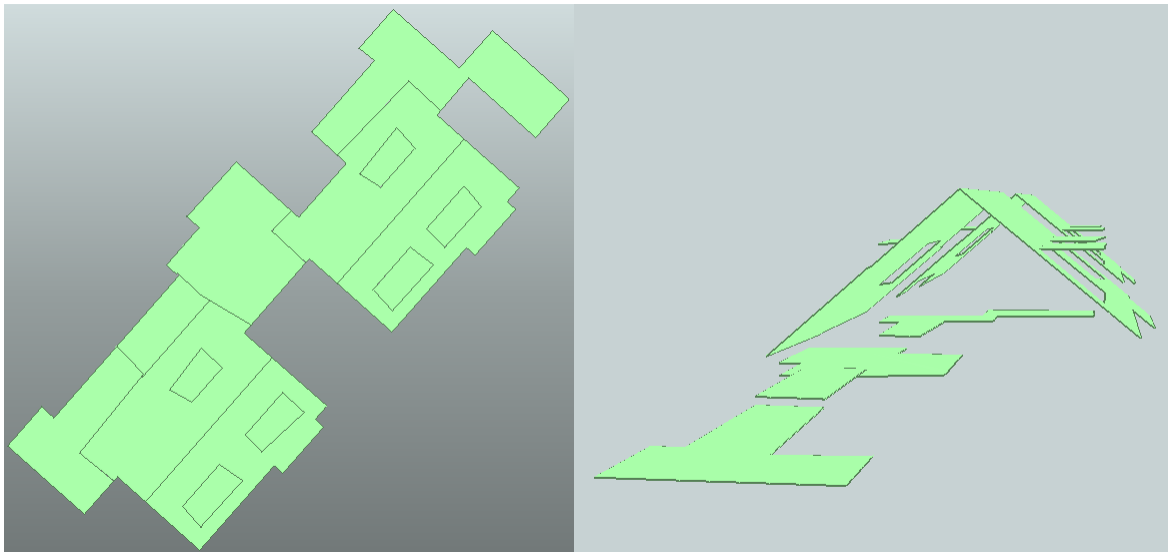


Figure 7.7: The results of plane creation in 2D (left) and 3D (right).

A little bit more work is required to make the method work on all point clusters. As a line can be used by many point clusters, multiple copies of the same line will exist, each of which with a different combination of direction and measures. These copies will be linked by their ID and any change affecting one of them will affect all of them. This includes movements, splits or being marked as fixed. With this mechanism in place it is safe to construct the cluster boundaries one at a time, resulting in a connected set of a lines for each cluster. These lines are combined to form a set of surfaces, each one representing a roof part.

7.1.4 Height assignment

Now that all surfaces are available in 2D, it is time to bring them back into the third dimension. Each surface is split into its separate vertices. Using the information about the previously detected planes, and the local building origin, each vertex is mapped to the correct height. These points are connected back in their original order to form a surface. As points on a single surface sample from the same plane, they form a planar surface in 3D. The set of these surfaces forms a complete 3D roof of a building. An example of the results of plane creation and height assignment can be found in Figure 7.7.

7.1.5 Solid construction

Even though the conversion from roof surface to solid object was not actually implemented for LOD2, the method used for LOD1, described in Section 6.2, can be re-used for the most part. The differences in the input structure are the horizontal roof surfaces and the possibility for neighbouring points to be on the same height. Three changes to the LOD1 method would be necessary to allow this method to construct valid LOD2 solid objects:

- For balconies, create no surface if each pair of points is on the same height.
- For balconies, create two surfaces instead of one if the two lines, which are identical in 2D, cross vertically.
- For walls, accommodate points on the boundary that share the same height.

7.2 Results

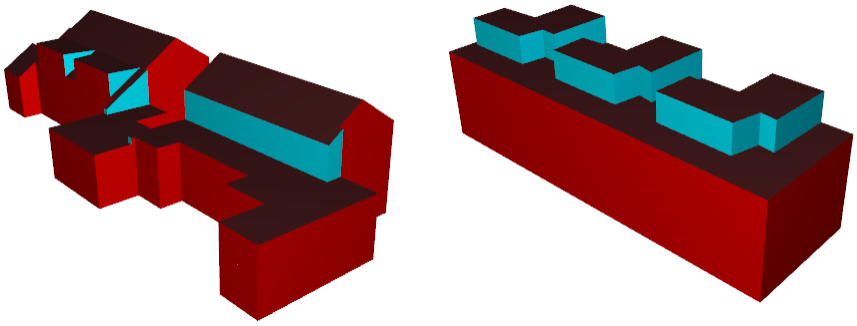


Figure 7.8: A visualisation of two buildings modelled correctly on LOD2.

After testing the method mentioned above on a set of 44 buildings it became clear that there is still some work to be done. Of those buildings only a ten (23%) resulted in the kind of roof structures the method attempts to generate. This is not entirely unexpected, as the approach consists of a large number of steps and the different types of buildings make for a variety of different cases. Of the building that were not processed correctly, twenty (45%) failed during plane detection while another fourteen (32%) failed at the surface creation stage, with both small and large errors occurring in both parts. Any error, whether large or small, counts as a failure of the entire building. As the percentage of correct results is low, no further efforts were made to evaluate. However, the few buildings that did succeed presented good models for their objects, as can be seen in a (non-valid) result visualization in Figure 7.8. A numerical evaluation of height differences between roof surfaces and input points would surely have given good results. The following list describes the most important observed problems in the approach, a few of which are shown in Figure 7.9:

- The estimated normals combined with the small bin sizes for the Hough transform result in points of a surface being spread out over multiple bins, each bin containing a small number of points. This can lead to the selection of a decent but sub-optimal plane. As a result of this, a single surface may be split over multiple planes or no plane may be found.
- Currently the plane detection and clustering are two subsequent steps. This can lead to points being assigned to planes incorrectly, as the consideration of possible clusters would make it clear that the point belongs on a different plane.
- Narrow sections of point clusters can lead to problems during line detection and plane creation, as lines may overlap and concave hulls may not be correct. The clustering combines all points that are on the same plane within a certain distance. This allows narrow sections in both the case where they are correct as well as in the case where it leads to problems.

- The execution of the movement connection during plane creation can move the intersection option on the other side into an impossible configuration.
- Gaps in the original data can lead to invalid concave hulls, providing an incorrect base for plane creation.
- Newly created lines during plane creation can not be properly re-used as there are no measures available for its copies.

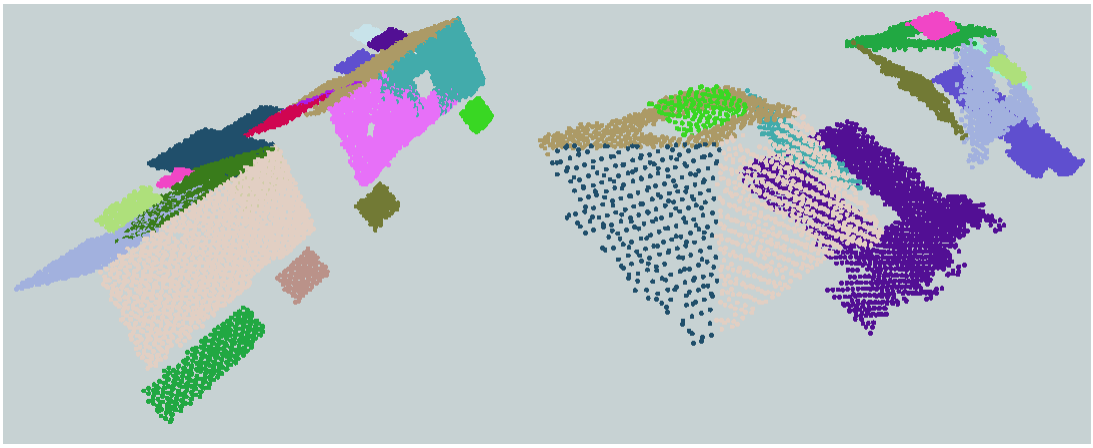


Figure 7.9: A single surface assigned to two planes (left) and an undetected surface which was filled later on (in brown) even though a similar surface (in blue) did get detected (right).

Even with this many problems the overall approach can be considered a relative success. The problems mentioned above are problems of the specific steps, not of the overall approach. The majority of problems are not hard to overcome and can be fixed with some more time and effort. Some of these fixes are given in Section 7.3. Once all steps are working as intended the approach will produce correct roof surfaces, which can be used to construct LOD2 solid models.

7.3 Further improvements

As can be seen in the previous section, there are still some improvements left to make. Not only in the way of improving the current functionality, but also in the way of extending this functionality. A number of possible improvements are explained in the following sections.

7.3.1 Plane detection

The plane detection step can cause issues by inaccurate point normals or because of the accumulator bins being too small. A simple solution would be to increase the bin size to allow more variety in points. The downside of this is that the detected planes will be less accurate, as there are less possible planes. A better option would be to improve the estimated normal accuracy. This could be done by calculating a plane through a number of neighbouring points and using the normal of that plane as the point normal, instead of taking the average normal of surrounding TIN triangles. A number of methods for normal estimation that could also be used are given by Dey et al. [2005].

7.3.2 Immediate clustering

The split in steps between plane detection and clustering allows for incorrectly assigned points. To prevent this problem the plane detection and clustering steps could be merged into a single step. Instead of detecting a plane, assigning points to it and clustering each plane individually, clusters could be formed during plane detection. For the best plane found the single largest point

cluster is created. All unused points are available for the next round. This means that a single plane may be found multiple times, but points are no longer assigned to planes where they can not be clustered.

7.3.3 Movement

The movement option during plane creation is the only option that affects the other side of the lines it connects. This can make other connections impossible. One simple fix would be to remove the movement option altogether, but it is quite useful for connecting subsequent almost-parallel lines. Another option would be to split a moved line in two so that the movement of one side does not propagate to the other side. Last but not least, a function could be added that constructs the best alternative once a connection has become impossible.

7.3.4 New line measures

It is hard to incorporate newly created lines in the plane creation step as they do not have the required attributes. These new lines should be treated as all others, mapped to the point cluster hulls it overlaps and distances measured for each of the endpoints. In the current implementation, mapping of lines is done in FME while the plane creation itself is written in Python. There is not really a way to output a new line, map it, and re-use it again in Python. A complete solution would have the line mapping done in Python so that newly created lines can be mapped as well.

7.3.5 Gap filling

Gaps in the coverage of the point clusters can exist because of missing input data or non-planar surfaces on the building. Currently these gaps are filled with a horizontal surface. In the recommended IMGeo 3D requirements, proposed as a result of the 3D Pilot NL, it is suggested that these gaps should be filled with a TIN surface representing the non-planar shape inside the gap. This could be an interesting way to fill gaps that resembles reality a lot closer than the current horizontal surface. However, care should be taken that the TIN model is not over-fitted and it does not introduce detail not fitting for LOD2. The biggest problem will be the connection between the TIN surface and the surrounding planar surfaces. It is not yet clear how this would work.

7.3.6 More connection options

At this point four different line connection options are included. However, these can be expanded quite easily. As long as a way to connect the two lines is available, together with an evaluation function, any line connection can be added. This way it is possible to include for example a method that focuses on perpendicular connections between lines, providing a low penalty to connections where such a right angle is possible.

Chapter 8

Conclusion

Throughout this thesis a number of different methods have been described that automatically construct 3D IMGeo models on different levels of detail. Some expand on existing approaches, other methods are entirely new, but all provide a fresh look on ways to construct models in a data-driven and reliable approach.

The goal on LOD0 was to build a land-covering 2.5D terrain model, built up of a large number of IMGeo objects. A standard triangulation of input points, cut in parts by object boundaries, provides a reasonable result that is easy to construct. Attempts were made to see how feasible it would be to improve clearly visible errors in this reasonable result by adjusting the triangulation using break lines. Steep walls were detected and included, water was made horizontal instead of jagged and a method was added to include simple bridges in the model. The result was a decent model that describes the general terrain of an area.

On LOD1 the focus shifted towards buildings, trying to model them as block models. A simple block model can be created by extruding the building footprint. For larger buildings or buildings with different heights the building can be split into multiple parts, each with a single horizontal roof surface. One approach based on Voronoi lines between clustered points builds these subdivided block models. While the roof surfaces itself may not be entirely accurate, the method produces reasonable subdivisions based on height differences and valid solid geometries in the large majority of cases. A second method based on the construction of a quad tree builds a block model out of actual blocks, representing the roof as a large number of squares. This method aims to have all points within each square within a specific vertical distance, showing that if no actual structure is required in the building a representative model can be made quite easily.

The highest level of detail discussed was LOD2, where the focus became the accurate reconstruction of roof shapes based on point clouds. A number of different steps change these point clouds first to points clustered on a single 3D plane, then to lines subdividing those clusters and finally to 3D surfaces representing the planar roof structures. While the individual steps need more work, the overall approach seems promising and allows for easy expansion of plane creation methods. When fully completed it should guarantee that each detected point cluster will become a correct roof surface in the result.

With all these different methods and results in mind, together with the knowledge about data sources and model uses, the next section will answer the questions raised in Section 2.1. A list of possible future work can be found in Section 8.2, including both improvements on methods in this thesis and work to be done on related subjects. Finally Section 8.3 gives a number of recommendations about 3D model construction in the Netherlands and how these models could be used.

8.1 Questions & answers

Which uses do 3D IMGeo models have? In Chapter 3 a number of possible uses of 3D models for local councils have been given, divided over the use types visualisation, analysis and registration. The models constructed for this thesis can mainly be used for analysis, providing valid solid geometries and land covering terrain models to be used in analysis revolving around air and noise pollution, and flood risks. The LOD0 terrain model combined with LOD1 block models would be enough to calculate water flow and affected areas during floods. All the LOD1 and LOD2 models would be useful to calculate noise and air pollution with as they represent the general structure of buildings quite well, especially the detailed models of LOD2.

The lack of textures makes the models not ideal for visualisation purposes, but they could be of good use for the workers at the council itself who are more interested in 3D town layout than pretty images. Lastly, while the models are not usable for the types of registration mentioned in Chapter 3, they could be used for example to automatically calculate building volumes for the councils building tax registration.

Which data sources are available and can be used for the construction of 3D models? The three data sources used for this thesis, IMGeo, BGT and AHN2, are described in Section 4.3. These data sources, representing 2D and height data, will be available nationwide allowing the methods in this thesis to be used across the country. Of course, other data sources could be used, both for 2D geometries as for height data, as long as they have similar properties. For 2D geometries, information from the BAG is a possible source, although this may lead to discrepancies when building LOD0 terrain models. For height data, point clouds can be constructed from stereo aerial images, which has the added advantage of regular updates.

In which ways can different types of models be constructed? The existing work mentioned in Section 4.2 shows only a small sample of papers dedicated to this subject. Methods range from model-based to data-based, from combining multiple data sources to only using height data, with constructed object types containing trees, bridges, roads and buildings. This thesis contains methods for the construction of LOD0 terrain models and buildings on LOD1 and LOD2, described throughout the Chapters 5, 6 and 7. Even for this restricted set a large number of different methods exist, each with different properties and results. The hardest part may actually be to pick the best construction method given input data and output requirements.

Which limitations come with automatic model construction? While implementing different methods the limitations became painfully obvious. One limitation is that the models are only as good as the input data. Gaps in input data will not be handled as easily by a computer as by a human. It also means that the construction of tunnels and bridges, or objects underneath bridges, is severely restricted by lack of available data. Another limitation comes from the fact that there are a lot of unique buildings, providing lots of different scenarios that should all be handled by the construction method. If all buildings were made from a limited number of types, automatic construction would be much easier. Instead, structures like rounded roofs or jagged walls make guaranteed accurate models infeasible. The third limitation comes from the sheer volume of input data. Construction on point clouds, whether it be roof construction for a single building or terrain creation for a whole area, are generally slow and require lots of memory. Another limitation is occlusion, missing height data caused by other structures being in the way of the aerial scan. As the majority of scans will be slightly sideways, roof sections could be missed as they are blocked out by higher building sections. The last limitation is a lack of evaluation options, a method that says whether a model is good or bad does not really exist. This means it is quite hard to say how effective an automatic construction method really is.

How feasible is the automatic construction of 3D IMGeo models? With the sub-questions answered it is now time to answer the overall research question. As there is quite a

lot of data available to use for automatic model construction, and there are plenty of possible uses for these models once they exist, the feasibility will really depend on the method of model construction. As models have been made during this thesis the automatic construction of 3D IMGeo is considered feasible, but only under certain restrictions and with limitations on the results.

The two largest limitations can be roughly categorized in two groups: evaluation and structure recognition. The lack of proper evaluation functions make it hard to repair or improve construction methods, as it is not necessarily clear whether a model is a success or not. The three types used here, height difference errors, technical validity and visual checking, are not enough to validate a large 3D dataset. The second limitation comes from the fact that a lot of tasks during model construction are recognising some structure in unstructured data. Whether it is recognising planes in point sets, lines through points or polygons from lines, is that there is no single best way to do it. Such a type of recognition always depends on assumptions and restrictions. This is the fundamental issue with automatic model construction and also the reason why so many construction methods exist.

The methods for building and terrain creation given in this thesis only deliver a small part of a full 3D IMGeo model. To build a complete IMGeo 3D model, construction methods for buildings, terrain, trees and other objects will all have to be combined into a single approach, with each of these methods being entirely different. With this in mind the overall conclusion is the following: it is feasible to automatically construct usable 3D IMGeo models, at least for LOD0 terrain models and buildings on LOD1 and LOD2, from 2D and 3D data sources available nationwide, but these models are not necessarily accurate representations of reality.

8.2 Future work

There is still a lot of work to be done in the area of automatic model construction. Improvements for the LOD0, LOD1 and LOD2 construction methods described in this thesis have already been given in Sections 5.4, 6.4 and 7.3 respectively. These improvements range from fixes in the current implementation, changing existing functionality to provide better results or actually expanding on the method functionality.

Some more work is also possible outside the scope of this thesis. Prime candidate for this would be a general function to quantify the quality of a 3D model. This could be based on differences with the input data, but also on the shapes contained in the model. Combined with a solid geometry validation method it would be possible to measure the success of a specific construction method.

Finally, it would be interesting to see a construction method for city furniture object types. For the majority of object types at least some information was already available, but for city furniture this was not the case. To get a complete 3D IMGeo model a method like this will also be necessary. These objects are probably just as hard to model in 3D as the other ones. Problems will also be quite different, for example detecting the specific type of city furniture and its orientation. It would be interesting to find out which problems and solutions would arise.

8.3 Recommendations

Any 3D model is only as good as its input data. And to build a complete 3D IMGeo model a complete input set of IMGeo data is required. Which is the reason that anyone aspiring to create such models in the future is recommended to create and maintain a full IMGeo registration, providing the most possibilities once such a transition to 3D is being made.

Another recommendation is about a 3D IMGeo standard. Right now the 3D Pilot NL group has made a document containing example requirements and specifications which can be used by IMGeo proprietors during procurements. The group has also released an XML schema modelling IMGeo as CityGML. The XML approach has a few downsides, like mixing Dutch and English, very inefficient data storage and incorrect multiplicity relations for newly added attributes. However,

it does provide a good method for storing, transferring and visualizing small datasets. It would be recommended to add both of these items, the specifications document and the XML schema, to the IMGeo 2.0 standard. This would provide a complete 3D standard for the Netherlands describing all objects, with their attributes, requirements and allowed geometries, and ways of exchanging and validating such 3D objects.

The last recommendation is perhaps more of a vision about how to use 3D models such as those presented in this thesis. As the construction process can take quite long and changes to data are infrequent, it would be best to run the construction process on scheduled intervals, or whenever new data is available. Resulting models would be stored in a database and be re-used until the next update. As models are automatically generated no form of model maintenance is required, which would be a challenge in itself. To use the models within software, a user could select a specific area or viewpoint on the map, only loading the required 3D models from the database. Settings could include level of detail selection and draw distance. The resulting view would allow users to do 3D analysis or create 3D visuals for reports. This approach gives the user access to 3D whenever it is needed, but does not force a 3D environment when it is not needed. This would combine the speed and simplicity of 2D maps with the detail and perspective of 3D models, truly offering a combination of 2D and 3D GIS.

Appendix A

Den Bosch Haverleij

The Haverleij area in municipality of Den Bosch, in The Netherlands, is used as a test site both for the 3D Pilot NL as well as for this thesis. The area shows a variety in terrain, containing canals, a river, wooded and agricultural terrain, houses, apartment buildings and a golf course. An overview of this area can be seen in the figures below. Note that it looks like buildings are missing in the bottom right urban area. This is not a result of any of the construction methods, instead these buildings were mislabelled in the original input data. Also note that some buildings in the 3D model appear flat. These are the very few buildings for which LOD1 construction failed.

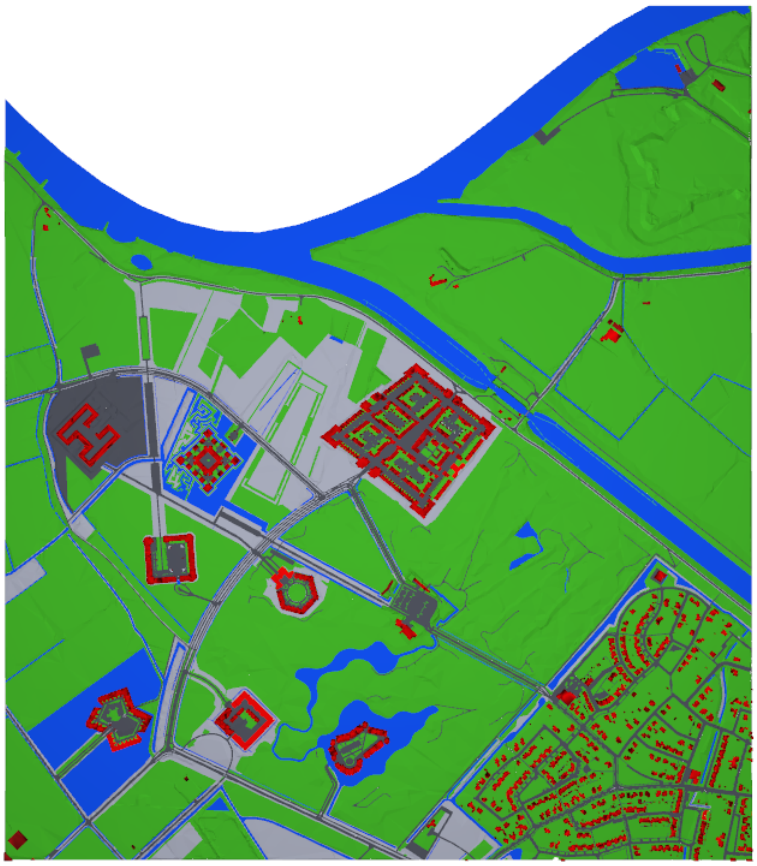


Figure A.1: A 2D overview of the Den Bosch Haverleij area.

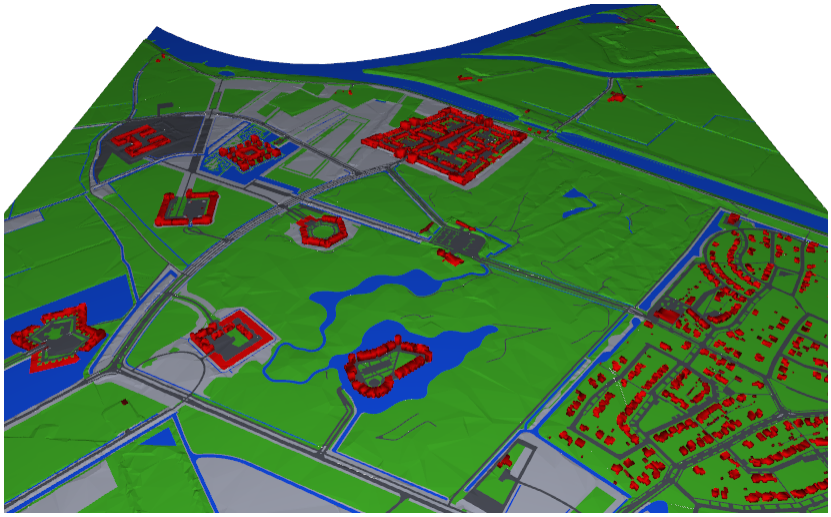


Figure A.2: A 3D overview of the Den Bosch Haverleij area.

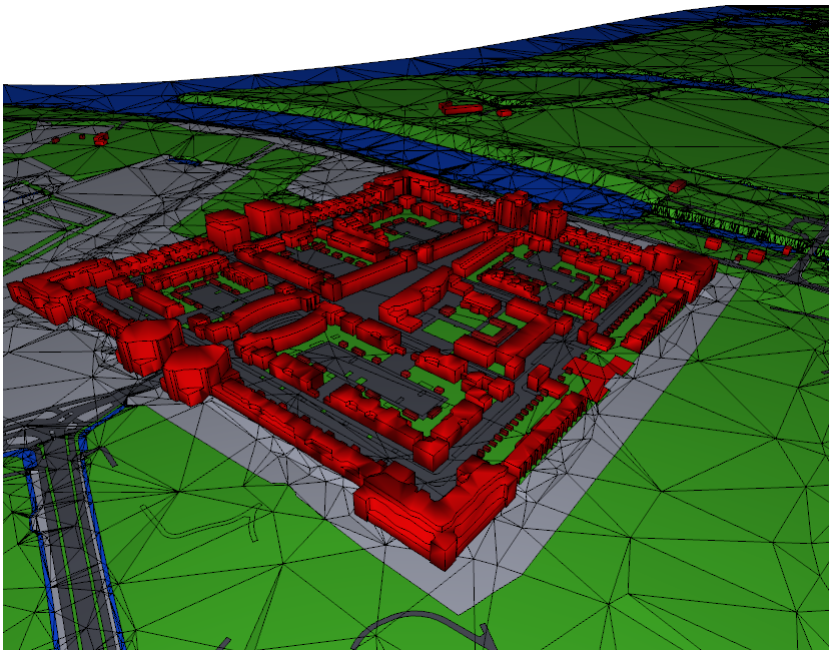


Figure A.3: A zoomed section of the Den Bosch Haverleij area, shown as a coloured wire frame.

Appendix B

Usage of FME

Throughout this thesis project the FME program by Safe Software has been used extensively. It would be safe to say that without FME this thesis would not have contained so many methods as it did. This appendix gives a small overview of my own experiences using FME.

Versions

When I originally started this project the FME version I used was 2011. A lot of times when I ran into some form of limitation, upgrading to a new version proved to be a solution. Whether it was because of the inclusion of a new version of CityGML, bug fixes in transformers or entirely new features, I upgraded my way through some version 2012 service packs and quite a few 2013 beta versions. The usage of betas was not without downsides: upgrades sometimes resulted in workbenches no longer working and instability could occur. This was largely made up by the large amount of available features and the frequency of updates either usable by me or specifically made for me. Looking back at all the versions i can safely recommend FME 2013 for anyone who wants to work with 3D or CityGML.

Readers, writers and transformers

The standard functionality of FME takes away two tasks I would have to write myself otherwise. The first is reading and writing all the different (spatial) data used in the project, which came in a large variety of formats and standards. Pretty much all of those could be read and written in a standard way, making sure I didn't have to worry about it. The only negative point was the CityGML and specifically the IMGeo XML support. Of course, as both of these formats were only released this year it would be unfair to expect a perfect implementation already. CityGML and IMGeo support was added halfway through my project. The only thing really left to solve was the writing of the IMGeo CityGML extension. The Dutch IMGeo attributes were defined as lists as they could occur multiple times according to the XML schema. This made it hard to impossible to write these attributes to file in the beta version used at the end of my thesis, although it seems this has been fixed at the time of writing. The last hurdle to take is to allow the writing of CityGML/IMGeo XML feature types without having an example feature available.

The second task is the implementation of specific construction methods. Using a number of available transformers a large of spatial operations could be executed on the data. In general a combination of transformers handle the pre- and post-processing for a single Python algorithm. Downsides that sometimes occurred are the transformer-by-transformer based nature of FME and the inability to repeat certain steps as no looping functionality is included. Overall the available transformers proved very useful, though as with any software library you will have to assume they work correctly and some may not do exactly what you think.

Python

The PythonCaller transformer allows Python scripts to modify features from within FME, using a module called FMEObjects. All complex algorithms in this thesis were written in Python. This method allows the available functionality of FME to be combined with the general flexibility of Python. While this combination proved invaluable, I would like to see the following functionality added:

- The possibility to access raster data as a 2D array from within Python.
- Customizable input and output ports on the transformer.
- A standard "group by" attribute on the transformer.
- Access to standard transformers from within Python.
- Better support for list attributes.
- The possibility to do calculations on feature coordinates. Currently coordinates have to be converted to for example NumPy arrays to allow vector multiplications and additions.
- Perhaps a better editor, though I think an outside editor will still have preference.

Bibliography

A. Akel, K. Kremeike, S. Filin, M. Sester, and Y. Doytsher. Dense DTM generalization aided by roads extracted from lidar data. *ISPRS WG III/3, III*, 4:54–59, 2005.

A. Alharthy and J. Bethel. Heuristic filtering and 3D feature extraction from lidar data. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(3/A): 29–34, 2002.

J. Benner, A. Geiger, and K. Leinemann. Flexible generation of semantic 3D building models. In *1st Intern. ISPRS/EuroSDR/DGPF-Workshop on Next Generation 3D City Models. Bonn, Germany, EuroSDR Publication*, number 49, 2005.

J. Binney and G.S. Sukhatme. 3D tree reconstruction from laser range data. In *IEEE International Conference on Robotics and Automation*, pages 1321–1326, may 2009.

C. Brenner. Towards fully automatic generation of city models. *International Archives of Photogrammetry and Remote Sensing*, 33(B3/1; PART 3):84–92, 2000.

F. Bretar, M. Chesnier, M. Roux, and M. Pierrot-Deseilligny. Terrain modeling and airborne laser data classification using multiple pass filtering. In *Proceedings of the XXth ISPRS Symposium*, pages 314–319, 2004.

M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational geometry: algorithms and applications*. Springer, 2008.

T.K. Dey, G. Li, and J. Sun. Normal estimation for point clouds: A comparison study for a voronoi based method. In *Point-Based Graphics, 2005. Eurographics/IEEE VGTC Symposium Proceedings*, pages 39–46, 2005.

E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1): 269–271, 1959.

D.H. Douglas and T.K. Peucker. Algorithms for the reduction of the number of points required to represent a line or its a caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

R.O. Duda and P.A. Hart. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, jan 1972.

M. Durupt and F. Taillandier. Automatic building reconstruction from a digital elevation model and cadastral data: an operational approach. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(3):142–147, 2006.

E-Overheid. Belangrijke invoerings- en releasedata. `http://e-overheid.nl/onderwerpen/stelselinformatiepunt/1493-basisregistratie-grootschalige-topografie-bgt`, 2012.

H. Edelsbrunner, D. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *Information Theory, IEEE Transactions on*, 29(4):551–559, 1983.

A.F. Elaksher and J.S. Bethel. Reconstructing 3D buildings from lidar data. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(3/A):102–107, 2002.

R.J. Fowler and J.J. Little. Automatic extraction of irregular network digital terrain models. *ACM SIGGRAPH Computer Graphics*, 13(2):199–207, aug 1979.

B. Fredericque and A. Lapierre. The benefits of a 3D city gis for sustaining city infrastructure. Technical report, Bentley, 2009.

Geonovum. *Basisregistratie Grootschalige Topografie: Gegevenscatalogus BGT 1.0*. Ministerie van Infrastructuur en Milieu, feb 2012a.

Geonovum. *"Basisregistratie Grootschalige Topografie: Gegevenscatalogus IMGeo 2.0*. Ministerie van Infrastructuur en Milieu, feb 2012b.

Het Waterschapshuis. Actueel hoogtebestand nederland 2. `http://www.ahn.nl/`, 2012.

H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *SIGGRAPH Computer Graphics*, 26(2):71–78, jul 1992.

M. Kada and L. McKinley. 3D building reconstruction from lidar based on a cell decomposition approach. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38(Part 3):W4, 2009.

V.K. Kurkula and M. Kuffer. 3D noise modeling for urban environmental planning and management. In *Proceedings of REAL CORP 008*, pages 517–523, may 2008.

H. Ledoux and M. Meijers. Topologically consistent 3D city models obtained by extrusion. *International Journal of Geographical Information Science*, 25(4):557–574, 2011.

D. Mioc. Flood progression modelling and impact analysis. In *Efficient Decision Support Systems - Practice and Challenges in Multidisciplinary Domains*, pages 227–246. 2011.

S. Oude Elberink. *Acquisition of 3D topography*. PhD thesis, Internationale Institute for Geoinformation Science and Earth Observation, University of Twente, 2010.

F. Rottensteiner and Ch. Briese. A new method for building extraction in urban areas from high-resolution lidar data. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(3/A):295–301, 2002.

F. Rottensteiner and Ch. Briese. Automatic generation of building models from lidar data and the integration of aerial images. In *3-D reconstruction from airborne laserscanner and InSAR data*, volume XXXIV, pages 174–180. ISPRS, oct 2003.

F. Rottensteiner, J. Trinder, S. Clode, and K. Kubik. Automated delineation of roof planes from lidar data. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36(3/W19):221–226, 2005.

R. Schnabel, R. Wahl, and R. Klein. Efficient RANSAC for point-cloud shape detection. In *Computer Graphics Forum*, volume 26, pages 214–226, 2007.

Safe Software. Fme transformers guide. `http://docs.safe.com/fme/pdf/FMETransformers.pdf`, 2012.

J. Stoter and M. Salzmann. Towards a 3D cadastre: where do cadastral needs and technical possibilities meet? *Computers, Environment and Urban Systems*, 27(4):395–410, 2003.

D. Tiede, G. Hochleitner, and T. Blaschke. A full gis-based workflow for tree identification and tree crown delineation using laser scanning. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 9–14, 2005.

P. Van Capelleveen. Urban drainage network modeling better analyzed using arcview 3D analyst. In *Esri International User Conference*, 1999.

O.R. Vincent and O. Folurunso. A descriptive algorithm for sobel image edge detection. In *Proceedings of Informing Science and IT Education Conference*, pages 97–107, 2009.

G. Vosselman. 3D reconstruction of roads and trees for city modelling. In *In International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, pages 231–236, 2003.

G. Vosselman and S. Dijkman. 3D building model reconstruction from point clouds and ground plans. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(3/W4):37–44, 2001.

G. Vosselman, B. G. H. Gorte, G. Sithole, and T. Rabbani. Recognising structure in laser scanner point clouds. In *Laser-Scanners for Forest and Landscape Assessment*, volume XXXVI-8/W2, pages 33–38. ISPRS, oct 2004.

G. Wang, F.H.M. van den Bosch, and M. Kuffer. Modelling urban traffic air pollution dispersion. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXVII(B8):153–158, 2008.

B. Warren-Kretzschmar and S. Tiedtke. What role does visualization play in communication with citizens? - a field study from the interactive landscape plan. In *Trends in Real-Time Landscape Visualization and Participation*, pages 156–167, 2005.