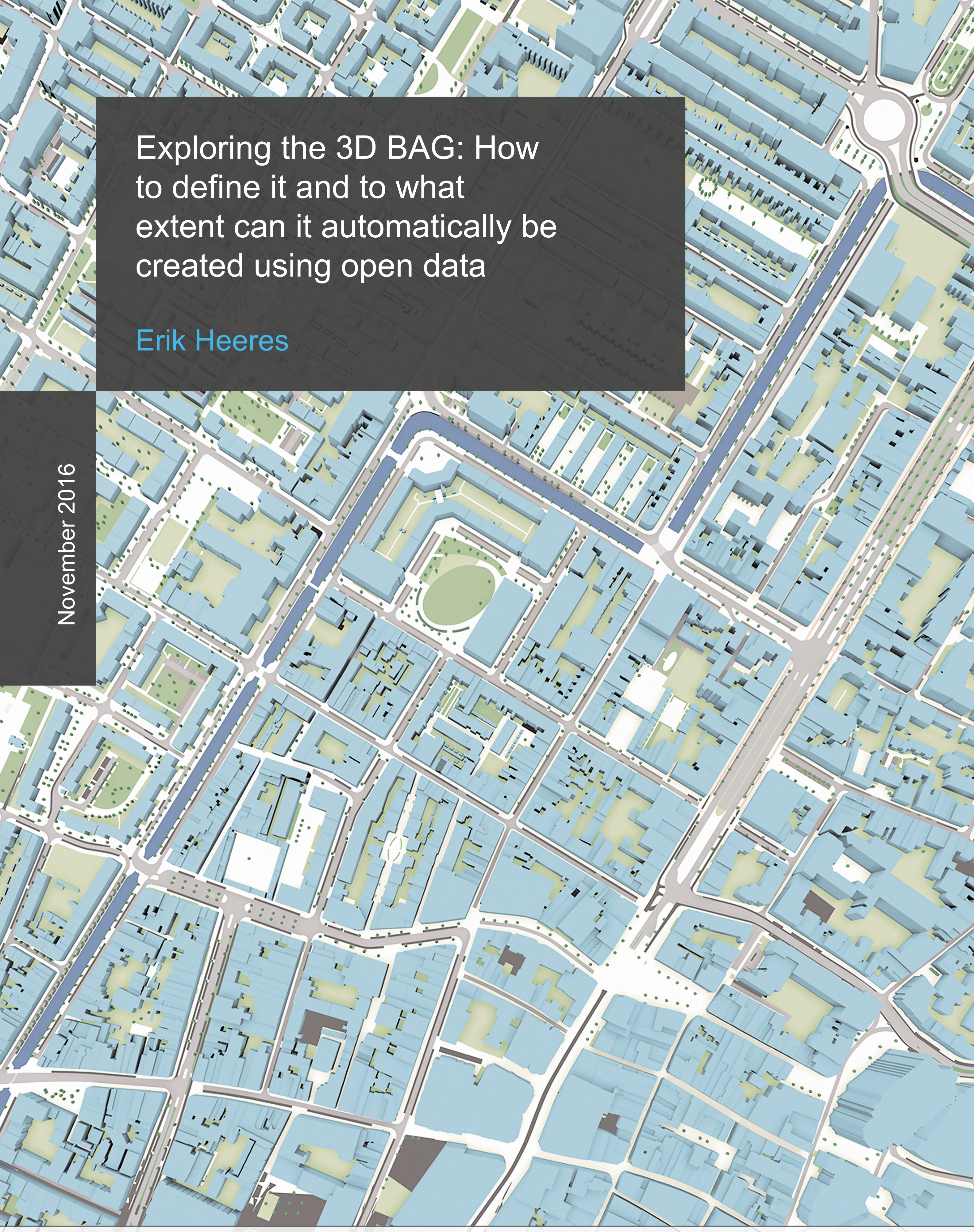


Exploring the 3D BAG: How to define it and to what extent can it automatically be created using open data

Erik Heeres

November 2016

Delft University of Technology

MSc thesis in Geomatics for the Built Environment

EXPLORING THE 3D BAG: HOW TO DEFINE IT AND TO WHAT EXTENT CAN IT AUTOMATICALLY BE CREATED USING OPEN DATA

A thesis submitted to the Delft University of Technology in partial fulfillment of the requirements for the degree of

Master of Science in Geomatics for the Built Environment

by

Erik Heeres

November 2016

The work in this thesis was made in the:

3D geoinformation group
Department of Urbanism
Faculty of Architecture & the Built Environment
Delft University of Technology

Supervisors: Prof.dr. Jantien Stoter
Dr. Hugo Ledoux
Co-reader: Dr. Martijn Meijers

ABSTRACT

The System of Key Registers is the main source of information for all governmental organizations in the Netherlands. It includes information that is used by the government on a regular basis, such as company names, personal data and spatial information. The *Basisregistraties Adressen en Gebouwen* (*BAG*) is part of this system, and stores all buildings and addresses within the Netherlands. It comprises non-spatial information such as the year of construction of a building, but also the 2D representation of buildings and units as polygons and points. Because of the increasing densification of the urban environment, it is more difficult to model the reality on a flat map without losing important information. As such, the 2D representation of the *BAG* has drawbacks and an improvement of this model is necessary.

The aim of this thesis was to explore the needs and possibilities to improve the *BAG* representation from a 2D into a 3D-model. I have especially focused on the main features within the BAG that are most influenced by the shift from a 2D to a 3D representation: the *Panden* (buildings) and the *Verblijfsobjecten* (units). In this thesis, I have firstly proposed a definition for the 3D BAG, and subsequently investigated to which extent the current available open data can be used to create 3D geometries of the buildings and units.

By the use of literature studies and interviews with experts in the field, I recognized a shift within the government regarding the use of 3D data. However, current efforts are scattered and on different scales. Here I proposed a 3D model that connects spatial information of different key registers harmonizing these efforts, and also solving current problems with the 2D representation of the *BAG*. Hence, it provides new possibilities for the data

Moreover, this thesis I proposed methodology for creating 3D geometries of the *BAG* buildings and units. It is based on the use of different, nationwide spatial datasets that are made available as open data by the government. This methodology resulted in a workflow that can theoretically create a nationwide 3D model of the *BAG*. The workflow consists of different steps, starting with the classification of underground buildings and followed by the calculation of the number of storeys, placement of the units over the storeys and dividing the storeys themselves. The workflow was implemented and tested for three areas to investigate its applicability to different environments; the highly densified area of The Hague, the urban area of Hoofddorp and the rural of Schoonhoven. The resulting dataset of The Hague was validated using a reference dataset provided by the municipality.

Results of the implementation of a 3D model of the *BAG* validated the potential of the designed methodology. For most buildings in the Netherlands, a 3D model can be created with reasonable accuracy. However, this thesis also reveals uncertainties within the *BAG* that makes it impossible to create accurate 3D models for all buildings and units with the developed methodology. When aiming at improving the results and overcoming these uncertainties a wide approach is needed. This approach should not solely focus on improving the developed methodology, but also on the acquisition of data that provides more information about the location and geometry of buildings and units.

ACKNOWLEDGEMENTS

I would like to express my gratitude to all those who supported me during this graduation project. First, I would like to thank my supervisor Hugo Ledoux for his excellent guidance, suggestions, and feedback on my work. Moreover, I would like to thank Jantien Stoter for being my graduation professor and providing me useful contacts within different governmental organizations. Furthermore I would like to thank my co-reader, Martijn Meijers, for his useful comments and advise during the entire thesis. Special thanks to Natasja de Vries for her support during my graduation period and for proof reading my concept thesis's. Finally, I would like to thank all my family and friends for their continued support during my student life.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

DEM	digital elevation model	69
DTM	digital terrain model	69
GIS	geographical information system	24
LV BAG	*Landelijke Voorziening BAG*	6
BAG	*Basisregistraties Adressen en Gebouwen*	iii
BGT	*Basisregistratie Grootschalige Topografie*	5
BA	*Basisregistratie Adressen*	6
BG	*Basisregistratie Gebouwen*	6
WOZ	*Waardering Onroerende Zaken*	11
BRT	*Basisregistratie Topografie*	5
BRO	*Basisregistratie Ondergrond*	5
LIDAR	Light Detection And Ranging of Laser Imaging Detection And Ranging	20
AHN	*Actueel Hoogtebestand Nederland*	30
AHN2	*Actueel Hoogtebestand Nederland 2*	
AHN3	*Actueel Hoogtebestand Nederland 3*	31
PDOK	*Publieke Dienstverlening Op de Kaart*	69
GML	Geographic Markup Language	63
XML	Extensible Markup Language	97
UML	Unified Modeling Language	99
LoD	Level of Detail	4
BIM	Building Information Model	20
OGC	Open Geospatial Consortium	21
ADE	Application Domain Extension	98

1 INTRODUCTION

The introduction of the computer in the late twentieth century has been of great influence on how we handle spatial information. Instead of using physical paper maps, it was now possible to store spatial information on the computer. This made it possible to create a centralized datasource which could be shared with everybody. The Dutch government recognized this shift and began developing policy that took advantage of the newly created possibilities [van Duivenboden and de Vries, 2003]. One of the results of this development was the creation of the System of Key Registers: Centralized databases in which essential information is stored that is used by governmental organizations.

The *BAG* is part of this system of key registers and stores all buildings and addresses within the Netherlands. It is a combination of two key registers: the register for addresses and the register for buildings. The registers provide information on for instance the year of construction of a building, the net floor area of a dwelling and the intended use of a unit. Also the geometry of buildings is included in the *BAG*; the units (with their address) are stored as points and the buildings as polygons. In Figure 1.1 the two datasets are shown on top of each other. The *BAG* is mostly used because it models the current state of all buildings and units in the Netherlands, although it also includes the history of these units.

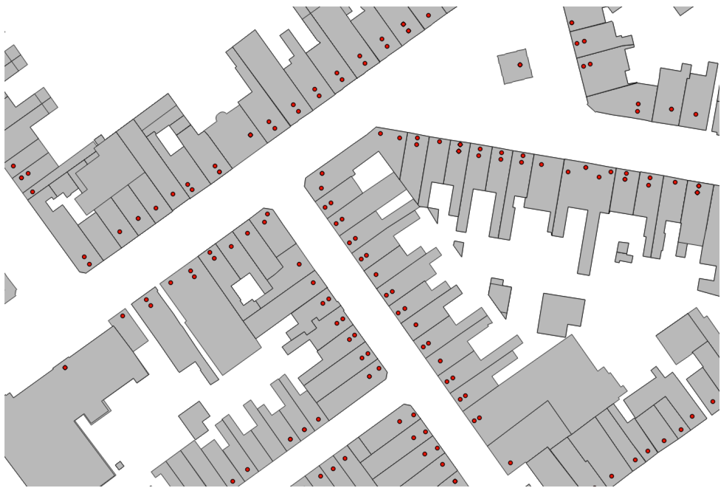


Figure 1.1: A section of the *BAG* with the two datasets overlaid: the buildings (gray polygons) and the units (red points). Buildings can include multiple units

The information that is included in the *BAG* is not only used by the government, the private sector also benefits from this information. Examples are architecture firms, real estate companies, banks and the government. In almost all situations the *BAG* is combined with other data sources in order to create valuable information.

But when more detailed analyses are desired, the *BAG* is not sufficient enough. A 3D model of buildings and units would allow them to perform more detailed analysis about separate units, e.g. the heat loss could be estimated. All these factors have an influence on the value of a unit, but can only be calculated using a detailed 3D model. Another situation in which the 3D model of the *BAG* is highly valuable constitutes the fire prevention of a building. In case of a fire in a big apartment building, firefighters can quickly develop a plan to combat a fire when they have a 3D model available. The shape of the building and the location of the separate apartments is in that case of great additional value.

In order to create a full accurate 3D representation of the *BAG*, all data should be acquired again with that purpose in mind. However, this is not feasible in practice due to the costs and time it takes. The current efforts by the municipalities of Rotterdam and The Hague aim at creating a 3D *BAG* by combing different datasets which are already available. They are currently working on the creation of a 3D model of their city including the *BAG*. In their work, they combine an an extended version of the *BAG*, named the BAG+ (discussed in more detail in Section 2.2.2.) with a 3D city model. The preliminary results of their work are promising, however further research is required for improvements of this combined model. Especially the method for placing a unit within a storey with use of a voronoi diagram gives unrealistic results in corner areas as shown in Figure 1.2. The method that the municipality of The Hague used to create their 3D *BAG* will be discussed in Section 3.3.1.

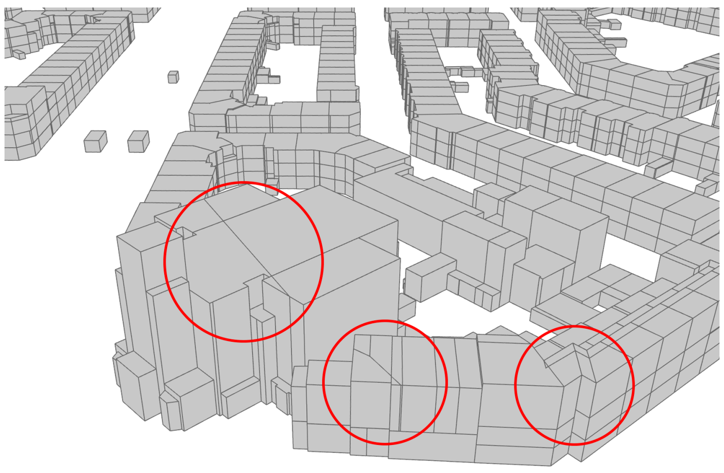


Figure 1.2: A section of the 3D *BAG* of the municipality of The Hague. Around the corners the unrealistic placement of units is clearly visible.

However, the biggest problem is not the unrealistic placement of the units, but the data which is used to construct the model. For the creation of their version of the 3D *BAG* both municipalities relied partly on data that is also available as open data. Open data is freely available to everyone to use and republish as they wish, without restrictions from copyright, patents or other mechanism of control [Auer et al., 2007]. But both municipalities also used other datasets in order to create their 3D BAG. Both municipalities used a *BAG+* dataset and a LOD2 city model. These datasets are not available nationwide and therefore the methods that both municipalities use for creating their 3D *BAG* is not scalable for other parts of the Netherlands.

The different datasets that are created by the municipalities and how they trie to combine these in order to create a 3D model of the *BAG* shows that a common approach is absent. The current efforts are small scaled, inefficient and scattered. Since every municipality develops its own method, the output will be different as well. The models are not uniform and cannot be compared. This problem was noticed by the *BAG BOA*, the board that is responsible for the strategic direction of the development of the *BAG*. Recently they have set up a working group to further develop and define the 3D *BAG* (Patrick Schmidt, personal communication, April 26, 2016). This thesis may contribute to this development because it is in line with this research.

1.1 RESEARCH OBJECTIVES AND QUESTIONS

This thesis explores the needs and opportunities when improving the *BAG* representation from 2D to 3D. This research especially focusses on the placement of *BAG* units within the buildings. Furthermore this research aims to develop a methodology which enables the creation of a approximate 3D version of the *BAG* units from open data. Furthermore, this research aims at identifying which crucial information is missing for the development of 3D geometries of the *BAG* units nation wide. Hence, the research question addressed in this thesis is:

How to define the 3D BAG and to what extent can the current available open data be used to create 3D geometries of the BAG units?

In order to answer the research question and test and validate the applicability of currently available information a prototype has been developed. This prototype tries to create a 3D model of the *BAG* from the current available information in which the units are represented as volumes, rather than points. The subquestions required to answer the main research question are:

- What is the 3D *BAG*?
- Why are municipalities interested in a 3D version of the *BAG*?
- How can the number of storeys of a building be calculated?
- How can underground parking lots be detected?
- What logic can be used for placing units within a building?
- How can floors be divided using the information within the *BAG*?
- Which uncertainties had the most influence on the automatic creation of the 3D *BAG*?

- Is it possible to combine the 3D geometries of the *BAG* with other key registers?

1.2 RESEARCH SCOPE

In order to maintain a clear research focus and to more strictly define the scope of this research, the following boundary conditions are set:

- This thesis only focusses on the geometry part of the *BAG* The system of key registers includes many rules and regulations, but this is outside the scope of this research.
- Data acquisition is not part of the research. The developed method will be based on data which is made available by the government as open data and is (or will be) available for the entire country. For validation and and improvement of the method, additional non-open information from the municipality of The Hague is used.
- To minimize temporal differences, datasets will be selected on a selected date of acquisition as much as possible. The final outcomes of the implementation will therefore not be a model of the current state.
- The exterior geometry of the BAG will be in level of detail 1 (LOD1), the height of the building will therefore be considered the same for the entire building. To which extent improving the Level of Detail (LoD) will influence the result will be discussed in the conclusions as future work (Section 7.2.2).

1.3 OUTLINE OF THIS THESIS

The chapters in this these are structured as follows:

- Chapter 2 describes the current state of System of Key Registers and the *BAG*.
- Chapter 3 explores the needs and opportunities for a 3D model of the *BAG* and concludes with a proposal for how it can be modelled.
- Chapter 4 presents the methodology developed in this thesis for the creation of rough 3D volumes for the *BAG* units.
- Chapter 5 describes how the proposed methodology is implemented and tested on a selected area.
- Chapter 6 explores the possibilities for storing the geometry of the *BAG* within the CityGML IMGeo standard.
- Chapter 7 summaries the most important findings and conclusions of the research. Moreover, recommendations for future work and data acquisitions will be listed.

2 BACKGROUND INFORMATION ABOUT THE SYSTEM OF KEY REGISTERS OF THE NETHERLANDS

Before the 3D *BAG* is discussed, this chapter will give more insight into the system of key registers of the Netherlands. The focus of the research lies on the creation of 3D geometries of the *BAG*. Hence, the emphasis of this chapter will be on the geometrical features of the *BAG*. First the origin of the system is discussed followed by a detailed overview of the *BAG*. The final chapter will discuss the *Basisregistratie Grootschalige Topografie* (*BGT*), a dataset which is of importance for this research.

2.1 OVERVIEW OF THE KEY REGISTERS

In the year 2000 the Dutch government started the two year program "Stroomlijning Basisgegevens". This program focussed on the vital information that is used by the government and other organizations with a public task [van Boxtel, 2001]. Examples of vital information are personal data, companies records and information about buildings. The goal of the program is to create the situation that citizens and companies only had to provide their data to the government once and only to one authority. The government in the Netherlands is obliged to use this data and are not permitted to collect any data that already exists in a key register. The authentic key registers function government-wide as the main source of information. There are twelve different key registers, which are all (soon to be) connected (Figure 2.1). The content of these key registers is determined by national law.

The key registers that contain geometry are:

- *Basisregistratie Topografie* (*BRT*) - Topographical maps with multiple scales,
- *BGT* - Large scale topographical map, this register will be discussed at the end of this chapter,
- *Basisregistratie Ondergrond* (*BRO*) - This registers will contain information about the underground,
- *BAG* - The register for buildings and addresses.

The system of key registers can be seen as one connected system which enables the government to serve its public task regarding the registration of physical objects.

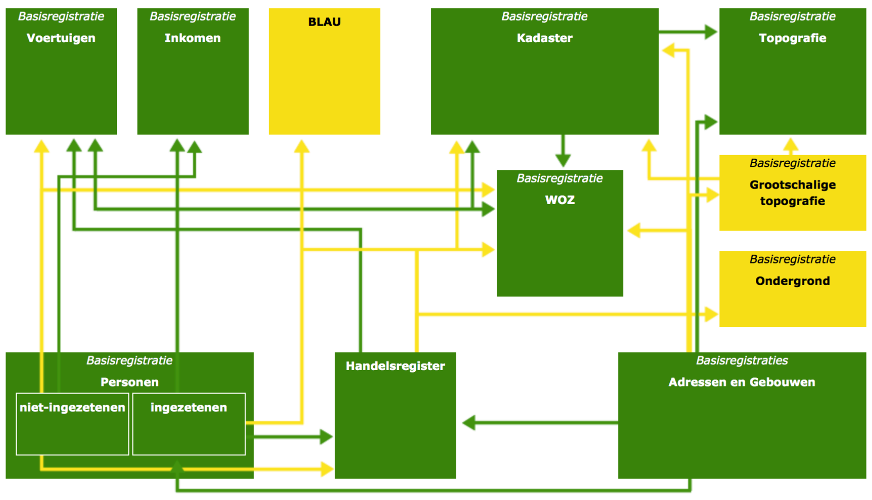

Basisregistratie Voertuigen
Basisregistratie Inkomen
BLAU
Basisregistratie Kadaster
Basisregistratie Topografie
Basisregistratie WOZ
Basisregistratie Grootschalige topografie
Basisregistratie Ondergrond
Basisregistratie Personen
niet-ingezetenen
ingezetenen
Handelsregister
Basisregistraties Adressen en Gebouwen

Figure 2.1: The state of the system of key registers in 2014. The green parts had already implemented, the yellow parts were still work in progress (Figure by Ministerie van Binnenlandse Zaken en Koninkrijksrelaties [2014]).

.

2.2 BASISREGISTRATIE ADRESSEN EN GEBOUWEN (BAG)

The BAG is composed of two connected key registers, namely the *Basisregistratie Adressen* (BA) (registration for addresses) and *Basisregistratie Gebouwen* (BG) (registration for buildings). Municipalities are source holder of the BAG and are responsible for the quality and completeness of the data. The data from all municipalities is gathered in a database called the *Landelijke Voorziening BAG* (LV BAG) and maintained by the Kadaster, the organization responsible for the Dutch cadastre. The Kadaster is responsible for the distribution of the data to organizations in the public sector, companies and individuals [Ministerie van Infrastructuur en Milieu, 2016]. Since the BAG was developed in 2009, all data has been stored permanently. Therefore, not only the current state of units is known but also their history.

2.2.1 Object classes

There are seven object classes that must legally be incorporated in the BAG and they are described in the *Catalogus Basisregistraties Adressen en Gebouwen* by Rietdijk [2009]. In Figure 2.2 relations between these object classes are visualized. The classes coloured green are part of the BA, the others are part of the BG. When creating a 3D model of the BAG, the *Pand* and the *Verblijfsobjecten* will be affected most. These affected classes will be discussed in more detail below.

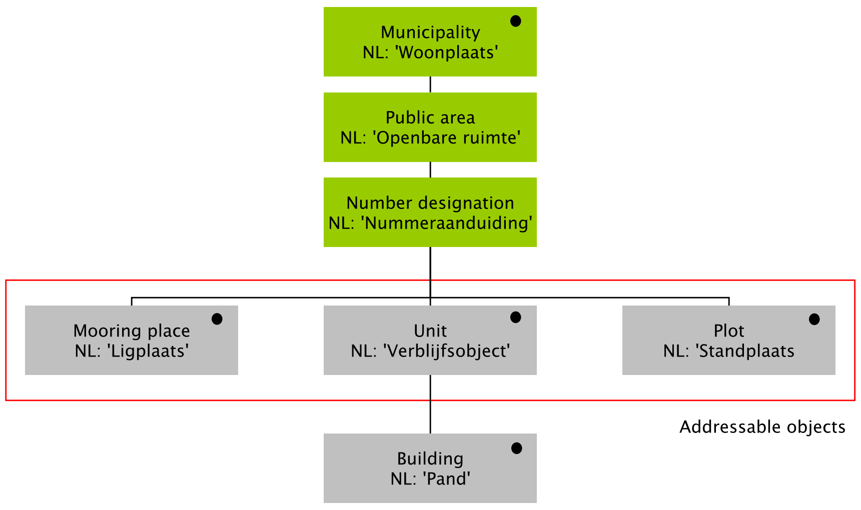


Figure 2.2: The relationship model of the BAG object classes. The green classes are part of the key register for addresses, the classes in gray are part of the key register for buildings. The classes with a black point include geometry.

Verblijfsobject

The *verblijfsobject*, which from now will be called an *unit*, is the main subject of this research. The definition by Rietdijk [2009] for a unit is:

> *"Een verblijfsobject is de kleinste binnen n of meerdere panden gelegen en voor woon-, bedrijfsmatige, of recreatieve doeleinden geschikte eenheid van gebruik die ontsloten wordt via een eigen toegang vanaf de openbare weg, een erf of een gedeelde verkeersruimte en die onderwerp kan zijn van goederenrechtelijke rechtshandelingen."*

Meaning it is the smallest unit suitable for living, professional or recreational purposes which is connected to the public road and which may be subject to property law acts. A unit can can be located in one or multiple buildings and is accessible by a public area such as a public road. This class is not only connected to the building class by the geometry, the point of a unit has to lie within the polygon geometry of the building, but is also made explicit by an identifier which is included as attribute of the units. Although it should not be possible, in practice it can be the case that points are located outside the building polygon. It is therefore preferred to use the identifier to relate the unit and buildings. Other attributes of the object class that will be used in this research are; the intended use of a unit, the floor area and the geometry. These attributes will be discussed separately. All attributes of the units can be found in Table 2.1

Dutch	English	Building	Unit
Identificatie	Identification	✓	✓
Aanduiding data in onderzoek	Indication data in research	✓	✓
Begindatum tijdvakgeldigheid	Start date of validity	✓	✓
Einddatum tijdvakgeldigheid	End date of validity	✓	✓
Documentdatum mutatie	Document mutation date	✓	✓
Documentnummer mutatie	Ducument mutation number	✓	✓
Indicatie geconstateerd	Indication detected	✓	✓
Geometry	Geometry	✓	✓
Pandstatus	Property status	✓	
Oorspronkelijke bouwjaar	Year of construction	✓	
Aanduiding Hoofdadres	Designation main address		✓
Aanduiding Nevenadressen	Designation additional addresses		✓
Gebruiksfunctie	Use function		✓
Oppervlakte	Floor area		✓
Pandrelatering	Relation with buildings		✓
Verblijfsobject status	Status		✓

Table 2.1: Attributes of the BAG buildings and units.

Unit attributes

There are eleven different intended use functions that a unit can have, ranging from residential to industry. These functions are based on the *Bouwbesluit*[1], the building regulations set by the government. Combinations of these functions are also possible. It is for instance not uncommon that residential and store functions are combined. The functions types that a unit can have are:

Dutch name	English translation
Woonfunctie	Residential function
Bijeenkomstfunctie	Gathering function
Celfunctie	Cell function
Gezondheidszorgfunctie	Health care function
Industriefunctie	Industry function
Kantoorfunctie	Office function
Logiesfunctie	Accommodation function
Onderwijsfunctie	Educational function
Sportfunctie	Sport function
Winkelfunctie	Store function
Overige gebruiksfunctie	Others use function

Table 2.2: Possible use functions of units

The floor area included in the unit follows the NEN 2580 standard for usable floor area. The measured area is the surface bounded by the outer walls, so it also includes the inner walls. Details about this area are described in the *Oppervlakte Verdiepingsdocument voor gemeenten* [Ellenkamp et al., 2007]. The areas excluded are:

- Surfaces in which the height is less than 1,5 meters;

1 http://www.bouwbesluitonline.nl

- Elevator shafts;
- Stairwalls, lunettes and vides with an surface larger than 4 m^2;
- Detached building constructions, with an area bigger than 0.5 m^2;
- Supporting walls.

The last attribute of interest is the geometry. The geometry can be stored as polygon- or point geometry, the latter being mainly used in practice. Drawbacks of this representation will be discussed in Section 3.1.

Pand

Connected to the unit class is the *Pand*, the building class. The definition by Rietdijk [2009] for a building is:

> *"Een pand is de kleinste bij de totstandkoming functioneel en bouwkundig-constructief zelfstandige eenheid die direct en duurzaam met de aarde is verbonden en betreedbaar en afsluitbaar is."*

Meaning that it is the smallest entity that was constructed independently in both functional and structurally ways. It is permanently connected to the earth, enterable and lockable. In this research the geometry of the building will be used. The geometry of the buildings follows the outer boundary as seen from above. In section 3.1 drawbacks of this representation will be discussed.

2.2.2 BAG+

The *BAG+* is the name for an optional extension of the BAG that some municipalities created for internal use. The content is not described by law, thus can differ between municipalities. It is not part of the system of key registries and therefore also not distributed by the Kadaster. Information about which municipalities have created a *BAG+* and what the content of this dataset is per municipality, is not publicly available. The same is the case for the data itself. A major reason for the inaccessibility might lie in the restrictions on the data that is used for the creation of the *BAG+*. Often it is created by combining datasets from different organizations within the government which all have different restrictions for publishing their data. Examples of BAG+ attributes are given in Table 2.4.

For the municipality of Amsterdam and Rotterdam information is available about what they include in their *BAG+*. On their website[2], the municipality of Amsterdam provide an overview what additions they made on the BAG. However, the data itself is not made available. The number of floors of a unit, on which floor level the entrance of a unit lies, the number of rooms within a unit and how the unit was paid for are examples of additional information that is added by the municipality [Gemeente Amsterdam, 2016b]. The municipality of Rotterdam also includes additional information about buildings and addresses in their *BAG+*. In Tables 2.4 and 2.3 the attributes are shown which are included in the *BAG+* of both municipalities. The *BAG+* attributes of the municipality of Rotterdam are described by Boeters

2 https://www.amsterdam.nl/stelselpedia/bag-index/informatiemodel-bag/

[2013]. Some of attributes are quite similar but defined differently. For example the number of floors of a unit: Rotterdam includes the lowest and highest floor of the residential unit, whereas Amsterdam only includes the number of floors. Because the definitions of the attributes are not harmonized, it is hard to compare or combine the data even if the data was made publicly available.

Attributes	Amsterdam	Rotterdam
Identification (local system)	✓	
Building type		✓
Function (more detailed)	✓	✓
Type of a unit	✓	✓
Lowest floor residential unit		✓
Highest floor residential unit		✓
Number of floors	✓	
Entrance floor	✓	✓
Access location	✓	
Number of rooms	✓	
Number of rentable units	✓	
Positioning of unit	✓	
Ownership type	✓	
Funding method	✓	
Subsidy		✓
Target group		✓
Reason for erasement	✓	
Reason for generation	✓	
Indicator-expired	✓	
Mutation by	✓	
Status coordinates	✓	
Indicator housing stock	✓	
Within area	✓	

Table 2.3: BAG+ attributes corresponding to the unit class.

Attributes	Amsterdam	Rotterdam
Identification (local system)	✓	
Name of Building	✓	
Lowest storey		✓
Highest storey		✓
Number of floors		✓
Building type		✓
Within building block	✓	
District heating		✓
Indication-expired	✓	
Mutation by	✓	

Table 2.4: BAG+ attributes corresponding to the buildings class.

The source of this information are often datasets that are also used within the municipalities to determine the *Waardering Onroerende Zaken* (WOZ). This key register includes the value of real estate for tax purposes. This value is calculated every year by the municipalities. For taxation a lot of information is used, e.g. information about the type of a unit and on which level the units is located. This is more detailed information than is included in the *BAG*. Because the key register describes the same units as the *BAG*, these additional datasets can easily be combined to enrich the standard *BAG* data. Another example of a municipality that stores additional information about *BAG* features, is the municipality of The Hague. They have datasets in which the unit floors are stored as separate entries, which include information about the floor levels, floor area, the type of the unit and in which neighborhood the unit is situated. What information is used by the municipalities and the information itself is not publicly available.

2.2.3 Usage of the *BAG*

As mentioned in the introduction, the *BAG* is used by the public and private sector. All public authorities must use the key registers for the execution of their public tasks. This means that all municipalities, provinces, water boards and other organizations with a public task use the key registers, including the *BAG* [Digitale Overheid, 2016]. From the *Tevredenheidsonderzoek BAG*, a report about the user appreciation of the *BAG*, can be concluded that there is a wide range of private organizations that use the the *BAG* as well [Statisfact, 2015]. The target group of the research is limited by the selection that the Kadaster and the Ministry of Infrastructure and Environment made. However, the research does provide insight in the users and applications of the *BAG*. There are a lot of cases where the geometry of the *BAG* is used, for instance for:

- Visualization - Municipalities, real estate et al.
- Determination of acces routes - Emergency services
- Appreciation of real estate - Government agency, real estate et al.

- Implementation in geo-applications - Consultancy, geo-companies et al.

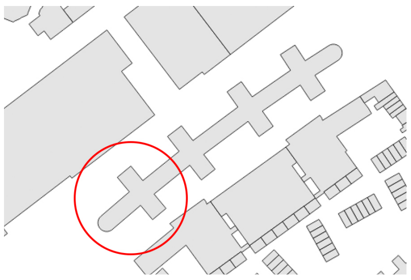
These use cases are only a small selection of the current use cases in which the geometry of the *BAG* is the main subject. However, it shows the value of this part of the register.

2.3 BASISREGISTER GROOTSCHALIGE TOPOGRAFIE (BGT)

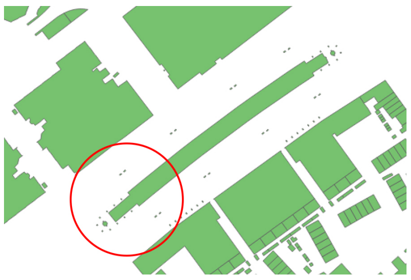
Along with the *BAG*, the *BGT* is also part of the system of key registers in the Netherlands. It is a detailed topographical map of the entire country. The *BGT* includes all physical objects such as buildings, roads, water and vegetation [Ministerie van Binnenlandse Zaken en Koninkrijsrelaties, 2016] and only shows the current situation. As is the case for the *BAG*, the goal of the *BGT* is to provide one uniform and unambiguous dataset for the entire country. The *BGT* has multiple source holders, not only governmental but also private parties, e.g. utility providers, contribute to the creation of the registry. The government uses *BGT* for the planning and maintenance of the environment.

The part of the *BGT* that is interesting for this research is the *Pand* features. The buildings in the *BAG* and the *BGT* use the same identification in their attributes, so there is a strong relation between these datasets. In the *BGT* also the geometry of the buildings is included. In practice not everybody is familiar with the fact that there can be a difference between the geometry of the *BAG* and the *BGT*. Where the geometry of the *BAG* follows the outer perimeter as seen from above (including the underground structures), the geometry of the *BGT* follows the footprint of the building. In most cases the geometry of the *BAG* is used for the creation of the *BGT*, because the geometry of the ground level is similar to the geometry of the outer periphery. But there can be a difference when part of the construction is under or above the ground level. This difference is shown in Figure 2.3. Because there is a solid connection between the *BAG* and the *BGT* but a difference in representation, this difference between representation provides extra information about the reality. This information will be used to find the underground structures and when placing the units (Section 4.1).

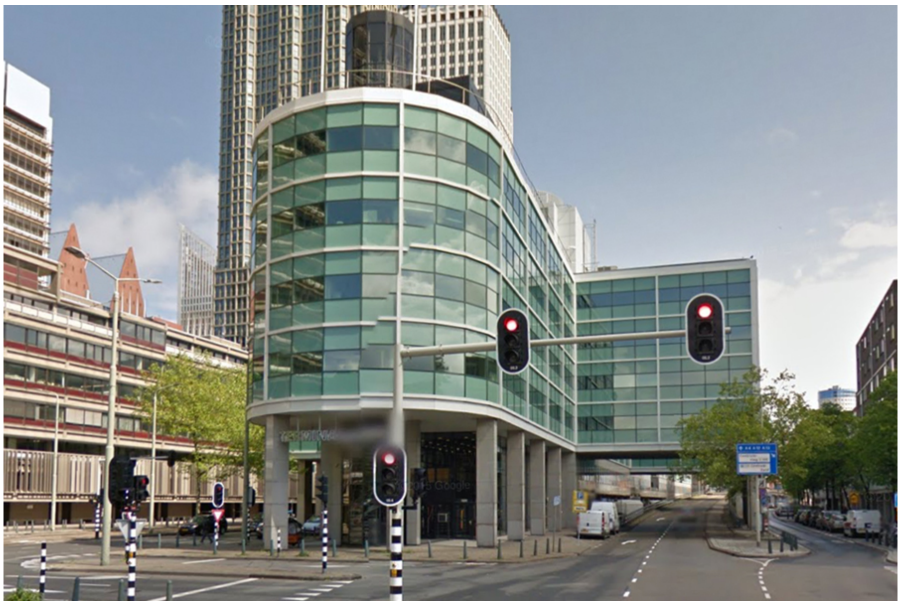
What should be noted is that the key registers are maintained by different organizations. In theory, both the registers contain the same information but in practice there can be variation in time. Because information about when the data was mutated are kept in both datasets, it is possible to check what information is most up to date.



(a) BAG



(b) BGT



(c) Streetview by Google maps [Google, 2016]

Figure 2.3: The difference between the BAG, BGT when modeling the reality. The buildings of interest is the office building of Stadsgewest Haaglanden (Location in EPSG 28992: 81886.6, 454907.6).

3 NEEDS AND OPPORTUNITIES FOR A 3D VERSION OF THE BAG

The increasing demand for a 3D model of the *BAG* is due to several factors. In this chapter the origin of this demand will be investigated. First the drawbacks of the current model are discussed, followed by the current trend toward the use of 3D data by governmental organizations. The current work the creation of 3D geometries for the *BAG* will be discussed next, along with my proposal for one complete city model of which the *BAG* can be part of.

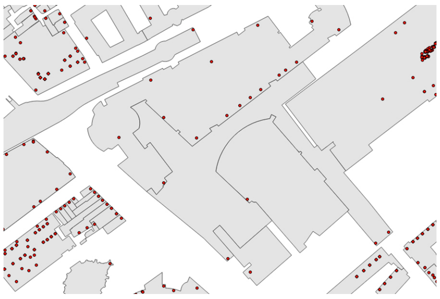
3.1 DRAWBACKS OF CURRENT 2D BAG REPRESENTATION

Currently, a 2D representation is used for the *BAG* geometries. Advantages of this 2D representation is that working with the data is relatively simple and that most software packages support 2D geographic information. However, this 2D representation has some critical limitations. These limitations mainly involve uncertainties about the configuration of buildings and units. In this section three examples of these uncertainties will be discussed: Underground and above-ground structures, complex building configurations and the location of units.

3.1.1 Underground and above-ground structures

In the *BAG* no distinction is made between underground and above-ground structures, not in the geometry but also not at attribute level. This can be important information, especially in densifying urban areas where the number of underground structures increases. An example of is the Spui, a square in The Hague. Under the square there is an underground parking garage, which is not distinguishable as such from the *BAG*. The difference between the *BAG* representation and the reality is shown in Figure 3.1. In this example the geometry fits between the surrounding geometries, but there are also cases in which the geometry overlaps partly or completely with other geometries. In both cases it is not possible to figure out the real situation.

Because the geometry is a combination of the perimeter of the underground structure, the ground level and the superstructure, it is possible that the geometry represents a combination of these structures (Figure 3.2). The *BAG* shows the underground parking garage which is part of the surrounding buildings, but from this representation the user might think that there is no square at all.


(a) BAG geometry.

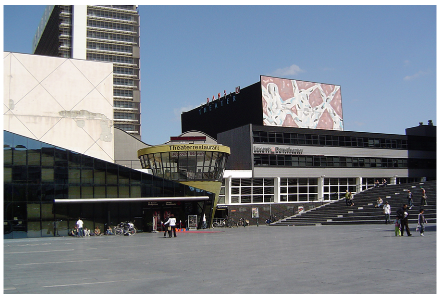
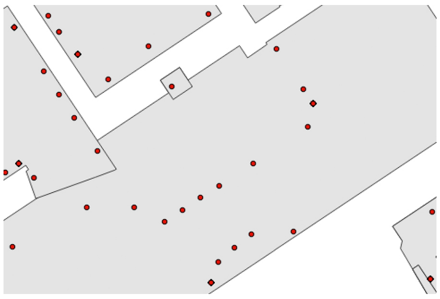

(b) Picture, Wikimedia Commons [2007b]

Figure 3.1: Spui square in The Hague, difference between the BAG and reality.


(a) BAG geometries

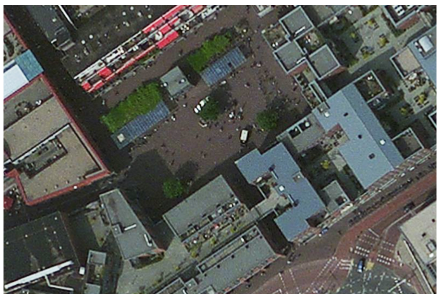
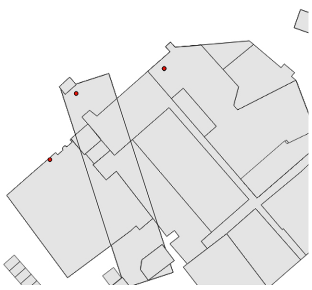

(b) Bing aerial view, Microsoft [2016]

Figure 3.2: The difference between the aerial view and the BAG geometry.

3.1.2 Complex building configurations

Another problem with the current 2D representation is the modeling of complex building configurations. In these situations different buildings are located above each other, or have an even more complex configuration. For instance, buildings can interlock one another. An example is the office of Unilever in Rotterdam (Figure 3.3). From the *BAG* it is not clear what the building looks like in reality. This is an extreme example and not very common, but these configurations are not rare at unit level, the next drawback of the 2D representation.


(a) BAG geometry.

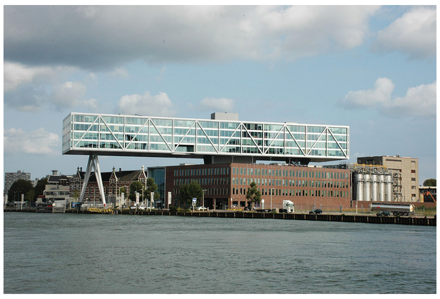
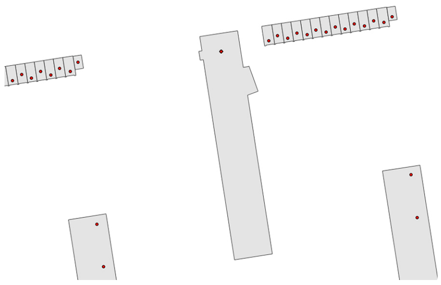

(b) Picture, Wikimedia Commons [2014]

Figure 3.3: The Unilever office in Rotterdam.

3.1.3 Unit location

The geometry of the units in the *BAG* are modelled as points. There are no constraints about the location of these points, except that is has to lie within the polygon of the related building. In Figure 3.4, the red points in the largest building are in fact hundred points with the same coordinates. From this data it is not clear where in the building these units are situated.


(a) BAG geometry.

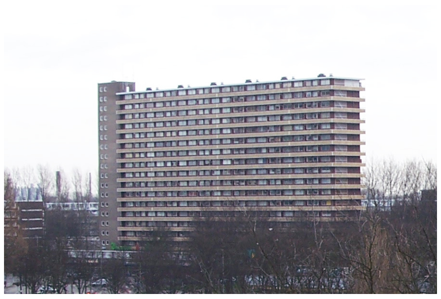
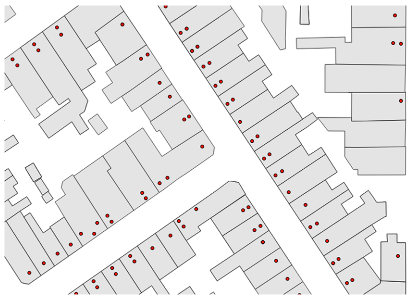

(b) Picture, Wikimedia Commons [2007a]

Figure 3.4: Apartment building in Delft, difference between the BAG and reality.

Some municipalities have a structure of placing the points (placing them in the center of the building polygon or placing the points that are on the ground floor closer to the edge of the building polygon), other municipalities do not seem to follow any structure. Figure 3.5 shows the differences between the municipalities of Delft and The Hague. For some buildings in the section of The Hague, the placement of the points looks almost random. In contrast to the section of Delft, where there seems to be a structure.


(a) Section of Delft

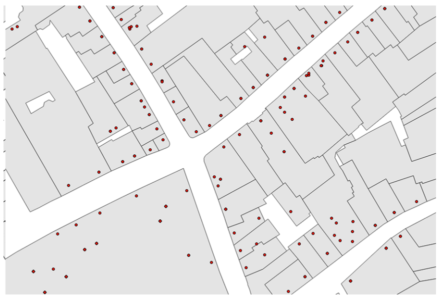

(b) Section of The Hague

Figure 3.5: Differences between placement of point geometries between two municipalities.

Because there is no constraint about the location of the points, it is not possible to make unambiguous conclusions about the location of the unit within the building: Not on which floor the unit lies, but also not where the units is located within the plane of the building polygon.

The uncertainties on building and unit level limit the usability of the *BAG* geometry. When these uncertainties are solved, new possibilities for the data will arise. These possibilities will be discussed in Section 3.4.

However, most buildings in the Netherlands do not lie in a highly densified area. In the less densified areas most buildings that have a related unit, only have one of them. To put this in perspective the *BAG*; almost 92% of the buildings in the Netherlands only have one related unit (Figures 3.6). From the other buildings, the case that there are two related units is most frequent (Figure 3.7). This is a nationwide average and will differ between rural and urban areas.

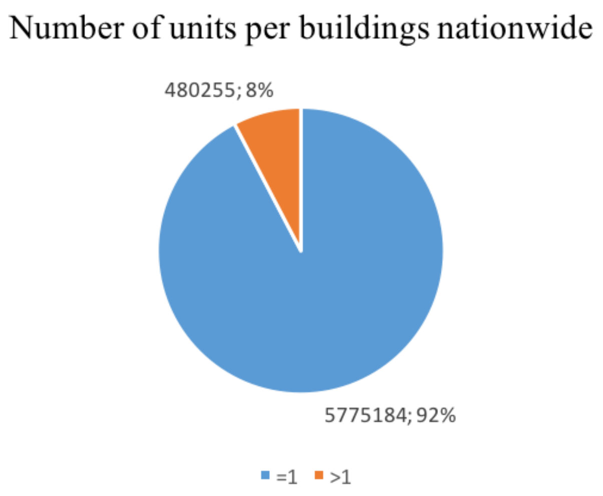


Figure 3.6: Number buildings that only have one related unit.

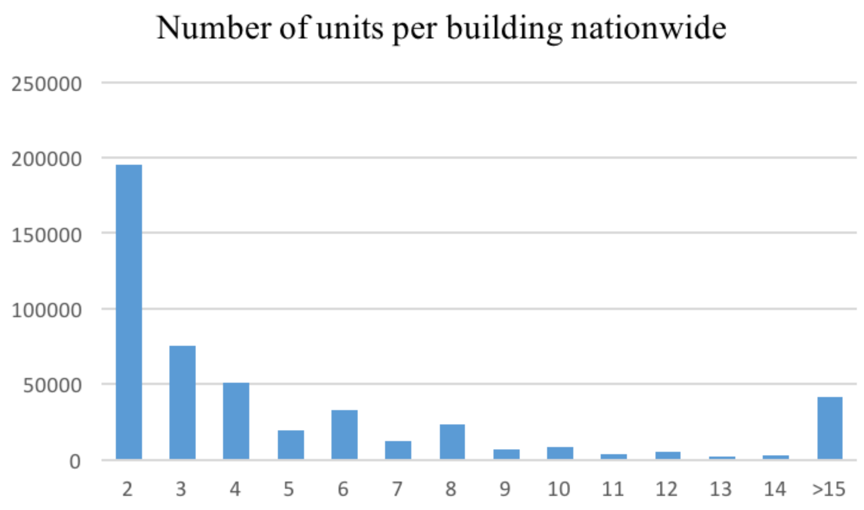


Figure 3.7: Amount of buildings with a certain amount of related units. This information is retrieved from the BAG dataset of the entire Netherlands.

3.2 CURRENT MOVEMENT TO 3D INFORMATION

In recent years, there has been a trend towards 3D data in the geographic information sector. This trend is also visible in the Netherlands. There is not a specific demand that drives this development, but it is driven by the potential that 3D data has for a wide range of applications. Software developers are responding on this trend, resulting in a growth of software packages that are capable of processing 3D data. The government also plays an important role in this development because it is an important user of geographical information and possesses large amounts of geographical data. The 3D cadastral registration and the 3D *BGT* are examples of how the government is moving to 3D data. In this sections several examples will be discussed of which are the result of this trend.

3.2.1 Dutch Cadastre

In March 2016, the first property deed with the rights visualized in 3D was registered by The Netherlands' Cadastre, Land Registry and Mapping Agency in short Kadaster. It concerns the railway zone in Delft, an area where multiple parties are involved in the ownership agreements. The complexity of the different parcels was a driving factor for moving to 3D in this case. The different parcels had overlap underground and aboveground, which made a clear visual representation in 2D almost impossible. Therefore they developed the possibility to visualize it in 3D (Figure 3.8). This complexity of overlapping parcels is getting more common in the densifying urban environment of the Netherlands, increasing the importance of using 3D in the cadastral registration.

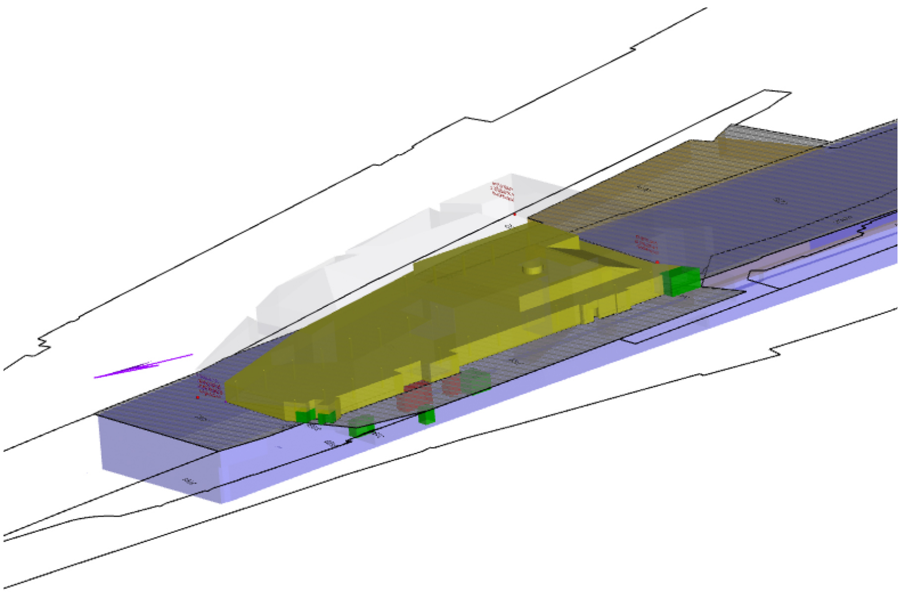


Figure 3.8: 3D registration of the train station in Delft. (Image by Kadaster [2016])

3.2.2 Top10NL 3D

TOP10NL is a topographical map of the Netherlands, created and maintained by the Kadaster. In 2015 a 3D version of the TOP10NL was created, the TOP10NL 3D. This was done automatically by combining 2D information with Light Detection And Ranging of Laser Imaging Detection And Ranging (LIDAR) data [Elberink et al., 2013]. The full dataset can be requested, but there is also a test area available for download[1]. Capturing the LIDAR data for the entire country is not done on a regular basis, which is the main reason that the dataset is not up to date.

3.2.3 Municipalities

At smaller scale, some larger municipalities are also introducing 3D data into their workflows. In 2011 the municipality of Rotterdam finished a 3D model of their city and made it publicly available[2] [Gemeente Rotterdam, 2011]. Similar to the city model created by Den Hague[3] (Figure 3.11), there where no logistics to maintain the model so within a couple of years both models were outdated. Because the model of Rotterdam proved to be of big value, the municipality is currently developing a new city model which also includes the underground. But this time the model will be maintained regularly (Patrick Schmidt, personal communication, April 26, 2016). This means that there is going to be a complete and integrated program: regular acquisition of data, software and hardware solutions to store and update the model and human resources. This project is still in his early stages and will not be up and running soon, but it is a promising development.

The municipality of Utrecht works with 3D on a neighborhood scale. One current project is a visualization of the Veemarktterrein, a new neighborhood in Utrecht, in cooperation with the ministry of Infrastructure and Environment. Another project is called Leidsche Rijn Centrum, a large new building project in which the municipality works along with a real estate- and housing corporation to develop a complete new city centre for Leidsche Rijn. The complete area is included in one Building Information Model (BIM) in which multiple architects include their designs [Hfb, 2016]. This creates one large and detailed digital maquette (Figure 3.9).

The municipality of The Hague has a vision for the use of their 3D data called: Dynamic 3D model. This model should become a complete 3D city model of the city, but it should contain more information i.e.functional maps, noise pollution and other restrictions. In the planning phase the architect should be able to upload his design and place it within this model. This model is then tested against all these restrictions and will be accepted or declined.
Recently, the municipality had a pilot in which they wanted to show proof of concept for this idea. The selected area was a small business park, De Brinkhorst. With use of open data they were able to calculate and visualize the impact of the uploaded designs. The pilot was a succes and the idea will be further developed by Geodan [Geodan, 2016]. Screen-captures of

1 http://www.kadaster.nl/web/artikel/producten/3D-kaart-NL.htm
2 http://www.rotterdam.nl/links_rotterdam_3d
3 https://data.overheid.nl/data/dataset/3d-model-den-haag/resource/32f598ca-8368-483d-aa74-ecc50b9e604e

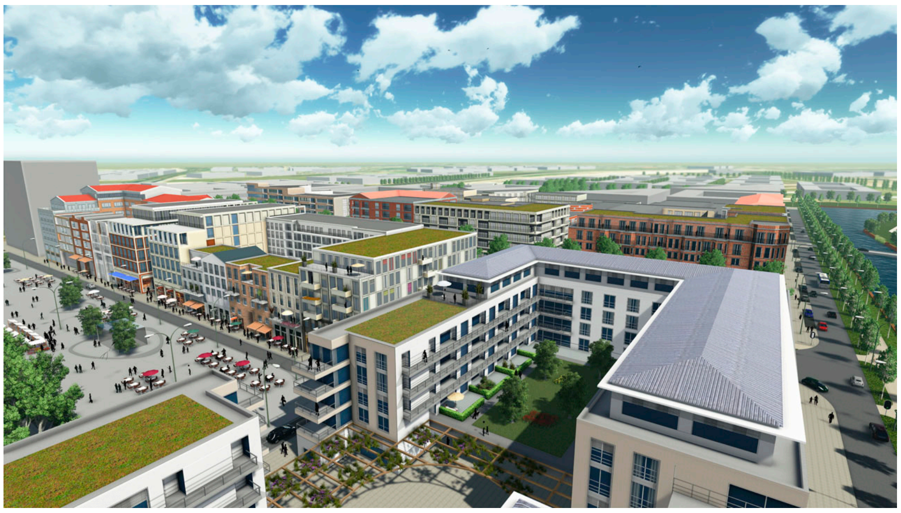


Figure 3.9: The BIM model of Leidsche Rijn Centrum. Because the model is so detailed, it can directly be used for visualization. (Image by Hfb [2016])

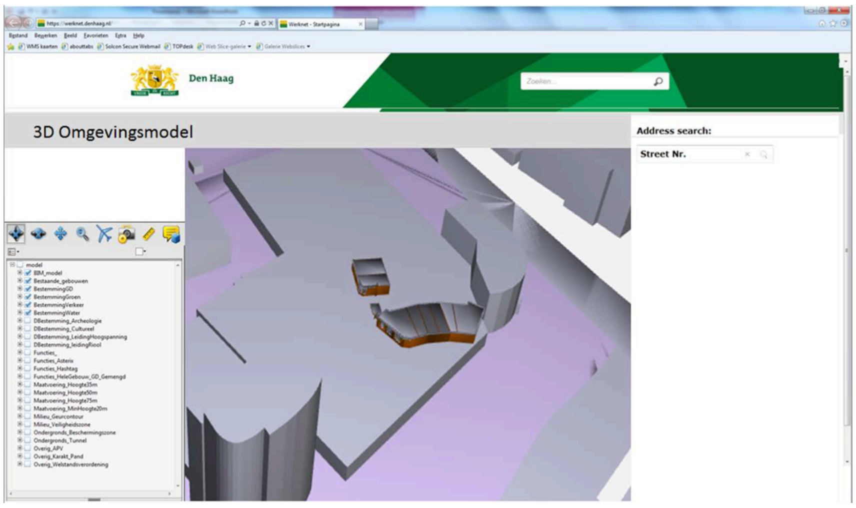
this pilot are shown in Figure 3.10a and 3.10b.

3.2.4 BGT IMGeo

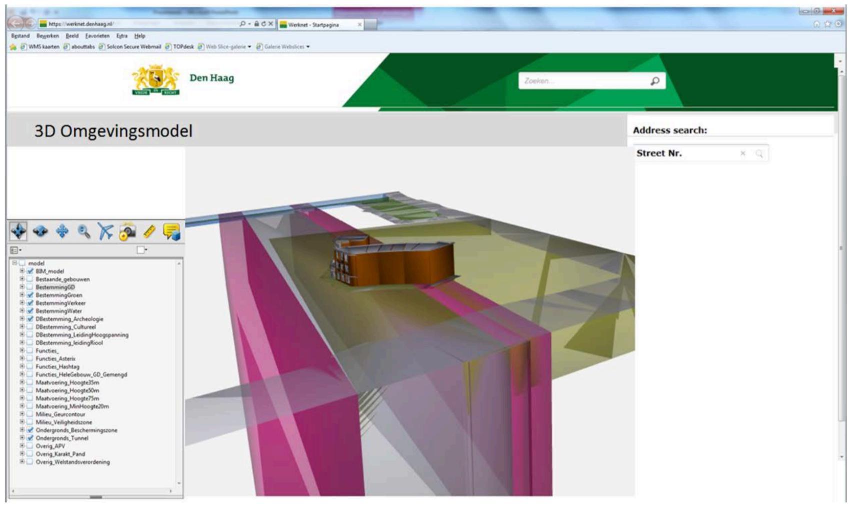
An other development that shows the trend of moving to 3D data is visible in the *BGT*. The information model used for the *BGT* is called: IMGeo and was especially developed to make it possible to store 3D geo-information [van den Brink et al., 2013a]. This was made possible because it has a complete integration with the international Open Geospatial Consortium (OGC)-standard CityGML, a standard for city and landscape models (this standard will be discussed in more detail in section 6.1). This standard makes it possible to include levels of detail in one model, e.g. a building can be included in the model as a polygon but also as a full 3D geometry. Because the information model is so flexible, it is relatively easy to make the step from 2D to 3D. It is therefore to be expected that this key register will be the first register in 3D. IMGeo will be discussed in more detail in Section 6.2.2.

3.3 CURRENT WORK ON THE DEVELOPMENT OF THE 3D BAG

Possibilities for a 3D version of the *BAG* are currently investigated on different levels. In may 2016, a working group was established by the Kadaster to identify the needs and opportunities to include 3D in the *BAG*. This is the first step, which may eventually lead to a nationwide 3D model of the *BAG*. In addition, efforts to create a 3D model of the *BAG* are made on the municipality level. In contrast to most 3D models as discussed previously, the main focus of these efforts is at unit level. This means that besides modeling the building itself, units within buildings are also taken into account



(a) The architect can upload and locate his model.



(b) The model is tested against the planning rules.

Figure 3.10: The pilot of the municipality of The Hague for interactive use of 3D within the planning phase.

3.3.1 Den Haag

The Department of Urban Development of the municipality of The Hague recognizes the potential of a 3D *BAG*. As such, they started a pilot for creating a 3D model of the *BAG* of their city. They developed a workflow with a 3D city model and the *BAG*+ as input, and thereby creating a 3D *BAG* (Figure 3.11). In this model, only units that are part of a building are stored, while the buildings itself are not stored.

In Figure 3.12 the partitioning workflow is shown for a single building with six units (A-F) and their floor levels (1-3). In order to find the floor height, the gutter height (h), included in the city model, is divided by the highest floor level found in related units. The floors that contain multiple objects are subsequently partitioned using a Voronoi diagram with the points of these units as input. Using the location of the points was in this test-case possible because the points of the units where located with a defined structure: in the middel of the unit seen from the road side. As already

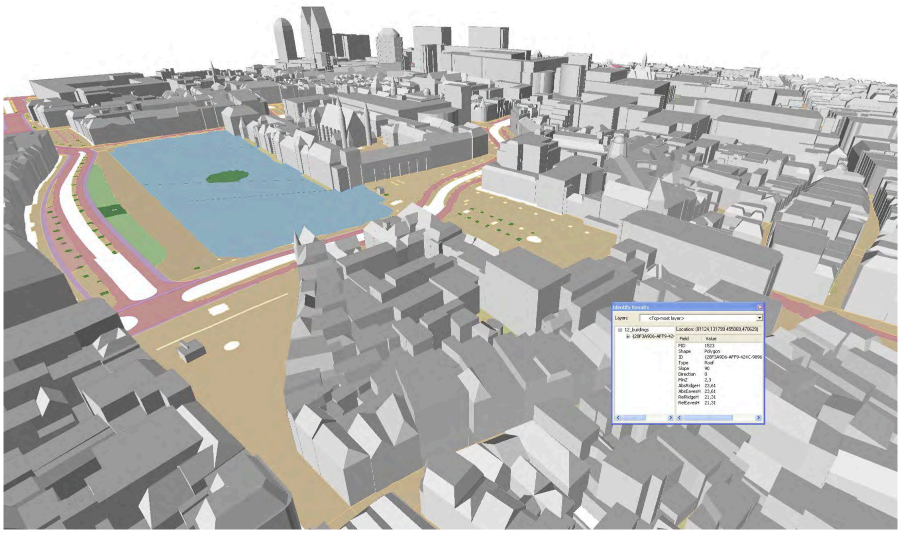


Figure 3.11: The 3D city model that formed the basis for the 3D *BAG* of The Hague.

discussed in Section 3.1, this is not always the case. These partitioned floor areas are then assigned to the units. To create the volumes, the floor areas are extruded with the floor height (Timo Erinkveld, personal communication, March 1, 2016).

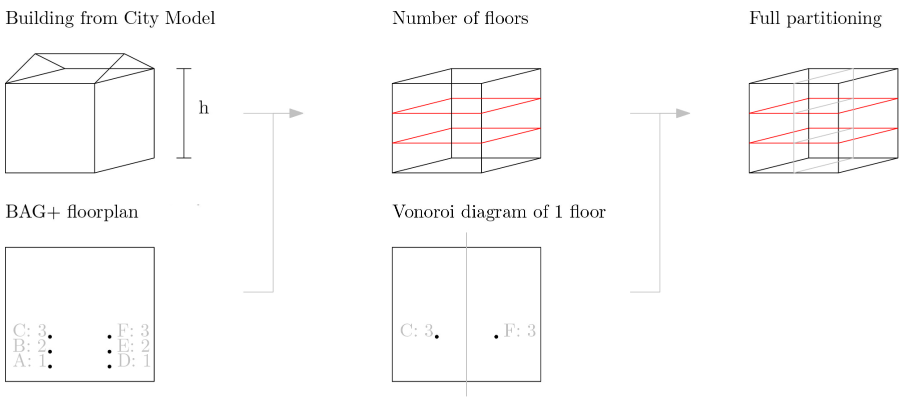


Figure 3.12: The partitioning workflow of a building with six addresses

This method showed the potential of using current available information to create a 3D *BAG*. However, improvement of this method is necessary for a better accuracy of the model and a wider range of applications of the model nationwide. The usage of the Voronoi diagram creates unrealistic partitions if the building configuration is not straightforward as shown in the introduction. In addition, the voronoi diagram does not take into account the floor area, which are known in the *BAG*. Another problem that occurs when using this method with the 'normal' *BAG* is that for apartment buildings, the points can lie on exactly the same coordinates, making the use of a Voronoi diagram impossible.

3.3.2 Rotterdam

The buildings in the current and future 3D city model will have a connection with the *BAG*. Additionally, research has focused on the way the placement of the point geometries of *BAG* units could help to create a 3D model in the future. The goal of this research was to find a logic structure of locating the points within the building, related to the location of the units in reality. As discussed in section 3.1.3, currently no guideline exists for the placement of the point geometries. Hence, recent research of the municipality focuses on creating the building models without the interior (Patrick Schmidt, personal communication, April 26, 2016). The placement of the objects within the buildings remains to be studied in future work.

3.4 MY PROPOSAL FOR THE 3D BAG

The trend in the previous chapters shows that there is interest in 3D information and a 3D model of the *BAG*. But there are multiple interpretations of how the 3D *BAG* should be defined. Experts on in this research field see the 3D *BAG* as a 3D model, but the inclusion of what kind of information and how this information should be modelled differs. In this section a vision statement is defined on how the spatial information within the key registers could be joined within one model. Thereafter, it is discussed how the 3D *BAG* geometry fits between this model is discussed next. The section finishes with several opportunities that will arise for the *BAG* when it is part of this complete 3D model.

3.4.1 One complete model

One of the most active research of the past years within the geomatics field has been the integration of spatial and spatiotemporal aspects together with thematic information from diverse application domains [Thomas H. Kolbe, Gerhard König, 2011]. Not only research has been done on the modeling of topographical features above ground, but also how to integrate the underground [Emgård and Zlatanova, 2007]. When all aspects are integrated, many problems that geographical information system (GIS) specialist face when combining different spatial datasets are solved.

In line with this vision, the municipality of Rotterdam has the long-term goal to create one complete model of their city. In this model all data that is used by the municipality is included in order to deliver their public tasks. The model will not only comprise all buildings and apartments, but also also trees, soil types, underground structures, roads and other topographical features (Figure 3.13). When establishing this model, there is no need to have multiple registers for different types of spatial information (Patrick Schmidt, personal communication, April 26, 2016).

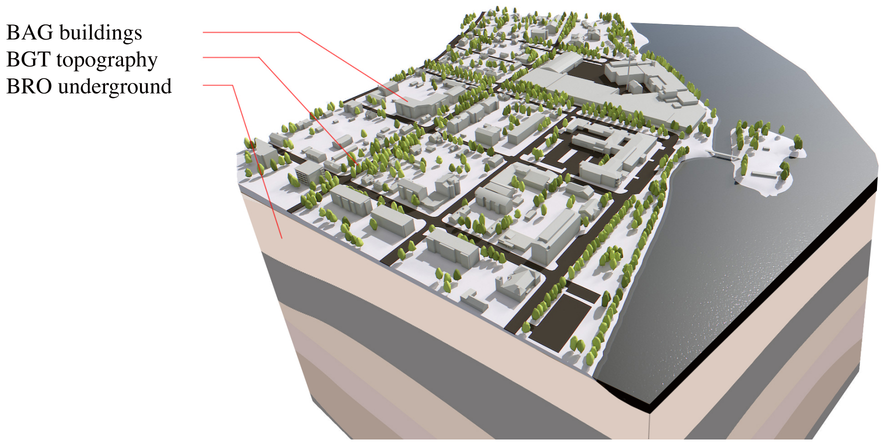


Figure 3.13: How the key registers could be joined in the future (Image adapted from Fodor [2015]).

3.4.2 How the 3D BAG geometries should be modelled to fit within this model

In order to achieve such a vision, all registers that contain spatial data, such as the *BGT*, the *BRO* and the *BGT*, should be joined in one complete model. As discusses in section 3.1, there are already a lot of problems and uncertainties exist in the geometry of the *BAG* dataset. This number of problems will only increase when the dataset is combined with other datasets. Therefore, the complete model should be in 3D, along with the geometry of the *BAG*. To exclude all special uncertainties in the *BAG*, also the units should be modelled in 3D. But this is not the main reason why the focus should lie on the units, another reason is that the units are the places where people stay. Moreover, these units are connected with other key registers such as the *WOZ*, the *Handelsregister* (commercial register), and the register for persons. All these registers can directly benefit from a 3D model.

The amount of details of the building models influences the applicability of model, i.e. for the determination of the average floor area in a neighborhood it is not necessary to include the windows in the model. However, in most cases it is easier to simplify a model than adding more detail. In an ideal model, the exterior of the buildings should be modelled as close to reality as possible. The same is the case for the shared areas within a buildings. Modeling the units themselves in more detail, e.g. including the interior, is not needed because this will raise new privacy issues. When the units and shared spaces are modelled, it will not only make spatial analyses more accurate, but will also be valuable for other use cases. For example, when the location of staircases, common spaces and entrances of apartments buildings are known, 3D spatial analyses functions can be used in emergency response situations, such as navigation and buffering [Lee and Zlatanova, 2008].

Since it will not be possible to create a completely detailed 3D model of every building, it should be possible to include different level of details concept for the *BAG* buildings and units (Figure 3.14). This proposed LoD concept makes it possible to model the current *BAG* data, but also to extent the data with 3D geometries. The order of the level of detail is based on the

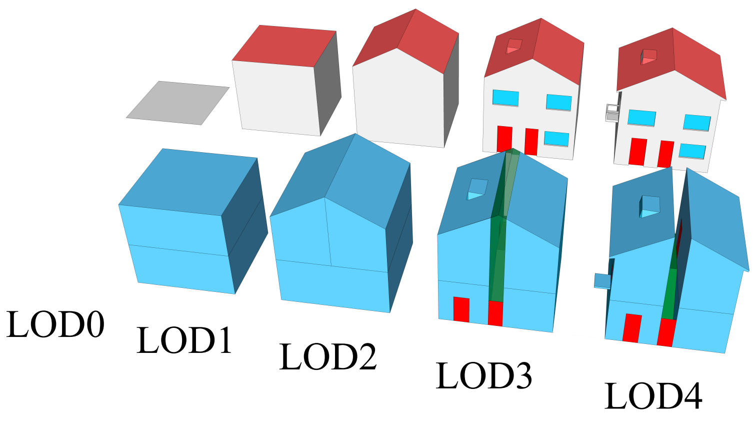


Figure 3.14: The proposed LoD concept makes a difference between the interior (objects) and the exterior of a building. The interior (in blue) will always follow the boundary of the exterior of the building.

type of data that is needed and the complexity of the creation of the model; for LOD0 the current data is sufficient, LOD1 and LOD2 need additional height information and LOD3 and LOD4 need detailed information about the indoor environment. For municipalities it would therefore be possible to gradually increase the LoD of the models when more data comes available. This LoD concept enables the storage of the current *BAG* but also the 3D geometries. The proposed LoD concept will include five levels for the units:

- LOD0: The units are part of the building and therefore do not have their own geometry.
- LOD1: The interior is divided by storeys. The units are related to the these storeys: the geometry of the storey is shared by the related units.
- LOD2: The units are modelled separately. For every storey of a unit, a solid is created. The aggregation of these solids form the total geometry of the unit. This enables to store unconnected spaces, such as storage areas, and separate storeys.
- LOD3: Shared areas are included as volumes and the location of the doors is modelled.
- LOD4: The shared spaces are modelled in detail and the location of the doors are included.

How this LoD concept can be stored within the current CityGML standard will be discussed in Chapter 6

3.4.3 The opportunities that will arise when the BAG geometry moves to 3D

A 3D model of the buildings and units will solve the spatial uncertainties of the *BAG* and will make it possible to include the *BAG* geometry within the complete model. In the previous subsection, a few use cases were already mentioned of how users of the *BAG* would benefit when it moves to a 3D representation. In this subsection more examples of how governmental organizations that use the *BAG* would benefit from the 3D model are discussed. Besides these governmental benefits, the market will also respond when the data is made publicly available. As such, private parties will find new use cases for the 3D model.

When creating an urban plan for a municipality, urban planners use functional maps of the project area. These maps provide vital information for the creation of new plans. Previously, these maps where made by labor-intensive fieldwork. By the introduction of the *BAG*, this process can be automated [Geonovum, 2015]. The 2D representation makes it still hard to interpreted the maps when units with different functions are overlapping. Hence, a 3D model would make it easier to interpeter information (Figure 3.15).

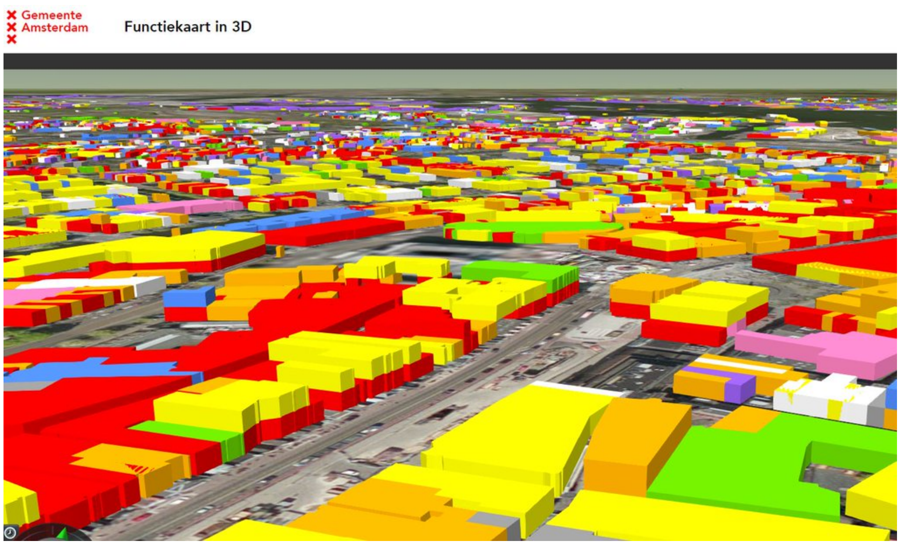


Figure 3.15: An experiment of the municipality of Amsterdam with the aim of creating a 3D functional map. Image by Gemeente Amsterdam [2016a].

During the planning phase, a 3D model can give better insight in the effect of the plan on individual addresses. Not the buildings are the main topic of these predictions, but the units where people stay. Topics with a 3D component (e.g. sunlight, noise pollution, particulate matter, line of sight and urban heat island effects) can be predicted predicted with a higher precision. This would make it possible to not only affect large scale, but also individual objects. The possibility to do direct predictions based on a detailed 3D model fits directly within the vision of the municipality of The Hague.

The tax authorities are also interested in the 3D *BAG*. When there is an accurate version of the 3D *BAG*, it is possible to detect fraud or errors in their data. In practice, units can have more storeys or more floor area than disclosed in their data. The value of property can therefore be calculated wrongly, which has direct influence on the tax. A 3D model of the *BAG* would make it possible to estimate the floor area in more detail, enabling the detection of inconsistencies between the data of the tax authorities and the *BAG*.

These applications are only a small selection of all possible applications that come along with the inclusion of the 3D geometries of the *BAG* in one complete model. In this section only direct opportunities of the 3D model for the municipalities and the tax authories were described. But also different levels of the government would benefit; e.g. flood models could predict more exactly which homes/businesses experience damage and which people have to leave their home. Moreover, the taxes on property-value can be calculated in more detail.

4 MY METHODOLOGY FOR CREATING 3D GEOMETRIES FOR THE BAG UNITS

In this chapter, my methodology for creating 3D geometries of the *BAG* units will be described. Based on this methodology, a prototype was implemented, as described in the next chapter. First, an overview is provided of the general workflow. This workflow can be divided in seven parts, which will be discussed separately. When relevant, related work is included in this discussion.

The goal of this methodology is to create approximate, but accurate 3D geometries for the *BAG* units. The geometry of the units will form the interior of the building geometry. This rough version includes volumes for every storey on which the living area of this unit is located, this volumes will be called the unit-storey (Figure 4.1). This is the second level of detail for the units, as proposed in Section 3.4. Since the information that is required to achieve a higher level of detail is not available, the inclusion of feature like shared spaces and building entrances is outside the scope of this research.

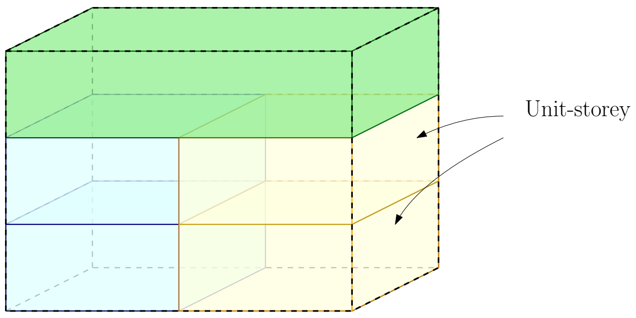


Figure 4.1: The goal of the workflow is to create the object storeys.

The workflow is divided into seven parts (Figure 4.2). The methodology is based on theory and practice. It is not possible to give an unambiguous result for every building since information is missing and inconclusive. Although for these cases it is possible to proceed the workflow based on assumptions, this will give an undesirable result in most cases. The uncertainties are discussed in every step and range from temporal difference between the height data, to not knowing with certainty if a building is above- or underground.

Because the *BAG* is used as a key register, the reliability of the result is important. In case there is no certainty about the result of the step, this will be noted and the feature will be excluded from further processing. In Figure 4.2, these excluded features are shown in the right column. Not every building will therefore follow the complete process. The number of buildings that are excluded in practice will be discussed in the following chapter.

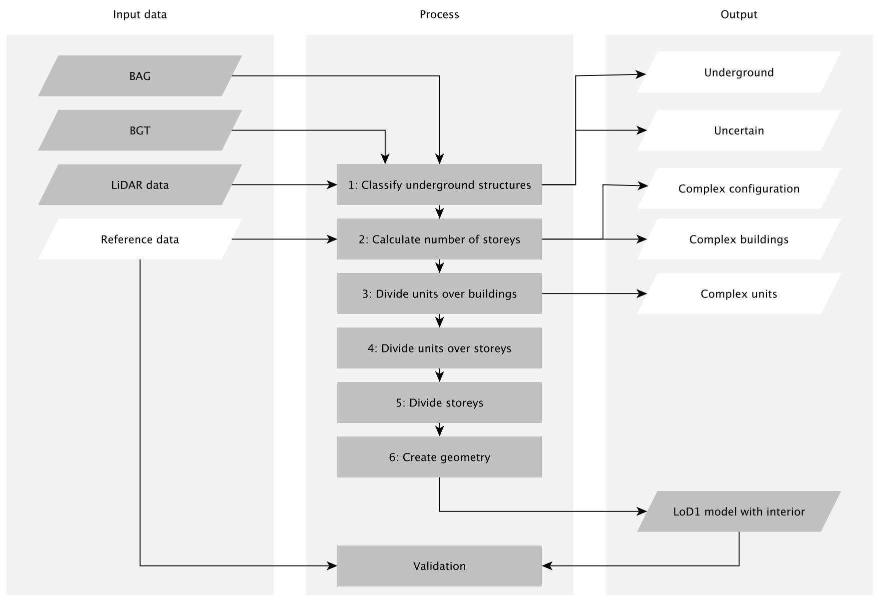


Figure 4.2: The general workflow. The six steps to create the model are shown in the process column. The created model is then validated with use of the reference dataset. This dataset is also used to determine missing parameters.

4.1 CLASSIFYING UNDERGROUND BUILDINGS

Underground structures are not registered as such in the *BAG*. For creating a 3D model, this is vital information. There is currently no related work for detecting underground structures available. As such two methods are combined. The first method, the LIDAR method, is based on combining the geometry of the *BAG* and LIDAR data. The second method, the BGT method, uses the differences between the *BAG* and the *BGT* (Figure 4.3) and can only detect if the building is above or underneath the ground level. By combing these methods, not only buildings that lie mostly or completely underground can be detected but also buildings that lie (partly) above ground. The buildings that give an inconclusive result are excluded from the workflow. At the end of this section these cases will be discussed.

4.1.1 Using density of LiDAR data to detect underground structures

Height data can be used to calculate the height of buildings and is therefor vital for creating a 3D model. The most common method for acquiring height data is with LIDAR. The acquisition cost for LIDAR data are relatively high. Hence, hardly any LIDAR datasets are freely available at the moment. The *Actueel Hoogtebestand Nederland* (*AHN*) is an exception. The *AHN* is a nationwide LIDAR data set (Figure 4.4). The data was captured for the Dutch

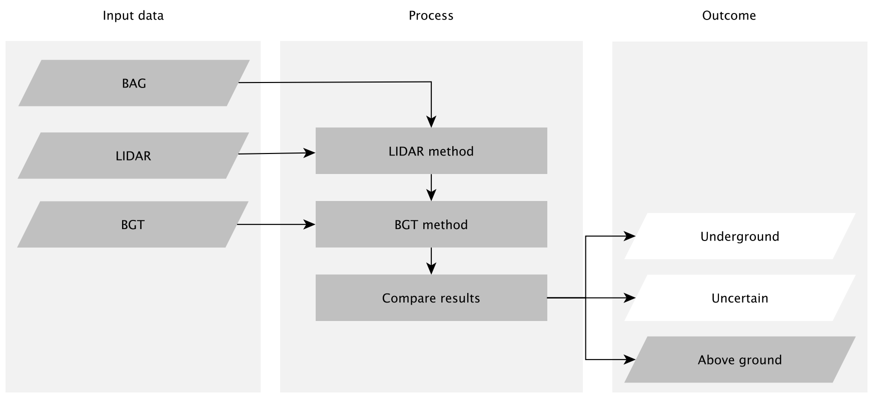


Figure 4.3: Workflow for classifying underground structures.

government for water and flood defense management[1]. Later, the government made it available as open data.

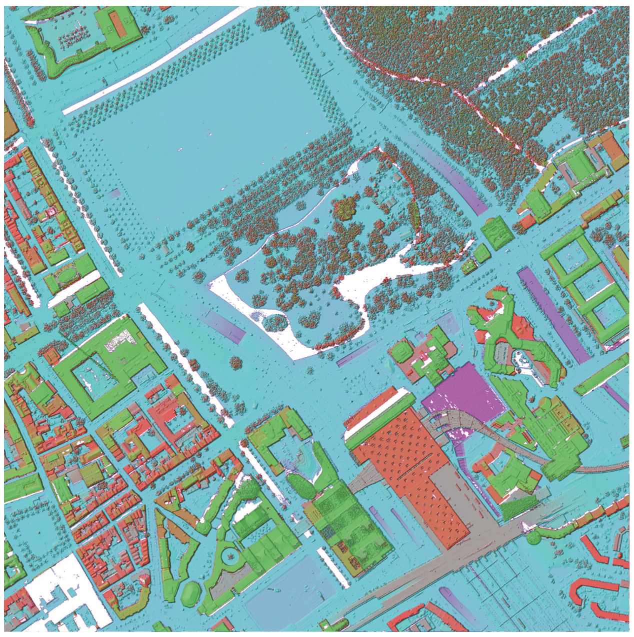


Figure 4.4: Section of the *AHN2* dataset.

The most recent version of the *AHN* is the *Actueel Hoogtebestand Nederland 3* (*AHN3*). The difference between the versions is the temporal aspect of capturing the data (*AHN2*: 2007-2012 and *AHN3*: 2014-2018) but also what is included in the data itself. The average point density is still 8 point per square meter, however in the latest version the points have more attributes. The data not only includes time of capturing, but also the classification of the individual points. As such, every point can easily be filtered: ground vegetation, building, and so on, whereas the previous version was only filtered by ground and non-ground points.

1 http://www.ahn.nl/

As the point can now be filtered based on the classification, it is possible to only take into account the points related to a building when calculating the height. In addition, the classification of points also provides the possibility to calculate the density of building points. This is useful because this will prevent errors when calculating the height of buildings. As shown in Figure 4.5, clipping the filtered point cloud by the geometry of an underground structure can result in a selection of points that belong to surrounding buildings. Calculating the height from these points will result in a positive building height, that is not the case in reality (figure 3.1 shows the same area). However, the number of points is very low, possibly meaning that it does not involve an above ground structure.

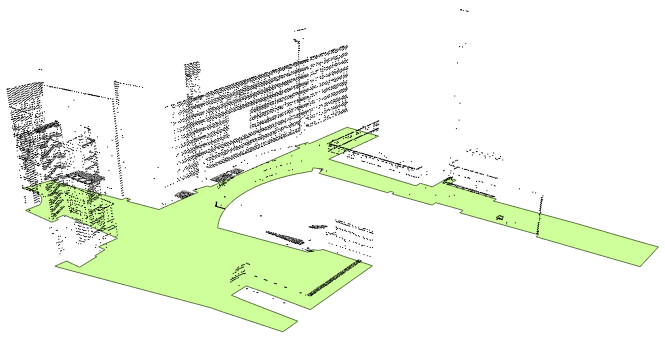


Figure 4.5: The points with the class building.

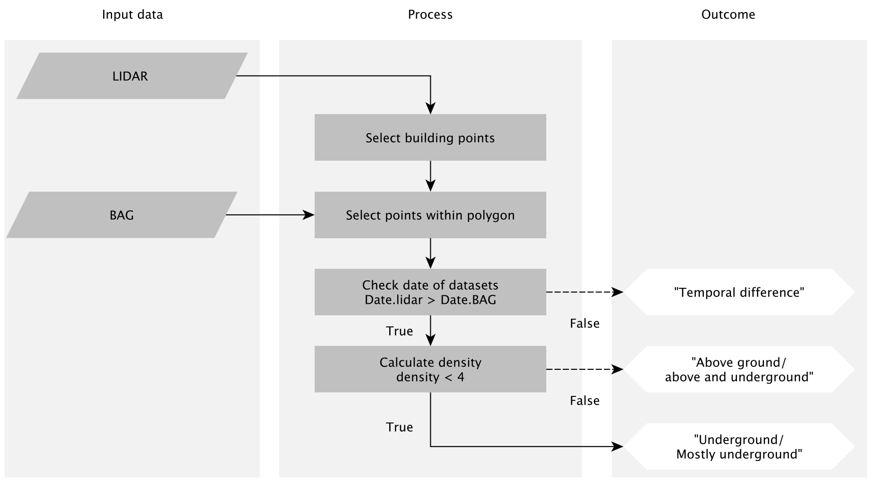


Figure 4.6: Workflow for density check.

The workflow for detecting underground structures (figure 4.6) starts with filtering the LIDAR point cloud, so that only building points are left. The building points are then clipped with every building polygon of the *BAG*. It can be the case that the capturing date of the *AHN3* is earlier than when the building was constructed, which will result in a wrong classification of the *BAG* object. This wrong classification is prevented by comparing the time of acquisition of the LIDAR points and the construction date of the object. When the acquisition date is before the construction of the building, it is not possible to use this method. But when this is not the case, the density for the building points within the building polygon is calculated. In situations where less than halve of the points are classified as buildings points, the building is classified as underground.

4.1.2 Using the *BGT* to detect underground structures

When the underground structure lies (partly) under another structure, the method discussed above will not be effective. Therefore, another method is needed to detect underground structures. The *BAG* geometries can overlap each other in some cases. The exact relation between these polygons can not be concluded using only the overlap of *BAG* geometries. As discussed in section 2.3, the *BGT* only includes buildings that are on ground level, while the *BAG* where the *BAG* includes all buildings. This information is used to exclude buildings buildings that are not located on ground level.

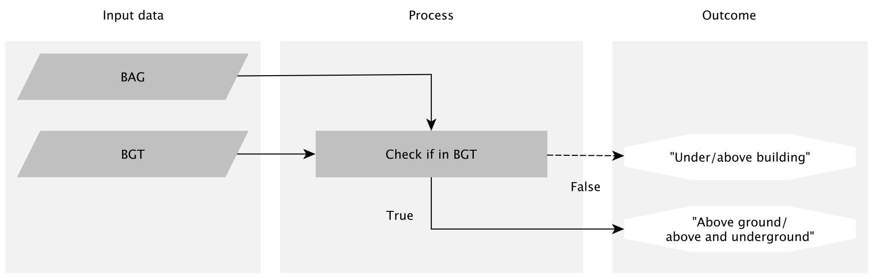


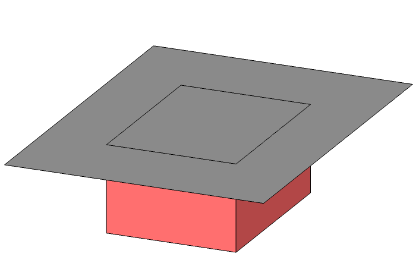
Figure 4.7: Workflow for *BGT* check.

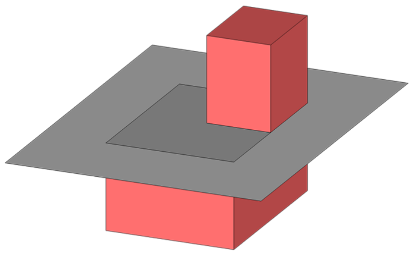
This method does not involve any geometrical computations. The identification numbers that are used in the *BAG* are also included in the *BGT*. When combining both datasets, buildings that are not included in the *BGT* are considered to be underneath ground level. When working with these datasets, it seemed that there is still a difference between the *BAG* and *BGT* that is not the result on this difference. Full certainty can therefore not be given solely based on this method.

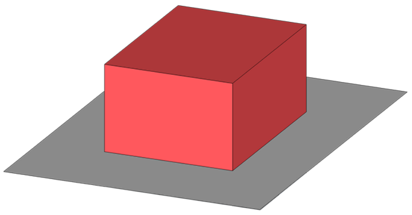
	Underground (LIDAR)	Above ground (LIDAR)
On ground level (BGT)	Uncertain	Above ground
Above/under building (BGT)	Underground	Uncertain

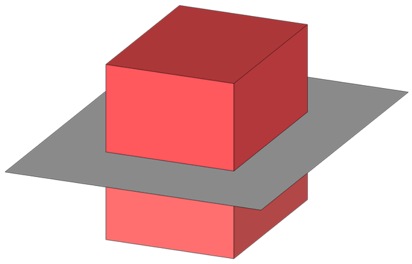
Table 4.1: Comparing the BGT and LIDAR methods to get the final result.

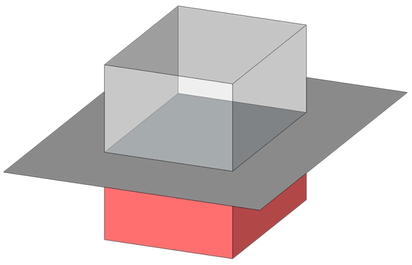
The method that uses the LIDAR data detects underground structures that are (mostly) underground. But when there is a structure on top, this method fails. The other method has the uncertainty that both dataset can differ in practice, without certainty why this is the case. Therefore the result of both methods are then compared to give a final result as shown in Table 4.1. The result of the classification will be: mostly underground for structures that are partly above ground but more than 50% underground, underground, above ground and uncertain. What should be mentioned is that buildings that are classified as above ground can also lie partly underground, i.e. of a basement. These uncertainties can not be detected with the methods. The result of this step is that every building in the workflow is classified, which will be further used in the process. The buildings that are classified as uncertain or mostly underground will not follow the general workflow because it is impossible to calculate the number of storeys for these cases (as discussed in 4.2.1).


(a) Underground.


(b) Mostly underground.


(c) Above ground.


(d) Both under- and above ground.


(e) Under building.

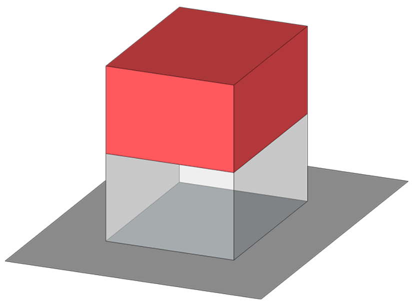

(f) Above building.

Figure 4.8: A selection of possible building configurations.

Figure	LIDAR	BGT	result
A	Underground	Above/under building	Underground
B	Underground	On ground level	Mostly underground
C	Above ground	On ground level	Above ground
D	Above ground	On ground level	Above ground
E	Above ground	Above/under building	Uncertain
F	Above ground	Above/under building	Uncertain

Table 4.2: Results of the LIDAR method and BGT method. As these results show, both methods do not give conclusive results. Two cases (A and B) can be classified with certainty, by combining both methods.

4.2 CALCULATING NUMBER OF STOREYS ABOVE GROUND

In this section the calculation of the number of storeys of a building is discussed. First the related work regarding this step is discussed and evaluated. Based on this related work a methodology is developed which will suit the desired result. The goal of this step is to calculate with certainty the number of storeys above ground using one value for the building height.

4.2.1 Related work for calculating the height and the number of storeys of buildings

A lot of research has been done on the creation of 3D city models from 2D data and LIDAR. Extracting building models was the main topic of this research. In 2003 Rottensteiner and Briese [2002] presented a method for automatically creating building models without the use of ground plans by segmentation the points cloud. Verma et al. [2006] included roof topology which enhanced the roof reconstruction. From the buildings, the focus then moved to creating complete city models from LIDAR and photogrammetry. The approach that Kada and McKinley [2009] presented produces LOD2 models from existing ground plans and airborne LIDAR. This is done by partitioning the building footprints and comparing the LIDAR data from this sections to standard shapes. This partitioning will also be discussed in Section 4.5.1 . Lafarge and Mallet [2012] presented a robust method for modeling cities which combined geometric primitives and mesh-patches created from LIDAR. Another robust method for creating a nationwide 3D dataset is given by Elberink et al. [2013]. They generated 2.5D and 3D representations using the 2D geometry of the TOP10NL, a nationwide topographical map, and the *AHN*. Per object of the TOP10NL, the LIDAR points are selected and processed using 3D modeling constraints of every object class. This ensures that, for example, roads and water are modelled as surfaces and buildings as 3D units. The buildings are modelled as solid blocks with flat roofs (LOD1).

While the LoD increases for the exterior of the building and other topographic features getting included in the models, the interior is not often part of these 3d city models. On a smaller scale there is done a lot of research on the creation of indoor 3D models. These are often created with use of indoor point clouds, i.e. the research of [Hong et al., 2015] and Díaz-Vilariño et al. [2015]. Because the used data is not available on a large scale, it is not possible to extent this research to the desired scale. But efforts are made to include indoor information to city models based on exterior information, such as calculating the number of storeys from the building height. In practice, the most common way of doing this is to divide the height of the building, extracted from LIDAR, by a fixed storey height. The value that is used for dividing the building height is often picked around 3.0, but it differs. Same as the value that is used for the maximum building height, the median or maximum of the clipped point cloud. Even the OGC standard CityGML 2.0, which includes the LoD concept, gives no clarity about how the building height for LOD1 models should be measured [Biljecki et al., 2016a].

I also investigated if it is possible to use the floor area included in the *BAG* to calculate the number of storeys of a building. This would provide the possibility to detect whether a building has a basement. But after experiments and comparing the results with a reference data set, the results were poor. The connection between the area of the building polygon and the number of storeys did not relate to the area of the building units. It was expected that there is a difference because the *BAG* units include the nett floor area and the building would represent the gross floor area. There are multiple tables available to convert these areas, multiplying the nett floor area by the right factor would indicate the gross floor area. But when checking multiple buildings manually, this factor can differ in reality from 1.2 to more than 4. Because of this difference, it is not possible to use the nett floor area despite available the conversion tables. Using LIDAR data gave much better results.

Extracting the building heights by combing LIDAR and a 2D geometry has proven to be a robust method. The height that is chosen for creating LOD1 models is subject to different interpretations and depends on the purpose of the data and context [Biljecki et al., 2016b]. Also the calculation of the number of storeys from a height value is an issue that is faced by different researchers in the 3D geoinformation field ([Boeters et al., 2015], [Alahmadi et al., 2016]). The research on the creation of an LOD1 model without additional height data described by Biljecki et al. [2017], shows that attributes such as the function of the building and the age have a slight influence on the average storey height. More research on this topic and also comparing the average storey heights of different cities will make it possible to predict the height parameters more accurate. For this research a fixed parameters for the average floor height will be used calculated from reference data. But in the future more parameters could be included. Because I use the building height to calculate the number of storeys, I will use the top 90th percentile of the points for the building height.

This research focusses on creating the geometries of the *BAG* units for a LOD1 model of the *BAG* buildings. Meaning that the building polygon will get one height for the complete building, calculated from the the points that lie within the building polygon. But using a LOD1 model will also mean that all storeys will have the same floor area. Using this method will cause problems when building polygons overlap (Section 4.2.2) or when there is a big height difference within one building (Section 4.2.4). Detecting these cases is therefore an important step of the developed workflow (Figure 4.9).

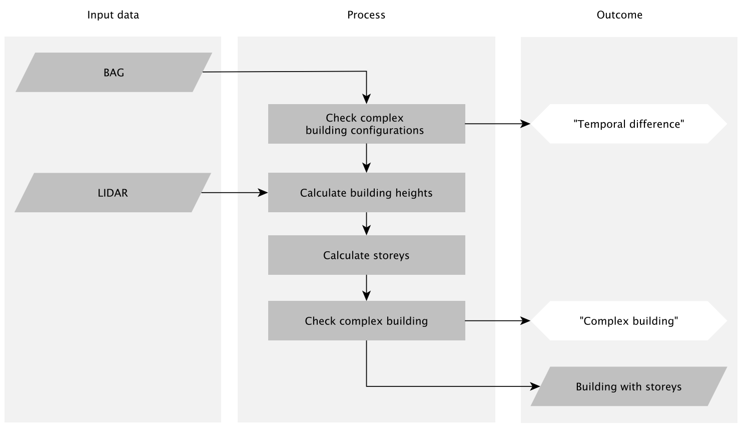


Figure 4.9: Workflow for building the number of storeys.

4.2.2 Exclude complex building configurations

When building polygons overlap it is unsure to which building the points of the point cloud belong to (Figure 4.10). When a building is partly underneath another building, the calculated median from the point cloud will always be higher than in reality. To find overlapping buildings, the geometry of the *BAG* is used. Most building polygons in the *BAG* do not overlap each other, except of minor overlaps due to accuracy errors in the geometry and if one of the buildings is underground. However, in cases with considerable overlap in which both buildings are above ground, there will be an overlap that influences the calculated height of the building. Calculating the number of storeys of the building will give inaccurate results. Therefore these buildings will be excluded from the workflow.

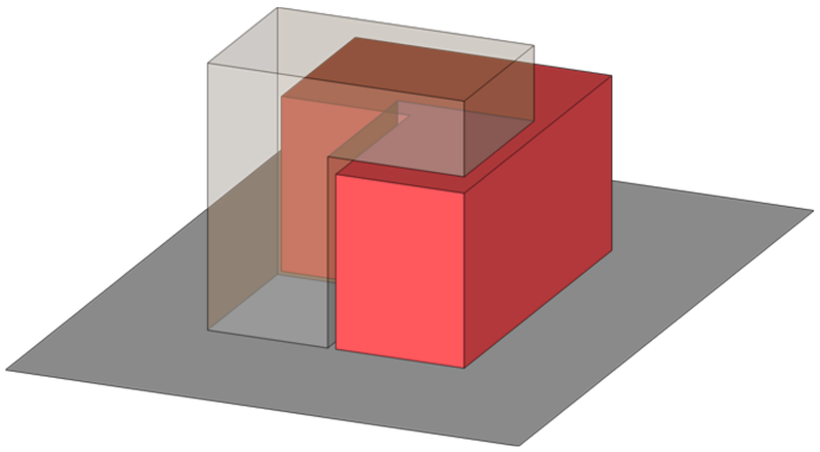


Figure 4.10: Overlap above ground.

4.2.3 Calculating the height of a building

For calculating the height of a building we need two heights, namely the height of the ground (Z_0) and the height of the building itself (Z_h). Both heights are extracted from the *AHN3* point cloud data and are subsequently subtracted.

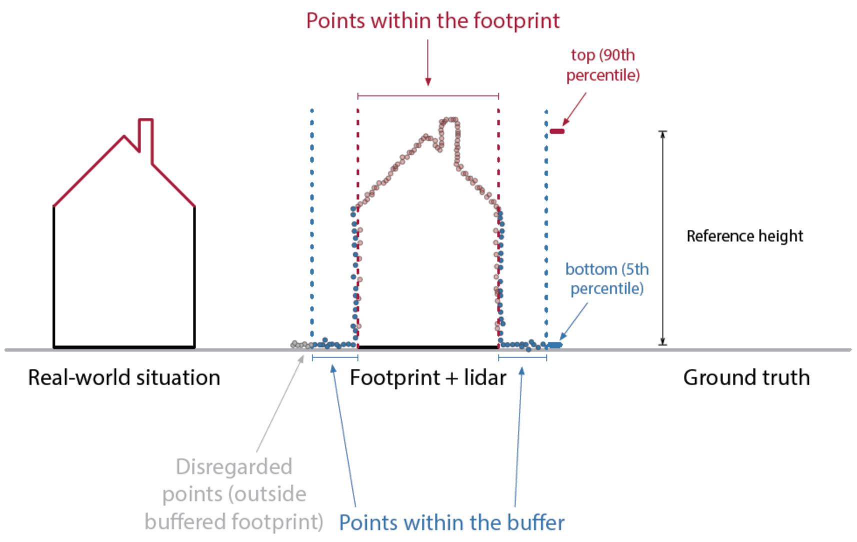


Figure 4.11: The determination of the heights of a building. Image by Biljecki et al. [2017].

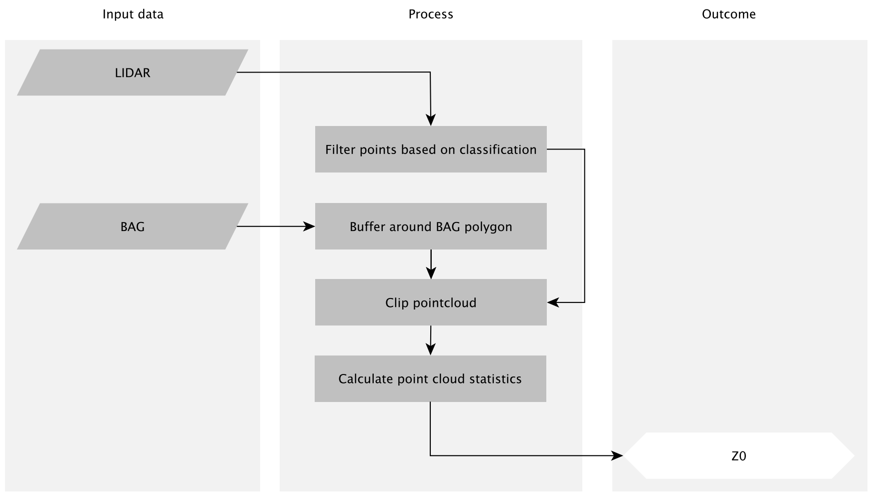

Input data
Process
Outcome
LIDAR
Filter points based on classification
BAG
Buffer around BAG polygon
Clip pointcloud
Calculate point cloud statistics
Z0

Figure 4.12: Workflow for Z_0.

To extract the Z_0 value, filtered points within a one meter of the building are used (Figure 4.11). The points within 1 meter of the buildings are used to calculate this value. Similar to the previous step of the workflow, these points are filtered based on the classification. This prevents that points belonging to other classes influence the result, e.g. points that belong to trees can create a significant error when not excluded. The Z_0 is determined by calculating the Z-median of these points.

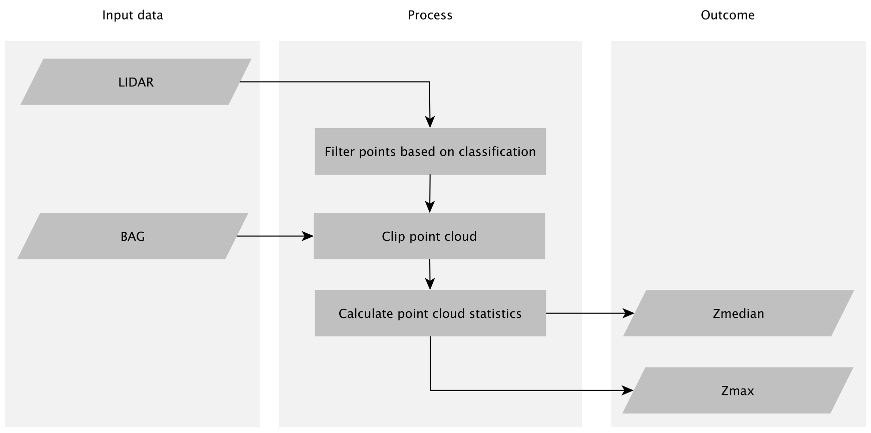

Input data
Process
Outcome
LIDAR
Filter points based on classification
BAG
Clip point cloud
Calculate point cloud statistics
Zmedian
Zmax

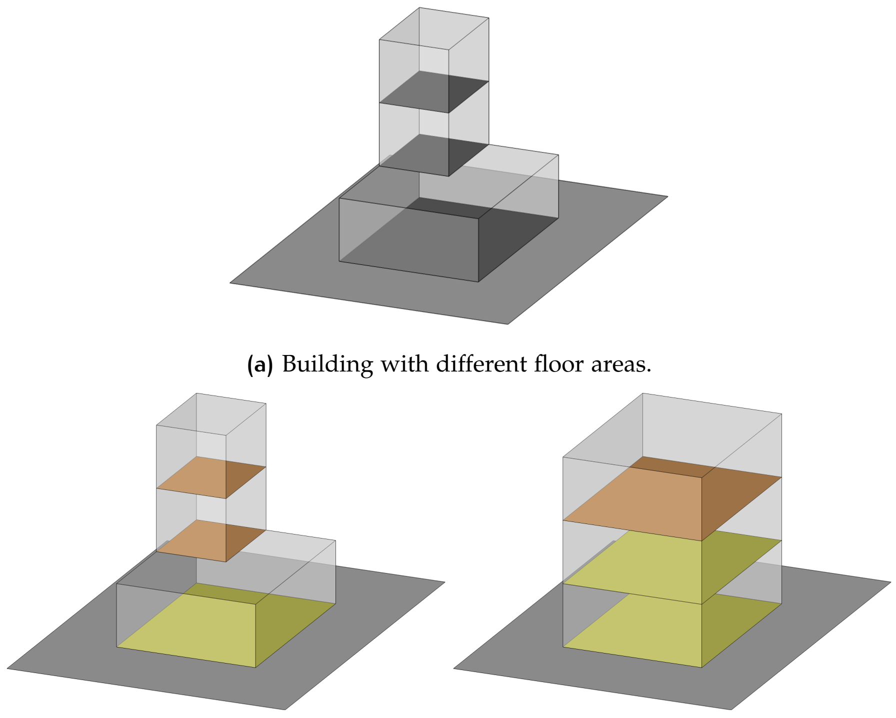
Figure 4.13: Workflow for Z_h.

The Z_h (height) is calculated with a similar proces, but then by filtering the point cloud on building points and only the points beloning to the building (Figure 4.13). Instead of only calculating the median of the points, also the maximum height is extracted. The number of storeys is calculated by dividing the Z_{max} by a parameter which will be determined from averaging the storey height value of a reference dataset. The value is rounded to get an integer.

4.2.4 Detect complex buildings.

The floor area will later be used for placing the units within the building. To make this possible within a LOD1 model, all storeys will be considered to have roughly the same size. Figure 4.14 shows the error that will occur when the LOD1 model is not sufficiently in modelling the reality. Dividing the floor area is only necessary when a building has multiple units, in case of zero or one unit, this error will not occur. Therefore, complex buildings (buildings that have big differences within the floor area) with multiple related units will be excluded from the workflow. Two types of complex buildings will be discussed.

The first type of complex buildings constitutes buildings with big variation within the building height. Although it is still possible to calculate the number of storeys, the calculated floor area will be completely different due to the different floors.



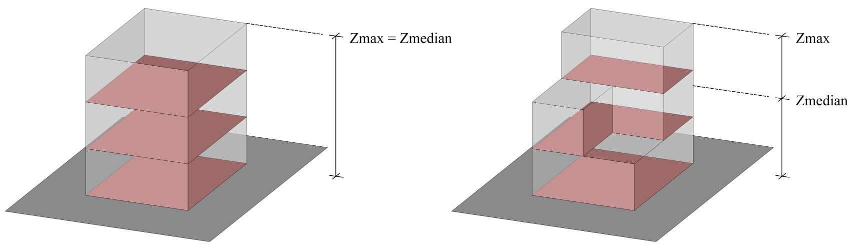
(a) Building with different floor areas.

(b) Placement of units in reality.

(c) Result when using LoD1.

Figure 4.14: In this example two units, A (yellow) and B (orange), are placed within building A). The area size of A is 2 times the area size of B. Figure 4.14b shows the placement of objects in reality, whereas figure 4.14c shows the division of floors when considering that all floor areas are the same.

In order to find out which buildings face this problem and, as such, should be detected, the median and maximum hight values are compared (Figure 4.15). When the difference between these values is above a certain threshold, the building is expected to be complex and placing units will not be possible. The threshold is set at a level in which buildings with a common, angled roof are not excluded from the proces.



(a) Simple building.

(b) Complex building.

Figure 4.15: Buildings in which there is not a large difference between the Zmax and Zmedian are considered to be simple buildings. When there is a larger difference, using LoD1 for modeling the building is insufficient.

Another type of complex buildings are buildings that have a large difference between the footprint and the outer perimeter (Figure 4.16). This will result in a similar problem as the cases discussed previously when placing the units. To detect this type of complex buildings, the geometry of the *BAG* is compared with the geometry of the *BGT*. As discussed in Section 2.3, a difference between these geometries suggests an overhang or understructure. As it is uncertain which floors are part of which geometry, the difference cannot be used for placing the units. In cases where the geometries are (roughly) the same, the building will not be excluded.

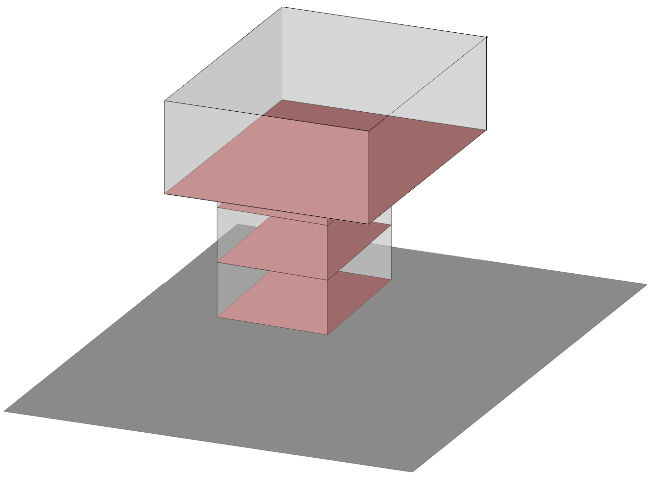


Figure 4.16: Complexity because of the difference between the outer perimeter and the footprint.

Theoretically, there can be a case in which the created methods to detect complex buildings are not sufficient. It is possible that the geometry of the *BGT* and the *BAG* is similar, but that within the building there is a large difference between the storeys. These buildings are not common in practice, but it is important to notice this uncertainty.

The result of this step of the workflow is that complex buildings and buildings with complex building configurations are excluded from the workflow. For all other building the number of storeys above ground is calculated. As previously discussed, the use of only one height parameter for calculating the number of storeys is not ideal. For building with high or low ceilings, the result may not be accurate, especially when a building has a lot of storeys. The result will be compared to a reference set to know the accuracy of this step. Also, this method is unable to detect basements or storeys that are partly underground. In cities where this is common, i.e. Amsterdam, this is a serieus drawback.

4.3 DIVIDE UNITS OVER BUILDINGS

For every unit within the *BAG*, the related building is included. There are two different cases possible: a building has multiple units or one unit can lie in multiple buildings. In very rare cases, both combinations can be present at the same time (in the test area of The Hague this occurred once). Before the buildings can be divided into multiple units, the units should be divided over the buildings to which they are related. As earlier discussed, the relation between a unit and a building is stored in the *BAG* geometrically and by attribute. The latter is used because in practice there are problems with locating the points as discussed in section 2.2.1. Further in the proces, the nett floor area of the units in combination with the combined nett floor area of all units is used for dividing the building. Therefore, this total floor area is also calculated.

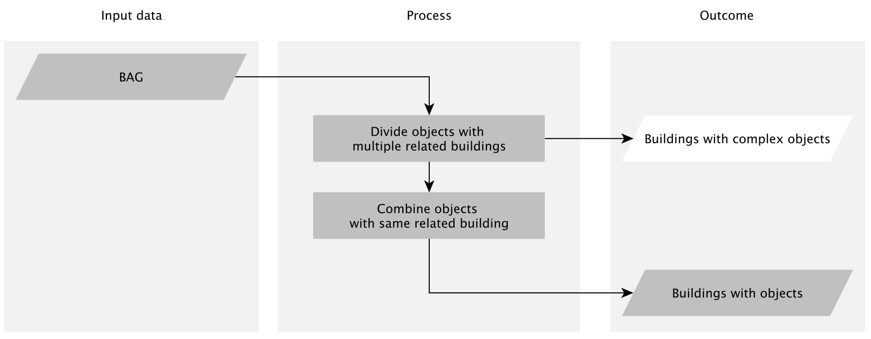

Input data
Process
Outcome
BAG
Divide objects with multiple related buildings
Buildings with complex objects
Combine objects with same related building
Buildings with objects

Figure 4.17: Workflow

4.3.1 Assign units to multiple related buildings

Although is not very common in reality, it can be the case that one unit has multiple related buildings (this is the case for only 0.6% of the units in the city center of The Hague). In order to divide the surface in a fair way, a default area is introduced. The default area of a building is calculated by multiplying the number of storeys by the floor area of the *BAG* geometry. This is also the reason why complex building are excluded in the previous step. The ratio between these calculated floor areas of the buildings is used to divide the nett floor area of the units (Figure 4.18). When an object relates to a small building and a large building, the latter will get a larger portion of the net floor area of the object than the small building.

The division of the units (and their nett floor area) over multiple buildings using the default area are based on the assumption that there is a correlation between the total floor area of buildings and the nett floor area of the object. However, this will not necessarily be the case. When there are multiple of these cases and they relate to the same building, the division based on this default area will be even more speculative. Therefore, the results for the partitioning of the buildings with units that have multiple related buildings may not be accurate.

4.3.2 Combining multiple units within one building

One building can have multiple units. This is a very common case in reality. All units that have the same related building belong together. The sum of the nett floor area of theses units is the total net floor area of the building (Figure 4.19).

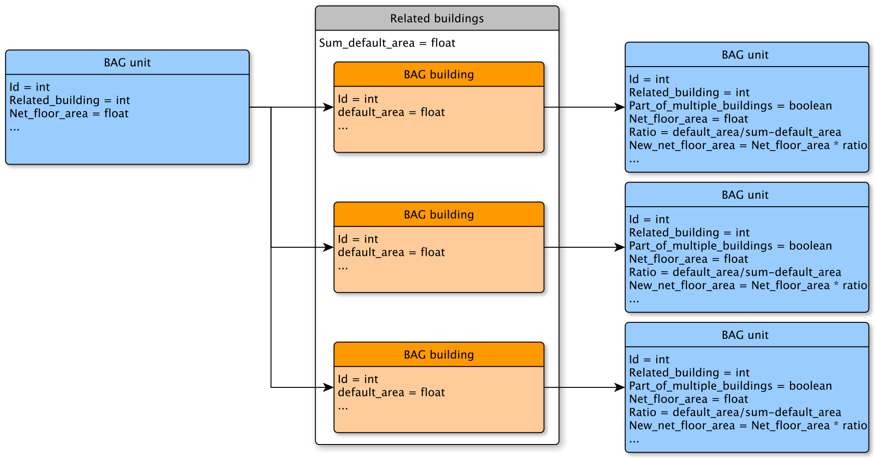


Figure 4.18: Single unit within multiple buildings. In this case the unit is duplicated and the the nett floor area is divided over these duplicates.

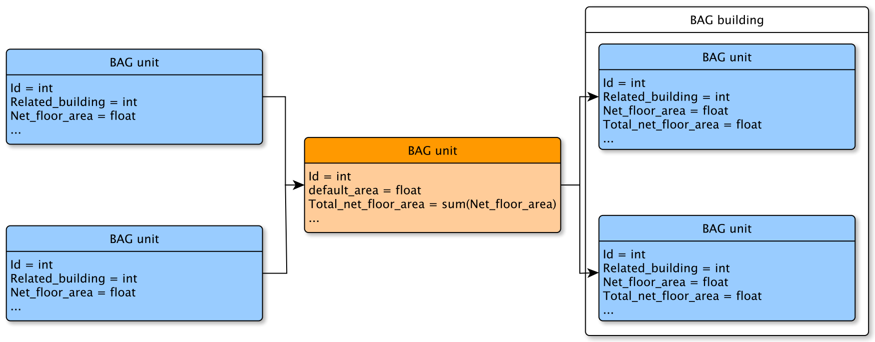


Figure 4.19: Single building with multiple units.

As a consequence, the related units for every building are included. The combined net floor area of these units is calculated and stored. However, only the first case provides certainty about the total net floor area. As such, there should also be a distinction between the two cases. In the latter, further division of the buildings using these divided units will give undesirable results and therefore these buildings are excluded from the workflow. It is not expected that this step will influence the accuracy of the methodology.

4.4 DIVIDE UNITS OVER STOREYS

The next steps in the workflow are focus on placing the units within a building. There is no additional open information available that will provide certainty about the results of these steps. Placing the units will therefore be largely based on logic, heuristics and assumptions.

Firstly, an ordering is made for the units based on attributes that are included in the *BAG*. Then sections are created based on the additions and letters of the house numbers. The last step is to divide the units over these sections and to calculate the floor level for the units. The consequence of these steps will be storeys containing the related units. In the objects the occupancy of the floor level by the object will also be included.

4.4.1 Ordering the units

The ordering of units is based on three different criteria, namely the number addition, the function of the units, and the net floor area of the units or the numbering itself in cases where the first two criteria are not sufficient (Figure 4.20). The number addition of units will give the most certainty, followed by the function. The house number itself will not provide much certainty, especially in cases with only two units within one building.

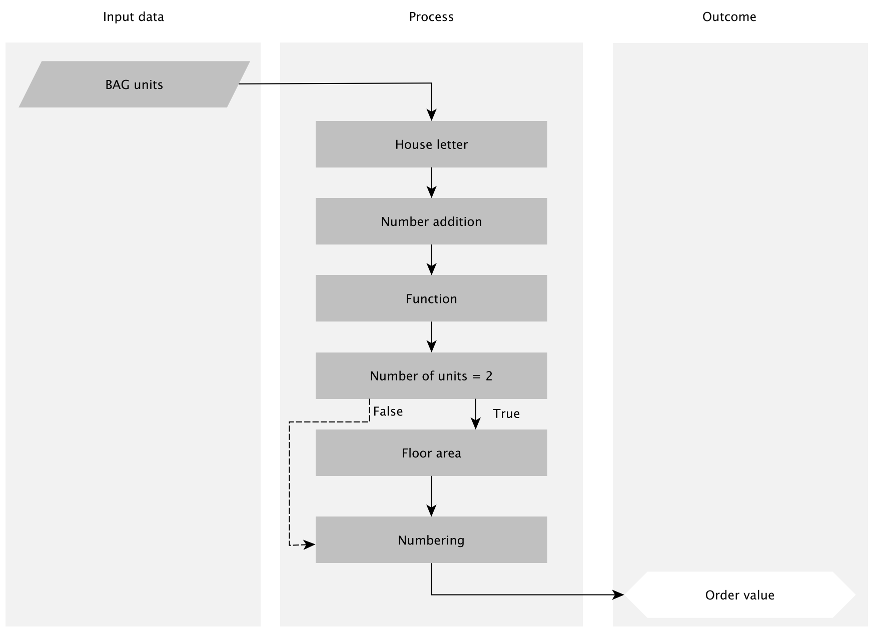
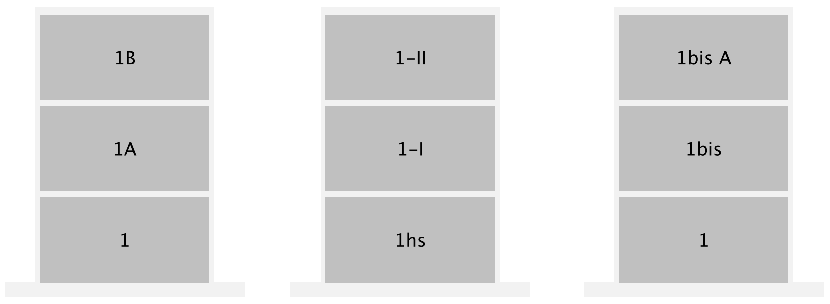


Figure 4.20: The ordering of the units.

Ordering based on number additions

In situations with multiple units within one building, the units can have either a unique house number, or an addition. In The latter case provides reasonable certainty about the order of the units regarding the floor levels. One problem with the number additions is that it can vary between municipalities, i.e. in the city of Amsterdam often a latin number is used as addition (Figure 4.21b) , while in the city of Utrecht the addition *bis* (Figure 4.21c) is often used. In general, the addition of simple letters in the order of the alphabet (Figure 4.21a) is most commonly used.



(a) Letters. (b) Latin additions. (c) Other additions.

Figure 4.21: Possible additions that indicate the location of units within buildings. The first case is most common in general, but municipalities can have their own ways i.e. Amsterdam (Figure 4.21b and Utrecht 4.21c).

Ordering based on function

In cases in which the addition of numbers to order units is not available, a ordering based on function will be tried. This only works when there is a difference between the function of the units. As discussed in Section 2.2.1, a unit can have eleven different intended functions. Not all these functions will be combined in practice, e.g., a building with a residential unit and a unit with a cell function will not be very common. In contrast, the combination of a store and residential function is quite common. In practice, in most cases the residential object will lie above the store object because the entrance of a store is on the ground level. This assumption will be used to order the units.

Ordering based on size and numbering

When the previous criteria are still are still not fulfilling the ordering of units, the floor area and the numbering itself are used. In situations with only two objects within the buidling, the building is commonly divided into a downstairs and upstairs apartment in the Netherlands (in Dutch it is called an *Beneden/bovenwoning*). The report about house prices provided information about the floor area [Visser and Van Dam, 2006]. The average floor area of the upstairs apartments is around 10 square meters larger than the area of the downstairs apartments. In cases with more than two units, no further information is available that can be used to order the units. As such, the standard house number is used.

4.4.2 Placing the units on floor levels

After ordering, the floor levels will be calculated for the units. This will be done using the following steps: first create sections with the same building and additions, calculate the number of storeys in the section, divide the units over the section floors and at last add the real floor level (Figure 4.22 and 4.23). These steps will be discussed in more detail in de following sections.

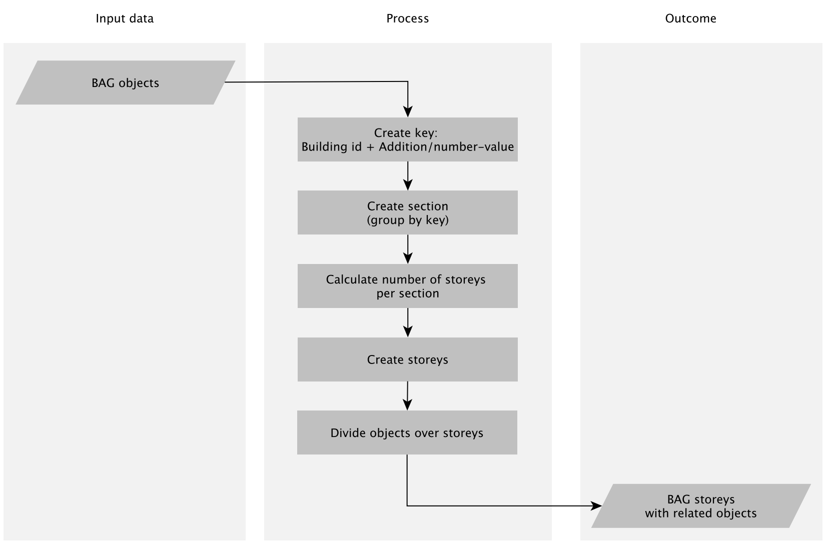
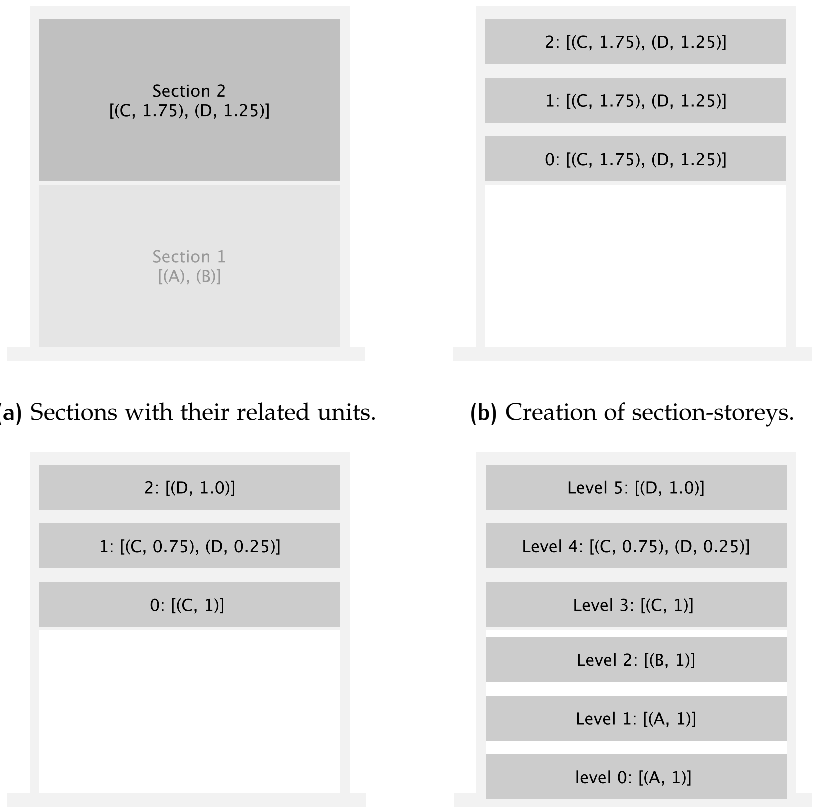


Figure 4.22: The workflow for placing the units on storeys.



(a) Sections with their related units.

(b) Creation of section-storeys.

(c) Divide units over storeys.

(d) Add floor level.

Figure 4.23: The order of the workflow to place units on the storeys.

Creating sections

Additions almost always indicate the storey on which the units are located: a unit with a different addition will not be located on the same floor. However, in cases with multiple objects containing different numbers with the same additions, they will lie on the same floor. Therefore, sections are created first (Figure 4.23a). Sections are groups of units that are considered to be located on the same floor, based on additions. To use the areas that are included in the *BAG* units, only ratios between the areas are used. First, the `Unit-building-ratio` is calculated. This is the results of dividing the *Net-floor-area* of the unit by the sum of the `Net-floor-area` of all units that have the same related building. This value indicaties how much of the entire building is occupied by one unit. By multiplying this value by the number of storeys of a building, the `Unit-storey-ratio` is retrieved. This indicates how many storeys a unit occupies. This is used to calculate the number of storeys in a section. In case there are no additions, the section will contain all related units. In case that the sections fill up a storey for less than 25%, the sections is merged with the other sections. This prevents that single, small units occupy complete storeys which can cause errors because the other units will not fit within the building.

Create section floors and divide units

The section will now be divided into section floors. Because I assume that all building floors are the same, the `Unit-storey-ratio` indicates the amount a unit fills a storey. The sum of this value for all related units corresponds to the number of storeys (Figure 4.23b). When the number of section floors are calculated, the `Unit-storey-ratio` will be recalculated to fit exactly with the sections. As s section will not always fill up exactly a round number of storeys, this is a necessary step. It is not yet clear where the units lie within these section floor. Two methods will be used to place the units on the storeys.

In case there are additions or house letters used within the building (and the section), all units with the same addition will also have the same storeys. All the section storeys will therefore be divided with the `Unit-storey-ratio` between these units. An example is shown in figure 4.24.

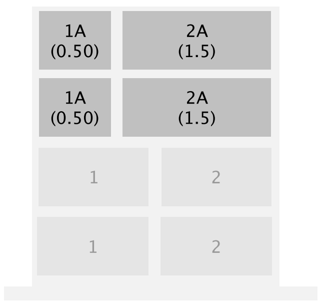


Figure 4.24: The units 1A and 2A are part of the same section. The sum of the `Unit-storey-ratio` is 2.0, meaning that there are two section floors. The ratio of the units is used to divide both section floors.

When no additions or letters are present, the ordering of the units is used to fill up the entire section. The `Unit-storey-ratio` is used to fill up these section storeys from the bottom. The starting point is the end point of the previous object (Figure 4.25). It is possible that an object lies on multiple floors. These units are divided and the sub-ratio for the floors are calculated (Figure 4.23c). The sum of these sub-ratios of the units that lie on the same floor will always be 1.0. The last step is to determine the real floor level by combing all the floor-sections of all sections (Figure 4.23d).

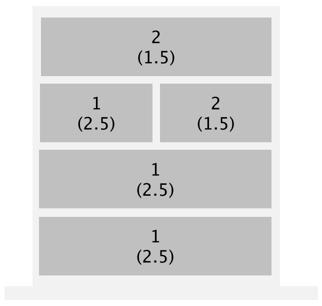


Figure 4.25: In case there are no additions, the units are filled up using the order defined in the previous section.

In practice, the ratios will not always count up to one storey which can result in very small areas. In cases where the units lie on multiple floors, it will be checked whether the difference between these multiple floors is limited. The storey ratio of one floor will be divided against the storey ratio of the previous floor. If the outcome is larger than a certain threshold value, the area of the first part of the object will be removed. On the other hand, if the outcome is lower than the threshold, the last part will be removed (Figure 4.26).

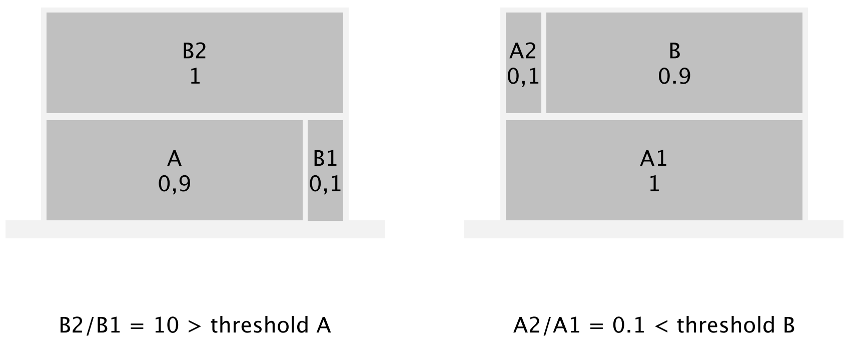


Figure 4.26: A threshold is set to prevent small areas in the net floor area ratios, which can otherwise cause inaccuracies.

The steps for ordering the units and using the ratios of the floor area to place the units on floor levels are largely based on assumptions. As a consequence, his is the most uncertain part of the entire workflow. The ordering of units will be less certain from top down: number additions are most certain, whereas the ordering by numbering will be less certain in case there are only two units. The order of the units will highly influence the placement of the units. To improve the accuracy of the model, more information about the location of the units is desirable. As discussed in 2.2.2, some municipalities already included information about the floor level of the units in their *BAG+*. In that case the uncertainties of these steps are removed, thereby improving the accuracy of the model tremendously. Unfortunately, this information is not publicly available and, therefore, will not be used in this research.

4.5 DIVIDE THE STOREYS TO CREATE THE UNIT-STOREYS

In situations that involve multiple units on one floor, the storey has to be divided. Different ways for partitioning polygons exist. Hence, this section will firstly provide an overview of the related work. This will be followed by the partitioning method especially created for this research.

4.5.1 Related work for area partitioning

There are different methods for partitioning polygons. I divided them into three categories: The *Area Partitioning Problem*, the voronoi diagram and data driven approaches. Although they all achieve a partitioning of an area, their respective approaches differ. The main differences between these three methods is the input data used to create the partitioning (Figure 4.27). Multiple criteria should be taken into account for choosing the desired partitioning method. Firstly, the method should be able to partition all simple polygons and polygons with holes. In addition, it should also be possible to do the partition based on an area parameter. Furthermore, it should give a realistic partition considering it represents a floor plan. Finally, the last criteria is that there is an robust implementation available.

Partitioning Problem

The definition that Bast and Hert [2000] use for the problem of partitioning is the problem for dividing a simple polygon into a given number of smaller connected polygons that have the same size. This problem is common in computer science, that have already offered many different solutions with various constraints and input parameters.

The algorithm created by Armaselu and Daescu [2015] partitions convex polygons in same sized areas and introduces the constraint that all perimeters should be of equal size. The algorithm can partition the polygon in pn^k convex parts, so the number of partitions is not flexible. The method of Lingas et al. [1982] is flexible in the number of partitions. Also it can cope with non-convex polygons and introduces the constraint that the cut lines are as small as possible, but the drawback of this algorithm is that it only works with rectilinear polygons, polygons with only corners of 90°. Finally, the algorithm created by Bast and Hert [2000] is also flexible in the number of partitions, has the same constraint (minimum cut length) but can only handle convex polygons. On his blog Khetarpal [2014] describes an algorithm which splits polygons into any number of equal areas, while ensuring the minimum length of line based cuts. It uses the medial axes to while parti-

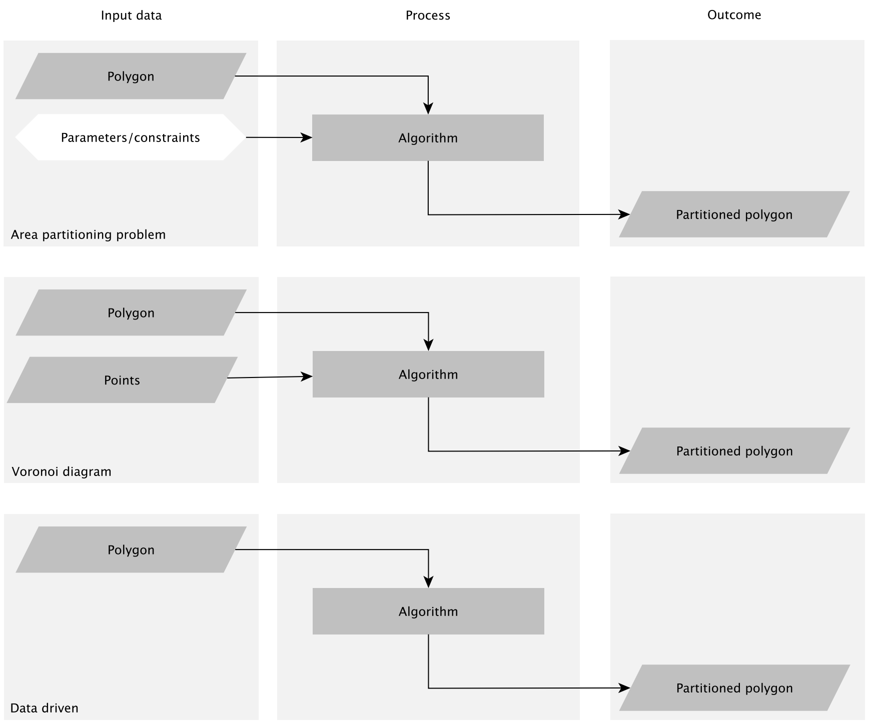


Figure 4.27: Three different partitioning approaches.

tioning. This solution is similar to the method of Bast and Hert, but can also handle non-convex polygons. Another difference between these methods is that the cutlines in the method of Bast and Hert will always have a start at a vertex. In Figure 4.28 the results of different partitioning methods are shown.

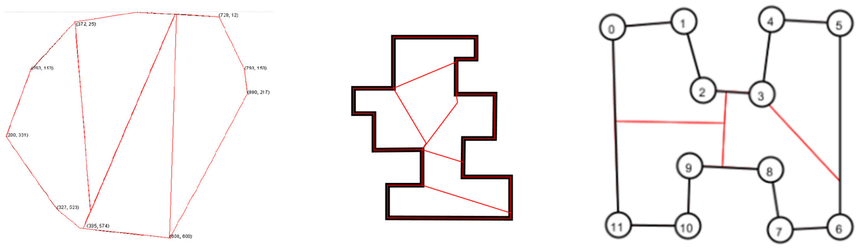


Figure 4.28: The results of the partitioning methods. At the left the result of the partitioning of a convex polygon in four equal sized areas by Armaselu and Daescu [2015]. In the middle the partitioning of a non-convex polygon in six equal sized areas by Bast and Hert [2000]. And at the right the partitioning of a non-convex polygon in four equal sized areas by Khetarpal [2014].

Advantages of these methods are the possible division of the polygon using the area. But these methods have their disadvantages. The algorithm created by Armaselu and Daescu has the extra restriction that the perimeters should have the same size, but it is restricted in partitioning convex polygons in pn^k convex parts. This makes it unusable for the partitioning of building polygons. The cutlines created by method of Bast and Hert all start at a vertex, which results in unrealistic partitions. In addition, this method only handles rectilinear polygons. The method of Ketherpal shows the most promising result, but it should still be modified in such a way that a polygon can be partitioned in different sized polygons. For all these methods there is currently no robust implementation available.

Voronoi diagram

The second partitioning approach is the voronoi diagram, which is also used in the workflow for creating the geometry for the 3D BAG of The Hague as discussed in 3.3.1. In this method the areas are partitioned with information about the location of the points. When considering a Euclidean plane, the line segments of the voronoi diagram contain all the points in the plane that have the same distance between the two nearest points. In 4.29 an example is shown of a voronoi partitioning.

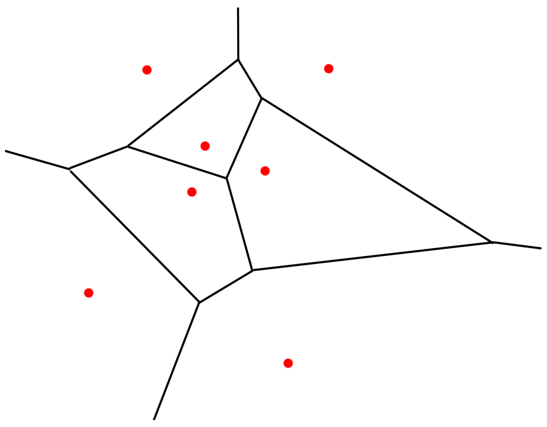
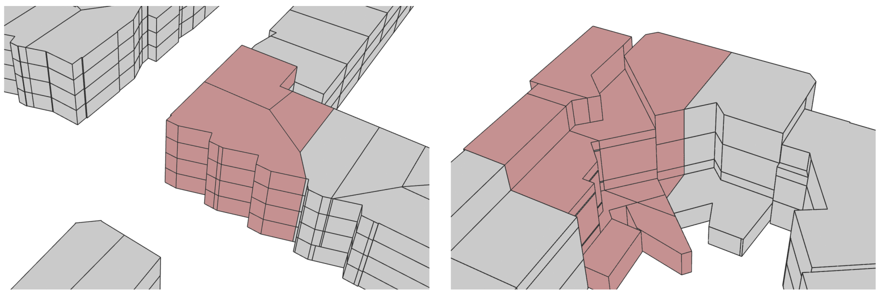


Figure 4.29: An example of a voronoi partitioning.

An advantage of this method is that it is widely used in the computer science [Aurenhammer, 1991] and that there are different robust algorithms available which can be directly implemented. However, this method also has important drawbacks. Because the method uses points as input, these points should relate to the location of the units within the building. As discussed in section 3.1.3, this is not always the case. Points can lie on each other, which makes this method completely impossible, or can have a random location within the polygon. Even if there is a logic placement of the unit points, i.e. the *BAG+* of The Hague, it will often still result in unrealistic partitioning (Figure 4.30).



(a) Corner-building. **(b)** To many points involved.

Figure 4.30: Unrealistic partitioning when using a Voronoi partitioning (in case of the *BAG*), usually occur in building corners. When there are too many points involved, the partitioning becomes a mess.

Data driven approaches

The last partitioning approaches are data driven. This means that only the geometry of the input influences the outcome. The methods discussed here are the partitioning in convex parts, triangulation, and the usage of directional lines. The results of these methods for partitioning are shown in 4.32.

Non-convex polygons can be partitioned into convex polygons. This means that the parts of the partitioning all have interior angles which are smaller than 180°, so all vertices point 'outwards'. As shown in in 4.32a, this method can help with partitioning a building in smaller building compartments, which in turn can be used for further partitioning.

Polygon triangulation is the partitioning of a polygon in a set of triangles of which the interiors do not intersect [de Berg et al., 2000]. Triangulation is widely used within the computer science and, therefore, different algorithms are available [Farin et al., 2006]. Usually, the result of triangulations are not unique, meaning that different solutions for one polygon exist. The result of this partitioning is shown in 4.32b.

The directional lines method is based on the first three steps of the generalization proces described by Kada [2006]. These three steps generate a 2D decomposition of a 3D building geometry by first calculating the plane equations of the facades. The infinite space is then divided by these plane equations, and the cells are identified that have a high percentage of overlap with the original ground plan polygon (Figure 4.31). Commandeur [2012] uses this idea for dividing buildings in partitions. But instead of a 3D model, he uses a line equations of the 2D geometry to partition the building. Afterwards, he uses the height, calculated from a point cloud, to determine which partitions belong together.

As the user has no control on how many partitions are created, these methods cannot be used alone. Partitioning a polygon in convex parts or in triangles is widely used and therefore available in multiple applications. This is not the case for the directional lines method, which is unfortunate because this would provide the most realistic partitioning for building geometry. All in all, this method could only serve as a first step because further partitioning/processing is needed in order to get the desired areas.

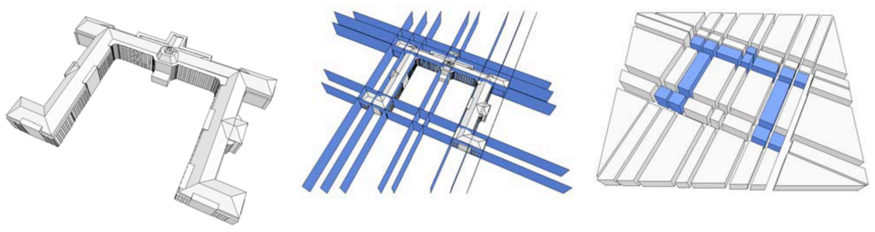
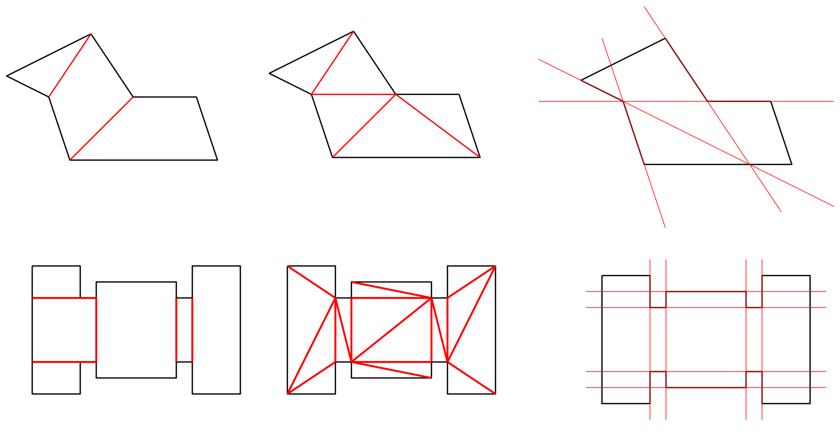


Figure 4.31: The generalization of a complex 3D building model using the plane equations (Figure by Kada [2006]).



(a) Convex parts **(b)** Triangulation **(c)** Directional lines

Figure 4.32: A complex building configuration in Delft.

Conclusion of related work

The desired method for the partitioning of the building polygons would take as input any polygon and the desired number of partitions with their desired area sizes. The result of the partitions should look realistic visually, because the partitions represent building units. Ideally, the corners of the partitions would be 90°.

There is currently no partitioning method available that will provide this desired result. The methods that are most promising are the partitioning method of Khetarpal [2014] and the use of directional lines by Kada [2006] (Table 4.3). When combining these methods, the directional lines method should first divide the building in compartments and the (modified) method of Ketherpal would do the further partitioning. Unfortunately, for both methods there is currently no robust implementation available which makes implementing it within a workflow impossible considering the timeframe of this research.

Method	Constraint	Area parameter	Realistic	Available
Armaselu and Daescu [2015]	Convex	Yes, kp partitions	No	No
Bast and Hert [2000]	Rectilinear	Yes	No	No
Khetarpal [2014]	No	Yes	Semi	No
Voronoi	No	No	No	Yes
Convex parts	No	No	Semi	Yes
Triangulation	No	No	No	Yes
Directional lines (Kada [2006])	No	No	Yes	No

Table 4.3: Comparison of the partitioning methods. The criteria are based on what is needed for this research. For partitioning we need a method that also can partition non-convex, non-rectilinear polygons. Moreover, it is important that it can partition an area based on a parameter, give a realistic result and that there is an implementation available.

To realize the partitioning, I experimented with developing new partitioning methods that fits the above mentioned criteria. The first step off all these methods was to find a starting point by creating the centerline (Section 4.5.2). This starting point will indicate from which side the storey will be filled with units. This is done to make use that in case of long buildings, the numbers will be successive. The first approaches were based on data driven methods, followed by a region growing (Section 4.5.3). The second approach was based on iteration over the centerline and cutting the area of the polygon (Section 4.5.4). To make a decision about which method is most suitable, the results of both methods are compared visually.

4.5.2 Creating the centerline

To find the start point for both partitioning approaches, first the centerline is calculated. This is done by calculating the straight skeleton of the building geometry. A straight skeleton is a method of representing a polygon by a topological skeleton. The straight skeleton of a polygon is defined by a continuous shrinking process in which the edges of the polygon are moved inwards parallel to themselves at a constant speed until they join[Aichholzer et al., 1995]. The straight skeleton is widely used as a way of constructing a polygonal roof structure above a general layout of ground walls (Figure 4.33). The straight skeleton provides a logical line structure for the building.

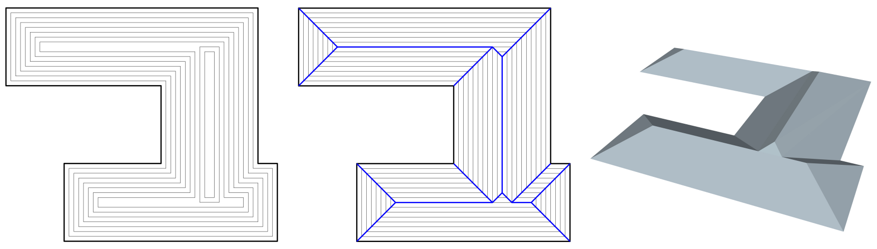


Figure 4.33: From left to right: the shrinking proces, the straight skeleton and the roof model.

The straight skeleton is a connected graph, but not yet usable for determining the starting point. To create one line, the longest path within the graph is calculated. This is done by calculating the shortest path between every end note of the graph. Dijkstra's algorithm was developed to find the shortest path between two nodes a graph network [Skiena, 2008]. For the weight the length of the different line strings is used. As shown in Figure 4.34b, this will result in a line that end is in a corner. Because it would be more logic to start the partitioning in the middle, the last lines are removed and then the line is extended till it hits the boundary of the polygon (Figure 4.34c). This point will be used as start or end point.

- Graph - longest path
- Remove ends
- Remove smaller than parameter
- Extend the two end edges
- Cut the edges with polygon
- Create centerline

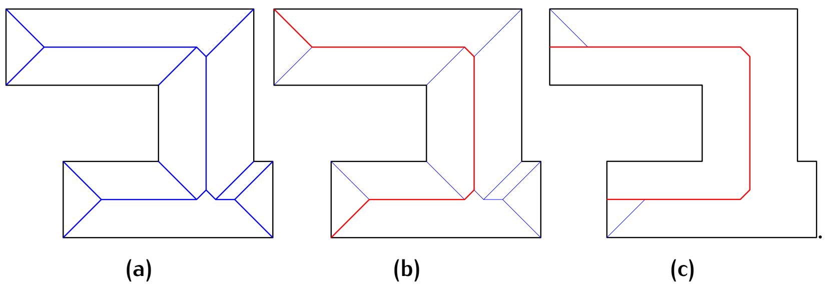


Figure 4.34: From a straight skeleton to the centerline.

To find which point is the starting point and which the end point, the numbers of the units in the surrounding buildings are used. First the buildings that belong to the same street and are closest from the starting and the end-points are detected. The building the building with the lowest unit house numbers, should be the previous building in the street. Therefore, the starting point is set at the points for which this building is the closest (Figure 4.35).

4.5.3 Data driven partitioning combined with region growing

Theses approaches are based on first partitioning with a data driven approach: dividing the polygon in triangles and rasterizing the polygon. The aim is to, from a starting point, add partitions till it is comparable with the desired area size of a unit. To place the objects with use of the area size, the areas of these partitions are first calculated. Then a graph network is created, which enables to do perform an adjusted Dijkstra's algorithm. Dijkstra's algorithm is normally used to find the shortest path between nodes a graph [Skiena, 2008]. It does so by checking every node within the graph. I adjusted the algorithm so it would add the area size of the nodes which it

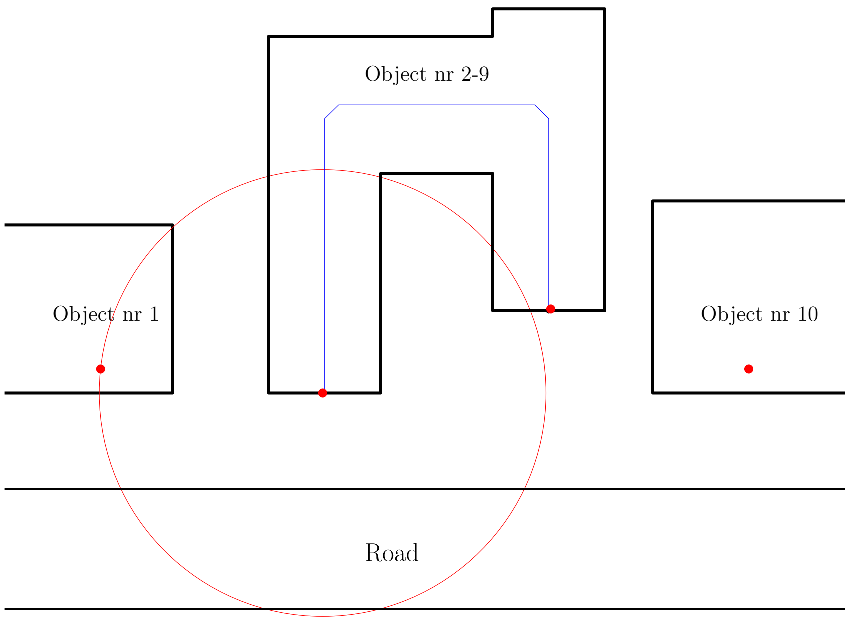
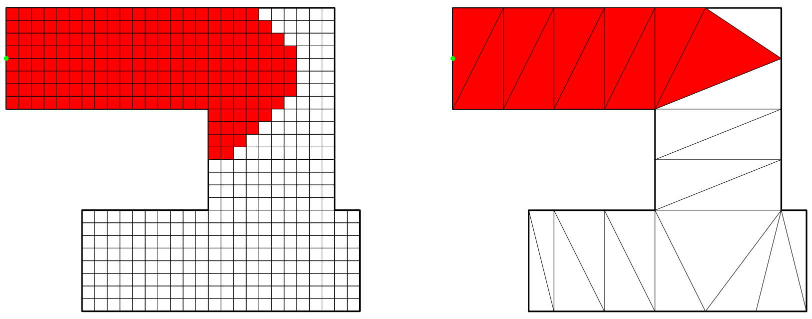


Figure 4.35: Finding the starting point of the centerline.

visited and it would stop when the desired area was obtained.

As shown in Figure 4.36, the result of these approaches will not result in a realistic areas. Also is there a problem when the region meets a corner, is is not sure if the areas will be connected. It is possible to create a line that divides the polygon, e.g. using the furthest partition and creating a line that divides it by sweeping 180°and selecting the shortest line. But when the areas are not connected or the spread is wide, this will not be useful because the line will be at the same location for multiple areas or it will divide the polygon in a unusable way.



(a) Using raster **(b)** Using triangles

Figure 4.36: The region growing on both rasterized and triangulated polygon.

4.5.4 Create cutlines by iterating over centerline

The next approach is based on using the centerline to provide a direction for the partitioning. It is known which units lie on which floor and the area ratios between these. This is used to divide the polygon. The main idea of the workflow is to iterate over the centerline and from the starting point and divide the polygon with cutlines; the combined line from that point to the two closest points on the polygon. The resulting area is checked if it fits with the desired area (Figure 4.37). The output of this part of the workflow are the separate geometries of the unites. The steps will be discussed in more detail in the following sections.

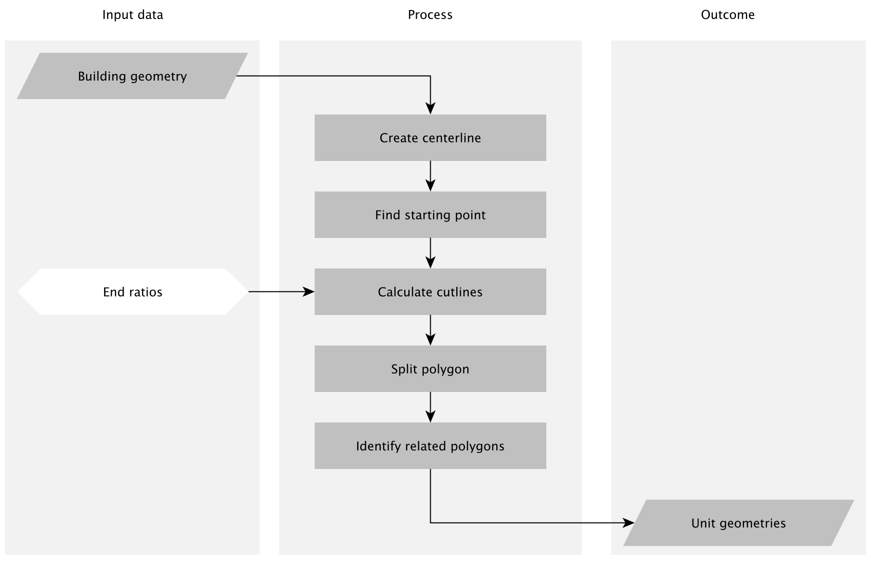


Figure 4.37: The method for partitioning floor areas with the geometry of the building and the floor area ratio of the object as input, and the geometries of the units as outcome.

Calculate cutlines

The created centerline will be used for locating the cutlines. All cutlines between the units are calculated separately. As input the ratios calculated in section 4.5 are used, along with the building geometry and the centerline. As already mentioned is the allocating of the cutlines an iterative proces, it iterates over the length of the centerline. Figure 4.38 shows the process for finding a cutline.

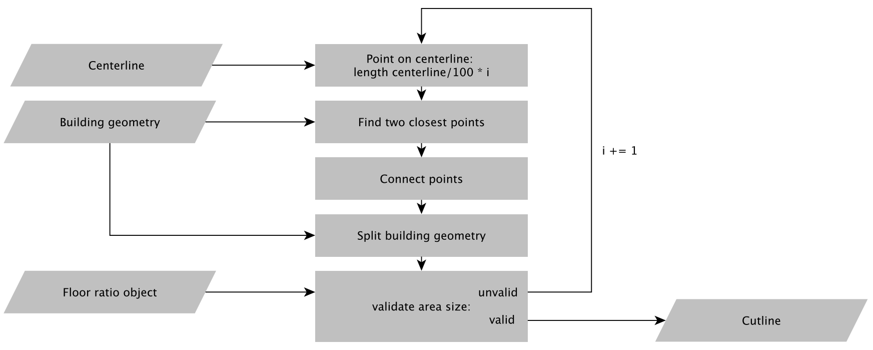


Figure 4.38: Iterative process to find the location of the cutlines.

The cutline is obtained by the following steps:

- Select point on the centerline. The distance from the starting point of the centerline to this point is the length of the centerline divided by $100 * i$.
- From this point, select the two closest points on the boundary of the building. To prevent that the points are on the same boundary, a constraint is introduced. The angle between these points has to be at least 135°. To prevent errors at the start and beginning, boundaries on which the start and end point lies are excluded.
- The three points, namely the point on the centerline and the two points on the boundary of the building polygon are then connected to create a cutline.
- Partition the building polygon using the cutline, creating a subarea.
- The size of the subarea is divided by the area of the complete building geometry to calculate the ratio. This ratio is compared with the floor ratio of the unit, when these ratios are equal the cutline is set. When the ratio is not equal, i is increased by one and the process starts over again.

The output of the proces is the cutline for the given ratio. When finding the cutlines for the other units, only the geometry and the part of the centerline which has not already been used are considered. This prevents that cutlines intersect (Figure 4.39).

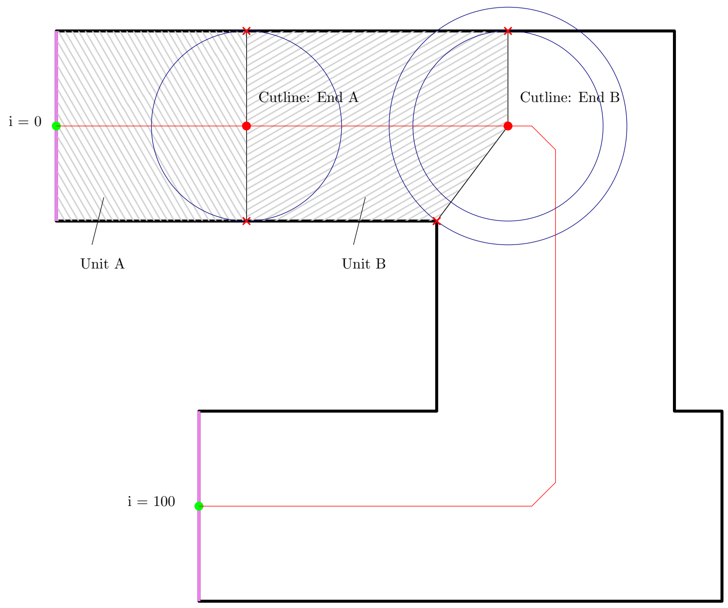


Figure 4.39: Example for finding the cutlines of one area with a start and end ratio. The line in red is the centerline of the building on which the proces iterates. Subarea B is the final geometry of the object. The boundaries in pink are not used when finding the closest points.

4.5.5 Comparison of the the developed partitioning methods

The developed data driven partitioning method based on rasterization and triangulation will result in unrealistic partitions. Especially in corner areas the partitioning methods will result in unrealistic partitions. However, even straight buildings will have this problem because the partitions do not follow straight or perpendicular lines. Moreover, these methods can also result in unconnected partitions which is also undesirable.

The method based on the iteration over the centerline gives a slightly better result. The partitions that are created are connected and for straight buildings the cutline will be perpendicular to the boundary of the building. Although the result of this method will still result in unrealistic partitioning in corner regions, the results is reasonable.

Although this method is capable of creating reasonable realistic partitions, the result of this step of the workflow is least certain. Using the direction of the centerline to divide building floors will result in partitions that look logic visually, but the reality can completely differ. Also can the partitioning fail in multiple cases, two of these cases are visualized in Figure 4.40. When there is a large building section but not large enough that it includes the centerline, the added area within one iteration can be to large. An other problem is when a building has a hole. Then the cutline will not result in a subarea. These problems can result in errors in the partitioning. The result should therefore only be used as proof on concept, not in practice. When more information about the location of the object is known, this method can be improved and the accuracy of the result would increase drastically.

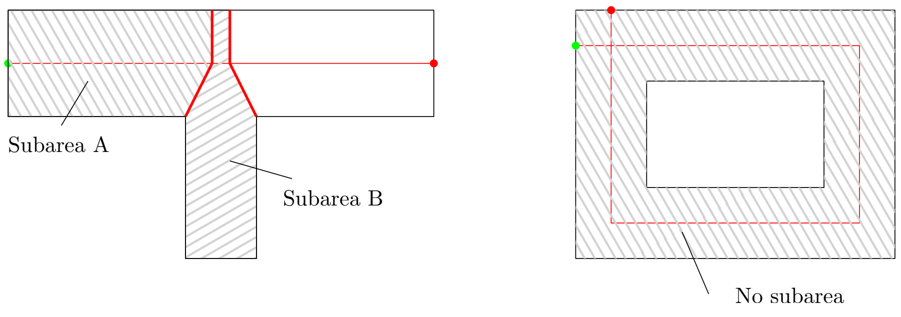


Figure 4.40: Problems in the partitioning can occur in case of large differences in area size perpendicular to the centerline. Also polygons with holes can not be divided.

4.6 CREATING AND VALIDATING THE 3D GEOMETRY OF THE UNIT-FLOORS

In this section the creation and validation of the 3D geometry will be discussed. First the method used to create the 3D geometries will be discussed, followed by the validation of these geometries.

4.6.1 Creating the geometry

For every unit-floor a 3D geometry is created. This means that a unit that has multiple storeys also consists of multiple geometries. The height of the storey is calculated by dividing the building height with the number of storeys, these values are already known (section 4.2). The value of Z_0 is used to place the object-floor on the ground and the starting heights of the storeys are calculated by:

$$Z_0 + (storey_n * storey_h)$$

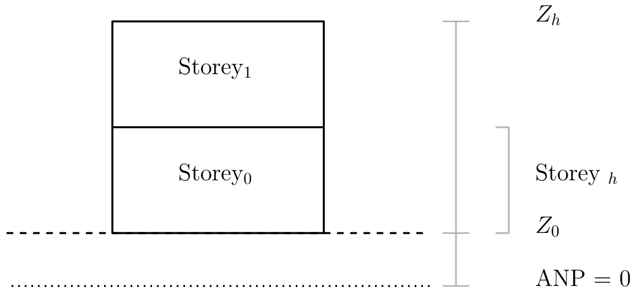


Figure 4.41: The different heights of a building.

To create the 3D volumes, a Z-value is added to every node of the 2D polygons. This Z-value is set at the starting height over the storey calculated with the previous formula. The polygon is then extruded with the storey height. This extruding is done by first copying the original polygon with then with a reversed orientation. The polygons are connected with polygons by iteratively walking through the ring of the original polygon. To create the side polygon two points are used of the original polygon, than the two identical points (except the z-values) of the copy.

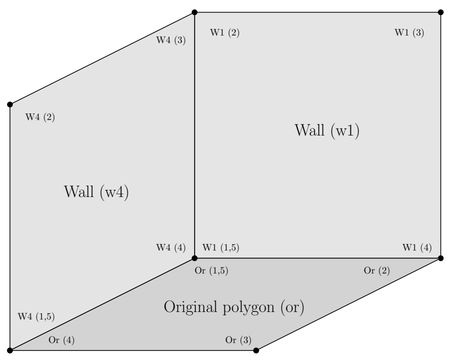


Figure 4.42: The reconstruction of walls.

Wall 1	
W1(1) =	Or(1)
W1(2) =	Or(1) + h
W1(3) =	Or(2) + h
W1(4) =	Or(2)
W1(5) =	W(1)
Wall 4	
W4(1) =	Or(4)
W4(2) =	Or(4) + h
W4(3) =	Or(5) + h
W4(4) =	Or(5)
W4(5) =	W4(1)

Table 4.4: The order of the points.

The result of this step is a 3D model with for all object-floors a geometry. The LOD1 geometry of a building is the aggregate of the object-floors of the related units. When exchanging and converting the created *BAG* geometry, it is important that the created units are geometrically valid. This will ensure that it is possible to do useful GIS operations when working with the created geometry of the *BAG*. In this section the international standard for geometrical primitives are discussed and how the created *BAG* solids can be validated.

4.6.2 Validating the geometry

There are multiple ways of how a solid can be stored. This can result in problems for the interoperability of datasets between systems. In order to counteract these problems, the ISO[2] and the OGC[3] developed standards which define the basic geographical 2D and 3D primitives are and the possible representation on the computer. The following definitions are taken from ISO [2003]:

- A GM_Solid is the basis for 3-dimensional geometry (Figure 4.43). The extent of a solid is defined by the boundary surfaces. The boundaries of GM_Solids shall be represented as GM_SolidBoundary;
- A GML_Shell is used to represent a single connected component of a GM_SolidBoundary. It consists of a number of references to GM_OrientableSurfaces connected in a topological cycle (an object whose boundary is empty.) [...] Like GM_Rings, GM_shells are simple.
- A GM_Object is simple if it has no interior point of self-intersection or self-tangency. In mathematical formalisms, this means that every point in the interior of the object must have a metric neighbourhood whose intersection with the object is isomorphic to an -sphere, where is the dimension of this GM_Object.

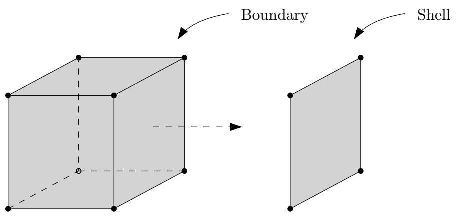


Figure 4.43: A solid is bounded by a number of exterior shells.

The different boundaries of a solid are allowed to interact with each other in certain circumstances. The implementation specifications are not defined by the OGC, but Ledoux [2013] described how the specifications for 2D can be generalized to 3D. The generalization results in the following assertions for the validity of a solid:

1. A solid is topologically closed;
2. Each shell of the solid must be simple, i.e. that it is a 2-manifold;
3. The boundaries of shells can intersect each other, but the intersection between the shells can only contain primitives of dimensionality 0 (vertices) and 1 (edges);

2 International Organization for Standardization: www.iso.org
3 Open Geospatial Consortium: www.opengeospatial.org

4. The shell is a 2-manifold and no dangling pieces can exist;

5. The interior of a solid must form a connected point-set.

To validate the results of the workflow and to ensure the validity of the created geometry, the result has to be validated against the assertions of the previous section. There are tools available for validating geometries. Wagner et al. [2013] created rules for validation of geometry based on the definitions from the Geographic Markup Language (GML) standard and made a tool that can validate 3D CityGML models based on these rules. Another tool to validate geometry is the val3dity. This tool validates 3D primitives and is made available through a webservice[4]. It is created and maintained by Hugo Ledoux (Part of the 3D GeoInformation group at Tu Delft). This tool can not only validate GML files, but also files with the OBJ, OFF, and POLY extension.

4.7 VALIDATION OF THE RESULTS WITH REFERENCE DATASET

It is hard to determine the accuracy of the created model. As discussed in Section 4.2.1 there is no strong connection between the nett floor area of the units and the gross floor area. Therefore it is not possible to validate the created area of the units. To give any certainty about the accuracy of the model, three steps within the process can be validated with the reference dataset: the number of storeys of a building, the number of floors of the units and on which storeys the units are located. The results of these validations will be discussed in the next chapter.

4 http://geovalidation.bk.tudelft.nl/val3dity/

5 IMPLEMENTATION & RESULTS

This chapter concentrates on the technical aspects of the implementation of the methods described in the previous chapter. Because the developed workflow follows a chronological order: the output of every step is the input for the next step. So every step will have its influence on the final result. To provide a proof of concept for the developed step, the result of every step in the workflow will be discussed using real data in order to provide a proof of concept for the developed step. In the first three sections, the tools, the selected area and the way the datasets are obtained will be discussed. Thereafter, the implementation of the workflow will be explained, combined with the results of every step.

5.1 TOOLS

The main tool used during this research is application FME[1]. It is an integrated collection of spatial transformation tools. This tool was selected based on the easy integration of datasets. Instead of focussing on the different datatypes, the focus can lie on combining and transforming the data.

Although FME provides a wide range of tools, not all necessary transformations are available. Especially for the partitioning steps, FME lacks flexibility and For instance, FME lacks flexibility for the partitioning steps. Therefore, these steps were implemented in the Python scripting language. This is a widely used and open source programming language. The following packages are used:

- pyshp [2]: Provides read and write support for the Esri Shapefile format.
- NetworkX [3]: Python language software package that provides functions for creation and manipulation of complex graph networks.
- Shapely [4]: Package that enables the manipulation and analysis of geometric objects in the Cartesian plane.

The *BAG* data is stored in a PostgreSQL[5] database. PostgreSQL is an open source, object-relational database system. The PostGIS extension[6] adds support for geographic data types, needed when storing the geometry of the unitss and buildings.

1 https://safe.com/
2 https://pypi.python.org/pypi/pyshp
3 https://networkx.github.io
4 https://pypi.python.org/pypi/Shapely
5 https://www.postgresql.org
6 http://postgis.net

5.2 TEST AREAS

The prototype is developed with the idea that it should work for the entire country. Three different areas are selected to test and validate the prototype. The selection will test the applicability of the developed methodology to different environments; the areas differ in level of urbanization. The selected areas are: the highly urbanized area of the center of The Hague (Figure 5.2), the urban area of Hoofddorp (Figure 5.3) and the rural area of Schoonhoven (Figure 5.4). The characters of these areas are summarized in Table 5.1 and differ in the number of buildings and number of units per building. Moreover, the building characteristics also differ between these areas which makes them good test cases for the developed methodology. The available data was also of influence for selecting the area. The *BGT* and *AHN3* datasets are not yet fully completed, but are already available for these areas (Figure 5.7 shows the tiles that are currently available).

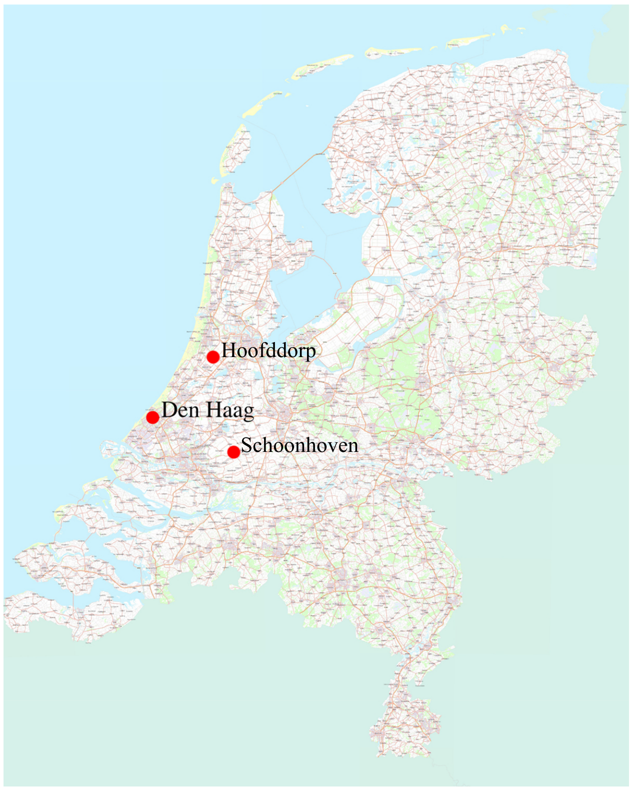


Figure 5.1: The location of the test areas within the Netherlands

Location (city/village)	The Hague	Hoofddorp	Schoonhoven
Setting (urban/rural)	Highly urbanized	Urban	Rural
Area (km2)	10	40	17
Number of buildings	30586	37313	8311
Building density (buildings/km2)	*3058,6*	*932,8*	*488,9*
Number of units	78890	36609	7460
Units per buildings	*2,6*	*1,0*	*0,9*

Table 5.1: This tables list the characteristics of the selected test areas. They differ between size, building density and number of units per building.

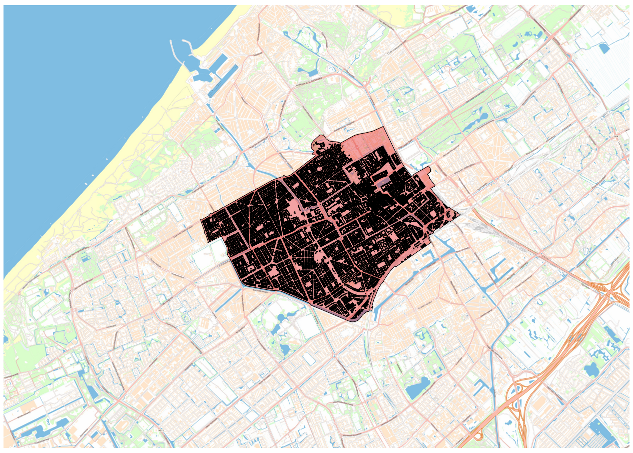


Figure 5.2: The test area of The Hague is highly densified. The selected buildings are colored in black.

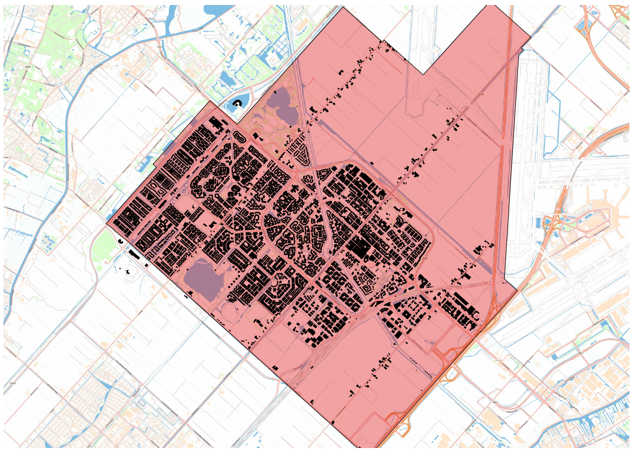


Figure 5.3: The test area of Hoofddorp is larger than the area of The Hague but less densified.

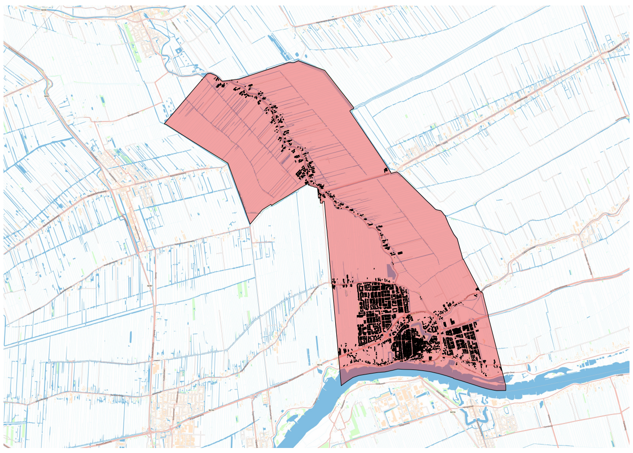


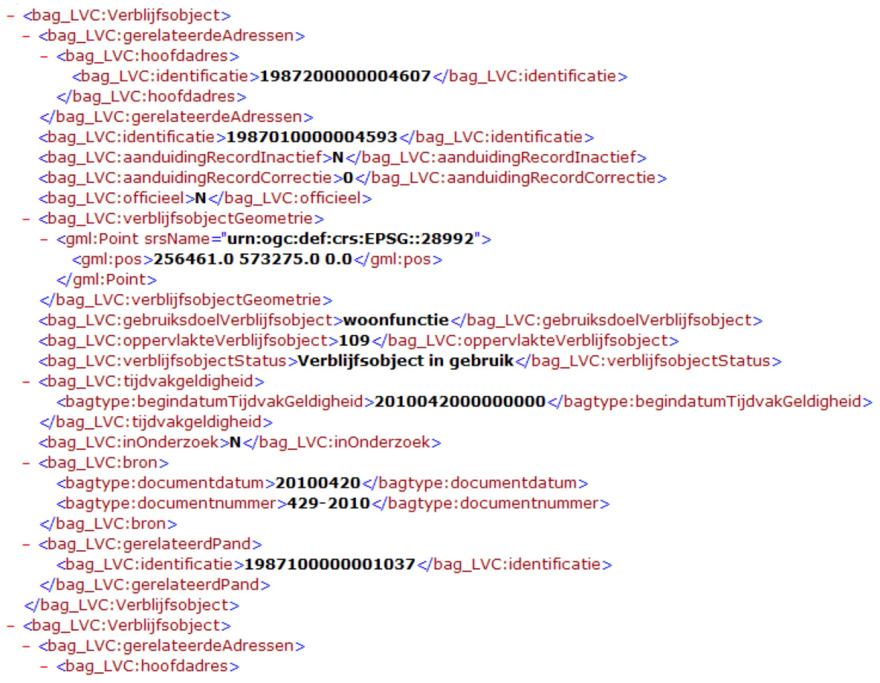
Figure 5.4: The test area of Schoonhoven-Vlist is a rural area, it includes farms and a small village.

5.3 OBTAINING AND PREPARING THE DATA

Obtaining and preparing the data is an important step before the data can be processed. The datasets have to be extracted for different internet-sources and have to be preprocessed before they can serve as input for the workflow. In the following sections, the steps that are needed for obtaining and preparing the data are discussed.

5.3.1 BAG Extract

The *BAG* is made available in the GML data structure (Figure 5.5). Although this data structure is useful for the *BAG* as it can include all non- and spacial data, processing this data structure is inefficient with the current available software applications. To solve this problem, NLExtract[7] created tools to convert this GML data to a spatial database (PostgreSQL/PostGIS). NLExtract is an open initiative that not only provides tools for converting the *BAG*, but also for converting other Dutch geospatial datasets [Broeke, 2016].



```
- <bag_LVC:Verblijfsobject>
  - <bag_LVC:gerelateerdeAdressen>
    - <bag_LVC:hoofdadres>
        <bag_LVC:identificatie>1987200000004607</bag_LVC:identificatie>
      </bag_LVC:hoofdadres>
    </bag_LVC:gerelateerdeAdressen>
    <bag_LVC:identificatie>1987010000004593</bag_LVC:identificatie>
    <bag_LVC:aanduidingRecordInactief>N</bag_LVC:aanduidingRecordInactief>
    <bag_LVC:aanduidingRecordCorrectie>0</bag_LVC:aanduidingRecordCorrectie>
    <bag_LVC:officieel>N</bag_LVC:officieel>
  - <bag_LVC:verblijfsobjectGeometrie>
    - <gml:Point srsName="urn:ogc:def:crs:EPSG::28992">
        <gml:pos>256461.0 573275.0 0.0</gml:pos>
      </gml:Point>
    </bag_LVC:verblijfsobjectGeometrie>
    <bag_LVC:gebruiksdoelVerblijfsobject>woonfunctie</bag_LVC:gebruiksdoelVerblijfsobject>
    <bag_LVC:oppervlakteVerblijfsobject>109</bag_LVC:oppervlakteVerblijfsobject>
    <bag_LVC:verblijfsobjectStatus>Verblijfsobject in gebruik</bag_LVC:verblijfsobjectStatus>
  - <bag_LVC:tijdvakgeldigheid>
      <bagtype:begindatumTijdvakGeldigheid>2010042000000000</bagtype:begindatumTijdvakGeldigheid>
    </bag_LVC:tijdvakgeldigheid>
    <bag_LVC:inOnderzoek>N</bag_LVC:inOnderzoek>
  - <bag_LVC:bron>
      <bagtype:documentdatum>20100420</bagtype:documentdatum>
      <bagtype:documentnummer>429-2010</bagtype:documentnummer>
    </bag_LVC:bron>
  - <bag_LVC:gerelateerdPand>
      <bag_LVC:identificatie>1987100000001037</bag_LVC:identificatie>
    </bag_LVC:gerelateerdPand>
  </bag_LVC:Verblijfsobject>
- <bag_LVC:Verblijfsobject>
  - <bag_LVC:gerelateerdeAdressen>
    - <bag_LVC:hoofdadres>
```

Figure 5.5: Example of GML data of a BAG unit.

NLExtract also provides a ready to use PostGIS dump of the *BAG* which I used for this research. These dumps are updated every month and can directly be imported in PostGIS. This is the most easy way of downloading the data and therefore used for this research. The PostGIS dump includes tables for every class of the *BAG* (Section 2.2). The data is extended with information about provinces and municipalities. As mentioned in section 2.2, features can have multiple relations, e.g. a unit can have multiple functions or can be located in multiple buildings. In order to store these relations, extra tables are included in the database. The NLExtract dumps also pro-

7 http://www.nlextract.nl

vide predefined views for the current and existing buildings and units. The relation between these views, which are used for this research, are shown in figure 5.6. For filtering and storing the data that falls in the selected test area, I created two extra tables: `units-selected` and `Building-selected`. The selection of the units and buildings is performed as follows:

1. Insert the polygons of the test areas in the PostGIS database. The polygons were created by selecting the neighborhoods in the *Wijk- en Buurtkaart* [8] dataset. This dataset contains all neighborhoods of the Netherlands and is available as open data.

2. Select the features in the building table that lie within the polygon of the test area. This is done by using the *ST-Within()* operation of PostGIS. The features that fall within the test area are stored in a new table (*buildings-selected*).

3. The selection of the units that fall in the test areas are selected by joining multiple tables. As shown in figure 5.6, two joins are needed (units - units in building - Building-selected). The units are stored in a new table (*units-selected*).

The outcome of this selecting process are two extra tables that contain all the current existing buildings and units that fall in the test area (Figure 5.34). The *building-selected* table contains 30586 features and the *unit-selected* table 78890. The database is directly connected to the FME workflow.

5.3.2 *BGT and AHN3*

Publieke Dienstverlening Op de Kaart (*PDOK*) provides the central portal for the dissimilation of governmental geographical datasets. The initiative for this facility came through a partnership between the *Kadaster* (Dutch cadastre), the ministries of *Infrastructuur en Milieu* and *Economische Zaken*, *Rijkswaterstaat* and *Geonovum*. Through this portal the *BAG*, *BGT* and *AHN* can be retrieved. The *BGT* is only available in the GML format. The *AHN* can be download as raster (digital elevation model (DEM) and digital terrain model (DTM)) and as raw point cloud. The latter is used because it contains more information, e.g. the classification of the points. The website uses a tile system, in which the user can select a tile for downloading a specific area (Figure 5.7). Using this tile system, the datasets are downloaded and stored locally.

8 https://data.overheid.nl/data/dataset/wijk-en-buurtkaart-2014-versie-1

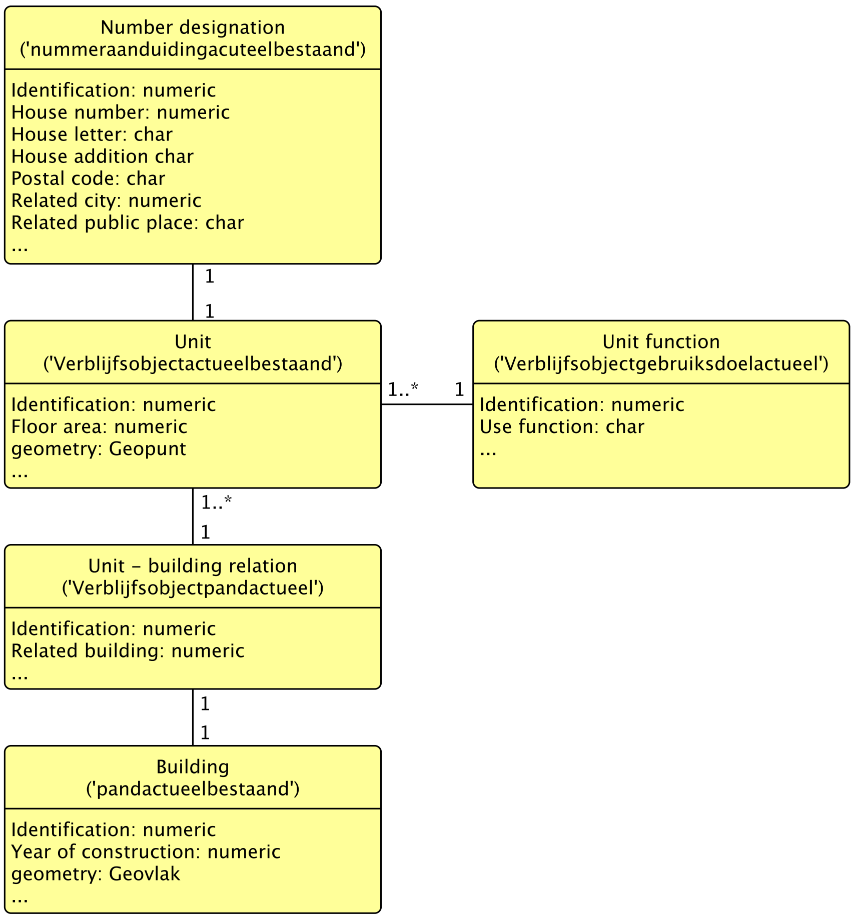
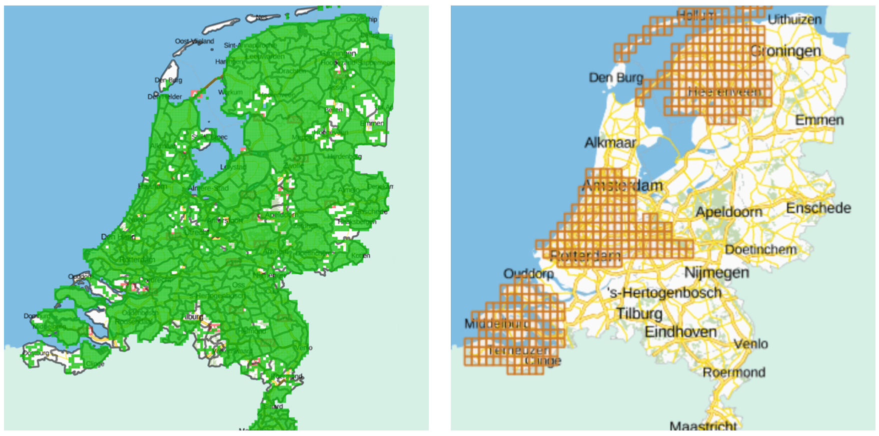


Figure 5.6: The relation between the buildings, units, and number designation in the **dbms!**.



(a) BGT tiles **(b)** AHN3 tiles

Figure 5.7: Tiles of the *BGT*[9] and *AHN3*[10] that can be downloaded from the *PDOK* website (November 2016).

5.4 STEP 1: CLASSIFY UNDERGROUND BUILDINGS

The first step of the workflow is to detect underground buildings. This section will describe how the methods that are developed to identify underground structures (Section 3.1.1) are implemented. A detailed workflow of this step is shown in 5.8. The implementations for the lidar- and bgt method will be described including the required parameter values. Because working with large point clouds datasets is time-consuming, I extracted all needed information from the data at once. As a consequence, not only information that is required for detecting underground buildings is extracted, but also other height values. The attributes that will be added to the *BAG* building features in this step are:

- `Area size`: The area-size of the geometry of the building polygon. This is used for calculating the density of the building points and will also be used later in the workflow (Section 4.4);
- `Zmin`, `Zmax`: the minimum and maximum height and the building, not only used for calculating the building height, but also for the creation of the geometry of the units (Section 4.4);
- `Hbuilding`: The building height used to calculate the number of storeys (Section 5.5);
- `Nstorey`: Number of storeys (Section 5.5);
- `Location`: indicates if a building is aboveground, underground or if the result is uncertain;
- `Complex-building`: indicates if a building has a big height (Section 5.5.2);
- *Temporal-difference*: if there is a temporal difference between the capturing of the point cloud data and the *BAG*.

5.4.1 LIDAR- and BGT-method

The *AHN3* point cloud is filtered based on the classification. The *AHN* uses the standard LIDAR point cloud format *LAS*, which means that the building points are classified with a value 6 and the ground point with a value 2. After the points are filtered based on this classification, they are clipped against the building polygon. The number of points is calculated and divided against the area of the building polygon. In cases where the number is below 4 points per square meter, the attribute `lider-method` is set to '`true`'. If there are no points that can be clipped at all, the feature will also get this attribute value. From the point cloud the date on which the data was captured can also be extracted. This date is compared with the *BAG* feature to find out if there is a temporal difference. If the date of the points is earlier than the date of the *BAG* building, the attribute `temporal-difference` is set to '`true`' and the building is excluded from the workflow.

To find out if the buildings are missing from the *BGT*, the datasets of the same area are tried to merge using the building identification number. In situations where the buildings are not merged because they are missing from

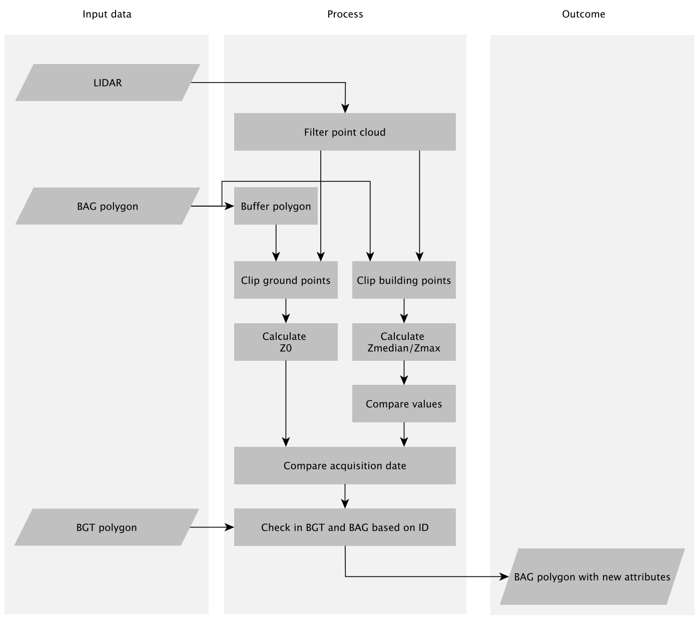


Figure 5.8: First steps of the implementation detects underground structures, complex buildings and calculates the heights which are used to determine the number of storeys.

the *BGT* dataset, the attribute `bgt-method` is set to '`true`'.

5.4.2 Result of underground identification methods

In this section, the consequence of the implementation for the selected test area will be discussed for the selected test areas. Outcomes of the different methods are discussed first, followed by the final classification.

The number of buildings that have a temporal difference differs between the test areas (Figure 5.9. In the test area of The Hague, approximately 2% of the buildings has a temporal difference. In the areas of Hoofddorp this percentage is around 3%. However, in the area of Schoonhoven this percentage is only 0.4%. These results indicate that for an accurate model, the *AHN* should be captured with a higher frequency in urban areas compared to rural areas.

The results of the LiDAR- and BGT method are shown in Figure 5.10. It is interesting to see that there is a considerable difference between the results of both methods for the area of Hoofddorp, where the results for the other areas are similar. This may due the accuracy of the acquisition of their *BGT*, or inaccuracies in the classification of the *AHN3*. When inspecting the result of the LiDAR method manually most of the buildings that where classified with the LiDAR method as underground, were in fact very small, above ground structures. Only in the test area of The Hague structures where classified with certainty as *Underground* (Figure 5.11). This test area also has the most buildings that are classified as uncertain. In Section 5.10.1, these buildings are validated manually.

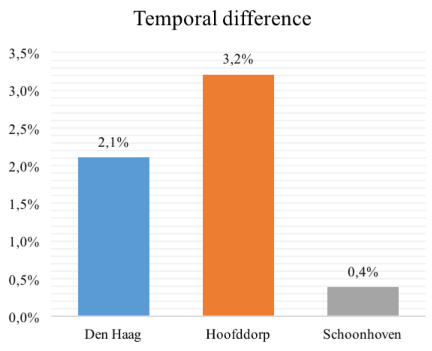


Figure 5.9: The temporal differences between the AHN3 dataset and the *BAG* data. In the area of Schoonhoven this percentage is very low, which indicates that only a few buildings are changed in the past few years.

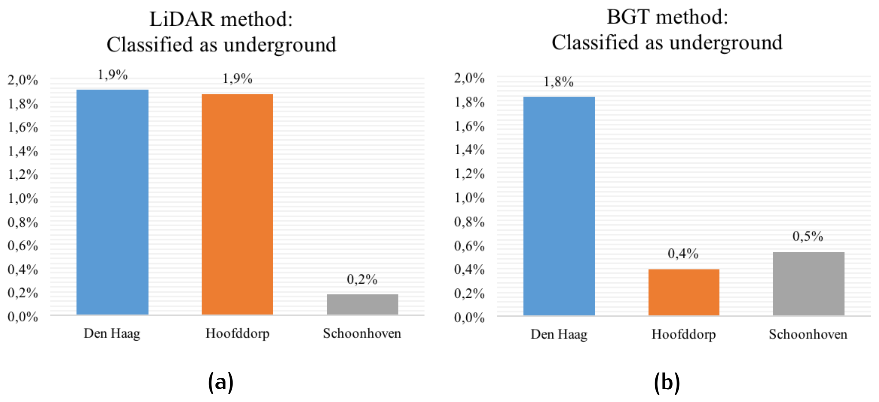


Figure 5.10: The result of the LiDAR and BGT method for the different test areas. Interesting is the difference between both methods for the test area of Hoofddorp.

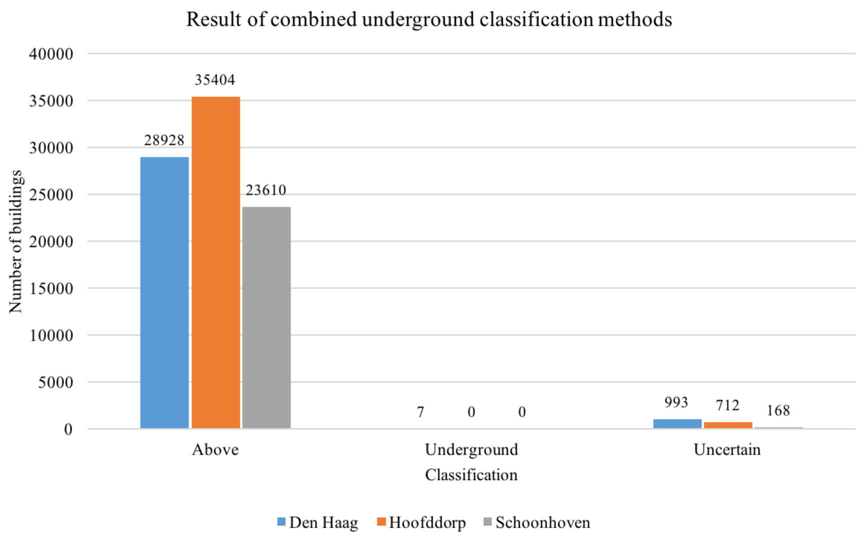


Figure 5.11: When combining both methods, only 7 buildings are classified as `underground`. As expected, most buildings are classified `above ground`.

5.5 STEP 2: CALCULATING THE NUMBER OF STOREYS ABOVE GROUND

The next step of the workflow is to calculate the number of storeys above ground and detect complex buildings. In this section the implementation of this step will be discussed. The way the height values of buildings are calculated along with the number of storeys will be discussed first. In section 5.5.2, the implementation of the identification of complex buildings and complex building configurations is discussed. This section concludes with the result of this step of the implementation. A validation and determination of the parameters for the building height and the threshold for the building complexity will be discussed in section 5.5.3.

5.5.1 Extract height values from point cloud

The height values are extracted during the same process as detecting underground structures (Figure 5.8). From the point cloud that is clipped by the building polygon, the median- and maximum Z-value is extracted. For the maximum Z-value the top (90th percentile) is used. To calculate the Zmin value of a building, first a buffer of 1.0 meter is created around the building polygon. This results in a polygon of the building that is extended 1 meter in every direction. This polygon is subsequently used to clip the point cloud. Instead of building points used previously, the point cloud are now first filtered on ground points (classification = 2). Thereafter, this filtered point cloud is clipped against the buffered building polygon to retrieve the surrounding ground points of a building. From these points the minimum Z is extracted.

The height of a building is calculated by subtracting the Zmax value by the Zmin value. This building height is then divided by 3.5 and rounded (section 5.5.3 describes how this value is determined). In few cases the number of floors turned out to be zero. All had a very small floor area (>10m2), which could be the consequence of the number of building points. For these cases, the number of storeys was set at 1 and the height of the structure at 3.5 meters.

5.5.2 Detect complex buildings and configurations

Complex buildings are identified by comparing the median and maximum Zvalue calculated in the previous step. If the difference between these values is greater than 3.0, the attribute `complex-building` is set to '*true*'. This value is calculated from the *WOZ*-dataset. How this is done will be discussed in section 5.5.3. Buildings that only have one related unit are not excluded from the workflow, because these buildings do not need further division; the interior of the complete buildings is assigned to one unit. How the number of related units is calculated is discussed in Section 5.6. To detect the other type of complex buildings, the geometry of the *BAG* and the *BGT* is checked. In practice, it is sufficient to compare the floor areas of both geometries in order to detect the presence of a large difference. If the difference is larger than 10%, the attribute is also set to `true`.

To detect complex building configurations (see Section 4.2.2), an overlap test is performed. In the geometry of the *BAG*, plenty unintended overlaps exist. It is therefore not possible to do a simple overlap test, but it is needed to find out if the overlap is compelling. For this purpose, I used the percentage of an area that overlaps. This is implemented by first intersecting the building polygons with each other. The overlaps are calculated and divided against the original area. When this is larger than 10%, the the building is classified as complex.

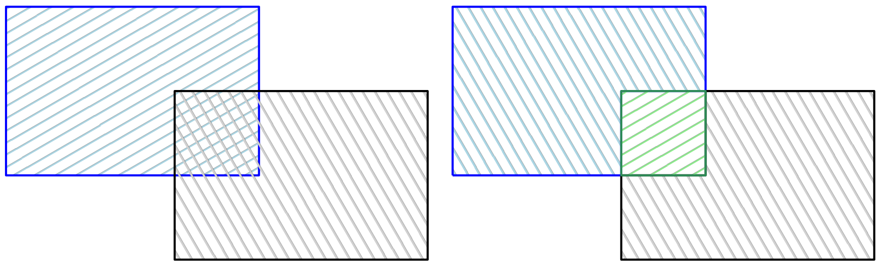

(a) Overlapping. **(b)** non-overlapping.

Figure 5.12: New polygons are created by intersecting overlapping polygons. The difference between the original area and the resulting polygon indicate how large the overlap is.

5.5.3 Calculate the average storey height and set complex building threshold with use of the reference dataset

To calculate the average storey height and to set the threshold for the complex buildings, a reference dataset (the dataset will be discussed in more detail in Section 5.10) is used. From this dataset the number of floors of a unit and the total number of floors can be calculated. When merged with the buildings that are also included within the The Hague area; the calculated number of storeys and the measured number of storeys can be compared. Moreover, the influence of a threshold for the Max- and Median values can be calculated.

I checked the result of 5 thresholds for the Max-median value, ranging from 1 until 3. Also is checked what the result would be if there was no threshold at all. After the filtering based on this threshold, the average storey height was calculated for all features that passed. This height was then used to calculate the number of storeys. Finally, the difference between this value and the reference set was calculated (Figure 5.13).

In table 5.2, the outcome of this workflow on the entire dataset is shown. As expected, a low threshold will result in the most accurate outcome, but will on the other hand exclude 85% of the buildings. It can also be noticed that the average storey height of the buildings is reduced when decreasing the threshold. Based on these outcomes, an optimal threshold of 3.0 is chosen. This threshold still provides reasonable results with the exclusion of only 27% of the buildings is excluded. Although more available reference data will lead to more accurate values, a complete accurate results will not be possible. Moreover, the reference dataset only included buildings in The

Threshold	None	3	2,5	2	1,5	1
Right number of floors	67%	82%	85%	88%	93%	95%
Buildings included	100%	73%	65%	54%	41%	15%
Storey height	3,7	3,5	3,5	3,4	3,4	3,2

Table 5.2: Result of the different thresholds. As expected, the accuracy of the calculated number of storeys increases when the threshold is lower.

Hague. This may not be completely representative for the situation in the entire Netherlands because of the difference in building types etc.

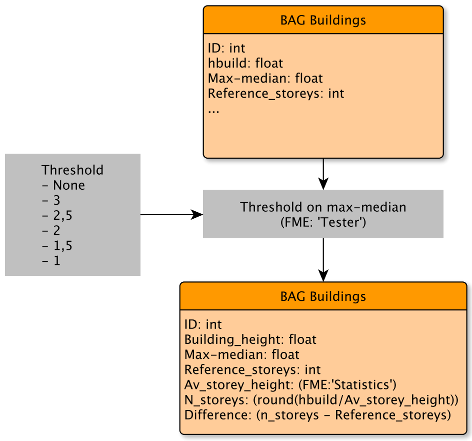


Figure 5.13: The implemented workflow for detecting complex building configurations.

5.5.4 Result of calculating the number of storeys

As a result of the chosen threshold for the complex building detection, a considerable part of the buildings in The Hague (16.5%) is excluded from the workflow. In the other test areas this percentage is around 2% (Figure ??). These values would be much higher when complex buildings with one related unit were also excluded; in the test area of The Hague this value would be almost be twice as high (28%). The number of complex configuration is very low, only in a few cases the overlap was substantially (Figure 5.14).

The exclusion of complex buildings has the most influence of the total number of excluded buildings (Figure 5.15). The difference between the test area of The Hague and the other areas may lie in the number of units per building. As shown in Section 5.6.2, the percentage of buildings with only one related units is considerably lower in the area of The Hague.

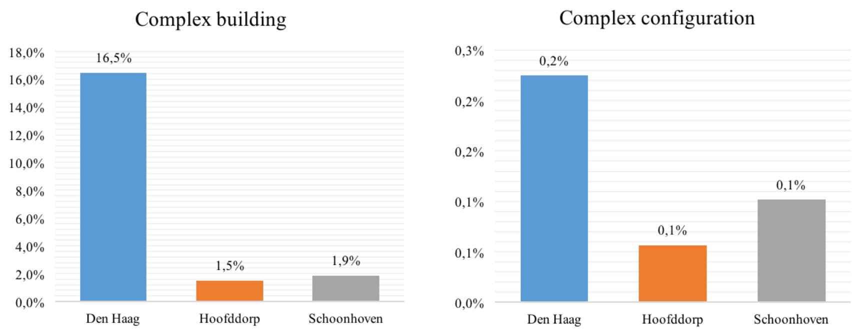


Figure 5.14: A considerable part of the buildings in The Hague was classified as a *complex building*, especially when compared to the other test areas. For all test areas, the percentage of *complex configurations* does not exceed 0.2%.

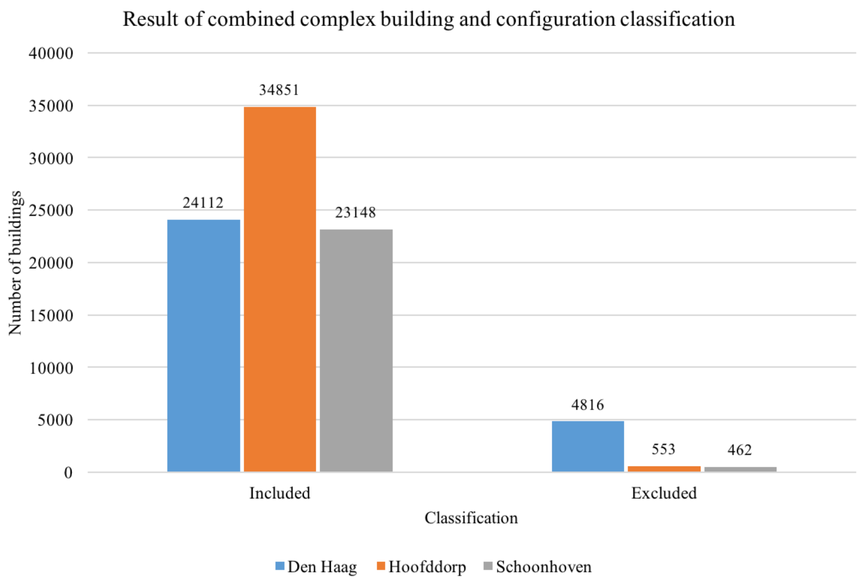


Figure 5.15: The total number of building that are excluded from the workflow. The number of buildings excluded in the test area of The Hague are considerably higher than in the other areas.

5.6 STEP 3: DIVIDE UNITS OVER BUILDINGS

Assigning the units to the buildings is the next step of the process. In this section two different scenarios will be discussed: a buildings with multiple units and a unit with multiple buildings. This section concludes with an overview of the results for the different test areas.

5.6.1 Assigning the units

As every unit has an ID of their related building this is straightforward. In this implementation step, the units will not be merged with buildings. The reason is that the units will be directly combined into floor-sections and then combined with the related building (See section 5.7) later in the implementation. Therefore, attributes are needed that will be created in this step of the implementation. In addition, complex units with multiple related buildings, have to be handled.

A detailed workflow is depicted in figure 5.16. First the default area (d-area) of the *BAG* units is calculated by multiplying the number of storeys with the floor area of the polygon. This value will be used to divide the net floor area of the units if necessary. To detect units with multiple related buildings and buildings with multiple related units, I used the relation table. This table shows the relations between the unit ID and the related building ID. A unit can have multiple related buildings and the other way around.

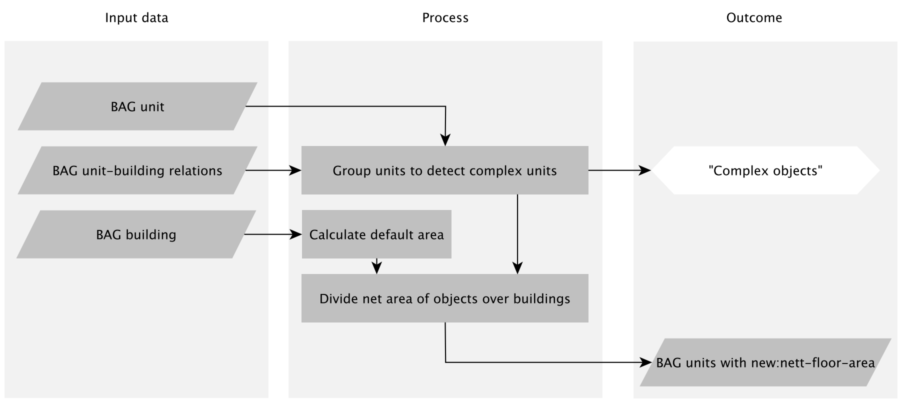

Input data
Process
Outcome
BAG unit
BAG unit-building relations
BAG building
Group units to detect complex units
Calculate default area
Divide net area of objects over buildings
"Complex objects"
BAG units with new:nett-floor-area

Figure 5.16: Dividing units over buildings and classifying complex units.

To find units with multiple related buildings, this table is grouped by unit ID. The resulting dataset includes unique units with a list of their related buildings. In cases with multiple items in the list, the unit is classified as `complex-unit`. The total default area is the combined default area of all related buildings. The lists are subsequently ungrouped and the new net floor area is calculated by using the ratio between the default area of the building and the default area of all related buildings of the unit.

To show the result of this step, The table is grouped on building ID, resulting in the unique *BAG* buildings with a list of the related units. It is now possible to merge it with the *BAG* buildings. In situations involving a complex units, the building is classified as `Complex-unit-building`. The total net floor area is included in a building is the sum of all these values for the related units.

5.6.2 Result of dividing units over buildings

The result of the unit placements is straightforward. All units are now assigned to their related building. Only 20 buildings with a complex units are present. These buildings were checked manually and most cases seemed to be errors in the *BAG*, which are just exceptions. The number of units that is related to a buildings is shown in figure 5.17. Most buildings only have one or zero related units. There is a clear difference visible between the test area of The Hague and the other areas, the number of buildings with more than one units is considerably larger. As shown in Figure 5.18 and5.19, these buildings are often small structures, e.g. sheds. The city center of Schoonhoven (Figure 5.20) is quite dense, however, most of the buildings still only have one related unit.

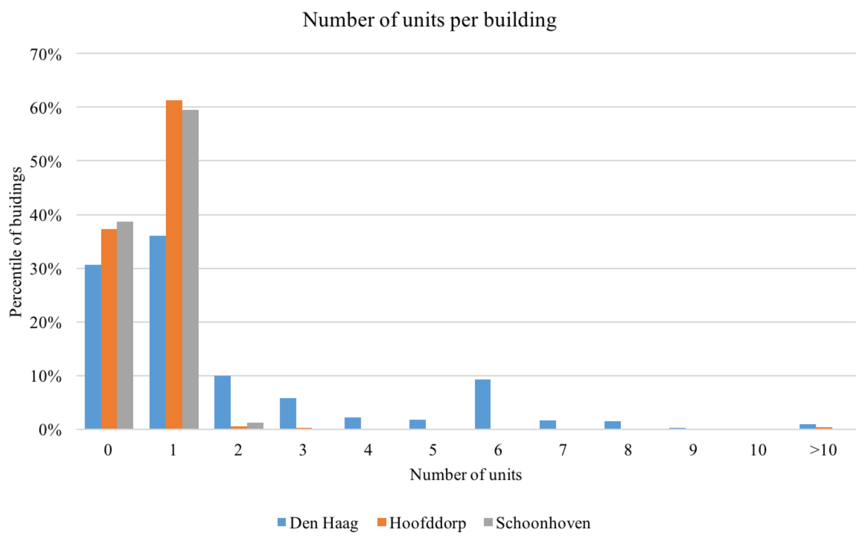


Figure 5.17: The number of units that relate to one building. Most cases are simple, even in the test area of The Hague most buildings only have one or zero related units. In the other test areas this percentage is even highter.

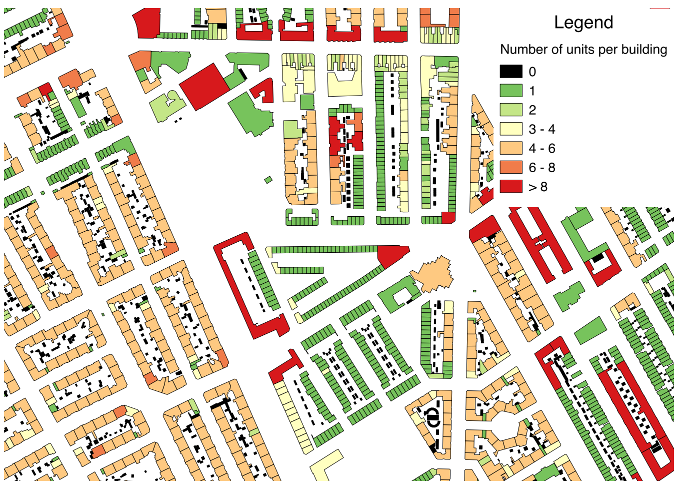


Figure 5.18: Compared to the other areas, the average number of units per building is much higher in the test area of The Hague.

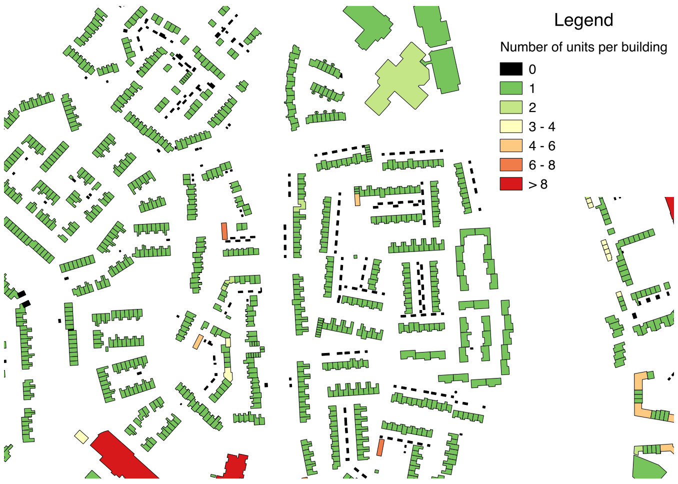


Figure 5.19: In the area of Hoofddorp, the sheds without related units are clearly visible.

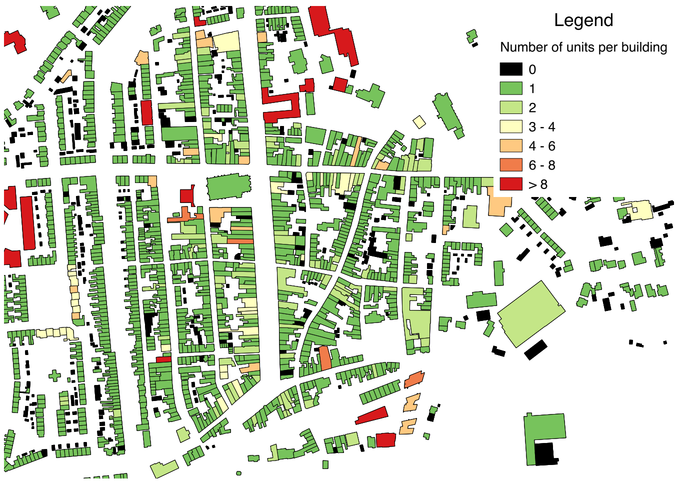


Figure 5.20: The buildings are colored by the number of related units.

5.7 STEP 4: DIVIDE UNITS OVER STOREYS

In this step, the units are divided over the storeys. The implementation is based on the methodology discussed in section 4.4. This implementation step starts with ordering the units by creating a key value that can be used to order the units. Thereafter, the sections are created containing units that are located on the same storey.

5.7.1 Ordering units

To order units, for every unit an ordering value is created. This value is build up in such a way that an unambiguous ordering is possible for every case. Units that are expected to lie underneath other units, get a lower value. Finally, the key value can be ordered from low to high. The order of this key value is as follows:

```
  [Letter value][Addition value][Function value][Area value][Number
value]
```

The letter and addition values are created by translating the characters in ASCII code, which are integers. These integers follow the same order as the alfabet and can, therefore, be directly used.

For calculating the function value, the functions table is used. Units with a residential functions will get a code of 1. In other cases the value will be set at -1. The units are grouped based on unit ID, because a unit can have multiple functions. The list values of this `function-order` are then summed up. In order to achieve only positive values for the key, 100 is added. In case the unit is residential, this will result in a value of 1+100=101, when a unit is a store it wil result in -1+100=99. However, in case a units contains both functions, it will get -1+1+100=100 as value.

Letter	Addition	Function	Area value	Number	Orderkey
A	0	Woonfunctie	46	54	**65**.048.101.000.0054
B	0	Woonfunctie	56	54	**66**.048.101.000.0054
C	0	Woonfunctie	56	54	**67**.048.101.000.0054
D	0	Woonfunctie	46	54	**68**.048.101.000.0054
0	0	Woonfunctie	50	2	48.**048**.101.050.0002
0	BIS	Woonfunctie	120	2	48.**066**.101.120.0002
0	0	Winkelfunctie	114	211	48.048.**099**.114.0211
0	0	Woonfunctie	150	209	48.048.**101**.150.0209
0	0	Woonfunctie	69	340	48.048.109.**069**.0340
0	0	Woonfunctie	98	338	48.048.109.**098**.0338

Table 5.3: The order keys for the units that belong to four different buildings. Bold text indicates the part in the key that influences the ordering.

In case there are only two units, ordering by house number makes less sense. If there are only two units will get an area-value in their key. As such, the units can be ordered from small to large. They area is calculated by dividing the net-floor-area of the unit by the total-net-floor-area of all units in the building combined. To get an integer, this value is multiplied by 1000 and rounded. The final step is the addition of the house number to the key. When ordering the units and all the other values are the same, this will be used. Table 5.3 shows examples of these keys for units that are related to four different buildings.

5.7.2 Placing the units on floor levels

The next step is to create the floors with the units and their ratios. This is a multi-step process that will be discussed separately. Firstly, sections are created of units with the same addition, which are then divided into section floors (Section 5.7.2). The last step is to calculate the floor level and merge the geometry with that of the *BAG* (Section 5.7.2). A workflow of these steps is shown in figure 5.21.

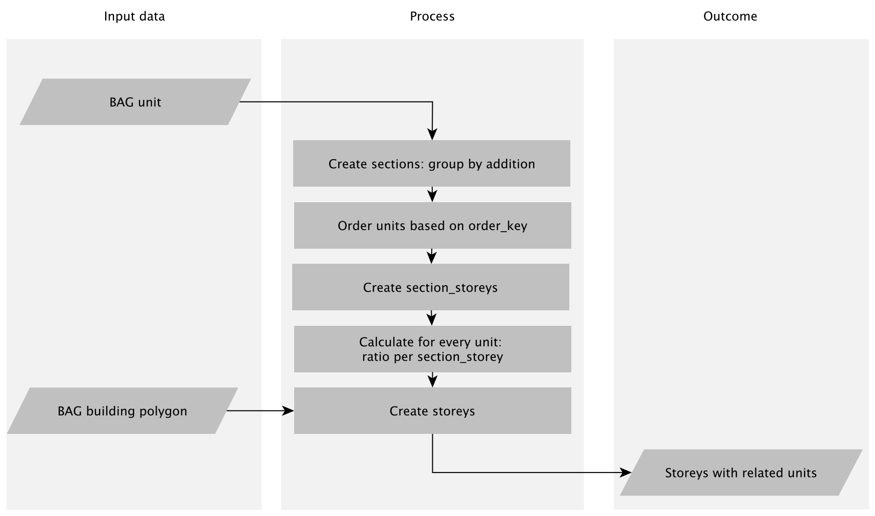

Input data
Process
Outcome
BAG unit
Create sections: group by addition
Order units based on order_key
Create section_storeys
Calculate for every unit:
ratio per section_storey
BAG building polygon
Create storeys
Storeys with related units

Figure 5.21: The implementation workflow for storeys with their related units.

Create sections

To group the units in sections, a key is created by combining the `Letter-` and `Addition value` with the building `ID`. This key is used to group the units in sections. The result is a section of units that have the same related building and the same addition value. The list of key units is sorted by the *order-key* for a clear order.

As discussed in the methodology, the `storey-floor-ratio` indicates how much floor a unit occupies. By summing these values for the units in the same section, the number of floors of the section is determined. When there are no additions and the section is the entire building, the normal number of floors is used. In case there are multiple floors, the required amount of copies of the floor are created. These floors get an index number that will be used in the next step.

Divide units over section floors

It is now clear how many floors a section has and how many units are part of this section. To determine on which floor of the sections the units lie, the `storey-floor-ratio` is used again. Firstly, this value is recalculated to fit the floor area of the entire section.

Firstly, a distinction is made between sections that include additions or house letters. These cases will follow a different step, namely in these cases, all floors will include all units. The `storey-unit-ratio` between these units is used to divide every section floor. When there are no additions, the sections will be filled up from below. First two lists are created, the `storey-unit-start` and the `storey-unit-end` lists. These lists include the starting and end point of every unit. They are created by summing up the ratios of previous units. By subtracting these values for every unit by the `section-level`, the rest values are calculated. When the rest value of the start is lower than one (-threshold) and the rest value of the end is higher than zero (+threshold), the units lie on the section floor. In these cases, the rest ratios are calculated for these units. An example is shown in the following figure and tables.

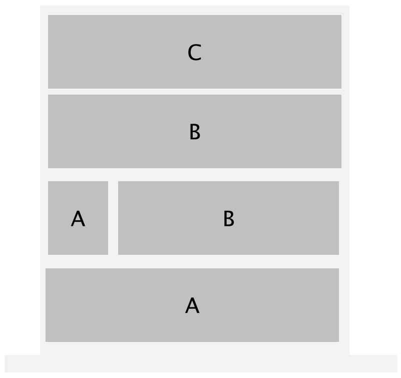


Figure 5.22: Example of section division for units A, B and C.

unit	Storey-unit-ratio	Storey-unit-start	storey-unit-end
A	1,25	0,00	1,25
B	1,75	1,25	2,00
C	1,00	2,00	3,00

Table 5.4: The lists for the units, in which the start and end indicate where the units is located.

Level	Rest-start	Rest-end	Rest-start <0.9	Rest-end >0.1	Result
0	1,25	2,00	FALSE	TRUE	FALSE
1	0,25	1,00	TRUE	TRUE	TRUE
2	-0,75	0,00	TRUE	TRUE	TRUE
3	-1,75	-1,00	TRUE	FALSE	FALSE

Table 5.5: For unit B, the result for every level is shown. The values of 0.9 and 0.1 are chosen as these will prevent small divisions of the unit.

To prevent that sections which include only a small area occupies a complete storey, a threshold is included in the workflow. When the summed Storey-floor-ratio subtracted with the integer of the Storey-unit-ratio value is lower than 0.25 or higher than 0.75, the units will follow the same workflow as the units without additions.

Calculate floor levels

Since floors can be created underneath the section, it is still unclear on which floor the units lie. Therefore, the floors are grouped again on building ID and then sorted on the addition value of the section. When exploding this list to section-storeys, an index is created that is the same as the floor level. This is exploded again to retrieve the the units within the section storeys. This ungrouping will result in multiple entries for one unit because for every storey, the unit will have one entry. The included ratio is used to investigate if the unit is part of this storey, when it is zero the entry is removed. The entries that are not removed will be joined if they have the same level. This will result in a storey with the units included that are part of this floor.

5.7.3 Result of dividing the units of the storeys

As a result of this implementation step, storey features with ordered related units are created. The ratio for the floor that will be used for partitioning the storeys is included in the units if necessary. As shown in Figure 5.23, the number of storeys that have to be divided is low. In the test areas of Hoofddorp and Schoonhoven this number does not exceed 5%. This is also clear from the 3D models shown in Figures 5.24, 5.25 and 5.26. The number of storeys without related units is remarkably high in the area of Schoonhoven. However, this can be explained because in this area there are many high barns. It can be chosen to set the number of storeys of these barns at one, however the uncertainty will then still not be solved because also structures without a related unit can have multiple storeys.

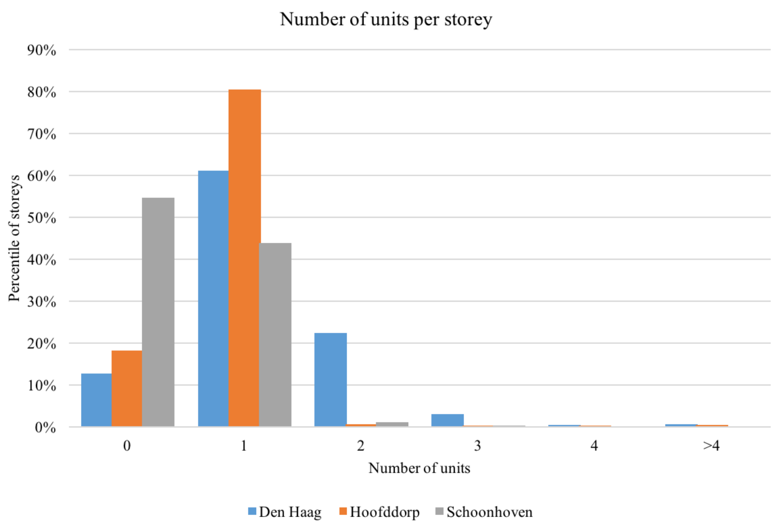


Figure 5.23: The number of units per storey for the different test areas.

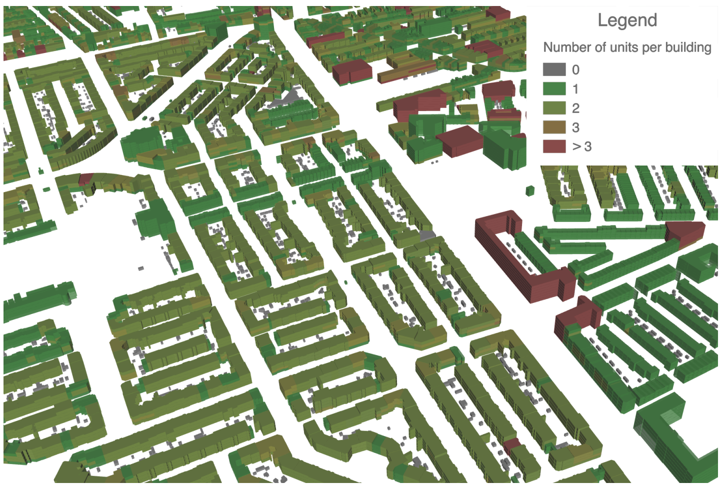


Figure 5.24: The unit-storeys colored by number of units per storey in The Hague.

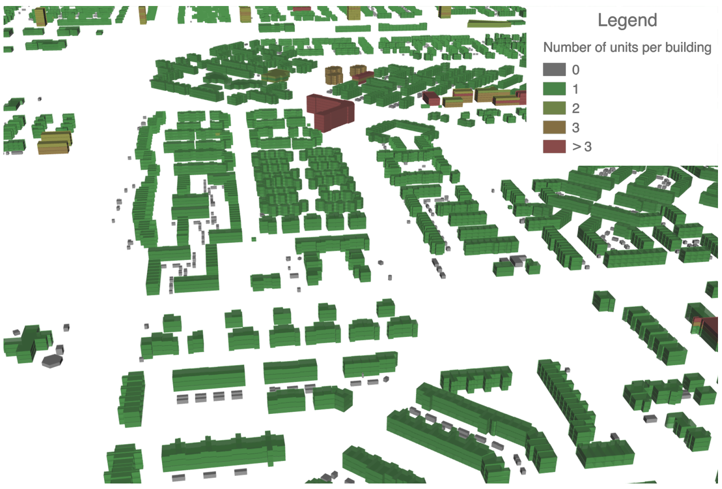


Figure 5.25: In the test area of Hoofddorp, the number of units per storey is lower compared to that of the test area of The Hague.

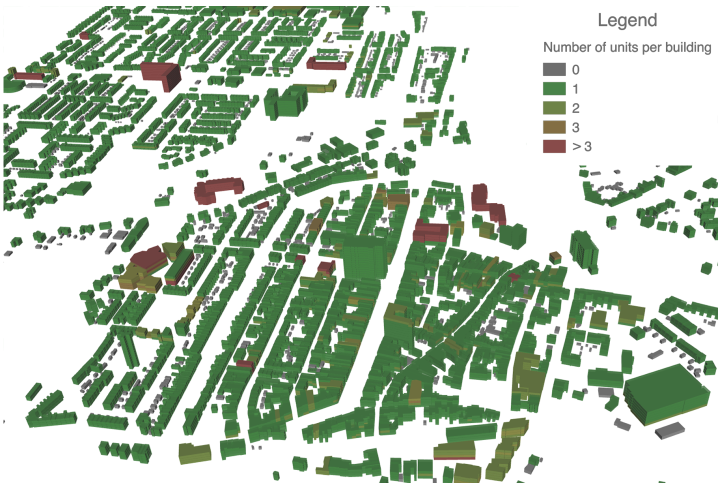


Figure 5.26: Also in the rural area of Schoonhoven, the appartement buildings are clearly visible (colored in red).

5.8 STEP 5: DIVIDE STOREYS

In this section the implementation of the developed partitioning method will be discussed, starting with the creation of the centerline (Section 5.8.1) which is done in Python, followed by the iterative cutline process (Section 5.8.2) done in FME.

5.8.1 Creating centerlines and finding starting points

Creating the straight skeleton of a polygon can be done in FME. The next step is to find the longest path within this straight skeleton. This is done calculating for every end node in the graph the shortest path to the other end nodes with Dijkstra's algorithm (Figure 5.27). This algorithm is implemented in the NetworkX package for python[11]. The weight of the edges is set to the Euclidean distance between the nodes of the edge.

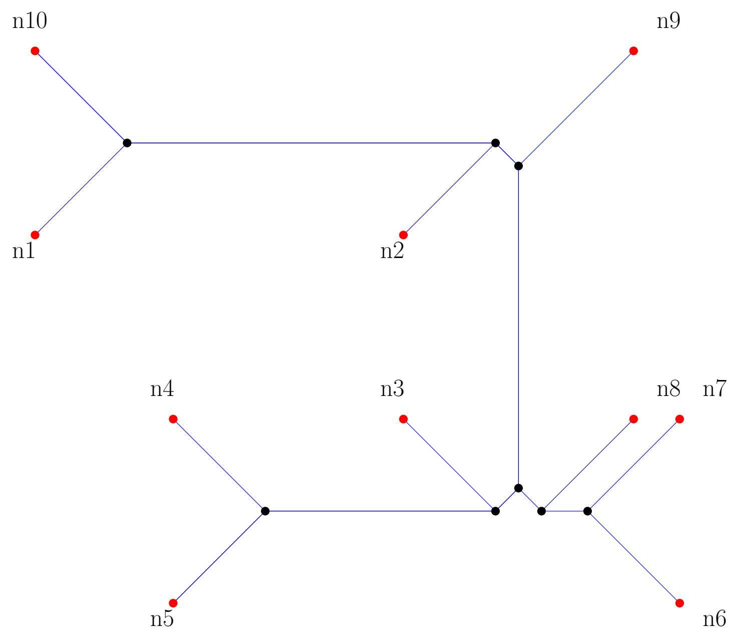


Figure 5.27: Graph network of the straight skeleton. Between every red node, the shortest path is calculated.

The path with the longest Euclidean distance is selected. Both edges at the end are then removed. When the end edges of the resulting path are smaller than 1.0 meter, these are also removed. This will repeated till both end edges are longer than 1.0 meter. These edges are than extended and intersected with the building polygon. The extension is done by first calculating the angle of the edge and the bounding box of the polygon. The line is extended till the bounding box, then the location of the intersection is calculated (Figure 5.28).

11 https://networkx.github.io/documentation

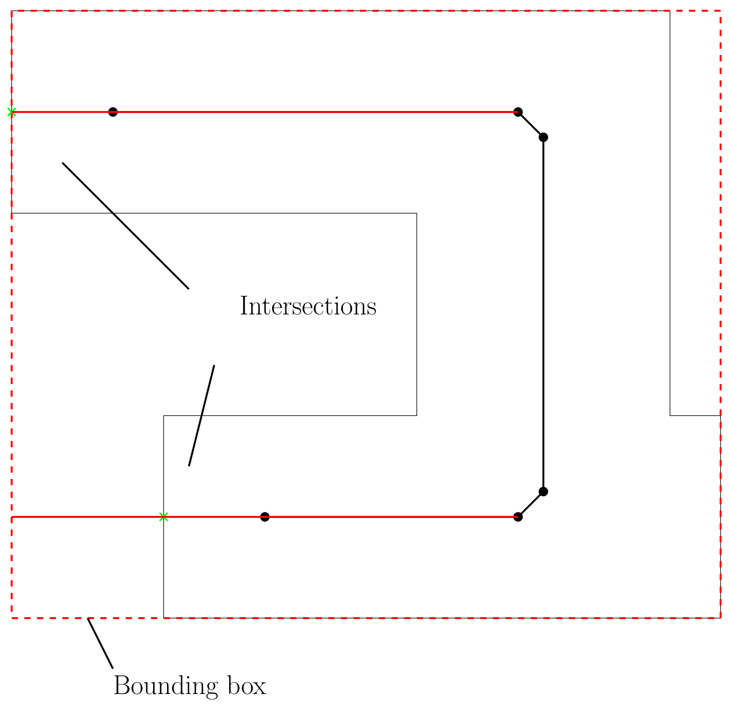


Figure 5.28: The edges are extended till the bounding box of the polygon. Then the intersection point is calculated.

These intersection points are the start or end point from the centerline. To determine which one is the starting point, for both points the closest buildings are determined. For every building it known which units are related, the building polygons will get the house number of the highest units. The point that is closest to the highest number (of the building), will be the starting point.

5.8.2 Creating cutlines by iteration

This sections follows the methodology described in Section 4.5.4. This part of the implementation was done in FME.

The length of the created centerline is calculated and divided by 100, this is used for the iteration process. For every step the two closest points on the polygon (the boundaries of the start- and endpoints excluded) is calculated, with the restriction that the angle between the two points and the point on the centerline is more than 135°. The cutline is created by joining these points with the point on the centerline. The polygon is than divided by this cutline, creating two sections. The area of first section is calculated and divided by the total floor area. When this is larger than the required ratio, the cutline is created. This procedure is repeated for the second cutline. The area between these cutlines is selected.

5.8.3 Results of dividing storeys

This step is not yet connected to the general workflow. To find which errors could occur when using this method, the partitioning method was tested manually. A couple of buildings where selected and the ratios for the desired areas were defined. In Figure 5.29, the result of the creation of the centerline is shown for two buildings.

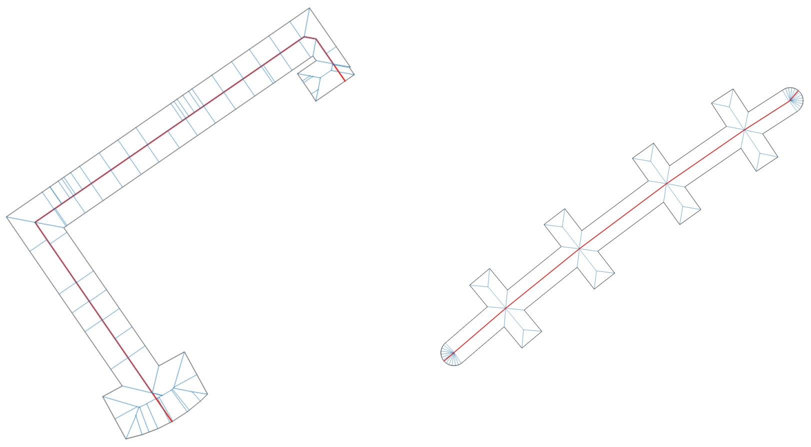
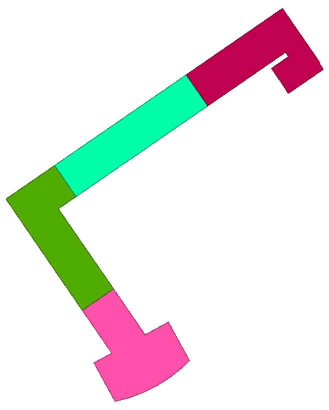
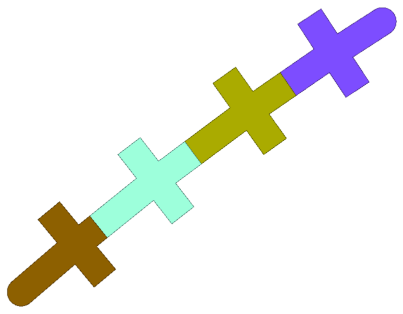


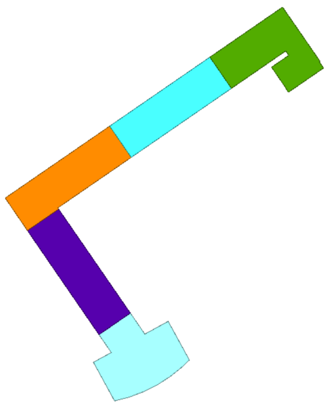
Figure 5.29: Two examples of the created centerlines. The lines in blue are the original straight skeletons of the polygons.

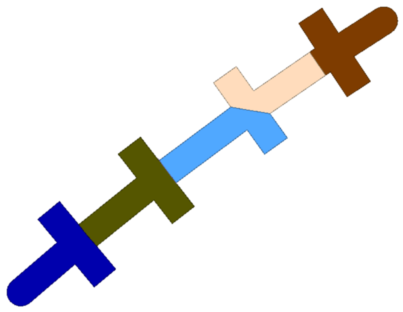
For these buildings the creation of the cutline was checked manually, because in reality these buildings only have one related unit. The partitioning method is also able to create units with different ratios, but for this test the building polygons were divided in 4, 5 and 8 equal parts. This makes it also visually easily to compare the results. As the results show, especially building B, the areas of the partitioning will not always have the desired area due to larger areas perpendicular to the centerline. In these cases, the cutline is also not perpendicular to the building, which results in a unrealistic partitioning. A solution would be to create the cutlines always perpendicular to the centerline, but this caused problems when the centerline went around a corner.

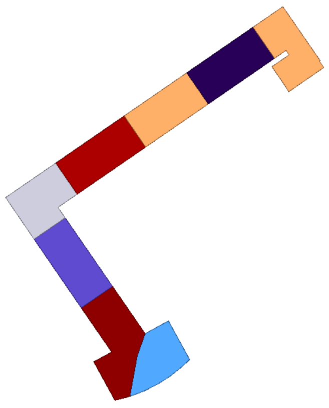
These results show that there is potential in this method. It is possible to partition the polygons based on the area ratios and for 'simple' buildings (buildings without large building compartments that are not in line with the centerline) this method provides an accurate partitioning. But when dealing with complexer buildings, this method should be improved. Suggestions for improving the method will be discussed in Section 7.2.3.


(a)


(b)


(c)


(d)


(e)

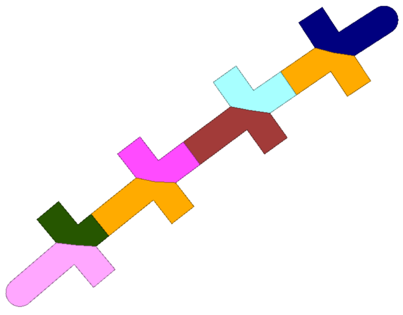

(f)

Figure 5.30: Two buildings that are partitioned in 4, 5 and 8 'equal' parts. As clearly visible in especially the last partitions, the desired area and the resulting area will not always be the same, due to large areas that lie perpendicular to the centerline.

5.9 STEP 6: CREATE AND VALIDATE THE GEOMETRY

In this final step, the 3D model is created and the geometry is transformed to a CityGML standard. The created geometry is then validated using validation software. The geometries that are created belong to `Unit-storeys` that only have one related unit. This because the partitioning method is not (yet) connected to the workflow.

5.9.1 Creating and writing geometry

The 3D model is created by extruding the building geometry with height values. Three parameters are used: the lowest height of the building (Zmin), the levels and the storey height. It will result in extruded volumes for all unit floors. Only the last parameter is not yet calculated. This is done by dividing the building height with the number of floors. The *3DForcer* transformer is used to change the 2D geometry of the floor into 3D. It takes as parameters a height, which in our case will be the lowest height of every floor level:

$$Zmin + (storeyheight * level)$$

Although the polygon now contains a height, it is still a 2D polygon. To translate the 2D polygon to a 3D solid, the polygon is subsequently extruded with the storey height. Currently, the created geometry is still in FME format. However, the software provides the possibility to write the geometry to a different file format, e.g. CityGML. The type is set at GenericCityObject, because there is no featuretype available that can define units (in Section 6.1 citygml will be discussed in more detail).

5.9.2 Result and validation of geometry

The geometry is created for the entire dataset. For every unit-storey a GenericCityObject is created. An example is shown in Figure 5.31, which consist of three `Unit-storeys`. It is possible to include all attributes of the related building and units, but for now is chosen to only include the ID, house numbering, year of construction and on which level the unit is located. The geometry is stored as a solid, which is build up from multiple surfaces. These surfaces are polygons described by an exterior, LinearRing. A section of the GML code is shown in figure 5.32.

The validation of the geometry is done with the web service val3dity. A couple of unit-floors are checked with this service, in all cases the result were positive.

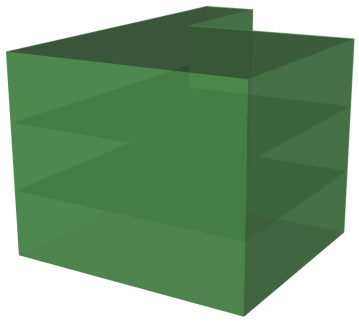
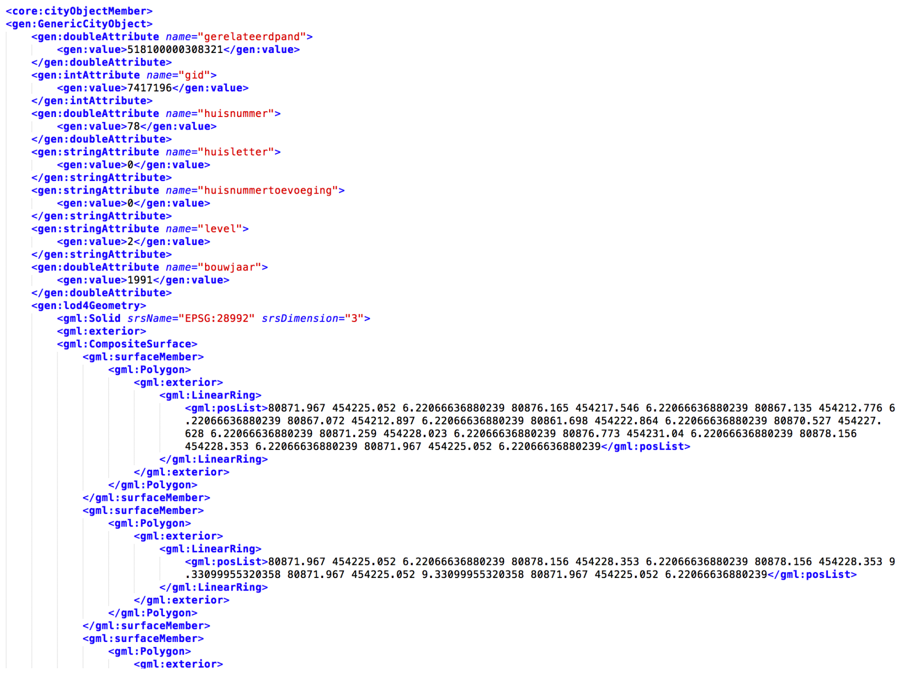


Figure 5.31: The three unit-storeys that are described in Figure 5.32.



```
<core:cityObjectMember>
<gen:GenericCityObject>
    <gen:doubleAttribute name="gerelateerdpand">
        <gen:value>518100000308321</gen:value>
    </gen:doubleAttribute>
    <gen:intAttribute name="gid">
        <gen:value>7417196</gen:value>
    </gen:intAttribute>
    <gen:doubleAttribute name="huisnummer">
        <gen:value>78</gen:value>
    </gen:doubleAttribute>
    <gen:stringAttribute name="huisletter">
        <gen:value>0</gen:value>
    </gen:stringAttribute>
    <gen:stringAttribute name="huisnummertoevoeging">
        <gen:value>0</gen:value>
    </gen:stringAttribute>
    <gen:stringAttribute name="level">
        <gen:value>2</gen:value>
    </gen:stringAttribute>
    <gen:doubleAttribute name="bouwjaar">
        <gen:value>1991</gen:value>
    </gen:doubleAttribute>
    <gen:lod4Geometry>
        <gml:Solid srsName="EPSG:28992" srsDimension="3">
        <gml:exterior>
        <gml:CompositeSurface>
            <gml:surfaceMember>
                <gml:Polygon>
                    <gml:exterior>
                        <gml:LinearRing>
                            <gml:posList>80871.967 454225.052 6.22066636880239 80876.165 454217.546 6.22066636880239 80867.135 454212.776 6
                            .22066636880239 80867.072 454212.897 6.22066636880239 80861.698 454222.864 6.22066636880239 80870.527 454227.
                            628 6.22066636880239 80871.259 454228.023 6.22066636880239 80876.773 454231.04 6.22066636880239 80878.156
                            454228.353 6.22066636880239 80871.967 454225.052 6.22066636880239</gml:posList>
                        </gml:LinearRing>
                    </gml:exterior>
                </gml:Polygon>
            </gml:surfaceMember>
            <gml:surfaceMember>
                <gml:Polygon>
                    <gml:exterior>
                        <gml:LinearRing>
                            <gml:posList>80871.967 454225.052 6.22066636880239 80878.156 454228.353 6.22066636880239 80878.156 454228.353 9
                            .33099955320358 80871.967 454225.052 9.33099955320358 80871.967 454225.052 6.22066636880239</gml:posList>
                        </gml:LinearRing>
                    </gml:exterior>
                </gml:Polygon>
            </gml:surfaceMember>
            <gml:surfaceMember>
                <gml:Polygon>
                    <gml:exterior>
```

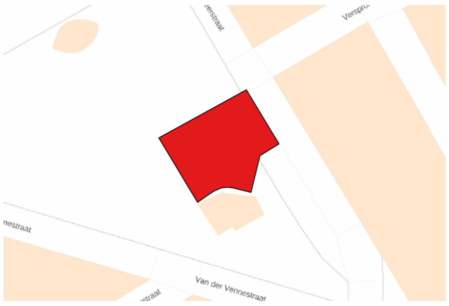
Figure 5.32: Section of the GML code from the unit-storeys.

5.10 VALIDATION OF THE RESULTS

In this section the validation of the results will be discussed. The validation of the buildings classified as `Underground` will be discussed first in Section 5.10.1. This part of validation is done manually. The other part of the validation is done with use of a reference dataset. The content dataset will be discussed in Section 5.10.2. Using this dataset it is possible to validate the following results: the number of storeys per building (Section 5.10.3), the number of storeys per unit (Section 5.10.4) and the location of the Unit-storeys (Section 5.10.5).

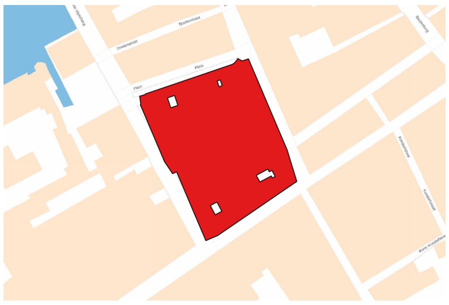
5.10.1 Validation underground classification

The validation of the classification of underground structures is done manually by using streetview. In the test area of The Hague, only seven buildings where classified with certainty as underground structures. The results of the validation is shown in Figure 5.33. The four large structures are classified correctly and are underground parking lots. However, Figure 5.33i shows that there are three small objects classified wrongly. These objects are missing in reality and also do not have any related units within the *BAG*. This may indicate that there is an error in the *BAG* and these items should be removed.



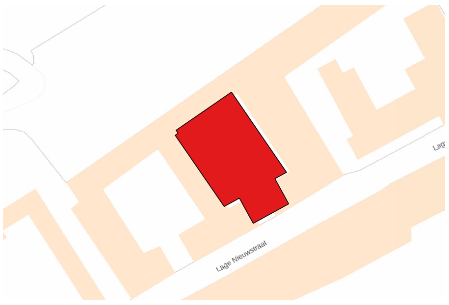
(a)

(b) Underground parking lot.



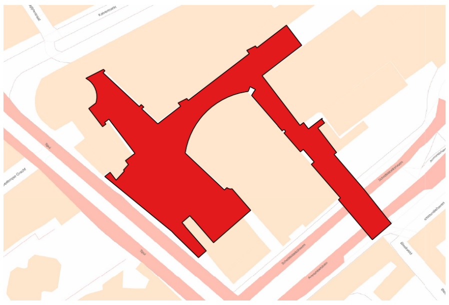
(c)

(d) Parking lot at Plein.



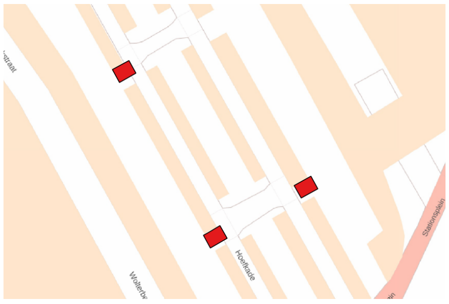
(e)

(f) Underground parking lot.

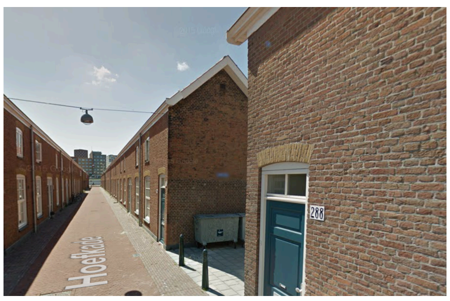


(g)

(h) Underground parking lot Spuiplein.



(i)



(j) Error in the BAG.

Figure 5.33: The seven buildings that are classified als underground structures. The figures on the left show the BAG-geometry and the pictures shows the entrances of these underground structures (Images from [Google Maps, 2016]). The buildings in the last figure are incorrectly classified. This may be due to an error in the BAG, further emphasized by the fact that the buildings do not have any related units.

5.10.2 The reference dataset used for validation

In this section the content of the reference dataset will be discussed. The which buildings are included in the dataset will be discussed first, followed by the structure of the dataset. The dataset is provided by the municipality of The Hague and is not available as open data.

Content of the dataset

The dataset describes buildings and units in the city center of The Hague. It contains 9469 buildings and 36275 units. Not all buildings and units are included in the dataset, only buildings with residential units. Hence, it is not possible to validate units, or extract the number of floors for buildings that have other functions. In figure 5.34 the difference between the test area of The Hague and the reference dataset is visualized.

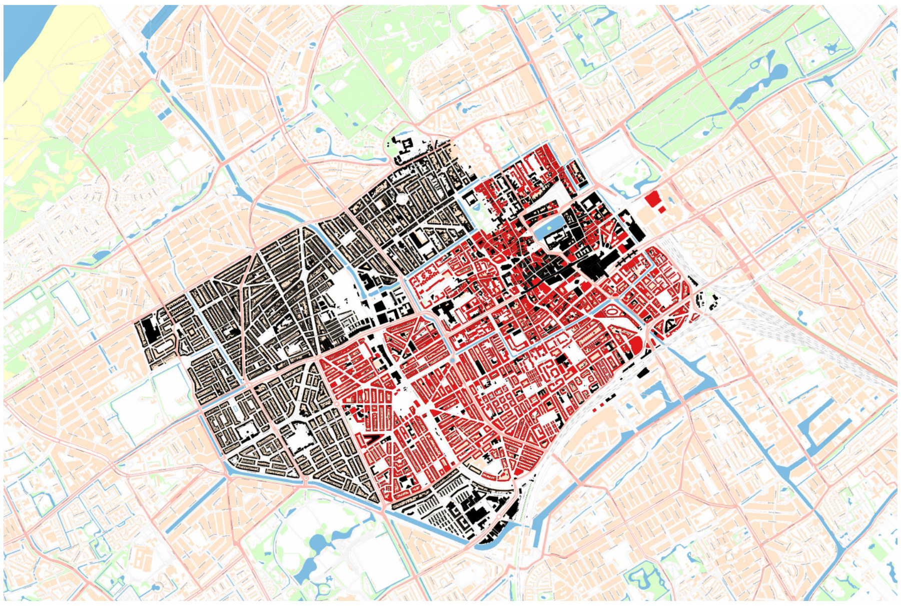


Figure 5.34: Not all the buildings of the selected neighborhoods are part of the reference dataset, because the buildings contain units with other functions or the buildings are outside the area. The buildings colored in red are the buildings within the reference dataset.

Structure of the dataset

The Building-ID's and the Unit-ID's are similar to the identification number in the *BAG* dataset which makes it easy to combine the data. In the reference dataset the type of the unit is described in more detail. However, the biggest difference between the *BAG* data and the data in the reference dataset is that the latter stores information of every unit for every storey (Table 5.6). The 'Start floor' column indicates at which floor the area that is used for living starts. The 'Floor number' indicates the relative floor number within the unit.

Building-ID	Unit-ID	Unit type (code)	Start floor (Unit)	Floor number	Floor area
0518100000339869	0518010000786435	A10	P	1	106
0518100000339869	0518010000387915	A22	1	1	46
0518100000339869	0518010000387915	A22	1	2	53
0518100000339869	0518010000753586	A22	1	1	59
0518100000339869	0518010000753586	A22	1	2	66
0518100000340011	0518010000856442	A75	P	1	38
0518100000340011	0518010000856442	A75	P	2	47
...	...	...	...	...	...

Table 5.6: Section of the reference *WOZ* dataset of The Hague. It is possible to extract the number of floors of a building from this data: i.e. the building with ID 0518100000339869 has three different units. One of the units is based on the ground floor, whereas the other two start at the first floor. These units also include the corresponding floor above and, as such, the building has a total of three floors.

5.10.3 Validation the number of storeys

Using the reference dataset, the number of storeys of a building can be calculated. By summing the *Start-floor* and the *Floor-number* and looking up the highest number for each building, the number of storeys is extracted. This results in a simple dataset that includes the *BAG* ID of the building and the number of storeys.

The accuracy of the calculated can be influenced by changing the threshold. The chosen threshold was set at 3.0 meters, which resulted that 27% of the buildings was classified as complex. Not all these buildings were excluded from the workflow because the buildings with only one related unit were not excluded. When considering only the non-complex buildings, the accuracy of the calculated number of storeys is 82%. Without any threshold, this percentage is 67% (Figure 5.35).

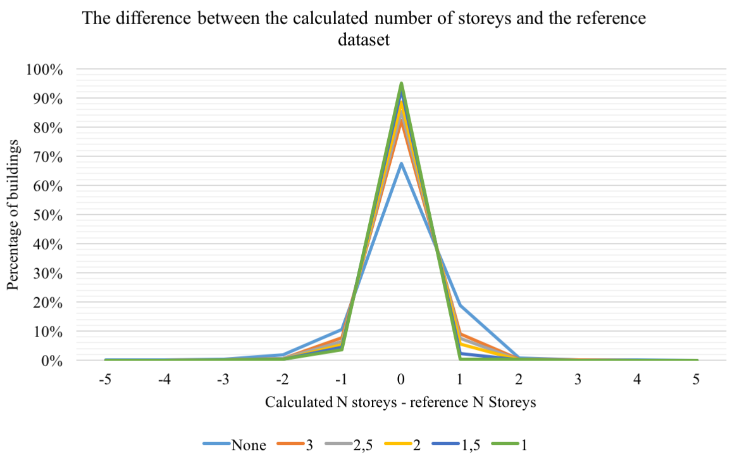


Figure 5.35: The accuracy of the number of storey calculation with the different thresholds.

5.10.4 validating the number storeys per unit

It is possible to calculate the number of storeys for every unit from the reference dataset. By grouping the Unit-ID's and counting the number of features in the groups this number is extracted from the data. This calculated number of storeys is compared to the number of `Unit-storeys` that the workflow created for every unit. Because every step of the workflow influences the accuracy of the result, the inaccuracy of the calculated number of storeys (accuracy of 82%) will also influence the accuracy of the number of created `Units-storeys`.

The comparison shows that the calculated number of storeys is in general slightly higher than the reference data (Figure 5.36). However, the calculated number of storeys is similar to the reference dataset in 72% of all cases.

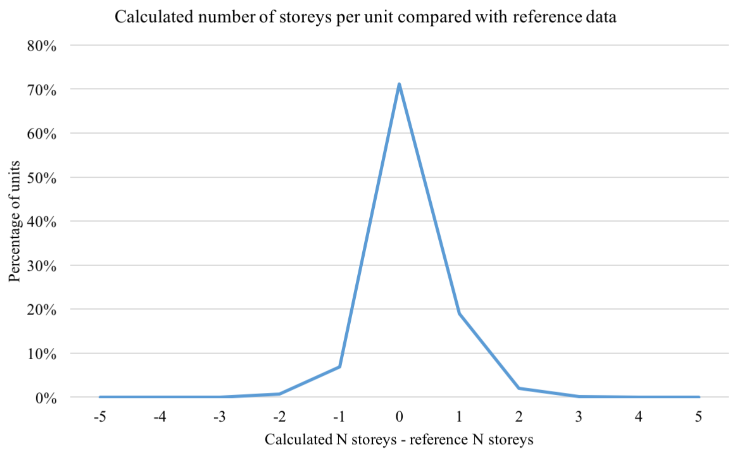


Figure 5.36: The number of `unit-storeys` created during the workflow compared to the reference dataset.

5.10.5 Validation the location of units over storeys

Since the `Unit-storeys` are similar to the way the reference dataset was set up, it is also possible to check how many `Unit-storeys` are on the right level. As such, a `Unit-storey-key` is created in both datasets. When both datasets are merged on these created keys, the number of `Unit-storeys` that are located right can be extracted. The result of this shows that 65% of the `Unit-storeys` are located on the right storey. This validation is very strict, it does not say anything about how close the result is to the reference dataset: only of it is exactly the same. When only considering the buildings that had the correct number of floors, this value is much higher (74%). However, it should be noted that the reference dataset only included residential units. Placing the units based on the function was therefore not possible.

The result of the validation shows that every step has influence on the correctness of the result. All steps contribute to an error: the accuracy of the calculated number of storeys per building (82% of the buildings correct), the number of storeys per unit (in 72% of the cases right) and finally the location of the `unit-storeys` (65%).

6 STORAGE OF THE 3D BAG

An open standard for storage and exchange of data is important for the widespread dissemination of data. For storing the geometry of the *BAG*, flexibility will be needed. This not only means that is has to be possible to model the current state, in case there is no 3D geometry available, but that it also has to provide the possibility to refine the model and add more detail. In this chapter the possibilities for modeling the objects and buildings of geometry of the *BAG* within CityGML is discussed. The chapter will conclude with a proposal for how to store the 3D *BAG* within CityGML.

6.1 CITYGML STANDARD

Currently the *BAG* is available as Extensible Markup Language (XML) data combined with GML for the spatial data. In this research we focus on the geometry part of the *BAG*. GML is the XML grammar which makes it possible to express geographical features. In 2008, OGC introduces the CityGML standard in 2008. Same as GML, it makes it possible to store information on a XML based encoding. But the CityGML differs because it provides a standard model and mechanism for describing 3D objects geometry, topology, semantics and appearance [Gröger and Plümer, 2012]. In particular the CityGML focusses on the representation, storage and exchange of virtual 3D city and landscape models. In this standard, it is not only possible to store building geometries, but many different types of features can also be included. This makes it an ideal model for the complete 3D model, including the *BAG* geometries. The use of a standard 3D model based on CityGML has already proven its validity by the *BGT*, as discussed in section 3.2.4.

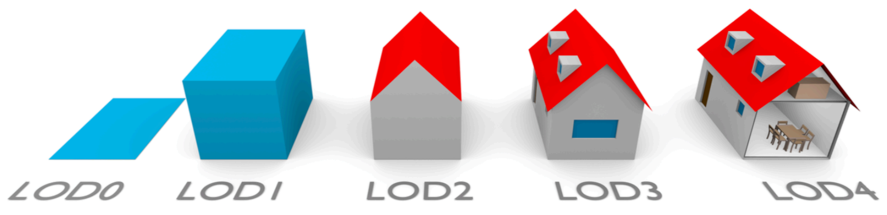


Figure 6.1: The LoD concept of CityGML includes five different levels for buildings: from flat geometries to fully detailed models including the interior.

However, the standard model in the current structure of City GML 2.0 cannot be directly used for the *BAG*, which represents a major drawback. The CityGML standard was developed to represent physical objects and object are more abstract; they do not follow the building hierarchy. This is not the only problem, the current LoD concept of the CityGML standard describes how detailed a building is modelled (Figure 6.1). There are five different level of details, ranging from building footprint (LOD1) to a fully detailed model (LOD4). Only in LOD4, the interior of a building is included. However, the building should be modelled in full detail in order to use the LOD4. Which is undesirable because the interior of buildings is far more detailed than the location of units within a building.

This problem is noticed by OGC and may be solved because the topic of refining and improving the LoD concept is currently under consideration [Löwner and Gröger, 2016]. Concepts of this version (CityGML 3.0) include a new LoD concept in which the buildings are not longer central, but the building features [Biljecki et al., 2016a]. Meaning that the building features, such as the rooms and the storeys, can be modeled in different LODs. This LoD concept would make it possible to store the *BAG* geometries in the appropriate manner, however it is still not ideal because of the relation schema of the CityGML buildings.

6.2 HOW TO FIT THE 3D BAG WITHIN THE CURRENT CITYGML STANDARD

Multiple options exist to include information not defined by the CityGML standard: (1) Store as Generic objects; (2) develop an application domain extension (ADE); (3) store the information as aggregates; (4) extend the CityGML 2.0 schema. Examples of these extension and their application to model the *BAG* objects will be discussed in the following sections.

6.2.1 Generic objects and attributes

For objects and attributes that not belonging to any predefined classes it is possible to create `Generic objects and Attributes`. As such, the building units can be stored as a new class. However, this class will not have a relation within the CityGML scheme, which is undesirable as the relation between the unit and building is inextricably connected.

6.2.2 Application Domain Extension

It is possible to extent the CityGML standard with an Application Domain Extension (ADE). This is an extra formal schema based on the CityGML schema definitions. The ADE allows the definition of classes, their relationships and attributes. For applications that require a large number of new features, the creation of an ADE is recommended, i.e. ADEs are available for noise and flood models. The ADE concept differs from the `Generic objects and Attributes` by the fact that an ADE has to be defined in an extra XML schema definition file with its own namespace. This file has to explicitly import the XML schema definition of the extended CityGML modules [van den Brink et al., 2013b].

As discussed in section 3.2.4, IMGeo is developed as a specialization of CityGML. This ADE extends CityGML with the Dutch 2D national Information Model for large-scale Geo-information (IMGeo) [van den Brink et al., 2013b]. This made it possible to extent the 2D data into 2.5D and 3D with the same principles as CityGML. The Unified Modeling Language (UML) diagram of the building class for the IMGeo shows which features were added to fit the building information within the CityGML standard. A new ADE element is created and called `Pand`. This element is related to the original CityGML feature type `BuildingPart`. Despite the link between the *BAG* and the *BGT*, no ADE element for the objects are included. This directly results in a problem within the schema; whereas the UML diagram shows that there is a connection between the feature: Address and the _AbstractBuilding, in the Netherlands a building is not an addressable object, only a unit.

ADEs have been proposed that can store units within buildings. The ADE presented by Çada [2012] can be used for the storage and exchange of immovable property tax records based on the Turkish Civil Law. By the ADE, the storage of condominium units that are part of a building is possible, in a similar way as the *BAG* units. In addition, it is possible to store joint facilities and cadastral parcels.

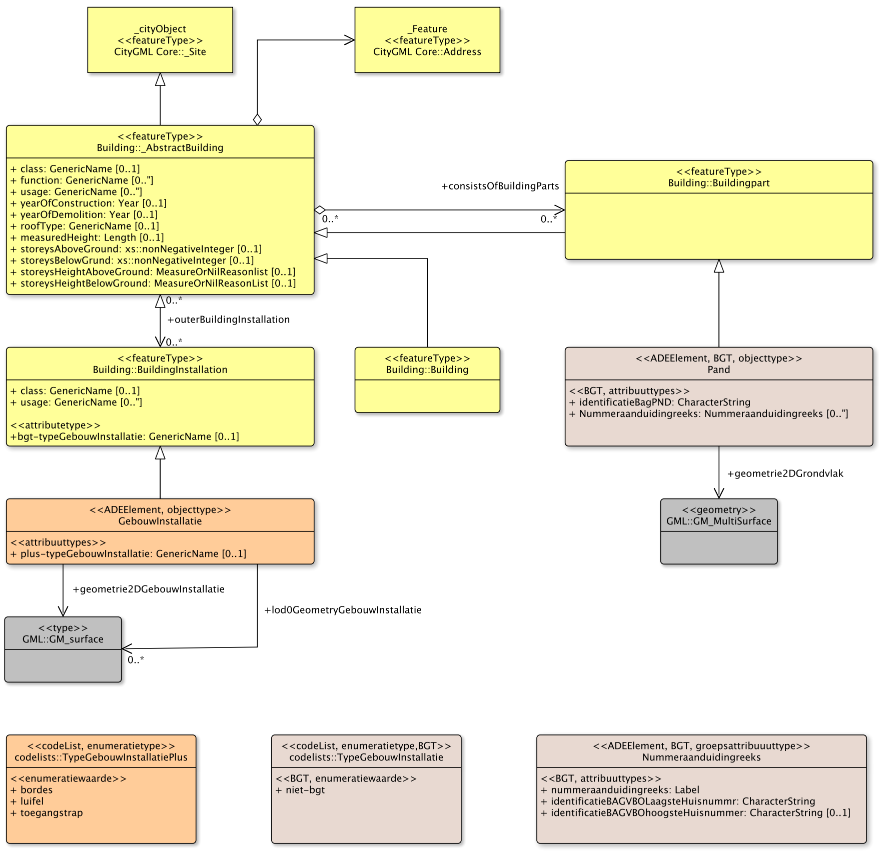

_cityObject
<<featureType>>
CityGML Core::_Site
_Feature
<<featureType>>
CityGML Core::Address
<<featureType>>
Building::_AbstractBuilding
+ class: GenericName [0..1]
+ function: GenericName [0.."]
+ usage: GenericName [0.."]
+ yearOfConstruction: Year [0..1]
+ yearOfDemolition: Year [0..1]
+ roofType: GenericName [0..1]
+ measuredHeight: Length [0..1]
+ storeysAboveGround: xs::nonNegativeInteger [0..1]
+ storeysBelowGrund: xs::nonNegativeInteger [0..1]
+ storeysHeightAboveGround: MeasureOrNilReasonlist [0..1]
+ storeysHeightBelowGround: MeasureOrNilReasonList [0..1]
+consistsOfBuildingParts
0..*
0..*
<<featureType>>
Building::Buildingpart
0..*
+outerBuildingInstallation
0..*
<<featureType>>
Building::BuildingInstallation
+ class: GenericName [0..1]
+ usage: GenericName [0.."]
<<attributetype>>
+bgt-typeGebouwInstallatie: GenericName [0..1]
<<featureType>>
Building::Building
<<ADEElement, BGT, objecttype>>
Pand
<<BGT, attribuuttypes>>
+ identificatieBagPND: CharacterString
+ Nummeraanduidingreeks: Nummeraanduidingreeks [0.."]
+geometrie2DGrondvlak
<<geometry>>
GML::GM_MultiSurface
<<ADEElement, objecttype>>
GebouwInstallatie
<<attribuuttypes>>
+ plus-typeGebouwInstallatie: GenericName [0..1]
+geometrie2DGebouwInstallatie
+lod0GeometryGebouwInstallatie
<<type>>
GML::GM_surface
0..*
<<codeList, enumeratietype>>
codelists::TypeGebouwInstallatiePlus
<<enumeratiewaarde>>
+ bordes
+ luifel
+ toegangstrap
<<codeList, enumeratietype,BGT>>
codelists::TypeGebouwInstallatie
<<BGT, enumeratiewaarde>>
+ niet-bgt
<<ADEElement, BGT, groepsattribuuttype>>
Nummeraanduidingreeks
<<BGT, attribuuttypes>>
+ nummeraanduidingreeks: Label
+ identificatieBAGVBOLaagsteHuisnummr: CharacterString
+ identificatieBAGVBOhoogsteHuisnummer: CharacterString [0..1]

Figure 6.2: UML diagram of the IMGeo. The colors indicate which classes belong to CityGML (yellow), BGT (beige) or IMGeo-optional (orange). Diagram adjusted from van den Brink et al. [2013c]

.

6.2.3 Store model as LOD4 and create `CityObjectGroups`

A workaround has been developed for cases that the LoD concept does not handle. CityGML 2.0 provides a method to group objects in case of building complexes by aggregating them using the concept of *CityObjectGroups*. This concept can also be used to aggregate objects that lie on the same floor to create storeys, such as Rooms, Doors, Windows, IntBuildingInstallations and BuildingFurniture Gröger and Plümer [2012]. In order to use this concept, more detailed features should be included in the model. This aggregation concept provides the possibility to model the unit-storeys as Rooms and aggregating these in objects. When improving the model by including the real rooms, this aggregation can still be used. However, the inclusion of the real rooms desires the movement of the complete *BAG* to the LOD4 level at once. Otherwise it will not be possible to possible to include all the units. This will be impossible in practice.

6.2.4 My proposal for extending the CityGML standard

The IMGeo provides a solid structure for storing topographical information. As the *BAG* buildings are currently already part of the IMGeo, my proposal is to extent the IMGeo schema so it can include both units and buildings. Shared spaces within a building will be included in the schema, providing a logic step for improving the model as discussed in Section 3.4.

Within the building class, two new ADE elements are introduced; the Units and the Shared-Space (Figure 6.3). These elements are part of the building; without the building it is not possible to model these elements. However, a building may or may not include objects or shared spaces. The predefined attributes for objects will mostly be the same as in the current situation, but there are small differences. The CityGML feature type Address is connected to the units, instead of the _AbstractBuilding, so it fits the Dutch situation. It should be noted that the usage defined in _AbstractBuilding is different to the usage defined for objects. The latter uses the classification defined in the *BAG*. In this way, it is possible to classify different uses within one building, which is currently not the case. The LoD concept discussed in Section 3.4 is included in this diagram.

The proposed extension of CityGML IMGeo will enable the storage of buildings and objects. Because the *BGT* is currently already available in this format, the two datasets can be connected directly.

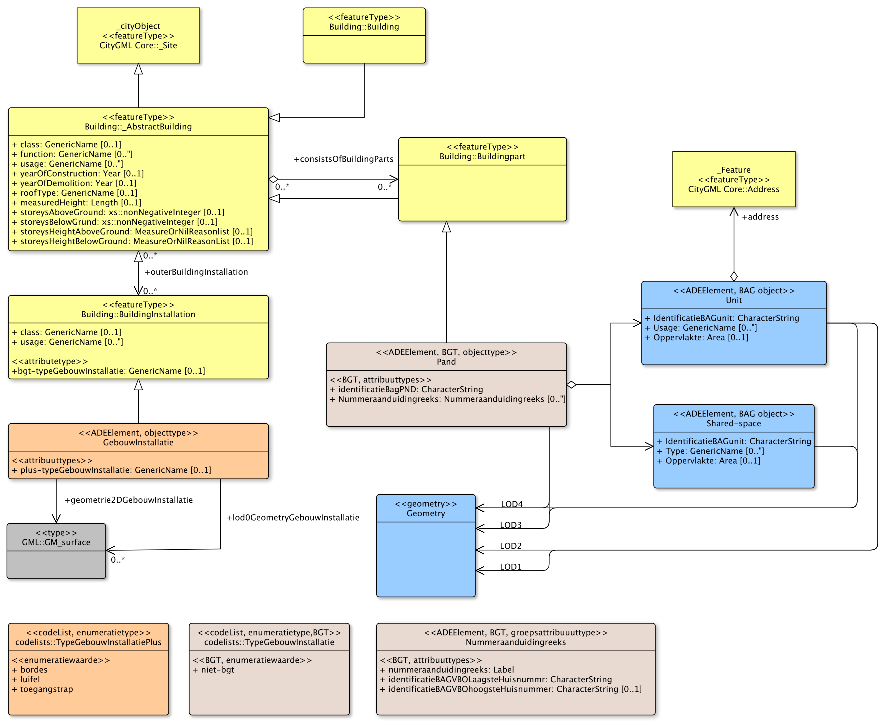

_cityObject
<<featureType>>
CityGML Core::_Site
<<featureType>>
Building::Building
<<featureType>>
Building::_AbstractBuilding
+ class: GenericName [0..1]
+ function: GenericName [0.."]
+ usage: GenericName [0.."]
+ yearOfConstruction: Year [0..1]
+ yearOfDemolition: Year [0..1]
+ roofType: GenericName [0..1]
+ measuredHeight: Length [0..1]
+ storeysAboveGround: xs::nonNegativeInteger [0..1]
+ storeysBelowGrund: xs::nonNegativeInteger [0..1]
+ storeysHeightAboveGround: MeasureOrNilReasonlist [0..1]
+ storeysHeightBelowGround: MeasureOrNilReasonList [0..1]
+consistsOfBuildingParts
0..*
0..*
<<featureType>>
Building::Buildingpart
_Feature
<<featureType>>
CityGML Core::Address
+address
0..*
+outerBuildingInstallation
0..*
<<featureType>>
Building::BuildingInstallation
+ class: GenericName [0..1]
+ usage: GenericName [0.."]
<<attributetype>>
+bgt-typeGebouwInstallatie: GenericName [0..1]
<<ADEElement, BAG object>>
Unit
+ IdentificatieBAGunit: CharacterString
+ Usage: GenericName [0.."]
+ Oppervlakte: Area [0..1]
<<ADEElement, BGT, objecttype>>
Pand
<<BGT, attribuuttypes>>
+ identificatieBagPND: CharacterString
+ Nummeraanduidingreeks: Nummeraanduidingreeks [0.."]
<<ADEElement, BAG object>>
Shared-space
+ IdentificatieBAGunit: CharacterString
+ Type: GenericName [0.."]
+ Oppervlakte: Area [0..1]
<<ADEElement, objecttype>>
GebouwInstallatie
<<attribuuttypes>>
+ plus-typeGebouwInstallatie: GenericName [0..1]
+geometrie2DGebouwInstallatie
+lod0GeometryGebouwInstallatie
<<type>>
GML::GM_surface
0..*
<<geometry>>
Geometry
LOD4
LOD3
LOD2
LOD1
<<codeList, enumeratietype>>
codelists::TypeGebouwInstallatiePlus
<<enumeratiewaarde>>
+ bordes
+ luifel
+ toegangstrap
<<codeList, enumeratietype,BGT>>
codelists::TypeGebouwInstallatie
<<BGT, enumeratiewaarde>>
+ niet-bgt
<<ADEElement, BGT, groepsattribuuuttype>>
Nummeraanduidingreeks
<<BGT, attribuuttypes>>
+ nummeraanduidingreeks: Label
+ identificatieBAGVBOLaagsteHuisnummr: CharacterString
+ identificatieBAGVBOhoogsteHuisnummer: CharacterString [0..1]

Figure 6.3: The proposed schema for extending the IMGeo. The objects colored in blue are added to the current schema to fit the objects. The object colored in beige belong to the IMGeo extension.

7 CONCLUSIONS AND FUTURE WORK

In this thesis, I have investigated problems with the current state of the *BAG* representation and the needs and opportunities for a 3D model of the *BAG*. In addition, I have proposed one complete model in which the key registers that contain spatial information are combined (Chapters 2 and 3). Thereafter, I have investigated the possibility of automatically creating a LOD1 model of the *BAG* units, using the raw LIDAR points of the *AHN3* dataset and the current 2D geometry of the *BAG*. Although it is quite common to create a LOD1 model from a LIDAR and 2D geometries in practice, I found multiple uncertainties that influenced the accuracy of the created model. Furthermore, I have identified multiple problems that occur when applying the method used in practice on the *BAG* geometry. Therefore, I have developed a methodology that not only created a rough version of the 3D geometry of the units, but also identified uncertainties and errors in the *BAG* (Chapter 4). The uncertain parameters for the storey height and the threshold for the complexity of a building, that ensures an accurate result were calculated using a reference dataset. A workflow was implemented with the *BAG*, the *AHN3* and the building geometry of the *BGT* as input. After identifying and excluding building with uncertain locations, complex geometries or buildings that include complex units, the implementation creates a 3d model including the unit-floors. These units were subsequently validated to ensure the geometrically correctness of the created geometries (Chapter 5). Finally, a proposal was created on how the current standard for the *BAG* can be extended so it can include the units and their 3D geometry (Chapter 6). The most important conclusions of the thesis will be summarized (Section 7.1), and future work (Section 7.2) and recommendations (Section 7.3) will be proposed.

7.1 CONCLUSION

This conclusion section provides answers on the research questions proposed in Section 1.1. First the sub research questions will be answered, After which the main research question will be addressed.

What is the 3D BAG?
When I started this reserach there was no definition for the 3D *BAG*. Hence, as discussed in Section 3.3, efforts are being made with the aim of creating 3D geometries of the *BAG*. The municipality of The Hague combined a LOD2 citymodel with their extended version of the *BAG* and also the municipality of Rotterdam is combining their *BAG*+ with 3D models. In these cases, the aim is to create geometries for the units within the building. However, it was not yet defined how the data should be modelled and what would be included. By exploring possible use cases and from discussions with experts, it became clear what should be included in the 3D *BAG*. In section 3.4

I proposed my version for the 3D *BAG*. In my proposed model, the *BAG* geometry is part of a model that also includes the spatial data from the other key registers. Importantly, a distinction is made between the geometry of buildings and units in my proposed concept. However, there is still a relation because the volumes of the units are bounded by the exterior of the buildings when modelled in 3D. This distinction between building exterior and the units enables the enhancement of the amount of detail in case more information is available for both exterior as interior, i.e. doors and shared spaced can be included.

Why are municipalities interested in a 3D version of the BAG?
There are different reasons why the municipalities are interested in a 3D version of the *BAG*. The drawbacks of the current *BAG* representations (Section 3.1) all involve uncertainty about the location and geometry of units and buildings. From the geometry (and attributes), it is not clear how the building relates to its surroundings or how the units are located within the building. A 2D representation can not model the complexity of the build environment in an unambiguous way. The interest in a 3D version of the *BAG* does not only has his origin in the drawbacks of the current representation. When the buildings and units are modelled in 3D, new opportunities for the *BAG* will arise (Section 3.4.3). A 3D model would make it possible to combine the *BAG* with other datasets, such as the *BGT*. Especially when the units are modelled separately, more accurate predictions can be done regarding spatial phenomena, such as noise and air pollution. Instead of the building being the center point of these predictions, the units were people stay become the subject. However, the municipalities also believe the opportunities of the 3D BAG for other organizations within the government, for instance emergency services the 3D model can be used for navigation within buildings, but the 3D model alone can already provide better insight in the situation. Moreover, the tax authorities are also interested in a 3D *BAG* due to the possibility that arises to detect fraud or errors in their data.

How can underground structures be detected?
Within the *BAG* there is no distinction between underground and aboveground structures. Two methods are combined in order to provide certainty about the location of the units. The LIDAR method is based on the classification of the LIDAR points and the density of these points. When there are more other points than building points, the building is considered to lie underground. The BGT method is based on the fact that the fact that buildings that are underground underground or that do not have a geometry on ground level are not stored in the *BGT*. Both methods have their uncertainties. However, by combining these methods buildings that lie completely underground and buildings that lie above ground (with or without a basement) can be identified with certainty.

How can the number of storeys of a building be calculated?
To calculate the number of storeys, different methods can be used. In Section 4.2.1 the current practice and research is discussed on these methods. In general there are two approaches. The first approach is based on the internal nett floor area of the building. By using a factor to convert this area to the gross floor area, an approximation of the number of floors can be given when divided by the area of the polygon. However, the accuracy of this method is discovered to be poor due to differences between the nett

floor area and the gross floor area. Hence, a conversion table based on the year and use of a unit is not accurate in practice. The second approach is based on extracting height data from a LIDAR dataset a dividing this height by an average storey height. Which height value for the building has to be extracted: the gutter height, median, maximum or average, is not defined. The storey height by which the building height has to be divided is also unclear. Biljecki et al. [2017] describes that there is a connection between attributes like building year and use of a building and the average storey height, but reference data is needed to define the parameters that enable the use of this knowledge.
For calculating the number of storeys of a building, the maximum height (top 90th percentile) is extracted from the LIDAR points that are classified as building points and are located within the building polygon. The minimum height of the building is extracted using the median of the ground points located around the building. By subtracting the maximum and minimum height, the building height is calculated. This height is divided by the average storey height calculated from a reference dataset.
Because a LOD1 model is created, there are cases in which this method is not sufficient. To make sure that the height extracted from the points belongs to the right building, building polygons that have large overlaps with other building polygons have to be excluded.
The difference between the maximum and median height of the buildings points influences the accuracy of the calculated number of storeys. A threshold was introduced regarding the difference of this maximum and median of the building points. When comparing the results with a reference dataset, a threshold of 1.0 would give in 95% of the cases a correct number of storeys. Hence, 85% of the building was excluded with this threshold. With a threshold of 3.0, the number of storeys was calculated right for 83% of the buildings.

What logic can be used for placing units on building storeys?
The first step is to find the related units of the buildings. The relation between the buildings and the units is defined both geometrically and by an identifier. Problems can occur regarding the location of the points. As such, the use of the identified is preferred. As discussed in Section 3.1.3, locating the units by using the location of the point geometries is not possible. Hence, a new method for placing the units had to be developed.

Although the nett floor area of the units is not directly related with the gross floor area of the building, the difference between the nett floor area of multiple units can be used to locate the unit within the building. By combining the ratios between the units with the number of storeys, it is possible to estimate how much of a unit occupies a storey. For example: in cases with two units within a three storey building, if the first unit has an nett floor area that is two times bigger as the second unit. It can be assumed that this unit occupies two storeys and the other unit only one.
To determine on which floors the units lie, the units should be ordered. Attributes used for this ordering include the house letter, the addition, the function, the area and the house number. Based on these attributes an order was created to divide the units over the storeys (Section 4.4.1).

How can floors be divided using the information within the BAG? Dividing the storeys in order to create the unit-storeys is strongly connected to the storeys of the units in the developed method. The area of the units can be used to divide the polygon (Section 4.5. Creating a centerline from the straight skeleton of the polygon, to place the units gives visually good results. However, in case of complex buildings the methodology can result in problems, visually and regarding the created area size (Section 5.8.3). More information is needed to improve the accuracy of the created storeys.

Which uncertainties in the BAG and lack of information can not be solved with information from other sources? There are uncertainties within the *BAG* that cannot be solved with the use of other available information. The location of the building cannot be unambiguously determined in every case. When combining the developed LIDAR and BGT method, buildings that lie partly underground or are located above the first level can not be identified as such (Section 4.1). However, even when both buildings are located above ground uncertainties occur. When building polygons intersect, it is not clear how the buildings are intersecting in reality. The main uncertainty in the *BAG* is the location of the units. In Section 3.1.3 this uncertainty is discussed. Currently there is no open data available that can provide certainty about the storey level or the location of the units within the storey. The created methodology is only based on assumptions and heuristics.

Which uncertainties had the most influence on the automatic creation of the 3D BAG? Because I created a LOD1 version of the *BAG*, it was not possible to create 3D geometries for every building. The buildings that had to be excluded because of complexity within the building was large, almost 28% (Section 5.5.4). Moving from LOD1 to LOD2 will solve this problem. As there was no information available about the location of the units with the buildings, this influences the final result. Compared to the reference set, 81% of the units that were modelled were located on the right storey (Section 5.7.3). This is a high numbers, because most buildings only have one or two related units. No information about the location of the units on the storeys was available, making it impossible to validate the results.

Is it possible to combine the 3D geometries of the BAG with other key registers? As proposed in Section 3.4, combining the 3D *BAG* geometries with other spatial data of the key registers would create new opportunities for the usage of the data. Although the *BGT* is currently integrated with the CityGML standard that enables the storage of 3D city models, it is currently not possible to include the created 3D *BAG* geometries within this model. The main reason is that feature type for units is currently missing. In Section 6.2.4 I proposed an extension of the current CityGML IMGeo ADE which would enable the storage of 3D solids of the units.

How to define the 3D BAG and to which extent can the current available open data be used to create 3D geometries of the BAG units?
The 3D *BAG* is the geometrical part of the *BAG* which models the exterior of the building, however it also includes the units and possibly the features in the shared spaces, such as the staircases and entrances. The geometries of the 3D *BAG* have a geographical location that enables the connection with the other key registers, such as the *BGT*.
In this research I explored the possibility of creating 3D geometries of the *BAG* units from open data. The uncertainties in the *BAG* and the lack of information about the buildings (the number of storeys, the location, etc.) and

the units (storey level, location on storey) made the creation of a fully accurate 3D model impossible. Hence, the developed methodology proved to to enable a correct 3D model of most buildings good results for most buildings. The largest part of the *BAG* buildings only have 1 or 2 related units (Section 5.6.2). And when dividing the units over storeys, the result showed that almost 75% of the storeys was only related to one unit (Section 5.7.3).
However, during my investigation of the possibility to create a 3D *BAG*, many uncertainties within the *BAG* were identified (Section 3.1). Although the created methodology and implementation could be further improved, the uncertainties and lack of information about the location of the units will always considerably influence the accuracy of the final result.

7.2 FUTURE WORK

In this section the future work will be discussed for the creation of the 3D geometries of the *BAG*. The suggestions that are provided in this section are only related to the developed methodology, not on the data acquisition. This will be discussed in Section7.3.1. These suggestions would improve the current result with the same data.

7.2.1 Storey height based on parameters

As discussed in Section 4.2.1, a parameter for the storey height based attributes, such as the year of construction, the function and the location, would possibly increase the accuracy of the calculated number of storeys. More research is needed to know the exact influences of these attributes on the storey height. Hence, more reference data should be acquired and researched.

7.2.2 Improving the city model

The next step would be to improve the level of detail of the building exterior. As discussed in Section 4.2.1, a lot of research has been performed on creating 3D city models from LIDAR, photogrammetry and building footprints. When the roofs of the buildings are modelled in more detail, also the storeys can be created more accurate. This would also enable the calculation of the floor area of the different storeys, which could then be used for placing the units. It would then be unnecessary to exclude complex buildings. However, the research aiming at improving the level of detail of the buildings do not always take into account the complex building configurations. For a full correct model these cases should also be modelled.

7.2.3 Partitioning

The developed partitioning method should be improved in order to create a more accurate approximation of reality. Area partitioning is a complex matter and not much research is performed related to the partitioning of floor plans. The partitioning based on the nett floor area of the units alone will not always provide the desired result. Two approaches will be discussed on how the partitioning can be improved.

To improve the current method, the partitioning could combine more knowledge about the building. An example is to create building compartments with the use of directional lines (Section 4.5.1). These building compartments would then be divided into smaller units with 'building logic'. The location and direction of the centerline should be designed in a way that residential units are always connected to an outer wall. Other restrictions could be included, such as the fact that building units are not smaller than 3 meters and the cutline is always perpendicular to the centerline.

Another approach would be to firstly identify the type of a building. Building primitives can be developed that include the topological relations between the entrances, shared spaces and units. This type could then be used to divide the building in a specific way. For example, gallery flats for example have a common floor plan. The topology between the main entrance, the staircases, the galleries and the residential units will be exactly the same in most cases. This knowledge could be used to partition the buildings and would enable the estimation of the location of shared spaces and entrances.

7.3 RECOMMENDATIONS

In this Sections recommendations are posed for the creation of a nationwide 3D *BAG*. The main focus is on the data acquisition and the possibility to combine the *BGT* with the *BAG*.

7.3.1 Data acquisition

There are many uncertainties in the *BAG* that cannot be solved using other data sources. To enable the creation of an accurate 3D *BAG* possible, data should be acquired with that purpose in mind. Not all data has to be acquired in the field. Combining the *BAG* with other datasets (Section 2.2.2) will make the creation of the 3D *BAG* more straightforward. However, this data will not be available in every municipality so an inventory of the available information should be made first.

To create the first level of detail proposed in section 3.4, only the number of storeys of a building is required and on which storey the units lie. This information may already be available in certain municipalities (the reference dataset provided by the municipalitie of The Hague includes this information). This data can be acquired with relatively small effort and will drastically improve the accuracy of the model.

To create the second level of detail, the geometry of the unit becomes important. In this level, the location of the unit is topologically ordered (neighbors are next to each other). The point geometry of the units will be sufficient in most cases, where the points are logically placed, e.g. locating the points in the middle of the units. This is already common practice in some municipalities, as is the case for the storey of the units. However, a defined standard for placing the points is desirable when using this information for the placement of the units.

The third and fourth level of detail include the shared spaces and the entrances of a building. To achieve this accuracy, more detailed information about the building is needed. For a fully accurate model, data has to be acquired manually or reconstructed from building plans. Hence, using the second approach discussed in 7.2.3, it would also be possible to approximate the reality by only creating topologic graphs from the building. However, more research is needed on the feasibility of this proposed method.

7.3.2 Combine BGT and BAG geometrically

To make it possible to store the data and adding value to both datasets the *BGT* and *BAG* can be combined. The difference between how the buildings are modelled (Section 2.3) will no longer cause interoperability problems when a the geometry moves to a 3D representation.
Combining the dataset is currently not possible, however the CityGML IMGeo standard provides a solid base that can be further extended (Section 6.1). The LoD concept of CityGML should be adjusted in order to store the geometry of the units without the need for storing the building geometry in LOD4. When these adjustments are made and the 3D *BAG* geometries are included within the model, new possibilities will arise for many current and future users of these key registers.

BIBLIOGRAPHY

Aichholzer, O., Aurenhammer, F., Alberts, D., and Gärtner, B. (1995). A novel type of skeleton for polygons. *Journal of Universal Computer Science*, 1(12):752–761.

Alahmadi, M., Atkinson, P., and Martin, D. (2016). Estimation of the Spatial Distribution of Urban Population using Remotely Sensed Satellite Data in Riyadh , Saudi Arabia. 44(0).

Armaselu, B. and Daescu, O. (2015). Algorithms for fair partitioning of convex polygons. *Theoretical Computer Science*, 607:351–362.

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., and Ives, Z. (2007). *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings*, chapter DBpedia: A, pages 722–735. Springer Berlin Heidelberg, Berlin, Heidelberg.

Aurenhammer, F. (1991). Voronoi Diagrams A Survey of a Fundamental Data Structure. *ACM Computing Surveys*, 23(3):345–405.

Bast, H. and Hert, S. (2000). The Area Partitioning Problem. *12th Canadian Conference on Computational Geometry*, pages 163–172.

Biljecki, F., Ledoux, H., and Stoter, J. (2016a). *An improved LOD specification for 3D building models*, volume 59.

Biljecki, F., Ledoux, H., and Stoter, J. (2017). Generating 3D city models without elevation data. *Computers, Environment and Urban Systems, Under review*.

Biljecki, F., Ledoux, H., Stoter, J., and Vosselman, G. (2016b). The variants of an LOD of a 3D building model and their influence on spatial analyses. *ISPRS Journal of Photogrammetry and Remote Sensing*, 116:42–54.

Boeters, R. (2013). Automatic enhancement of CityGML LoD2 models with interiors and its usability for net internal area determination. Technical report, Tu Delft.

Boeters, R., Arroyo Ohori, K., Biljecki, F., and Zlatanova, S. (2015). Automatically enhancing {CityGML} {LOD2} models with a corresponding indoor geometry. *International Journal of Geographical Information Science*, pages 1–21.

Broeke, J. (2016). NLExtract. Retrieved May 5th, 2016, from http://www.nlextract.nl/.

Çada, V. (2012). An application domain extension to citygml for immovable property taxation: A Turkish case study. *International Journal of Applied Earth Observation and Geoinformation*, 21(1):545–555.

Commandeur, T. J. F. (2012). Footprint decomposition combined with point cloud segmentation for producing valid 3D models. (March).

de Berg, M., van Kreveld, M., Overmars, M., and Schwarzkopf, O. (2000). *Computational Geometry*, volume 28.

Díaz-Vilariño, L., Khoshelham, K., Martínez-Sánchez, J., and Arias, P. (2015). 3D modeling of building indoor spaces and closed doors from imagery and point clouds. *Sensors (Basel, Switzerland)*, 15(2):3491–3512.

Digitale Overheid (2016). Stelsel van Basisregistraties.

Elberink, S. O., Stoter, J., Ledoux, H., and Commandeur, T. (2013). Generation and Dissemination of a National Virtual 3D City and Landscape Model for the Netherlands. 79(2):147–158.

Ellenkamp, Y., Rietdijk, M., and Hulscher, B. (2007). Oppervlakte Verdiepingsdocument voor gemeenten. Technical Report Versie 2.0 (december 2007), Ministerie van VROM, Den Haag.

Emgård, L. and Zlatanova, S. (2007). Design of an integrated 3D information model. *Urban and regional data management: UDMS annual*, pages 143–156.

Farin, G., Hoffman, D., and Johnson, C. R. (2006). *Triangulations and Applications*.

Fodor, Z. (2015). ArtStation - 3D City model, Zoltán Fodor.

Gemeente Amsterdam (2016a). Functiekaart in 3D. Retrieved May 6th, 2016, from https://pbs.twimg.com/media/Cg{_}43l1iUkAAwbNm.jpg:large/.

Gemeente Amsterdam (2016b). Objectklasse verblijfsobject. Retrieved April 11, 2016, from https://www.amsterdam.nl/stelselpedia/bag-index/catalogus-bag/objectklasse-0/.

Gemeente Rotterdam (2011). Rotterdam 3d. Retrieved April 5, 2016, from http://www.rotterdam.nl/rotterdam{_}3d.

Geodan (2016). Dynamisch 3D model voor Gemeente Den Haag - Geodan. Retrieved July 17, 2016, from http://www.geodan.nl/dynamisch-3d-model-gemeente-haag/.

Geonovum (2015). Eenvoudiger realiseren van functiekaart plangebied. Retrieved April 5th, 2016, from http://www.geonovum.nl/wegwijzer/inspiratie/eenvoudiger-realiseren-van-functiekaart-plangebied.

Google (2016). Ammunitiehaven. Retrieved April 4th, 2016, from https://www.google.nl/maps/@52.077103,4.3201057,3a,75y,30.08h,94.12t/data=!3m6!1e1!3m4!1sbYcIOHebcBVBVQEapY3suw!2e0!7i13312!8i6656.

Google Maps (2016). Den Haag Centraal - Google Maps. Retrieved April 4th, 2016, from https://www.google.nl/maps/place/Den+Haag+Centraal/@52.0812338,4.323752,18z/data=!3m1!5s0x47c5b71775dcaf9b:0xcfe11fee4473a618!4m5!3m4!1s0x47c5b7179eb1cb57:0x45853774d5de7d6d!8m2!3d52.0812762!4d4.3239063.

Gröger, G. and Plümer, L. (2012). CityGML Interoperable semantic 3D city models. *ISPRS Journal of Photogrammetry and Remote Sensing*, 71:12–33.

Hfb (2016). Project Leidsche Rijn Centrum. Retrieved July 7, 2016, from http://www.hfb-groep.nl/LeidscheRijnCentrum.

Hong, S., Jung, J., Kim, S., Cho, H., Lee, J., and Heo, J. (2015). Semi-automated approach to indoor mapping for 3D as-built building information modeling. *Computers, Environment and Urban Systems*, 51:34–46.

ISO (2003). ISO 19107:2003: Geographic informationSpatial schema.

Kada, M. (2006). Generalization of 3D Building Models for Map-Like Presentations. *Archives*, (2002).

Kada, M. and McKinley, L. (2009). 3D Building Reconstruction from LIDAR based on a Cell Decomposition Approach. *CMRT09: Object Extraction for 3D City Models, Road Databases and Traffic Monitoring - Concepts, Algorithms and Evaluation*, XXXVIII:47–52.

Kadaster (2016). Wereldprimeur: inschrijving met rechten in 3D.

Khetarpal, S. (2014). Dividing A Polygon In Any Given Number Of Equal Areas. Retrieved Juanuari 2th, 2016, from http://www.khetarpal.org/polygon-splitting/.

Lafarge, F. and Mallet, C. (2012). Creating large-scale city models from 3D-point clouds: A robust approach with hybrid representation. *International Journal of Computer Vision*, 99(1):69–85.

Ledoux, H. (2013). On the validation of solids represented with the international standards for geographic information. *Computer-Aided Civil and Infrastructure Engineering*, 28(9):693–706.

Lee, J. and Zlatanova, S. (2008). A 3D data model and topological analyses for emergency response in urban areas. *Geospatial Information Technology for Emergency Response*, pages 143–167.

Lingas, A., Lingas, A., Pinter, R., Pinter, R., Rivest, R., Rivest, R., Shamir, A., and Shamir, A. (1982). Minimum edge length partitioning of rectilinear polygons.

Löwner, M. O. and Gröger, G. (2016). Evaluation Criteria for Recent LoD Proposals for City-GML Buildings. *Photogrammetrie - Fernerkundung - Geoinformation*, 1:31–43.

Microsoft (2016). Sebastiaansplein. Retrieved July 22, 2016, from http://www.bing.com/maps/?FORM=Z9LH3.

Ministerie van Binnenlandse Zaken en Koninkrijksrelaties (2014). Interactieve Stelselplaat. Retrieved May 6, 2016, from https://www.digitaleoverheid.nl/onderwerpen/stelselinformatiepunt/stelselthemas/verbindingen/verbindingen-tussen-basisregistraties.

Ministerie van Binnenlandse Zaken en Koninkrijsrelaties (2016). Basisregistratie Grootschalige Topografie (BGT). Retrieved May 10, 2016, from http://www.digitaleoverheid.nl/onderwerpen/stelselinformatiepunt/stelsel-van-basisregistraties/1493-basisregistratie-grootschalige-topografie-bgt.

Ministerie van Infrastructuur en Milieu (2016). Basisregistraties Adressen en gebouwen. Retrieved June 4, 2016, from http://www.basisregistratiesienm.nl/basisregistraties/adressen-en-gebouwen.

Rietdijk, M. (2009). Catalogus basisregistraties adressen en gebouwen. Technical report, Ministerie van Volkshuisvesting, Ruimtelijk Ordening en Milieubeheer, Den Haag.

Rottensteiner, F. and Briese, C. (2002). A new method for building extraction in urban areas from high-resolution LIDAR data. *International Archives of Photogrammetry and Remote Sensing*, XXXIV(3):295–301.

Skiena, S. S. (2008). *The Algorithm Design Manual*, volume 1.

Statisfact (2015). Tevredenheidsonderzoek BAG 2015. Technical report, Statisfact, Utrecht.

Thomas H. Kolbe, Gerhard König, C. N. (2011). Advances in 3D Geo-Information Sciences. *Springer*, page 310.

van Boxtel, R. H. L. M. (2001). Brief minister over de voortgang van het programma Stroomlijning Basisgegevens.

van den Brink, L., Stoter, J., and Zlatanova, S. (2013a). Establishing a national standard for 3D topographic data compliant to CityGML. *International Journal of Geographical Information Science*, 27(1):92–113.

van den Brink, L., Stoter, J., and Zlatanova, S. (2013b). UML-based approach to developing a citygml application domain extension. *Transactions in GIS*, 17(6):920–942.

van den Brink, L., van Eekelen, H., and Reuvers, M. (2013c). Basisregistratie Grootschalige Topografie: Gegevenscatalogus IMGeo 2.1.1.

van Duivenboden, H. and de Vries, M. (2003). Stroomopwaarts! Kroniek van het Programma Stroomlijning Basisgegevens. Technical report, Den Haag.

Verma, V., Kumar, R., and Hsu, S. (2006). 3D building detection and modeling from aerial LIDAR data. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2, pages 2213–2220.

Visser, P. and Van Dam, F. (2006). De prijs van de plek. Technical report, Ruimtelijk Planbureau, Den Haag.

Wagner, D., Wewetzer, M., Bogdahn, J., Alam, N., Pries, M., and Coors, V. (2013). Progress and New Trends in 3D Geoinformation Sciences. *Progress and New Trends in 3D Geoinformation Sciences*, pages 299–314.

Wikimedia Commons (2007a). Delft Poptahof. Retrieved May 22, 2016, from https://commons.wikimedia.org/wiki/File:Delft{_}Poptahof.JPG.

Wikimedia Commons (2007b). Spui. Retrieved May 22, 2016, from https://commons.wikimedia.org/wiki/File:Spui.JPG.

Wikimedia Commons (2014). Unilever "De Brug". Retrieved May 22, 2016, from https://commons.wikimedia.org/wiki/File:Unilever{%}22The{_}Brug{%}22.JPG.

COLOPHON

This document was typeset using LATEX. The document layout was generated using the `arsclassica` package by Lorenzo Pantieri, which is an adaption of the original `classicthesis` package from Andr Miede.