Student thesis series INES nr 557

Creating a workflow of 3D building data in a municipality context

Tom Arnold Bendiksen

2021

Department of
Physical Geography and Ecosystem Science
Lund University
Sölvegatan 12
S-223 62 Lund
Sweden

Tom Arnold Bendiksen (2021).

Creating a workflow of 3D building data in a municipality context

Master degree thesis, 30 credits in *Geomatics*

Department of Physical Geography and Ecosystem Science, Lund University

Level: Master of Science (MSc)

Course duration: *November* 2020 until *June* 2021

Creating a workflow of 3D building data in a municipality context

---

Tom Arnold Bendiksen

Master thesis, 30 credits, in *Geomatics**

Lars Harrie

Department of Physical Geography and Ecosystem Science

Barzan Abdi

Malmö municipality

Exam committee:

Per-Ola Olsson, Department of Physical Geography and Ecosystem Science
Karolina Pantazatou, Department of Physical Geography and Ecosystem Science

Acknowledgements

First and foremost, I would like to thank my supervisor Lars Harrie for consistent and excellent guidance throughout this master thesis. Without your support I never would have been able to create a report at this level. Secondly, a big thanks goes to Barzan Abdi at Malmö municipality. From the day I started my internship at the city planning office you gave me the trust and tools I needed to develop myself in subjects regarding 3D city and building models. The internship ultimately led to the cooperation and implementation of this master thesis. Fredrik Nilsson at Kristianstad municipality also deserves a huge recognition for his support during the practical implementations of this thesis. You have been a great source of knowledge for me when learning about the structure and rules for 3D building data.

Abstract

3D city models are valuable tools for visualization and analytical purposes in the urban environment. A popular format for city models is CityGML, an open data model that can be used for storage and exchange of 3D city models. However, using city models based on CityGML has appeared to be a challenge as there is a limited number of software applications that are able to read them. Complexity in CityGML data sets is also a problem as the format allows for many variations of geometry handling. Also, 3D models in general are often created individually by producers and exchanging them between platforms and users is an issue. As a response to the challenges of 3D city models not being created in a common way, national standards have been created to satisfy internal needs and making 3D city data more software compatible. Sweden has also recognized these needs and is in the process of creating a national standard for 3D buildings called NS (National Specification) building. The data in NS building will be represented as NS LOD (National Specification Level of Detail), a lightweight 3D data type that focuses on storing simple geometries that can be transformed to CityGML. Another 3D city format is CityJSON. CityJSON is a format that has been adapted to the data model of CityGML. The creators of CityJSON have attempted to reduce the complexity of 3D city models to allow for better software development and usage on the web. Because of the complexity 3D city models, robust database systems that can implement all the features of a 3D city model are required. 3D City Database (3DCityDB) is a geodatabase that has been created to properly transform and store CityGML datasets.

This thesis has attempted to verify that NS LOD data can be used in NS building and be transformed to CityGML. This master thesis is also involved in a project (3CIM) between the three largest municipalities in Sweden (Stockholm, Gothenburg and Malmö) and Lund University. The objective of the project is to research how a 3D building model can be enhanced by making it interoperable to an information model. This will in return allow it to be used for analytical purposes. This thesis has created a 3D building environment based on 3DCityDB. The environment can import CityGML data (transformed from NS LOD) and export to CityGML and CityJSON. The environment can be further developed and used to control the information flow during import and export. The results of this thesis show that NS LOD data can be used in the national Swedish specification NS building to be transformed to CityGML. However, further development of the specification should focus on errors that are created during transformation from NS LOD to CityGML as they can hinder the usage of the transformed data. This thesis has also successfully created open-source processes that can be used to import CityGML data to 3DCityDB and export to CityGML and CityJSON. Still, if a user is to adapt the processes much work must be put in to ensure that data sets exceeding this master thesis can be handled.

Keywords: Physical Geography and Ecosystem analysis, 3D City Models, 3D Building Models, CityGML, CityJSON, 3DCityDB, NS Building, 3D City Information Modeling

Table of Contents

Abbreviations

2D	Two-dimensional
3CIM	3D City Information Modeling, Cooperation project between Stockholm, Gothenburg and Malmö municipality and Lund University
3D	Three-dimensional
3DCityDB	3D City Database, Database for storing CityGML data
ADE	Application Domain Extension, Application to extend the functionality of 3DCityDB
AEC	Architecture, Engineering and Construction
AR	Augmented Reality
BIM	Building Information Modeling
CityGML	City Geography Markup Language
CityJSON	City JavaScript Object Notation
COLLADA	Collaborative Design Activity
FME	Feature Manipulation Engine, Software for integrating spatial data
GIS	Geographic Information System
glTF	GL Transmission Format
GUI	Graphical User Interface
IFC	Industry Foundation Classes
ISO	International Organization for Standardization
KML	Keyhole Markup Language
LOD	Level of Detail
NS	National specification
OGC	Open Geospatial Consortium
SQL	Structured Query Language
UML	Unified Modeling Language
VR	Virtual Reality
XML	Extensible Markup Language

1. Introduction

1.1. Background

City models represented in 3D are a tool for city planning and are becoming increasingly popular for larger cities (Eriksson et al. 2020). These city models contain buildings and other objects and structures often found in the urban setting, and can be used for visualisation as well as for advanced simulations and analysis (Kolbe et al. 2019). Such analyses are useful when planning the construction of new buildings as they can provide valuable information such as sun exposure and noise pollution. To facilitate simulations and analyses it is important that a 3D city model follow common standards.

CityGML is one of the most used information models for representing objects and structures found in a city model (Gröger et al. 2012). CityGML is an open standard structured for both storing and exchanging data. The standard represents the physical representation of real-world objects, as well as topological and semantic information (Gröger et al. 2012). In addition, CityGML provides support for objects to be stored at different levels of detail (LOD). This means that a complete city model can be represented by objects of varying features. The most common object found in a city model is the building object, but city models may also contain several different objects found in the urban environment such as bridges, vegetation and water.

City models based on CityGML can become quite large in both size and complexity. Large data models are preferably stored in databases as it makes accessing and searching the data easier. Because of the complexity of city models challenges can occur regarding the database system handling the storage of the models (Yao et al. 2018). In addition to standards regarding city and data models, the database system must be able to implement the key components of the data. 3D City Database (3DCityDB) is an Open Source database extension, or database schema, created to properly handle 3D city models conforming to the CityGML standard (Kolbe et al. 2019). 3DCityDB has the functionality to support the numerous features of CityGML such as multiple level of details and different geometries.

Even though city models represented in CityGML are increasing in popularity, several studies have reported poor interoperability between CityGML data and the software used to handle the data (Noardo et al. 2020). There are two main factors behind this. Firstly, the geometries describing the objects in CityGML can be represented in many ways. Second, the (hierarchical) data structure of CityGML can become very complex when the data sets increase in size. These two factors create a high level of difficulty when developing software to read, visualize, analyse, and transform CityGML data (Ledoux et al. 2019). New version releases of CityGML will hopefully deal with these problems. However, there are also differences in national requirements when representing city models which differ greatly based on the countries developing them.

To increase the data interoperability and make 3D-solutions accessible for more users, the data should be reduced in terms of complexity. This would minimize the difficulties when developing software and make city models more accessible. CityJSON is a text-based data-format with emphasis on reducing the deep structure of CityGML, while at the same time offer the same functionality (Ledoux et al. 2019). This makes CityJSON a more compact alternative to CityGML. This is partly because features in CityJSON can only be represented in one way, and not several like CityGML (Ledoux et al. 2019). This applies for both geometries and thematic data. By making 3D-data accessible in CityJSON and increasing the interoperability,

more end-users will be likely to apply 3D-solutions and benefit of the advantages it can bring in city planning.

To support the continuous popularity of 3D city models, there is a need for national 3D standards conforming to a nation's individual needs. A national standard would allow for consistent information handling in the urban environment. Some studies regarding creating international 3D standards have pointed out the importance of relating the data in a city model to an international standard. This may increase the likelihood of software applications being compatible and able to properly read the data. Countries such as Germany (Gruber et al. 2014) and Netherlands (Brink et al. 2013) have implemented national standards for city models.

There is an ongoing work in Sweden to establish a national specification regarding 3D-buildings[1] (web site only available in Swedish). National specifications are important for nationwide solutions as it promotes better data exchange. The Swedish specification, called NS building, will specify how building objects found in a city model are to be modelled according to Swedish requirements. The specification will be based on mapping already existing geometrical components to the international standard CityGML as can be seen in Figure 1.1. Geometrical components, compared to complete 3D buildings, are easier to keep up to date and 3D buildings can be created based on needs (Lantmäteriet 2021b).

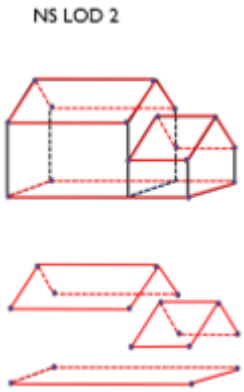


Figure 1.1: Mapping geometrical components from NS building (bottom figure) to CityGML (upper figure).

Stockholm, Gothenburg, Malmö municipality and Lund University are working on a project (3CIM). The project is researching modeling a 3D city model and how it can be applied in a rainfall analysis. The 3D building will be based on the previously mentioned Swedish national specification. The project is still in its early stages and research is being made on how to store and handle the 3D building. Malmö municipality is also looking into the possibility of using a 3D database environment for storing CityGML.

Because of the 3CIM project and Malmö municipality's internal needs, a 3D building environment is designed (Figure 1.2). This environment is based on the Swedish standard NS building, 3DCityDB, CityGML and CityJSON. The environment and its workflow are the topics of this thesis. The environment transforms data from NS building to CityGML, imports and stores it in the database environment and allows for exporting to either CityGML or the

[1] https://www.lantmateriet.se/sv/webb/smartare-samhallsbyggnadsprocess/nationella-specifikationer/

more lightweight format CityJSON. Lastly, the data is tested against applications to ensure its usability as it has gone through several altering processes.

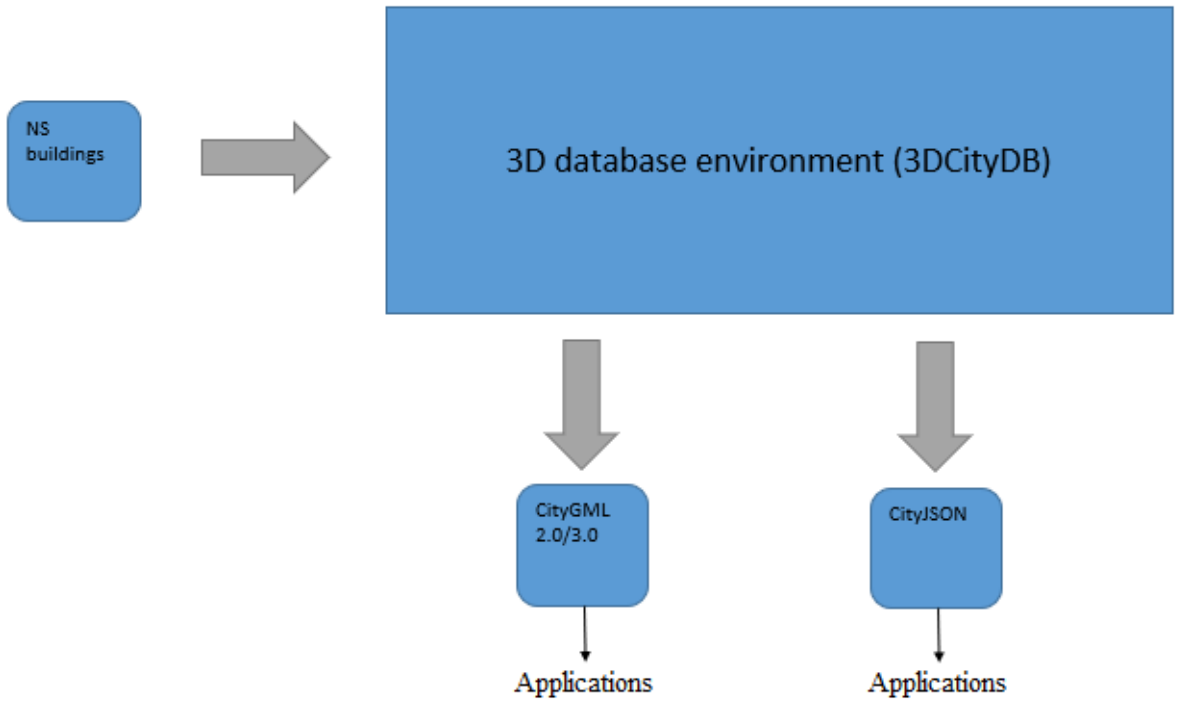


Figure 1.2: Vision model workflow and 3D database environment. The two topics of this master thesis.

1.2. Aim

This master thesis is a cooperation between Lantmäteriet (Swedish mapping authority), 3CIM and Malmö municipality, and therefore has several aims. For Lantmäteriet, who is developing NS building, it is crucial to verify that the geometrical components from the Swedish standard NS building can be transformed to CityGML. 3CIM and Malmö municipality need a process to import CityGML data to 3DCityDB to support proper storage. Lastly, the database system will be enhanced by creating an export process to two different formats, CityGML and CityJSON. This may allow more users, such as smaller municipalities, to use 3D building data. The specific objectives are:

1) Create a process which can transform data from the upcoming official Swedish building specifications (NS building) to the international standard CityGML.

2) Create a process to import the transformed CityGML data to the 3D city database environment - 3DCityDB.

3) Create a process to export data from 3DCityDB to CityGML and CityJSON.

4) Evaluate the applicability of the general workflow in Figure 1.2. Applications typically used by a user of 3D data should be tested after the data has gone through all the processes of the vision model. Open-source applications are used for testing purposes, as they are free and easily accessible.

1.3. Limitations

3D city models may describe several different objects of the urban environment, but this master thesis is limited to objects representing 3D buildings. Building is also the only theme where substantial work has been done for the new Swedish national specification (NS building). Therefore, no consideration is taken to other urban themes such as water, vegetation, transportation, etc.

Only technical feasibility is studies. There is no focus on organization, data flow or economic considerations. The technical applications of 3D building data are prioritized.

1.4. Disposition

After this introductory chapter this master thesis is split into seven chapters. Chapter two includes a description of national and international standards for city models in addition to tools that are commonly used in the 3D environment. Chapter three brings up recent studies regarding the standards and tools and how they can be used for real-life applications. Chapter four presents the case study and is followed by chapter five which includes the methodology used. Lastly, chapter six, seven and eight present the results, discussion and finally the conclusion.

All scripts made in this master this are available from the following GitHub web page: https://github.com/tomaben/Master-thesis-.

2. Standards and tools for city models

3D city models are virtual representations of the physical urban environments. Such environments contain many different objects found in a city, and city models must be able to represent these objects. Standards for city models help create common solutions for such challenges. The most common object in a city model is the building, and it is also the most complicated to represent correctly as a model. In terms of complexity, buildings can be very detailed and as a result there are many ways to represent them in a city model. Therefore, some countries have recognized the need for a consistent representation of buildings and developed national specifications to describe 3D building models based on their individual national requirements.

This section describes the most popular standard for 3D city models, CityGML, and how it has been adapted by national specifications for 3D buildings. Also, CityJSON, an alternative light weight format for city modeling, is discussed along with its positive and negative sides. Lastly, 3D City Database, a database schema for the storing of CityGML, is presented.

2.1. CityGML

In 2008 the Open Geospatial Consortium (OGC) declared CityGML as an international standard for presenting and exchanging 3D city models (Gröger et al. 2012). Since then the standard has been further developed; the currently used version, CityGML 2.0, was released in 2012. CityGML has the functionality to portray the geometries, themes and visual features of city building objects (Gröger and Plümer 2012). Thematic, or semantic, representations of the object models found in an urban environment is a central part of the standard. Such objects are e.g.: buildings and their corresponding parts (roofs, walls, windows etc.), surfaces and terrain, water and vegetation (Gröger and Plümer 2012). Not only does CityGML describe the semantic properties of the objects, but also the relationship between them. For instance, the relationship between a building, or more precisely the ground surface of a building, and the surface or terrain it is placed upon. The most central object model for CityGML is the building model.

For all models represented in CityGML there are clear definitions for semantic information, attributes, relationships and 3D geometrical representation (Gröger and Plümer 2012). These models, including building, are grouped into modules. CityGML allows for combination of different modules in order to satisfy a certain use-case (Gröger and Plümer 2012). However, not many analytical use-cases exist for these models, other than the building model, which means that their applications are primarily visualisations (Biljecki et al. 2015).

CityGML can describe building objects with the help of visual appearances on the outer boundaries, such as walls or roofs (Gröger and Plümer 2012). These appearances come in the form of different types of categorical data that are used in analytical tasks. Noise pollution, sun exposure, wind simulations, shadow cast simulations and energy demand estimation are some of the use cases (Biljecki et al. 2015). These abilities allow 3D city models not only to be used for visualisation purposes, but also for simulations and analysis.

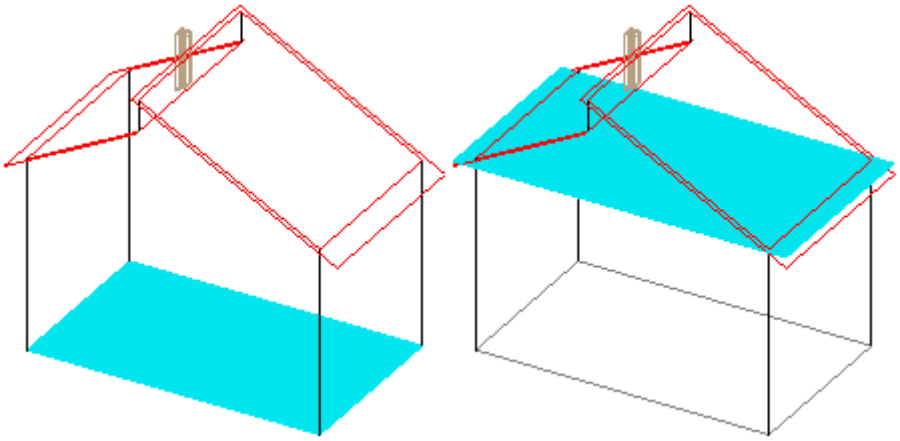
There are many use-cases in 3D city models that require additional functionality that exceeds the limits of CityGML (Gröger and Plümer 2012). These specific use-cases might require additional attributes, feature types or relationships that CityGML simply cannot offer. Therefore, CityGML has developed the Application Domain Extension (ADE). An ADE is a customisable application schema of CityGML which allows users to extend the functionality of a standard CityGML schema (Biljecki et al. 2018). By adding new associations and attributes it is possible to create new feature types in CityGML which are custom-tailored for a specific purpose. In addition, CityGML supports the use of several ADEs concurrently which makes 3D city models versatile.

2.1.1. Level of detail

CityGML uses five different levels of detail (LOD) to describe the complexity a 3D object can have (Gröger et al. 2012). The lowest level is LOD 0 and highest LOD 4. For each increase in LOD the models become more detailed for both geometric and thematic consideration. The following section will introduce LOD for 3D building models.

LOD 0

Building objects represented in LOD 0 are described by horizontal 2.5D polygons. The polygon is either placed at footprint- (Figure 2.1a) or roof height (Figure 2.1b) (Gröger and Plümer 2012). No solids are associated with LOD 0.

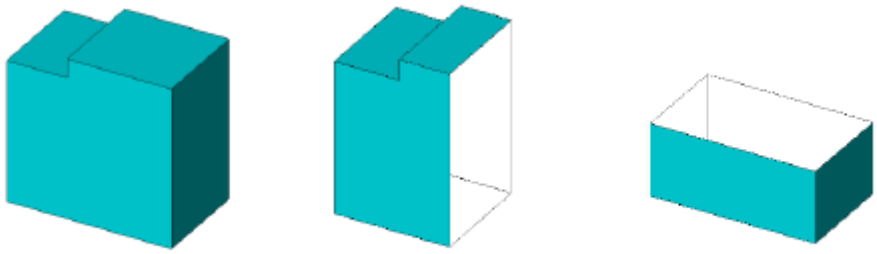


(a) (b)

Figure 2.1: LOD 0 described by either a footprint in cyan (a) or roof height in cyan (b) (Gröger et al. 2012, page 67).

LOD 1

LOD 1 adds solids and multi-surfaces to building objects with the help of complimentary surfaces. The objects are not very detailed, seeing as they are represented by cubes (Figure 2.2). A building can now be split into several building parts to take into consideration differences in geometrical or semantic data (Gröger and Plümer 2012). For instance, a building can have different heights (geometrical) or different roof type (semantic).



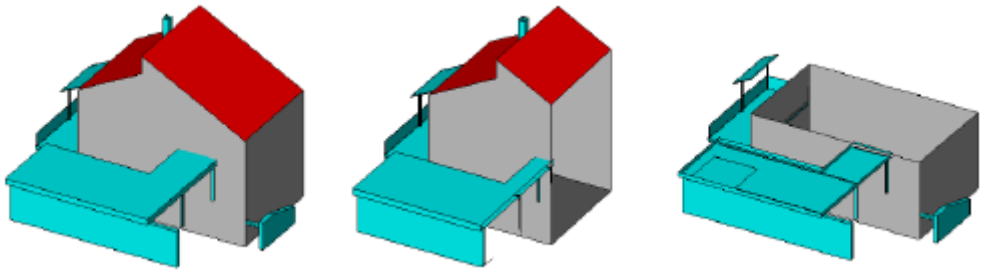
(a) (b) (c)

Figure 2.2: LOD 1 representations. The practice of different roof heights can be seen in (a). (b) and (c) shows the interior of the building (Gröger et al. 2012, page 67).

LOD 2

LOD 2 introduces roof shapes to LOD 1 (Figure 2.3a). In LOD 1, all roof types are represented by flat surfaces, but LOD 2 allows for more detailed descriptions. Another important feature added in LOD 2 is that the boundaries of a building can now be used for thematic representation (Gröger and Plümer 2012). This means that the surfaces are split into the following categories: *GroundSurface*, *WallSurface* and *RoofSurface*. However, some building objects cannot be represented like that. Buildings who are not fully closed, such as barns with large openings, are

added with a *ClosureSurface* which falsely closes the object and allows it to be fully represented as a volume object (Gröger and Plümer 2012).

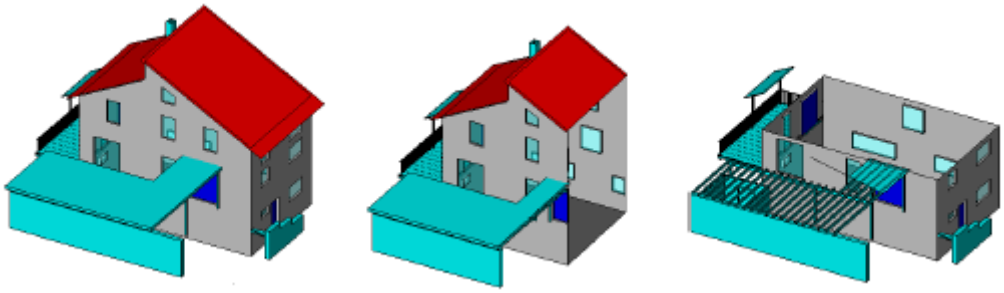


(a) (b) (c)

Figure 2.3: LOD 2 representations. Roof shapes has been added as well as some exterior details. (b) and (c) shows the interior of the building (Gröger et al. 2012, page 67).

LOD 3

LOD 3 adds windows, doors, detailed roof structures and detailed exterior surfaces (Figure 2.4) (Gröger & Plümer 2012).

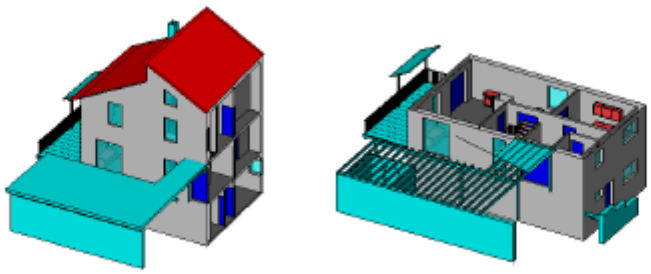


(a) (b) (c)

Figure 2.4: LOD 3 representations. Doors and windows are added to enrich the buildings. (b) and (c) shows the interior of the building (Gröger et al. 2012, page 67).

LOD 4

Finally, LOD 4 adds the interior objects of a building (Figure 2.5) (Gröger & Plümer 2012).



(a) (b)

Figure 2.5: LOD 4 representations. Interior objects have been added as can be seen in (a) and (b). No further details are added to the exterior (Gröger et al. 2012, page 67).

A new LOD concept for the future release of CityGML (CityGML 3.0) is being discussed by several CityGML professionals (Machl 2013; Löwner and Gröger 2016). There is a need to evaluate the current LOD system for CityGML to adapt it to future applications (Löwner and Gröger 2016). The current system is based on 3D models only being used for visualization purposes, but with the development in technology and applications visualization is now only one of many purposes (Biljecki et al. 2015). To help users and researchers of 3D city models Biljecki et al. (2016) has suggested a proposal for a new LOD concept. The new concept will not be limited to any specific 3D format, e.g. CityGML or CityJSON. Instead, the proposal should work for any software that can represent 3D building models. The proposal points out two main disadvantages to the current LOD representation. Firstly, the current LOD concept is missing a clear specification for each LOD level. Second, because of the first disadvantage, each LOD level is unambiguous and therefore allowing room for wider interpretation when defining which LOD an object should have. This may lead to different buildings having the same LOD, even though they have different levels of abstraction (Biljecki et al. 2016).

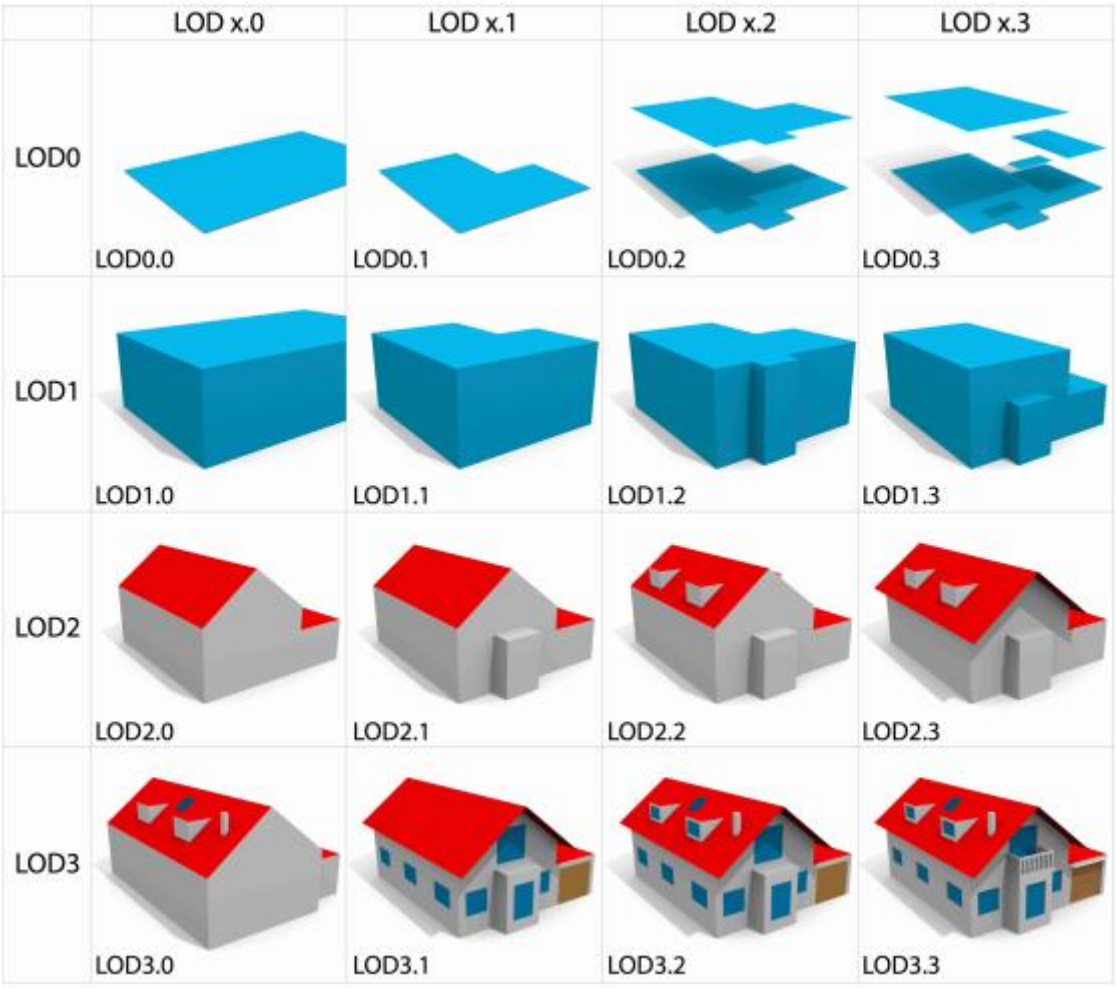


Figure 2.6: Expanded levels for LOD 0-3. Each level has been expanded to 4 sub-levels and therefore described in a more accurate way reducing the risk of misinterpretation (Source: Biljecki et al. 2016, page 28).

To overcome the shortcomings of the current LOD structure Biljecki et al. (2016) suggest widening the levels of detail for LOD 0-3 to 16 (Figure 2.6). The new definition will expand each LOD to 4 sub-levels, except for LOD 4 which introduces indoor elements and is not a focus in the proposition. The expanded LOD definition is based on a series of researches and studies in the 3D city community, one being national standards and guidelines. In addition, Biljecki et al. (2016) suggests national mapping agencies can implement the new LOD definition in their specifications and that practitioners can standardize their data. The new LOD concept has been adapted as an inspiration to the Swedish specification NS building mentioned in Section 1.1. The concept is used to describe how geometrical components can be stored in a database (Row 1, Figure 2.6) and be transformed to CityGML.

2.1.2. Modeling semantics and geometry in CityGML

CityGML has two official formats XML/GML and a database format (3DCityDB) (Gröger et al. 2012). XML is responsible for organising the different classes found in a building object, while GML handles the actual geometries (Ledoux et al. 2019) By being based on XML and GML, CityGML inherits both the advantages and disadvantages of these structures. As a result, as seen in Figure 2.7, XML tables may become complex when representing buildings in city models.

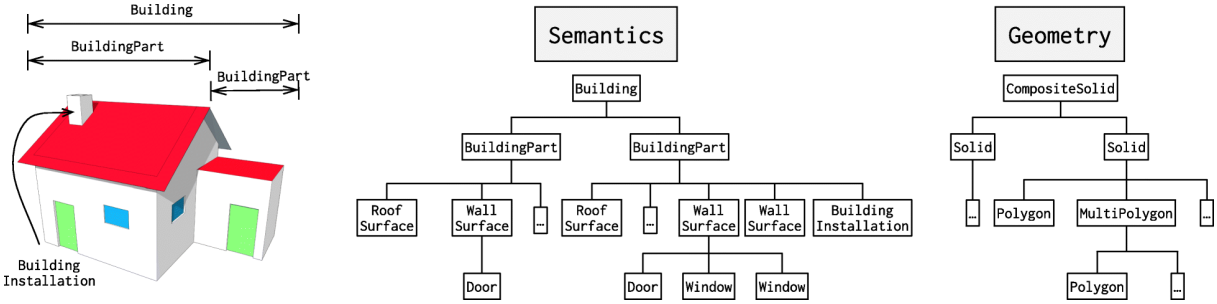


Figure 2.7: Representation of a single building object. The object results in a deep structure of tables required for representation (Ledoux et al. 2019, page 3).

This disadvantage is even greater when dealing with large city models as they contain many buildings. A drawback of implementing GML is that there are several different ways of representing a single geometry (Vitalis et al. 2020). This produces challenges for developing software to properly read the different representations of geometries from GML.

2.1.3. Geometry

CityGML LOD 2 allows thematic surfaces to be represented in a 3D building object. Such surfaces can be ground surface (*GroundSurface)*, wall surface (*WallSurface*), roof surface (*RoofSurface*) etc. These surfaces are then used as boundaries for the actual building solid (Figure 2.8). The solid can be used to calculate the volumetric properties of the building, and therefore some surfaces do not help define the solid. For instance, roof overhangs are part of the thematic *RoofSurface,* but they are not part of the boundary surface that encloses the solid.

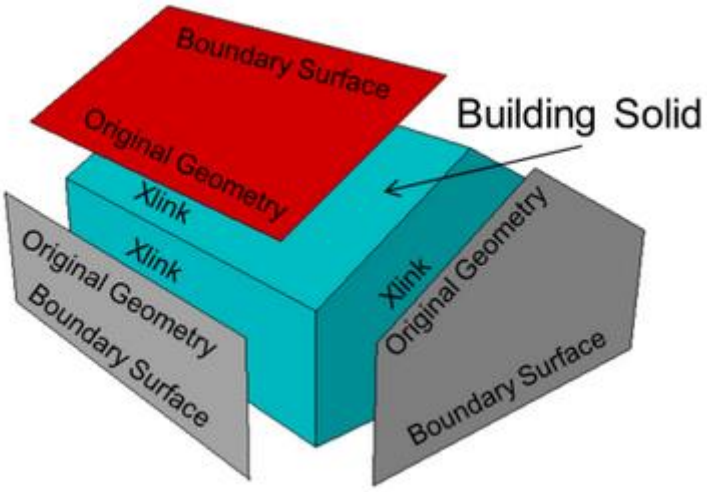


Figure 2.8: A 3D building object represented by boundary surfaces and a building solid (SIG3D 2017).

To avoid storing these geometries several times (boundary surfaces and solid), the CityGML standard uses xlinks. An xlink is an unique identifier that is used when some objects share the same geometries (Gröger et al. 2012). This means that the building solid (teal) in Figure 2.8 does not contain any geometric properties but has references to the boundary surfaces. In return this reduces the size of CityGML data sets, but as Figure 2.7 highlights the hierarchical structure might become complex for large data sets.

2.2. CityJSON

Because of the complexity of CityGML, Ledoux et al. (2019) have developed CityJSON as an alternative for city models. CityJSON is not an official OGC standard, but is an encoding of the CityGML 2.0 data model and has the potential to become a popular data exchange format for web apps (Ledoux et al. 2019). The main reasoning behind developing the format was to give software-engineers an easier format to work with and hopefully speed up the usage of 3D-data in practice, both for web applications and software development (Ledoux et al. 2019).

CityJSON reduces the complexity of city models by applying identifiers to reference the objects (see appendix B - the numbers in the boundaries list reference the list of vertices). The identifiers are stored in a dictionary which gives developers direct access to the information (Ledoux et al. 2019). Therefore, the hierarchical XML-structure is removed, and a flat data format which takes up less space is obtained (see Appendix B). Ledoux et al. (2019) reported an average of 6 times more compact datasets with CityJSON compared to CityGML. CityJSON also makes reading the files uncomplicated by only allowing one way to represent both geometries and semantic information.

Vitalis et al. (2020) point out the restricted access of GIS-software which can handle CityGML files. The software that can handle CityGML are mainly proprietary, such as FME and ArcGIS. Open-source software require tricky workarounds which in addition sometimes include external tools. There exists support from several programming languages to read and alter the structure of CityJSON files such as Java, C++ and Python (Ledoux et al. 2019). These programming languages are also some of the most popular when creating applications for the web, and thus creating or updating web applications for CityJSON becomes less problematic. Also, the usage of CityJSON might motivate further development of general software as the format is easier to read. Vitalis et al. (2020) developed an open-source plugin for QGIS[2] that allowed the import and visualization of CityJSON datasets. The plugin successfully loaded central information such as object types and attributes which is useful when displaying city models.

Challenges with CityJSON

Although CityJSON has been reported as an alternative to CityGML it is not effectively used in real-world applications (Nys et al. 2020). 3D building models based on CityJSON are not being constructed in a large enough scale, and the main reason behind this is because there are not many processes to generate them (Nys et al. 2020). However, two viable open-source generation methods exist: 3Dfier[3] and CityGML-tools[4]. The 3Dfier generates LOD 1 building models by extracting topographical data sets. CityGML-tools can convert already existing CityGML-files to CityJSON and backwards. However, this is done by a command line and not a user-friendly graphical user interface (GUI). A viable and flexible extension of an existing software, like 3DCityDB, might make CityJSON a more appealing alternative to implement.

2.3. Building Information Model – BIM

Another type of 3D building model is the Building Information Model, or BIM. The BIM abbreviation is not limited to 3D building models, and has several other definitions (Miettinen and Paavola 2014), but this section emphasises on how BIM can be used to digitally represent a building and building information. BIM is used within the architecture, engineering, and

[2] https://www.qgis.org/en/site/
[3] https://github.com/tudelft3d/3dfier
[4] https://github.com/citygml4j/citygml-tools

construction (AEC) industry to digitally represent a building during and after construction providing a unique life-cycle representation (Azhar 2011). BIM technologies store information for the building which can be used to make decisions concerning design, planning, construction, and management (Azhar 2011). BIM can be used to estimate costs for building projects as it can store material information; the same information can also be used to manage a building after its creation. A building has several installations such as electricity, plumbing, ventilation etc. and with this information represented in a BIM environment conflict detection can be checked in a fast and automatic way. Energy simulations for environmental purposes can also be applied for buildings. By providing precise 3D building models with integrated information, BIM creates several benefits such as better information flow, automated processes and reduced building project cost and time (Volk et al. 2014).

The most common open data format to represent BIM is the Industry Foundation Classes[5] (IFC). IFC is an open-standard format which focuses on improving the usage of BIM and BIM software (Laakso and Kiviniemi 2012). IFC, like CityGML, can store semantic information but emphasises on building and structural information (Julin et al. 2018).

2.4. National standards for 3D buildings

Development in technologies has made it easier to create and use 3D building models. Still, many challenges have been reported when incorporating 3D building models with applications that are used by professionals on a daily basis (Stoter et al. 2013). As a response, many countries have developed national 3D standards for buildings to conform for internal needs (Brink et al. 2013; Gruber et al. 2014; Ates Aydar et al. 2016; Soon and Khoo 2017). As further development and usage of 3D city models increases, more standards could be created at a national level to ensure consistent and stable models which are software compatible.

A German standard was developed because of the complicated connection between 3D buildings and the information required to do analyses for economic and administrative purposes (Gruber et al. 2014). Also, the 3D-building models that were being created did not meet the quality demands and improvement possibilities required to do such studies (Gruber et al. 2014). The standard was based on a CityGML Application Domain Extension (ADE), with limitations that mainly included building objects and their respective appearances (Gruber et al. 2014). Even though limitations were applied, some extended information were allowed through the ADE. For storage solution, the 3D standard applied 3D City Database (3DCityDB). Turkey went a similar direction as Germany, when establishing a 3D standard for buildings. The standard was a CityGML ADE which focused on bridging national data with the international CityGML standard (Ates Aydar et al. 2016).

The 3D standard which emerged from the Netherlands focused on connecting information from 2D information models to 3D and thus conforming to the international standard CityGML (Brink et al. 2013). A study regarding 3D- and GIS-software was executed to find the best suited standard to which the data would be mapped to (Stoter et al. 2011). CityGML, with the help of its corresponding ADE-functionality, was chosen in this standard as well as the above-mentioned German and Turkish standard. By implementing previously acquired information from 2D models, the study pointed out several beneficial points: The 2D models had abundant thematic information, and in some cases enough properties to generate 3D models. Already having access to semantic data is very valuable as there are limited possibilities to acquire this

[5] https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/

data on a large scale (Brink et al. 2013). In addition, and perhaps the most important, by using already existing data meant that the usage of the application areas could be implemented (Brink et al. 2013). A follow-up study was conducted by Stoter et al. (2013) to ensure that the outcome of the 3D standard from the previous study could be implemented in real-life applications. The study emphasized on structuring and preserving 3D data to adapt to the national standard.

Singapore also created a standard based on CityGML but did not limit itself to 3D building models. The standard adopted and incorporated other thematic modules from CityGML such as: vegetation, waterbody, bridge and tunnel (Soon and Khoo 2017). The data used to create the city model according to the standard had to be collected by airborne data collection, mobile imaging, and scanning. As a result the models became very large and extra considerations had to be taken to the database system responsible for storing the models (Soon and Khoo 2017). 3DCityDB was taken into consideration to help solve the database challenges, but since 3DCityDB allows for more than one geometry column in certain tables further research had to be taken. GIS only permits one geometry column per database table and this would impose technical issues (Soon and Khoo 2017).

As of this date there does not exist a national standard for 3D buildings in Sweden. However, a Swedish specification currently being developed (see Section 2.5). The specification is partly inspired by a case study conducted by Eriksson et al. (2020) where the aim of the study was to create a suggestion for a national building standard. The case study developed an ADE based on the CityGML version 3.0. with the following requirements: The standard should support the general development of 3D city models. It should be able to connect to Building Information Models (BIM) and relevant national registers. The standard should be built upon a classification system to help connect the information between BIM and 3D city models. The case study concluded that the data model of a 3D city model should be based upon an international standard, such as CityGML.

2.5. A Swedish national standard for 3D buildings

The following section describes the development of a Swedish national standard for 3D buildings. The standard, and the documents used in this master thesis to describe the standard, are not fully developed as of the writing of this master thesis. Therefore, information reported in the following paragraphs are subject to change.

The Swedish national specification for 3D buildings (NS building) is currently being developed (Lantmäteriet 2021b). The data in the specification will be stored as geometrical components in the Swedish national geodata platform. From the geodata platform it will be possible to transform NS building data to the CityGML 2.0 format. The transformation of the geometries is not limited to 3D buildings, but can also be transferred to base maps, 3D-printing formats, surface models, etc. (Lantmäteriet 2021a). Four different NS LOD will be supported and stored in the geodata platform: NS LOD 0.1, 0.2, 2.1 and 2.2 (Figure 2.9). Additionally, each LOD will have three sub-levels: a, b, and c.

The NS building specification has some limitations and will not include the following topics: indoor information, information regarding apartments, other urban constructions (bridges, tunnel etc.) and information regarding building construction type (material, colour etc.) (Lantmäteriet 2021b).

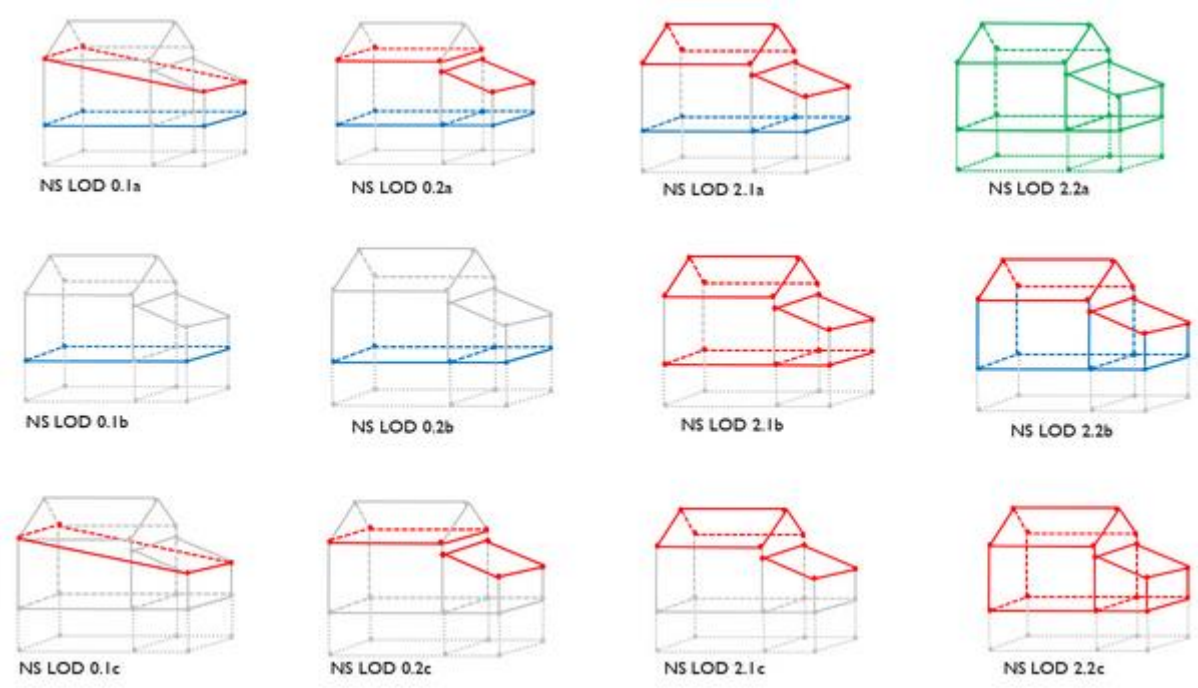


Figure 2.9: Work in progress of the Swedish national specification for 3D buildings. Different colours represent different methods of data acquisition. Red surfaces is the result of traditional terrestrial geodetic measurements, blue surfaces are from photogrammetry and green are from BIM data. Alternatively, terrestrial or airborne laser scanning can be used (Lantmäteriet 2021a).

Data collection for building information has previously been performed individually based on municipalities and their needs (Lantmäteriet 2021b). Structuring the already existing building information according to NS building is therefore a challenge as the information varies from municipality to municipality. Hence, the municipalities are responsible for gathering and updating data for buildings in their respective regions (Lantmäteriet 2021b). The gathered data can be used to create new models, or an already existing model can be updated. In both cases the data must be in accordance to NS building and when the models are created/updated they will be uploaded into the national geodata platform as NS LOD representations (Figure 2.9). The building data can be gathered by terrestrial geodetic measurements, photogrammetry, terrestrial or airborne laser scanning or even BIM, and will be updated regularly by the municipalities. From the national geodata platform consumers can retrieve whichever NS LOD is available for their purposes and transform it to CityGML. The consumers may choose to export to simplified CityGML models (LOD 1), or more detailed (LOD 2), but never at a higher LOD than available. Alternatively, a producer can choose not to store data in the national geodata platform but provide the data according to NS building by their own. For instance, Malmö municipality can create and publish their own models and will therefore not use the geodata platform.

The information flow of NS building can be seen in Figure 2.10. The geodata used to create NS LOD is first acquired [1]. From there the municipalities either create a new model or update an already existing model to correspond to NS LOD [2]. The next step is storing the NS LOD data in a database [3]. Two paths are available: either the municipalities use the national geodata platform provided by Lantmäteriet, or the municipalities can choose to provide their own means of storage. Either way the municipalities are responsible for gathering [1], creating [2] and storing [3] NS LOD data.

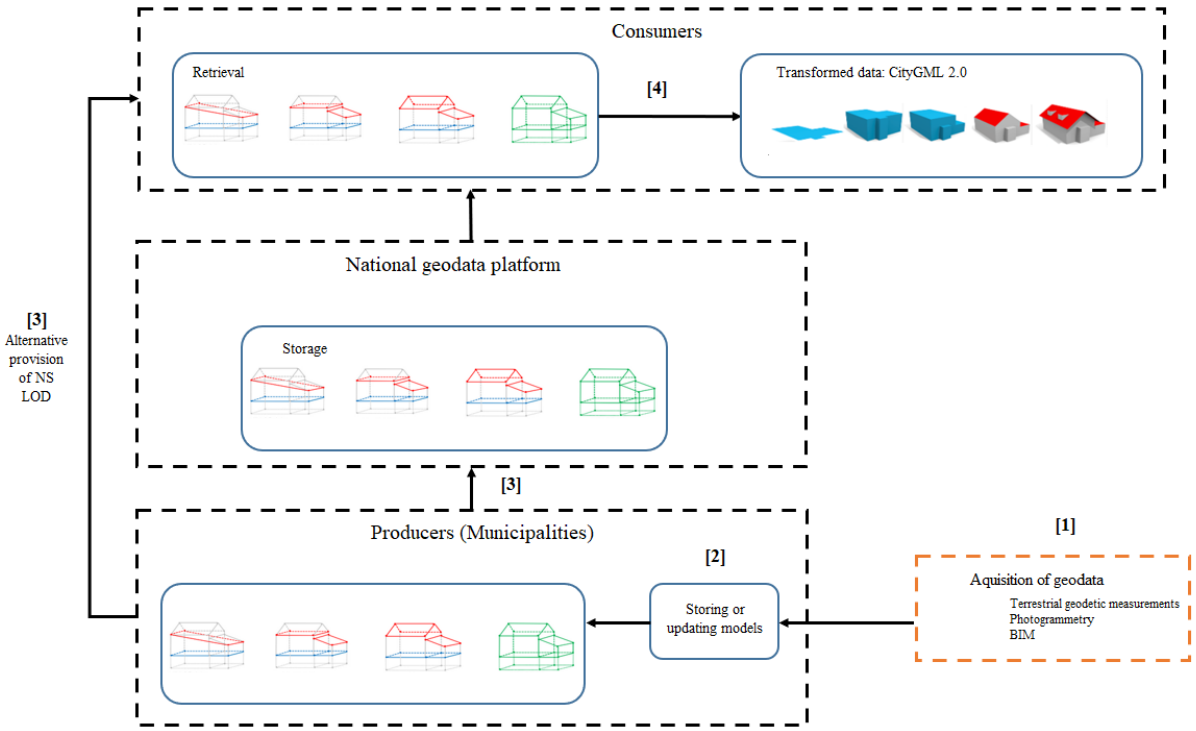


Figure 2.10: Overview of the information flow based on NS building. Producers create geometric components which are stored in the national geodata platform. From there consumers can extract the components and process them to create CityGML with various LODs. The figure is based and translated from Lantmäteriet 2021.

Lastly, consumers can retrieve the available NS LOD data and transform it [4] to CityGML version 2.0. The transformed CityGML data sets are based on what NS LOD is available in the national geodata platform, or alternate storage method provided by a municipality.

Figure 2.11 describes a preliminary UML diagram of a building and its sub-parts in the national geodata platform. The purpose of the model is visualisation and analysis at a national level, and the municipalities are responsible for most of the data needed for this purpose. In the NS a building object (denoted NS building) is defined as a long-lasting construction which consists of a roof, or roof and wall structure (Lantmäteriet 2021b). It is firmly placed on solid ground, or partially under, and is intended for humans to reside in. A building part is a sub-section of a building based on either physical, functional, or time-based aspects which can be considered a building. Building accessory is defined as a smaller establishment which is connected with the building. Geometry is the geometric properties of the objects.

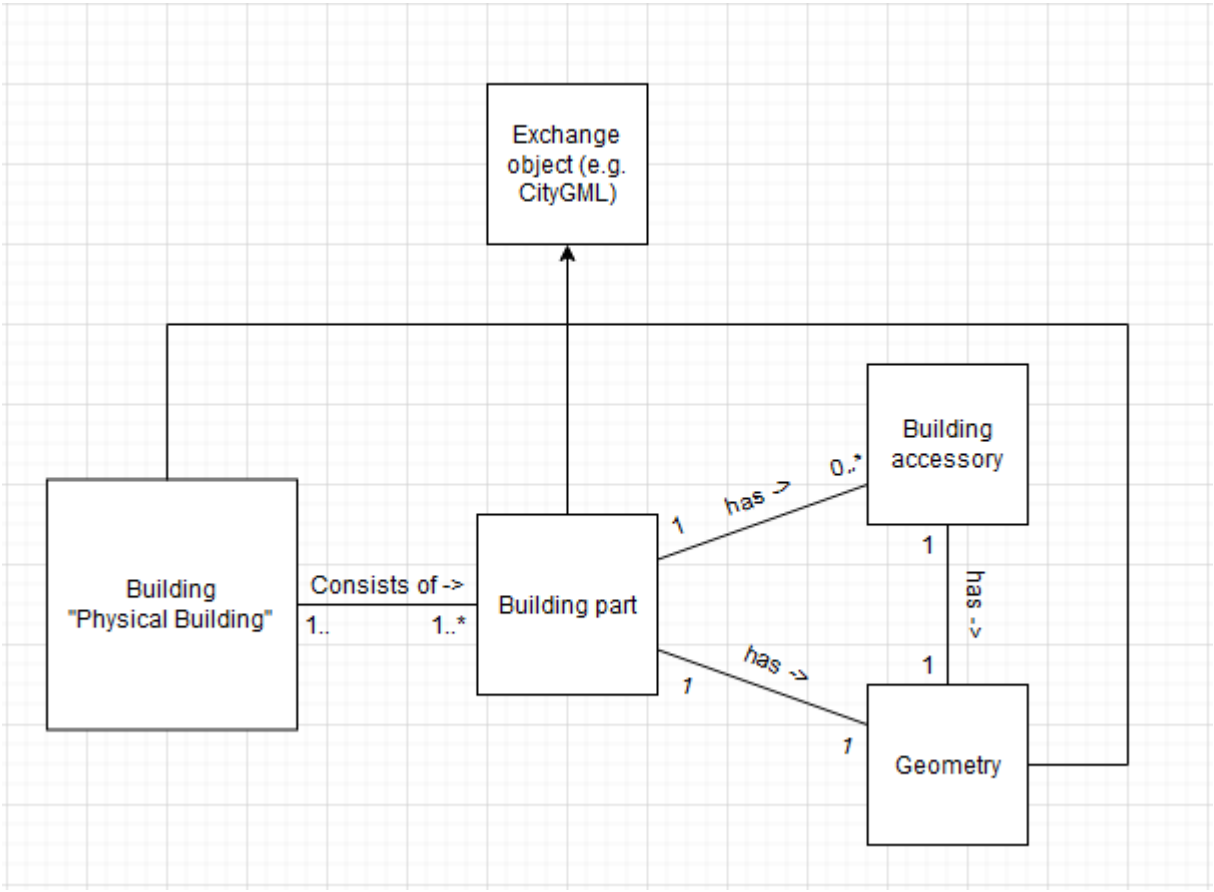


Figure 2.11: UML diagram of NS building in the national geodata platform. The diagram describes NS building and its sub part building part and building accessory. The figure is based and translated from Lantmäteriet 2021.

2.6. Measurement and modeling guidelines for 3D city models

When gathering new data or updating an already existing 3D city model, clear guidelines are required to enable consistent and complete models. Such guidelines can help define information flow and the content of 3D city models. Without these guidelines, decisions regarding collecting or updating data will often be influenced by the individual contributor. Two different guides will be introduced in the following sections: Modeling Guide for 3D Objects – created by SIG3D to help model 3D city models referring to the German and European standards (SIG3D 2017) and Mätanvisningar (measurement instructions) – created by Lantmäteriet to help users in Sweden exchange geodata between different organizations (Lantmäteriet 2018). It should be clearly stated that the Swedish measurement instructions are a work in progress and are subject to change.

The Modeling Guide for 3D Objects is a guide created by the Special Interest Group 3D (SIG3D) to help users model 3D objects. The guide is based on CityGML versions 1.0 and 2.0 and is limited to the exterior of buildings (SIG3D 2017). It is intended for developers, modelers and users responsible for 3D data. Several rules are brought up in the guideline considering overhanging building elements (roof overhangs), modeling, terrain intersection line, heights, level of detail, user defined attributes etc.

There are previous modeling guidelines, part of the project Swedish geoprocess, that aims at creating a common way of exchanging geodata so that it can easily be exchanged between administrative stakeholders (Lantmäteriet 2018). Lantmäteriet is conducting work to update the previous guidelines with measurement instructions partially inspired by the SIG3D Modeling

Guide. Types of geodata involved in the instructions are buildings, ground details, ground usage, roads, and heights. For buildings, both 2D and 3D are considered. Buildings in LOD 0 (footprints) are considered 2D while LOD 2-3 are 3D buildings. Indoor details are not part of the instructions for buildings. The measurement instructions provide explicit guidelines how data regarding 2D and 3D buildings shall be interpreted, gathered, documented and coded (Lantmäteriet 2018). This includes the LOD concept previously mentioned in the section regarding the Swedish specification NS building. In this way the measurement instructions and the NS LOD concept will complement each other and be helpful tools for gathering data to conform for NS building.

2.7. Geometry standards

To successfully use 3D building data in visualisation and analysis, it is a requirement that the basic 3D surfaces (3D primitives) defining the 3D buildings are in accordance with international standards. Rules regarding the structure of 3D primitives are specified in the international standard ISO19107 and are meant to ensure consistency when transferring and transforming datasets (Ledoux 2018). ISO19107 defines a set of geometric primitives which combined can be used to represent 3D objects (ISO 2019 described in Ledoux 2018). 3D objects are called solids in ISO19107. A solid is created by combining 2D surfaces. These surfaces are created by 1D curves which in turn are created with the help of points, which have zero dimensions, or 0D. The validation of these geometric primitives, and that they are properly combined to create 3D objects can be verified by the tool Val3dity.

2.8. Tools for validating geometries – Val3dity

Val3dity is an open-source software used to validate the geometric primitives in accordance with ISO19107. It is available as a web application as well as a downloadable program[6]. In addition to the rules set by ISO19107 it includes two restrictions: curves must be linear, and surfaces must be planar (Ledoux 2018). These restrictions are used in CityGML and many 3D GIS software and are therefore adopted by Val3dity. Commercial 3D and GIS software offer some validation methods, but the issue with these methods is that they seldom offer a complete validation approach that is satisfying for 3D building objects (Ledoux 2018).

Val3dity uses a hierarchical approach when validating the 3D objects (Ledoux 2018). First, 2D validation rules are applied to the surfaces, and by extension curves and points as well. Secondly the topology, intersections, orientation, etc of the shells defining individual solids are checked. Since solids can be defined by inner and outer shells, the relationship between these are also checked in a third step. Lastly, all solids are compared, and their interactions validated. This hierarchical approach means that if the validator detects an error in the first step it will not continue the remaining steps, but simply report all errors in the current step and close the validation. This is to avoid reporting errors that do not really exist but appear as a consequential error. If the original glitch is fixed potential consequential errors might not appear.

Val3dity can uncover many errors in a 3D city data set. One of these errors are non-planar surfaces (Figure 2.12). A non-planar surface is a surface where one, or more, of the vertices does not lie on the same plane as the others (Autodesk 2015). From Figure 2.12 two vertices can be observed outside of the plane (surfaces coloured green). Val3dity has a default tolerance value of 1 cm (distance from vertex to plane).

[6] https://github.com/tudelft3d/val3dity

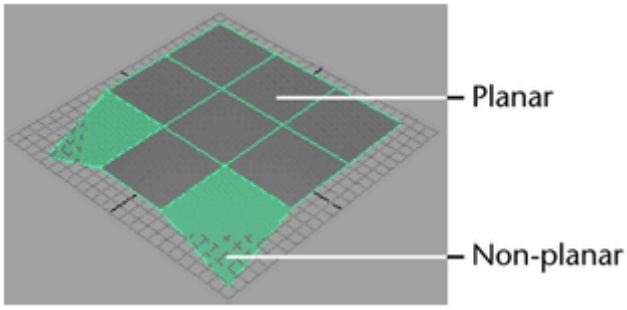


Figure 2.12: A non-planar surface. Surfaces that are coloured green which are below the netted surface are non-planar. Source: Autodesk.

2.9. 3D City Database

3D City Database (3DCityDB) is an Open Source extended database schema which is mapped to the CityGML standard (Kolbe et al. 2019). 3DCityDB is not a standalone database but is added as a database schema to a spatially-enhanced relational database management system (SRDBMS) (Kolbe et al. 2019). An example of this can be seen in Figure 2.13. It supports two database systems: Oracle[7] and PostgreSQL/PostGIS[8]. Oracle is a commercial database system, while the PostgreSQL/PostGIS database is Open Source. PostgreSQL is an object relationship database system and PostGIS is an extension which enables support for spatial data.

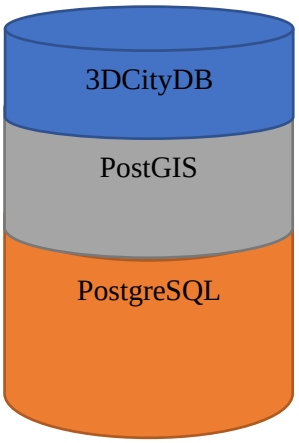


Figure 2.13: 3DCityDB and its relationship to a database. PostgreSQL is a relational database management system (RDBMS). The PostGIS database extension adds functionality to store spatial objects in PostgreSQL, making the database spatially-enhanced. Lastly, 3DCityDB is added on top to support CityGML

3DCityDB is based on the CityGML 2.0 standard and the main functionality is to support the storage of CityGML models of various sizes, levels of details (LOD) and geometries (Kolbe et al. 2019). This is done with the help of a graphical user interface (GUI) which has tools for importing, exporting, studying, and overseeing the stored city models. The database schema also supports the previously released version CityGML 1.0. No support currently exists for the future release of CityGML (CityGML 3.0), but it is expected to be implemented in the future (Kutzner et al. 2020).

3DCityDB has several tools which adds additional support for the users of 3D city models. In addition to exporting to CityGML, 3DCityDB can transform data to Keyhole Markup Language (KML), COLLAborative Design Activity (COLLADA) and the GL Transmission format (glTF). All three formats are primarily used in 3D-applications for visualisation and exchange purposes (Barnes and Finch 2008; Burggraf 2015; Bhatia et al. 2017), but they can also be used in Earth browsers such as Google Earth, ArcGIS explorer and Cesium (Kolbe et al. 2019). 3DCityDB also supports additional extensions to CityGML files by managing their corresponding ADEs.

The import and export functions of 3DCityDB can be used through a GUI or a command line interface (CLI). The database can import CityGML models, and as previously mentioned transform and export to CityGML, KML, COLLADA or glTF formats. The built-in import function does not support updating or versioning of CityGML data sets. Updating and

[7] https://www.oracle.com/database/
[8] https://postgis.net/docs/manual-3.0/

versioning are important functionalities as cities undergo changes throughout time and keeping track of them in a database environment can be useful. To update a CityGML data set at with 3DCityDB one may have to manually remove outdated data and then import the updated data. Also, 3DCityDB does not have any functionalities to detect and repair errors in a data set. The database can validate the schema of the data but does not detect errors like non-planar surfaces or double points.

3. Related studies

3.1. Geometric validation of 3D city model data

Wagner et al. (2015) created a series of methods to validate regular geometric requirements for 3D buildings in a city model. The motivation behind the study was that in individual applications quality requirements often were different, and that 3D data has a bigger source of potential errors, making them difficult to validate. Like Val3dity, the study was based on ISO 19107 and GML stating that these standards present a clear understanding of geometric rules. Several validation methods were created such as point accuracy, polygon validation, closeness (meaning the first and last point in a linear ring must be the same, or closed), duplicate points, validation of solids, etc. The study focused on the CityGML format, and only geometries. Schema validation was not a priority as commercial software yielded good results. As a conclusion, the study suggested that the validation of ISO 19107 and GML should follow a modular approach, meaning a set of rules must be validated before the validation of other rules are executed. This is because some validations are dependent on other validations, and this is the same basis on how Val3dity is programmed. Lastly, the study suggested that future research should focus on validating semantic/thematic data and the connection between geometries.

Another important type of validation is the classification process that is being used when creating 3D building models with data collection methods such as photogrammetry and remote sensing. It has been reported that such data collection methods can effectively recreate the 3D structure of a building and other urban objects (Wentz and Zhao 2015). The above paragraphs discuss geometric rules within the 3D building models after they have been constructed, but it is just as important to pay attention to how the 3D representations of the buildings have been created. Recreating 3D buildings with automatic extraction methods are becoming increasingly accessible and there is a need to determine the quality of the methods being used (Wentz and Zhao 2015). When validating the construction of 3D building models there are two central factors: reference data to directly compare with the constructed 3D models, and which method of classification was used (Wentz and Zhao 2015). 3D building data would be a good source for reference data, but not many sources exist. Therefore, deriving the footprints and rooftops of buildings from aerial images, web maps etc. is an alternative. Two popular classifications are the pixel- and object-based methods (Awrangjeb et al. 2010). Pixel-based approaches uses pixel values to classify, while object-based creates objects in an image and then attempts to classify them (Desheng and Xia 2010). Object-based approaches has been reported as the better method for classifying (Desheng and Xia 2010; Wentz and Zhao 2015).

3.2. 3D city model data in open source products

This section will briefly introduce the two applications that have been used to test the CityGML and CityJSON data sets created and exported in this master thesis.

FZKViewer

FZKViewer[9] is a free to use application which focuses on importing and visualising models from both the GIS and BIM fields. The viewer has an emphasis on open standard formats such as CityGML and supports ADE extensions. In addition, FZKViewer can represent semantic data such as roofs, walls, floors etc. and even edit them (Häufel et al. 2015). The application has been used in several research projects for various purposes such as: BIM and GIS integration (de Laat and van Berlo 2011; Floros et al. 2017), CityGML with solar analysis (Wate and Saran 2015) and spatial planning with 3D models (van Berlo et al. 2013).

CityJSON loader in QGIS

Vitalis et al. (2020) created an open-source plugin for QGIS to directly import 3D building models represented in the CityJSON format. Before this plugin existed no support was available for semantic 3D building models in QGIS (Vitalis et al. 2020). The plugin reads the code of an input CityJSON file and transforms it to QGIS features by implementing a series of processes. Attributes are first identified, and necessary layers created. QGIS features are then constructed and assigned to the previously created layers. Finally, the layers are appended to the active QGIS project. The plugin has functionality to automatically recognize and add a rule-based styling depending on the semantic surface types of a building (*RoofSurface, WallSurface* and *GroundSurface* described in Section 2.1). *RoofSurface* is coloured red and *WallSurface* white. Additionally, different object types can be split into different layers, and information regarding LOD (Level of detail) can be added as either an attribute or as a layer. Vitalis et al. (2020) proves the functionality of the CityJSON loader by importing three open data sets: city of Den Haag with LOD 2 buildings and terrain, city of Helsinki with LOD 1 and LOD 2 buildings and lastly a railway dataset with surrounding landscape. All three data sets were successfully loaded in the plugin. Both QGIS and the CityJSON loader are free to use and open-source software which is a great advantage to researchers and users of 3D data that may not have access to expensive proprietary software. Proprietary software is often required to read and visualise CityGML (Vitalis et al. 2020), and the CityJSON loader can be used as a replacement.

3.3. Benchmark study of CityGML software support

Noardo et al. (2020) conducted a comprehensive reference study to evaluate the software support for CityGML 2.0. Many practitioners of CityGML, both from an academic and practical perspective, have reported poor interoperability and end product when working with the format. Some of these issues were a result of the complex XML structure previously mentioned in Section 2.1. The reference study aimed to test the issues with CityGML in terms of interoperability with software in an academic matter. To do so, the study invited practitioners with a considerable variation in terms of skills and work areas within CityGML and software usage to gain a broad and fair answer. The practitioners were split into groups with varying skill and given three different CityGML datasets.

The experiment was then to measure interoperability by testing these datasets on various tools commonly used within the 3D-community, geographic information systems (GIS) and BIM

[9] https://www.iai.kit.edu/english/1648.php

software. A wide range of software were tested, but some of them were specialised to handle only 3D-data or traditional GIS-data. The results reported a poor interoperability between the format and software. In terms of reading and visualising the datasets, not many applications came through. Even fewer tools were able to export them without data loss. Several problems were outlined as possible factors of the results. CityGML and city models in general tend to be very complicated regarding geometrical and thematical data. In addition, CityGML requires much from the software that reads it because it contains both elements from 3D-graphics and basic geoinformation.

In conclusion the benchmark study pointed out the importance of restricting geometries in terms of representation. It is vital that applications can handle 3D-data without altering the geometries in the process. Working with less complicated formats, such as CityJSON, was also presented as an alternative to CityGML (Noardo et al. 2020).

3.4. Real world examples of 3DCityDB

3DCityDB has been applied by many different professionals to store CityGML datasets. Cities, mapping organizations, firms, etc. are some of the users of the database environment (Yao et al. 2018). In addition, 3DCityDB has been tested in scientific domains to store, model and analyse 3D city models with various applications. This section highlights three studies conducted for scientific purposes: a traffic noise analysis, urban energy modeling, and simulation and flood modeling.

Traffic noise analysis

As a response to the growing Indian population, Konde and Saran (2017) attempted to create a web based application to analyse traffic noise pollution for a portion of the Rajpur road and its surrounding areas. The CityGML format was chosen to visualise the buildings along the road in the study area. Two datasets were generated: a LOD1 and a LOD2 representation. By using the import function in 3DCityDB the datasets were stored in a PostGIS database with the 3DCityDB extension. A web server would then communicate with the database environment and deliver data to the web client for visualisation. To gather traffic noise data, 23 points were placed among the most exposed parts of the study area.

The study successfully generated noise data and the effect it had on the surrounding buildings. A traffic noise map, as can be seen in Figure 3.1a, was generated and visualised in the earth browser Cesium. A semantic query, seen in Figure 3.1b was performed to only visualise the building that had a noise impact greater than 65 decibels.

(a) (b)

Figure 3.1: Traffic noise map generated with noise data and CityGML. The road of the study area is the thick red line and the surrounding areas are generated with the help of LOD2 CityGML buildings. On figure (a) sound levels are represented as decibel by the colours of the 3D buildings. The values range from 58 (green) to 70 (red) or more. On figure (b) a query has been performed to only represent buildings who are affected with a decibel value greater than 65 (Konde and Saran 2017, page 256).

The study concluded that it had developed an economical solution for representing noise data and its impacts by only using open source applications such as CityGML and 3DCityDB. Future recommendations emphasised greater detailed 3D buildings to also analyse the noise impact inside the buildings. Such analysis would require LOD4 to describe details regarding the interior.

Urban energy modeling and simulation

3D City Database has been integrated in an augmented- (AR) and virtual reality (VR) environments for energy modeling. Santana et al. (2017) created an application to visualize energy simulations and model the results to buildings objects. This was done in an application that could transition between traditional virtual maps, AR and VR. To represent the 3D buildings CityGML was used. Administrators and authorities within the building process have a special interest in this sort of application as it allows them to study 3D buildings in place and gain an overview of the surrounding by switching to virtual maps (Santana et al. 2017). The application was developed for mobile devices to reduce the amount of equipment required in the field. For energy simulations, a solar radiation model with time series data spanning over a year was used and integrated with CityGML building models. The user was able to select a building of their choice and apply solar radiation data to view how it was affected. This specific use-case is beneficial when studying potential implementations of solar panels (Santana et al. 2017). The study was welcomed with great curiosity for its applicability in real life projects, and future research will focus on the expansion of new models with different simulations.

Flood modeling

Borkowski et al. (2016) conducted a study to implement a system for analysing and visualising a flood simulation. The inspiration behind the study was the potential environmental damage floods can cause in urban areas. Flood models can be used in decision-making when preparing or handling the destruction that floods cause on society. Such models can be created with the help of 2D maps and geographic information system, but they are inferior to 3D-models when attempting to replicate reality (Borkowski et al. 2016). Therefore, the study aimed at creating a

3D representation of the study area. To achieve this, a digital terrain model (DTM), in combination with buildings and trees represented in 3D was created. The buildings in this area were represented in the CityGML format with LOD2. To store the data 3DCityDB was chosen because of its ability to properly map CityGML for storage.

The importance of standardising spatial information to formats such as CityGML and 3DCityDB when creating flood simulations was highlighted in the conclusion. The study strived to use such formats, and as a result the flood system analysis created was open and independent when realising the 3D-solution.

3.5. BIM and GIS integration

Much time and effort have been spent on improving BIM and 3D GIS integration (Liu et al. 2017). Integrating these two data sources can provide many advantages to users of 3D city data. A 3D city model can for instance be updated using newly created BIM models with enriched information (Julin et al. 2018). However, there are several challenges with this integration as BIM and GIS has background in two very different fields. 3D GIS data, such as CityGML and CityJSON, focuses on describing spatial information through coordinates and can accurately place the location of an object. This information is then often used in analyses describing objects on a large scale. BIM focuses on describing a buildings life-cycle through information flow and management, but has challenges with georeferencing (Liu et al. 2017). By integrating GIS and BIM the fields would naturally complement each other as they have their own unique strengths and weaknesses. GIS can describe objects on a large scale with surrounding information through spatial analyses but has limited information regarding the objects itself. BIM on the other hand has abundant information, but limited possibilities to describe it with neighbouring objects.

One of the biggest challenges when attempting to integrate BIM and GIS is information loss and change during conversion (Liu et al. 2017). While GIS has been around for many years, BIM and IFC are relatively young conceptions. Also, the standards have different backgrounds and use cases. Liu et al. (2017) suggests that new developing standards might bridge BIM and GIS closer together by working around their challenges. These challenges have been researched in a report by Gilbert et al. (2020). The report had affiliations in both buildingSMART International (bSI) and Open Geospatial Consortium (OGC). bSI is responsible for the BIM standard IFC, while OGC is responsible for CityGML. The report presented the potential challenges and a proposal of actions to further integrate the standards. However, standards are very time and price consuming and no standard can cover all thematic objects found within an urban environment. Attempts have also been made to convert and extend the already existing standards (CityGML and IFC) to make them more interoperable. The issue with this approach is that the standards lack the appropriate structure for representing the entire lifecycle for urban objects (Liu et al. 2017).

4. Case study

4.1. Background – 3CIM

The three largest municipalities in Sweden, Stockholm, Gothenburg and Malmö and Lund University, have joined forces in a project focused on 3D city information modeling. The project, 3CIM, studies the requirements for information modeling in order to make 3D models operable for complex analyses and simulations. The information model generated by the project will be tested in a rainfall analysis. 3CIM points out the large potential such environmental

analysis has for the society, but that there is a gap between existing 3D models and the information required to do them. Current Swedish 3D-models mainly have information required for visualisation and there is a need to connect additional information from the municipalities. The building model created in 3CIM will be a Swedish profile based on CityGML which has been researched in the ongoing work for a national specification[10].

4.2. Malmö municipality

The Malmö city planning office is a main user of 3D building data. Their building data set covers all the buildings in Malmö that have a footprint larger than 30 square meters. Only existing buildings are modelled (i.e., no planned). In addition, Malmö municipality can provide complimentary terrain and surface models. These 3D models are made available for the public through a paid service by offering extractions which can be used for whichever purpose the buyer sees fit. Two examples are 3D-printing and visualisation. Malmö's current database environment is Microsoft SQL, but Malmö city are looking into the possibilities of storing CityGML in the database environment 3DCityDB[11]. This will allow Malmö municipality to store an almost complete 3D model of the city in a database environment. The environment can be used to visually experience how the city changes over time as new updates to the model can be added or altered.

4.3. 3D Building data environment

To enhance Malmö city planning office, and other municipalities, usage of 3D building data this master thesis creates a 3D building data environment. The environment has the functionality for transforming, importing and exporting 3D building data originating from NS building. A vision model of this environment was presented in the introduction and can be seen in Figure 1.2. The NS building data are transformed to CityGML and imported to a 3D database environment for storage. From there the data should be made available by exporting to the original format CityGML, but also CityJSON. By expanding to CityJSON more users of 3D building models can be reached as it can be an easier format to work with.

5. Methodology

5.1. Overview

The methodology is split into five steps (Figure 5.1). Step A involved mapping geometrical components from the Swedish national specification building (NS building) to the international standard CityGML. The building objects were then verified by Val3dity to ensure that the buildings were properly mapped. If not valid, the buildings were fixed according to the error messages from Val3dity and tested again until proper objects were made. The CityGML building object was then imported to a PostGIS database with the 3DCityDB extension for storage (step B). Step C involved exporting data from the database to CityGML (version 2.0) and the CityJSON format. In step D Val3dity was once again used to ensure valid 3D building objects had been exported. In the last section, step E, the models were tested in GIS and 3D software to ensure the usability. The CityGML models were tested in FZKViewer, and CityJSON models were tested in the CityJSON loader plugin available in QGIS. Both applications were mentioned and discussed in Section 3.2.

[10] https://www.smartbuilt.se/in-english/projects/open-call-8/3cim/
[11] https://www.3dcitydb.org/3dcitydb/

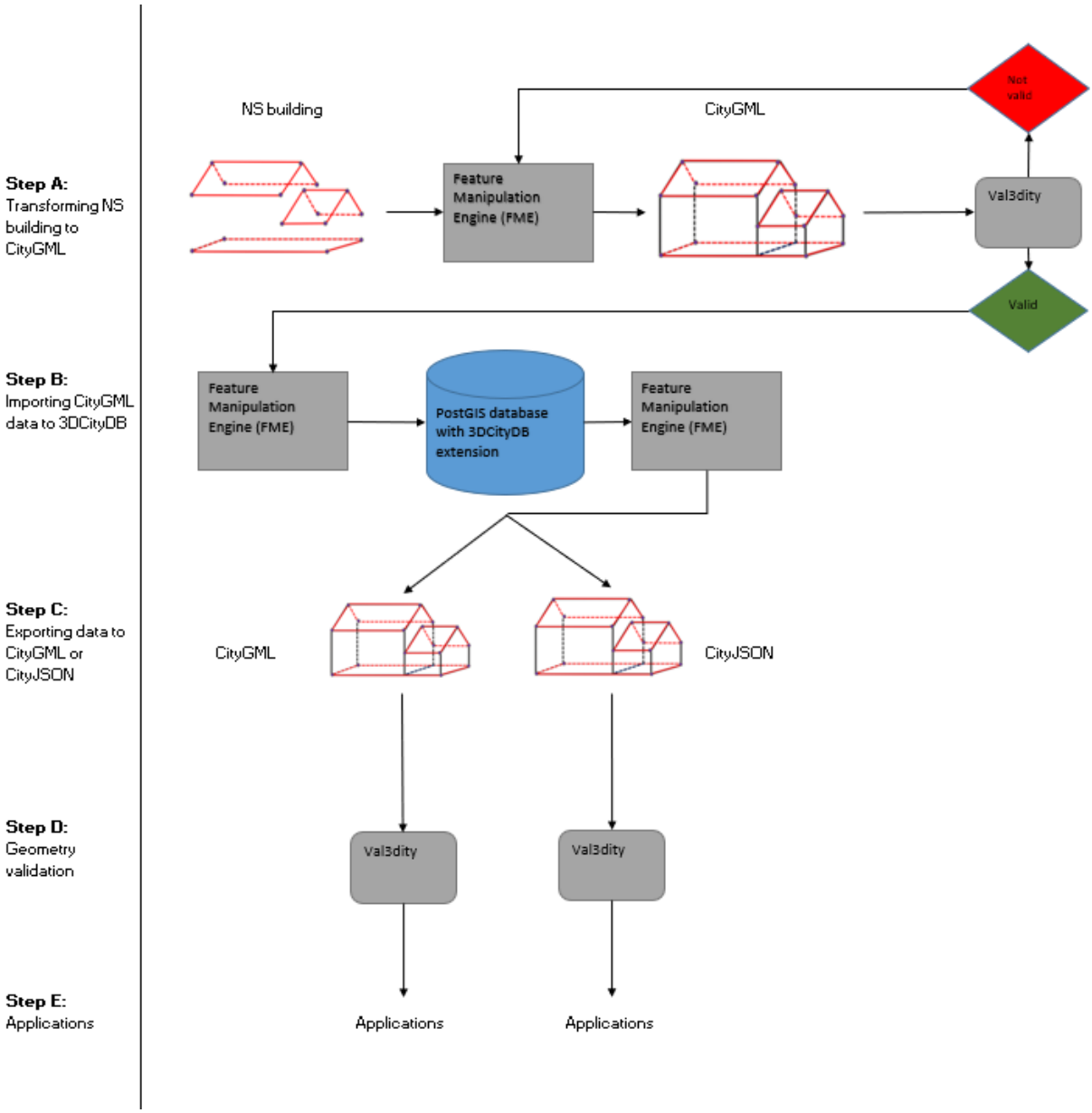


Figure 5.1: Overview of the general workflow. Step A transforms NS building data to CityGML. Step B imports the data to 3DCityDB. Step C exports to CityGML and CityJSON. Step D validates the geometries. Lastly, step E tests the data in applications.

To transform the NS building components, to handle the import and export to and from 3DCityDB the ETL (Extract Transform Load) tool Feature Manipulation Engine[12] (FME) was used. FME is a software created by Safe Software which offers powerful data transformation and manipulation processes. FME is highly suitable for handling 3D-formats as it can read and write to a vast list of different formats within the field. Two of these formats are CityGML and CityJSON. FME is also capable of validating geometries and schemas which is critical to ensure consistent transformations. FME also supports connections to databases such as PostGIS and has the functionality to import and export data. A central part of this thesis is to ensure the proper creation of geometries within FME based on NS buildings components. All FME scripts

[12] https://www.safe.com/fme/fme-desktop/

created in this master thesis are available for download from GitHub[13] with a BSD -3-Clause License[14].

5.2. Test data

Four buildings are used in the study:

- Kristallen, the municipality house in Lund, Figure 5.2a.
- Multihuset in Malmö, Figure 5.2b.
- Undervisningshuset at the Royal Institute of Technology (KTH) in Stockholm, Figure 5.3a.
- Kanaan's Garden Café in Stockholm, Figure 5.3b.

(a) (b)

Figure 5.2: Test data. (a) Kristallen (source: Lund municipality). (b) Multihuset (source: NCC).

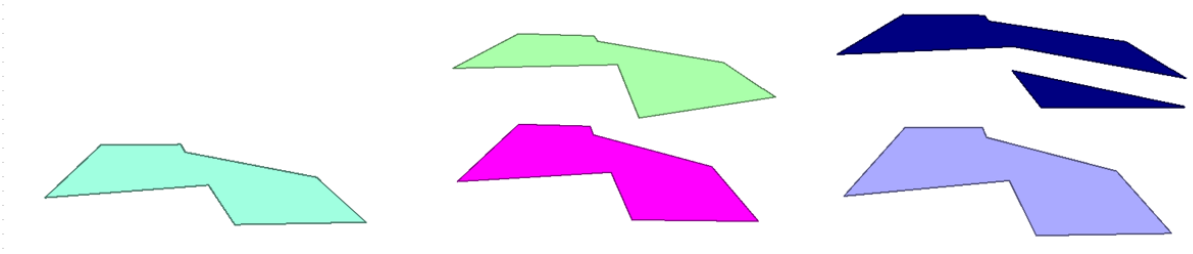
(a) (b)

Figure 5.3: Test data. (a) Undervisningshuset (source: Christensen & co arkitekter). (b) Kanaan's Garden Café (source: Stockholms stad).

The test data from Kristallen and Undervisningshuset were originally BIM models created by the architect company Christensen & Co. Architects and Kanaan's Café test data was also a BIM model, but generated from laser scanning (Sun et al. 2020). The last test model, Multihuset, was a product of modifying an already existing BIM model with cadastral

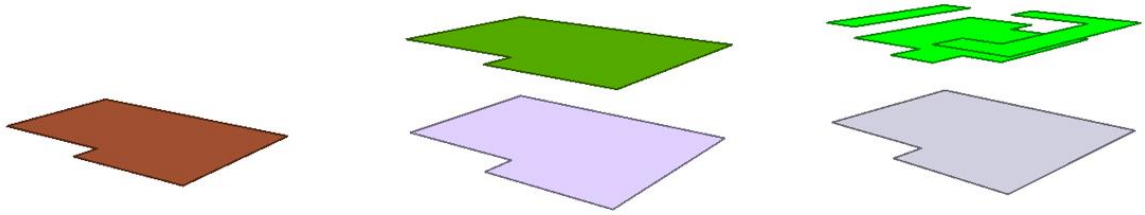
[13] https://github.com/tomaben/Master-thesis-
[14] https://opensource.org/licenses/BSD-3-Clause

information (Sun et al. 2019). All the four BIM models were converted to NS building in FME (made by Per-Ola Olsson, Lund University). In Figure 5.4 to Figure 5.7 the NS building data is shown.



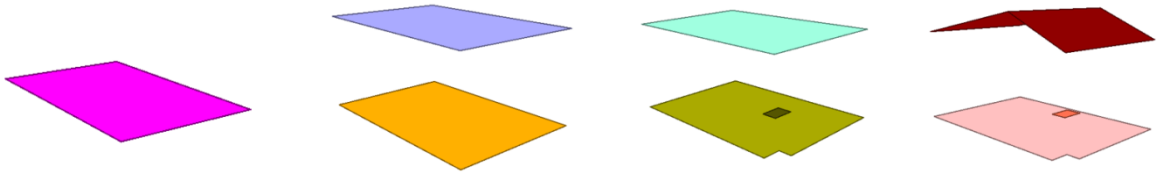
(a) (b) (c)

Figure 5.4: Test data NS building. (a) Kristallen Lund. NS LOD 0.1. (b) NS LOD 0.2. (c) NS LOD 0.3.



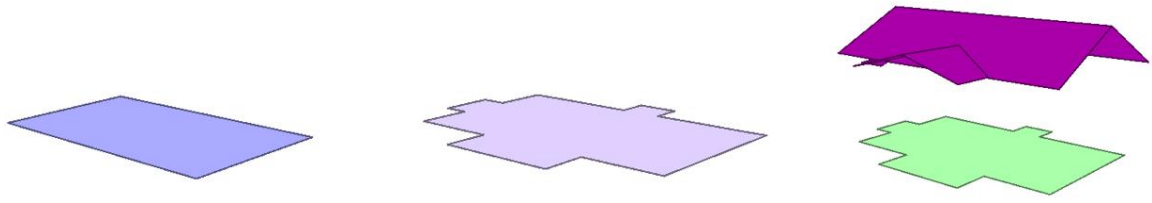
(a) (b) (c)

Figure 5.5: Test data NS building. (a) Multihuset Malmö. NS LOD 0.1. (b) NS LOD 0.2. (c) NS LOD 0.3.



(a) (b) (c) (d)

Figure 5.6: Test data NS building (a) Undervisningshuset Stockholm. NS LOD 0.1. (b) NS LOD 0.2. (c) NS LOD 0.3. (d) NS LOD 2.0.



(a) (b) (c)

Figure 5.7: Test data NS building (a) Kanaan's Garden Café Stockholm. NS LOD 0.1. (b) NS LOD 0.2. (c) NS LOD 2.0.

5.3. Step A: Transformation from NS building to CityGML

In this study the mapping of geometrical components from NS building to CityGML of different LODs was tested. This sort of mapping from NS building to CityGML had not been done before

and was vital to verify. Therefore, only geometries were tested as a first step, and no attributes were taken into consideration. Only attributes that were vital for the CityGML standard or the storage of objects in 3DCityDB were created.

From Figure 5.8 we can see how NS building data is transformed to CityGML: the first row, NS LOD 0.x, portrays two examples of how the geometrical components from NS building can be described and stored in a database. Additional components with more detailed roof structures can also be stored (Figure 5.8, Example 2). LOD 0.1 can be transformed to CityGML LOD 1.1, but attributes describing height are necessary. LOD 0.3 can be transformed to LOD 1 without any further attributes. All the buildings mentioned in section 5.2. were transformed, an overview of the transformation workflow is given in Figure 5.9.

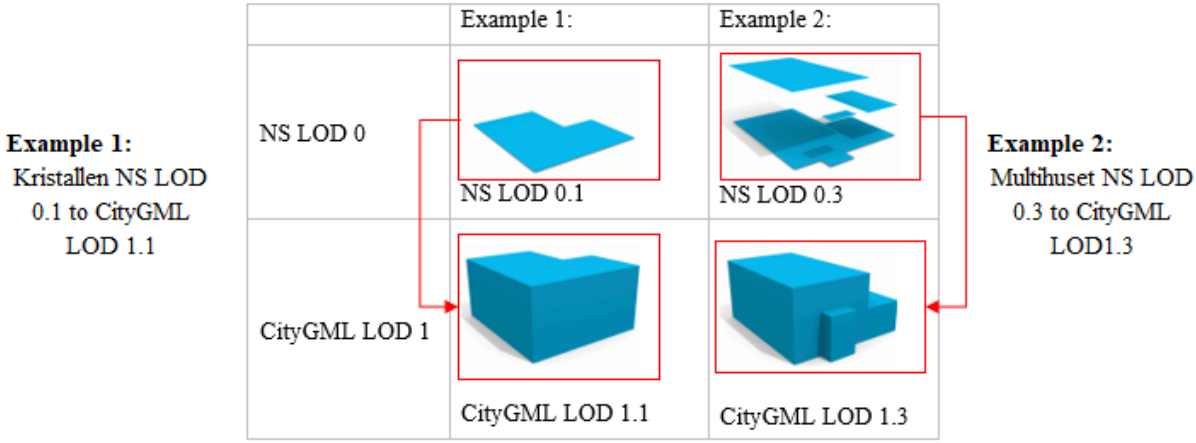


Figure 5.8: Two examples describing how NS LOD can be stored and transformed to CityGML.

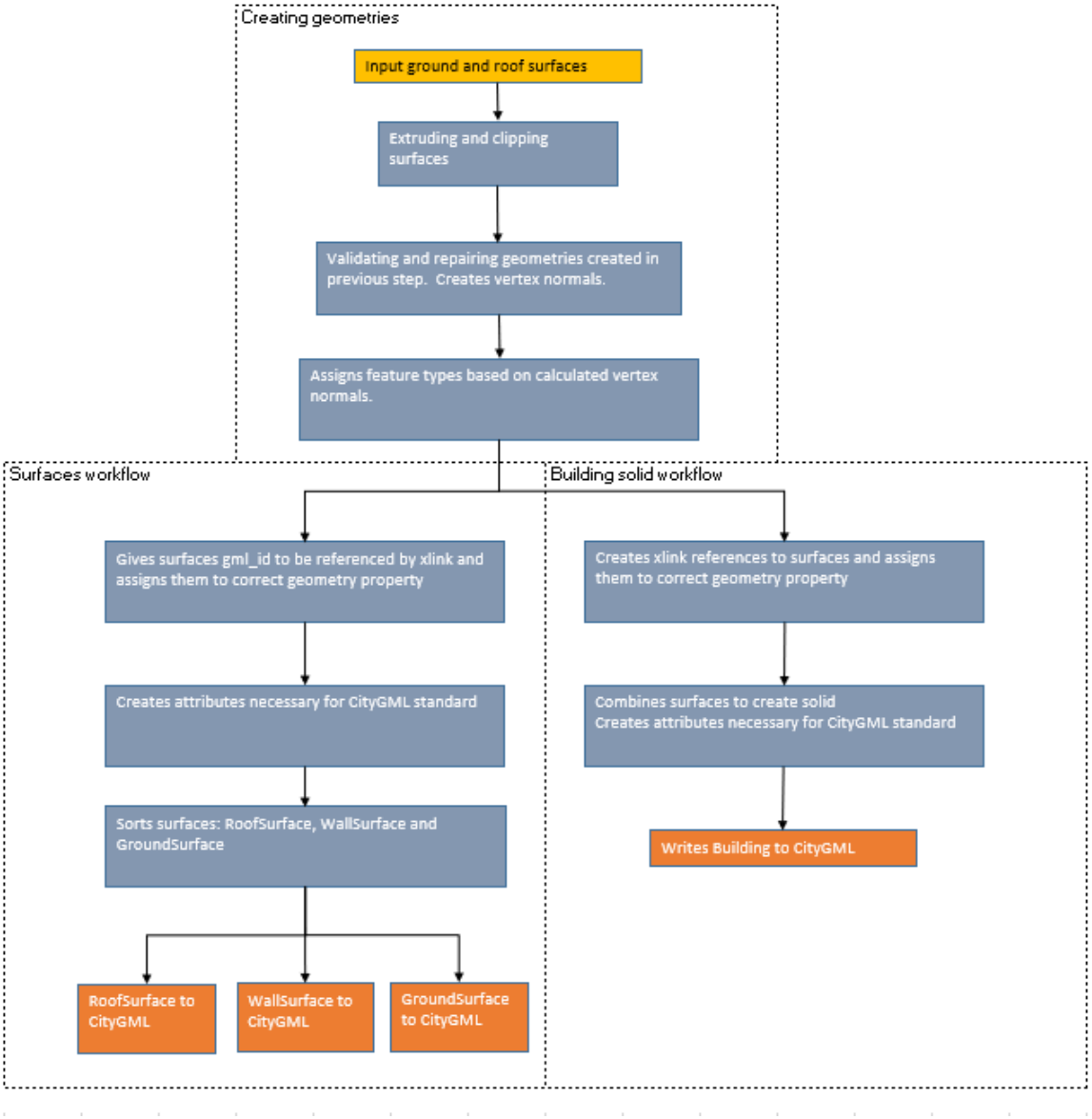
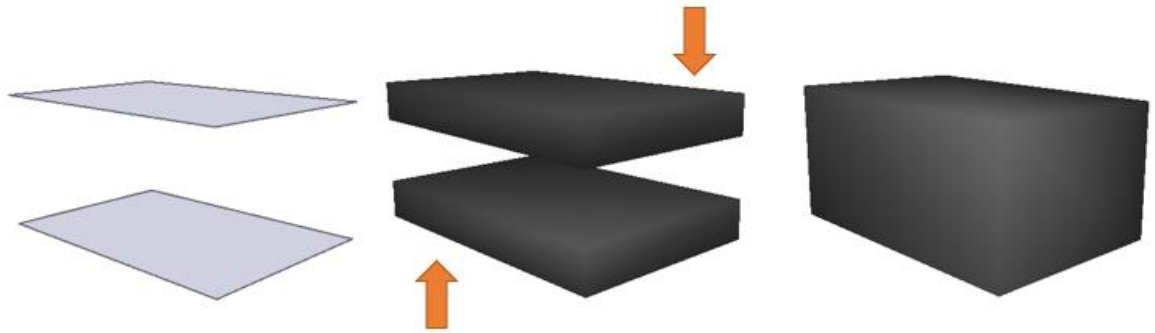


Figure 5.9: Workflow: Transforming NS building to CityGML.

The NS building data, seen in Figure 5.4 to Figure 5.7, was imported in FME and the roof and ground surfaces were separated. The roof surface was then extruded downwards below the lowest point of the ground surface, while the ground surface extruded upwards above the highest point of the roof surface. The intersection of these extruded surfaces was then clipped, and the common geometric areas of both surfaces was used as a solid representation of the building. An illustration of this can be seen in Figure 5.10.



(a) (b) (c)

Figure 5.10: The extrusion and cutting of NS LOD data. (a) Imported roof and ground surface. (b) Roof surface extruded downwards while ground surface extruded upwards. (c) The extruded surfaces cut.

For Kanaan's Café another extrusion process was necessary to correctly create the roof overhangs of the buildings. The same approach was employed, but instead of using the intersections of the geometries, the areas outside were extracted.

To ensure that the surfaces representing the 3D building were in accordance to 3D GIS standards, planar surfaces (flat two-dimensional surfaces) were checked and repaired if failed using built-in transformers in FME. Vertex normals were then calculated to separate the surfaces to three categories: *RoofSurface*, *WallSurface* and *GroundSurface*. The workflow was then split: The surfaces defining the building solid were sent into one workflow, while the surfaces defining the thematic portion of the building in another. Xlinks were then assigned to the building object which referenced the surfaces defining the solid.

Attributes unique to CityGML were created and assigned for both workflows. These attributes included *level of detail*, *gml_id* and *gml_parent_id*. *gml_id* and *gml_parent_id* attributes were mapped to correctly represent the hierarchy of the objects. Lastly, the complete building object and building surfaces were stored as CityGML using the CityGML writer in FME.

5.4. Step B: Import to 3DCityDB

Figure 5.11 pictures a simplified UML building model of the database environment used to store the CityGML data. The original UML building model from 3DCityDB was too complex and contained an abundance of attributes and tables that were not necessary for the implementation of this master thesis. Therefore, a modified model was created to better fit the CityGML data transformed from the NS building data in step A.

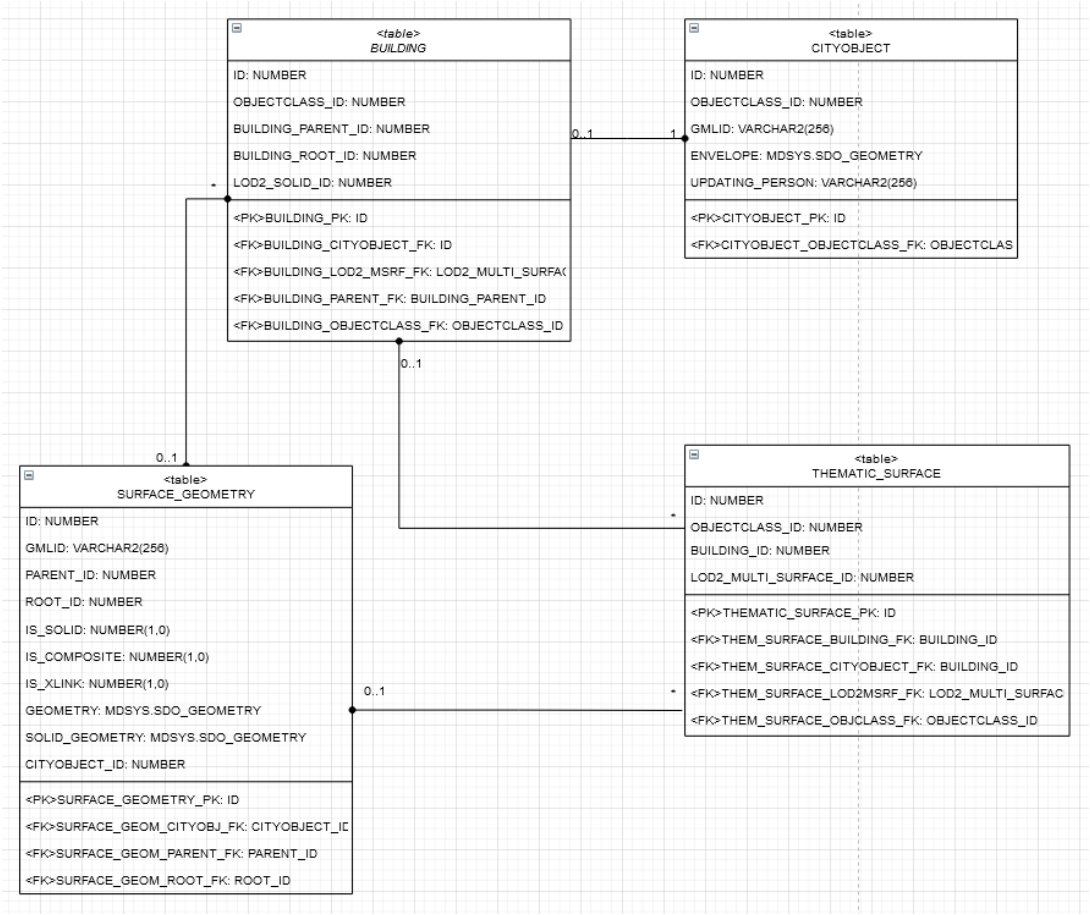


Figure 5.11: Simplified UML model 3DCityDB building.

3DCityDB has built-in functionalities that support importing CityGML data. It was a wish from the 3CIM project to create a process that could import CityGML data without using these functionalities. Doing so would allow for better information control and the created process could be modified in the future to support features that the built-in functions do not have. Such features could be updating, versioning or finding and fixing errors in a CityGML data set.

The CityGML data were transformed to LOD 1 and LOD 2, but only LOD 2 is described here as it was the most complex to import to the database. The import structure for LOD 1 is roughly the same as the first workflow for LOD 2, namely the building solid import. LOD 2 requires additional thematic surfaces, described in Section 2.1, like *GroundSurface*, *WallSurface* and *RoofSurface* to be explicitly modelled. A workflow of the import process can be seen in Figure 5.12.

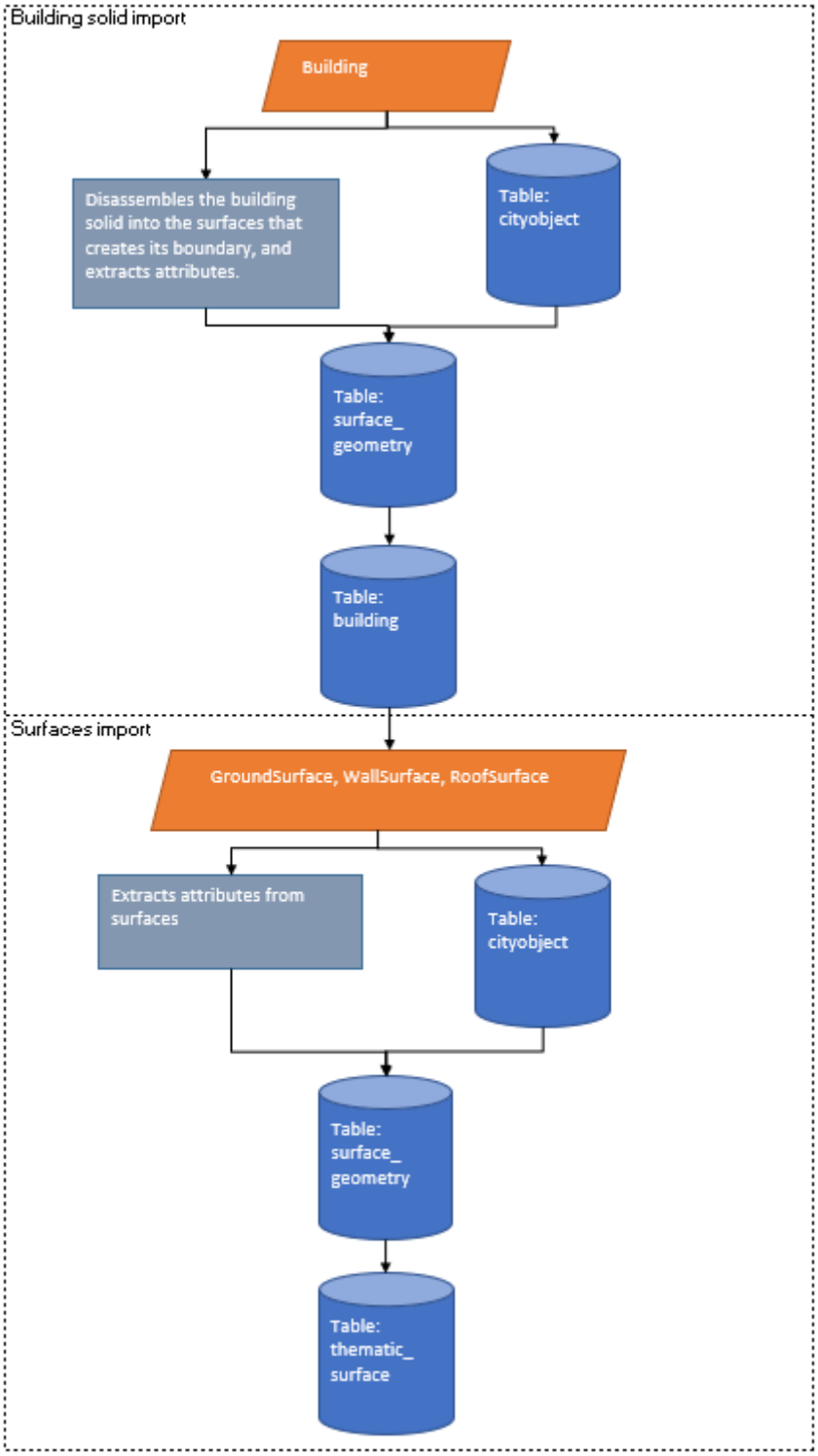


Figure 5.12: Workflow: Importing CityGML to 3DCityDB.

The building solid was disassembled into the surfaces defining the boundary of the solid. In a parallel workflow, the building object was stored in the *cityobject* table. Required attributes for the table *surface_geometry* were extracted from the model parts and the object was then split into three: the actual solid, the root object and lastly all the boundary surfaces defining the solid. These objects were then written to the table *surface_geometry* in the correct order and with correct attributes so the integrity of the table was kept. An SQL query was then executed against the *surface_geometry* table to retrieve the attributes *id* and *gml_id* which are used to link to the

building table. Since this query would retrieve all objects in *surface_geometry* the results were filtered based on the building object. When filtered, all attributes were written to the *building* table. The building object was then stored in the appropriate tables of the database and the next part was handling the thematic surface geometries.

When storing CityGML data in 3DCityDB, surfaces can be aggregated to describe several surfaces, or the boundary of a solid object (Kolbe et al. 2019). This means that a surface will act as a root surface with no geometries, while other surface(s) (with geometries) references the root surface. The first surface imported was the *GroundSurface*. Necessary attributes were retrieved from the surface, and at the same time *GroundSurface* was also written to the *cityobject* table. From here the surface was split in two: root surface and geometric properties. The objects were assigned SQL queries as text attributes and ID's determining which object they belonged to. Finally, the root and geometric objects were intertwined by sorting them based on their workflow ID and type. The objects were now sorted so that each surface was represented as a root object and geometric object, and then written to *surface_geometry* based on the SQL text attributes. A new SQL query was executed against the newly written *surface_geometry* table, but this time *root_id*, *gml_id* and *cityobject_id* was extracted to be prepared for the table *thematic_surface*. Similar to the building solid workflow, the correct objects were filtered using *gml_id* and assigned the previously extracted attributes. Lastly, the objects were assigned to the *thematic_surface* table. The process was then repeated in an identical fashion for *WallSurface* and *RoofSurface.*

5.5. Step C: Export to CityGML and CityJSON

The built-in functions of 3DCityDB supports export to CityGML, KML, COLLADA and glTF. To reach a wider range of users and create more applications to generate 3D city model data, the export process has been built to export to CityJSON as well as CityGML. Currently CityGML has more available software for reading data when comparing to CityJSON. CityJSON does however have a good potential for being a better format when considering web and mobile applications as reported by Ledoux et al. (2019). The software available for CityGML is also mainly proprietary.

5.5.1. CityGML

The export process shared many similarities to the transformation process (step A). However, there were some changes when handling the geometries and attributes necessary for CityGML and CityJSON. Instead of creating them based on NS LOD they were retrieved from 3DCityDB. A set of SQL-queries (Table 5.1 to Table 5.3) had to be executed on the various tables, and the results had to be altered and assigned to correct objects.

To start the export process the *gml_id* of the building object to be exported was used as an input variable as it is unique for each building. Figure 5.13 describes the general workflow for exporting CityGML LOD 2. CityGML LOD 1 is not explained, as it is very similar to LOD 2, the major difference being that LOD 1 has no surfaces, only a solid building object explained in the building solid workflow.

Figure 5.13: Workflow: Exporting CityGML LOD 2 from 3DCityDB. The bold dashed lines reference which SQL-query was used from Table 5.1 to Table 5.3.

The *gml_id* attribute was used to query the *cityobject* table and retrieve the *cityobject_id* of the building (Figure 5.13: SQL-query 1). Then *cityobject_id* was used to connect and query the *surface_geometry* table to retrieve the *gml_id* of all surfaces defining the solid (Figure 5.13: SQL-query 2). When the IDs were collected a new query was sent to *surface_geometry*, but this time all relevant attributes and geometries for creating the surfaces (*RoofSurface,*

WallSurface, GroundSurface) were collected (Figure 5.13: SQL-query 3). By using *root_id* the remaining root surfaces, with attributes, were collected (Figure 5.13: SQL-query 4). The data was then split in two: surfaces and building solid. The geometries of both workflows were then converted to the proper type for further processing. Lastly, in a similar approach as the workflow in step A, *gml_id* and xlinks were assigned to the correct geometries and necessary attributes for the CityGML standard were set. To classify the surfaces a last query was done on the *cityobject* table to retrieve IDs specifying the feature types (Figure 5.13: SQL-query 5). The surfaces and building solid were then stored in the CityGML format using the built-in writers in FME.

Table 5.1: SQL-query 1 and 2.

SQL-query	1	2
SQL-statement	SELECT id as cityobjectid FROM citydb.cityobject WHERE gmlid = '$(gml_id_export)'	SELECT gmlid FROM citydb.surface_geometry WHERE cityobject_id = '@Value(cityobjectid)'
Description	Finds the *cityobject_id* of the building to be exported from *cityobject* using input *gml_id.*	Finds all *gmlids* of the building solid using previously obtained *cityobject_id*.

Table 5.2: SQL-query 3 and 4.

SQL-query	3	4
SQL-statement	SELECT gmlid, root_id, geometry, cityobject_id FROM citydb.surface_geometry WHERE gmlid = '@Value(gmlid)'	SELECT gmlid as gml_id, '@Value(geometry)' as geometry FROM surface_geometry WHERE '@Value(root_id)' = root_id
Description	Retrieves necessary attributes for recreating the surfaces (*RoofSurface, WallSurface, GroundSurface*) using previously obtained *gmlid*.	Uses *root_id* to retrieve root surfaces with attributes.

Table 5.3: SQL-query 5.

SQL-query	5
SQL-statement	SELECT objectclass_id FROM citydb.cityobject WHERE gmlid = '@Value(gml_id)'
Description	Retrieves *objectclass_id* for each surface.

5.5.2 CityJSON

In CityGML a building object is the parent of all surfaces (*GroundSurface*, *WallSurface* and *RoofSurface)*. As mentioned in Section 2.2 the CityJSON format has reduced complexity compared to CityGML by removing all hierarchical dependencies, and therefore no such parent-child structure exists. Instead, CityJSON stores such thematic surface information as attribute values, and each surface that uses those values points to it (Ledoux et al. 2019).

Because of the simplified assigning of thematic surfaces in CityJSON the approach for exporting CityJSON models from 3DCityDB was very similar as the CityGML LOD 1 export process. The main difference was renaming and structuring necessary attributes to comply to the CityJSON requirements. The *gml_id* of the building object was renamed *fid* and *lod* renamed *cityjson_lod*. The semantic information (*GroundSurface*, *WallSurface* and *RoofSurface)* was stored as geometric properties for the surfaces that were applicable. The same tables were used to access the geometric values.

5.6. Step D: Geometry validation

Before importing the objects to 3DCityDB and after exporting them to CityGML or CityJSON, the objects had to be controlled that they were in fact valid 3D solids. For this the 3D validator Val3dity was used.

5.7. Step E: Applications

To ensure the usability of the exported CityGML/CityJSON data they were tested in applications freely available for download. For CityGML the FZKViewer was used. As previously mentioned, not many software has support for reading and visualising CityGML. If they do, they are proprietary and expensive licences apply. FZKViewer was easily available and therefore it was chosen. For CityJSON the CityJSON loader plugin created by Vitalis et al. (2020) was used. This plugin was available in QGIS for direct download. Both QGIS and the plugin were open-source and free to use, meaning that this solution would be more accessible for users without expensive licences.

6. Results

6.1. Step A: Transformation from NS building to CityGML

Table 6.1 shows the transformed NS building data in the CityGML LOD 1 format. All CityGML LOD 1.1 models were transformed from NS LOD 0.1. Likewise, CityGML LOD 1.2 were transformed from NS LOD 0.1. Lastly, CityGML LOD 1.3 were transformed from NS LOD 0.3. The exported models for Kristallen looked very similar but had different roof structure. Kristallen LOD 1.1 was completely flat, while LOD 1.2 had a slight tilt facing downwards on the right portion of the roof (when viewed in Table 6.1). Instead of a tilted roof structure, Kristallen LOD 1.3 flattened that portion of the roof and moved the structure one floor down.

Multihuset LOD 1.1 and LOD 1.2 were close to identical. The difference between them was the test data, where LOD 1.1 only contained a ground surface that was extruded to a calculated height, while LOD 1.2 had both ground and roof surface that were combined to create the same models. Undervisningshuset LOD 1.1 and 1.2 were created with a similar approach. Multihuset LOD 1.3 had additional roof details and Undervisningshuset LOD 1.3 was modelled with entrance area. No test data for Kanaan's Café NS LOD 0.3 were made available, therefore the cell is empty.

Table 6.1: CityGML LOD 1.x

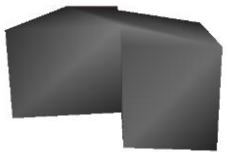
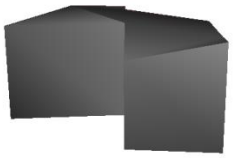
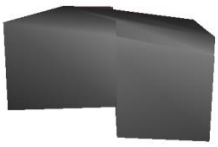
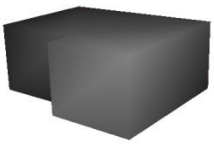
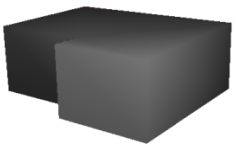
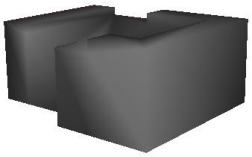
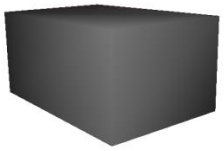
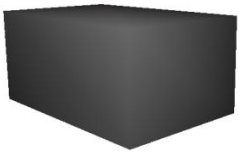
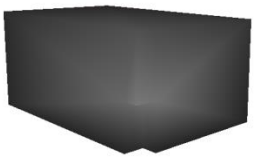
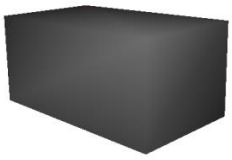
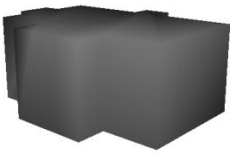
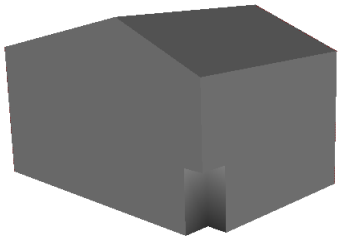
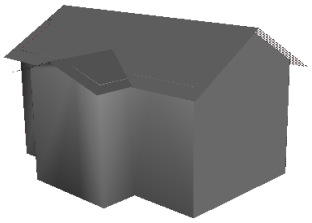
	CityGML LOD 1.1	CityGML LOD 1.2	CityGML LOD 1.3
Kristallen			
Multihuset			
Undervisningshuset			
Kanaan's Café			No test data

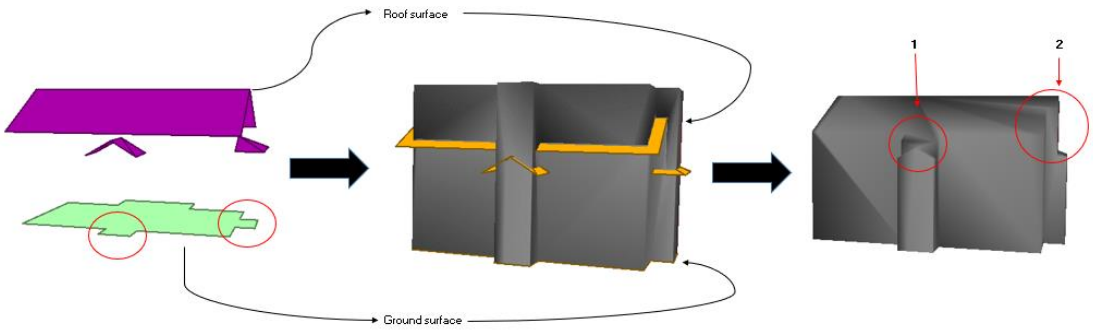
Table 6.2 shows two data sets in CityGML LOD 2 derived from NS LOD 2.0 using the workflow in Figure 5.9. Undervisningshuset LOD 2.0 were modelled with detailed roof structure in addition to the entrance area introduced in LOD 1.3. Kanaan´s Café LOD 2.0 also had detailed roof structure but with roof overhangs.

Table 6.2: CityGML LOD 2.0

	CityGML LOD 2.0
Undervisningshuset	

Kanaan's Café	

The ground surface (footprint) of Kanaan's Café was defined as both entrance areas and the actual base of the building (Figure 6.1a). As a result, the extruded surfaces creating the building solid also extruded parts of the entrance areas as the building solid (Figure 6.1b and c). These areas, highlighted in Figure 6.1c and marked as 1 and 2, were not actually part of the physical walls, but created because there were no possibility to separate entrance areas from base ground surface. Therefore, roof overhangs over the entrances were not correctly modelled.



(a) (b) (c)

Figure 6.1: (a) Kanaan's Café NS LOD 2.0). (b) Extruded ground and roof surface. (c) Final cut building solid.

When the detailed roof surface of Kanaan's Café was extruded downwards and intersected with the ground surface several double points and small triangles were created in the ground surface. Figure 6.2 shows the feature type *GroundSurface* of Kanaan's Café after intersection. Double points were set up to be repaired in FME by using a built-in tool, but this did not succeed. Small triangles were attempted to merge to create larger triangles, but this was mainly successful for the feature types *RoofSurface and WallSurface*, while *GroundSurface* experienced less fixes. The double points and small triangles of *GroundSurface* was never repaired, and therefore Kanaan's Café was not properly transformed.

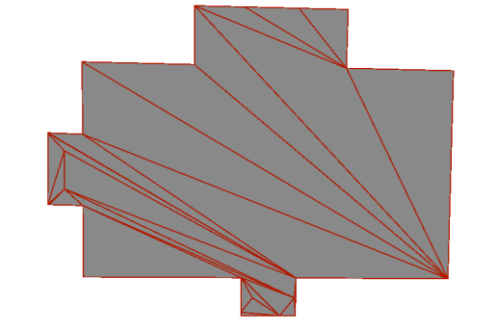


Figure 6.2: Double points and small triangles in GroundSurface in Kanaan's Café.

The calculated vertex normals used to determine which feature type the surfaces belonged to created some errors. Not all generated surfaces were planar, which led to the normals pointing in the wrong direction and thus being wrongly classified. In these cases, the surfaces had to be manually directed to the correct feature type. Mainly *WallSurface* experienced this error.

Figure 6.3 provides an example of a wrongly classified wall (red triangle) when transforming Undervisningshuset NS LOD 2.0 (Figure 5.6d) to CityGML LOD 2. The walls were expected to receive a normal value of zero which would translate to the normals pointing straight out from the surface. Because of the wall surface not being vertical to the ground surface, the normal instead received a very low negative number. This led to the script wrongly classifying it as a *GroundSurface*. This error had to be resolved by accessing the script and allowing it to classify such low negative numbers as *WallSurface*.

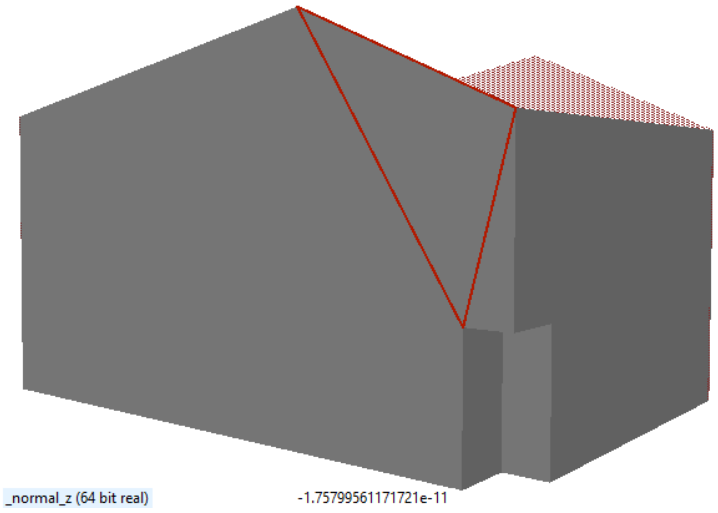


Figure 6.3: Wrongly calculated vertex normals for a WallSurface (red triangle) on Undervisningshuset LOD 2.0.

All CityGML models, except for Kanaan's Café LOD 2, were validated in Val3dity with a 100% valid result. Kanaan's Café had two errors: many consecutive/double points (previously explained) and one non-planar surface (Figure 2.12). The distance to the plane was 0.0101105 m, but Val3dity only allowed a tolerance of 0.01 m. By using FME's built-in *GeometryValidator*[15] an attempt was made to repair the non-planar surface, but the script was not able to recognize the error and therefore could not be repaired.

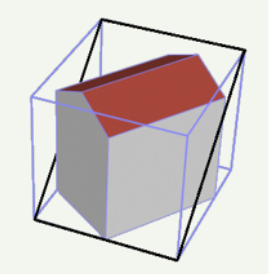
[15] https://www.safe.com/transformers/geometry-validator/

6.2. Step B: Import to 3DCityDB

In this section two of the CityGML data sets generated in step A are presented and it is described how they were imported to 3DCityDB. Multihuset LOD 1.3 (Table 6.1) and Undervisningshuset LOD 2 (Table 6.2). All CityGML data sets converted from NS building were imported, but only two of these data sets are explained in this section.

Figure 6.4a shows the resulting table *cityobject* after the two models were imported. The ID column stored the primary key of the table and was stored as an increasing integer value starting at 1. *Objectclass_id* registered which feature type the objects represented. In accordance to the 3DCityDB standard (Kolbe et al. 2019) the following *objectclass_id* was assigned for each CityGML feature type: *Building*-26, *GroundSurface*-35, *WallSurface*-34, *RoofSurface*-33. For storing *gmlid* the attributes created in Step A for each CityGML object were used. To create the *envelope* attribute a 3D polygon was generated for each object that represented the minimum and maximum coordinates forming a box around the object (Figure 6.4b). The final attribute, *updating_person*, was used to keep track of who was updating the table.

	id [PK] integer	objectclass_id integer	gmlid character varying (256)	envelope geometry	updating_person character varying (256)
1	1	26	gml_d54735ff-0803-41d2-905c-2972490b6cf7	01030000A0C00B0000010000000500000...	Tom_Bendiksen
2	2	35	gml_d8c0deac-41df-4fd8-b76e-4df88dbd6a7e	01030000A0C00B0000010000000500000...	Tom_Bendiksen
3	3	35	gml_e2ef0ed7-84ca-4942-b0cd-bf0f62f36ebe	01030000A0C00B0000010000000500000...	Tom_Bendiksen
4	4	34	gml_ab2384d9-8caf-4ab2-8af3-fd9dfe7974af	01030000A0C00B0000010000000500000...	Tom_Bendiksen
5	5	34	gml_316e7bd0-bf0b-4a7d-8204-f4e86db7c6e9	01030000A0C00B0000010000000500000...	Tom_Bendiksen
6	6	34	gml_e87f14c2-7435-485f-8812-644f60896148	01030000A0C00B0000010000000500000...	Tom_Bendiksen
7	7	34	gml_2f6284ea-37d5-4fed-ba3d-37aaad24984d	01030000A0C00B0000010000000500000...	Tom_Bendiksen
8	8	34	gml_708662c3-d3e6-443c-99ec-23f35f70ee7b	01030000A0C00B0000010000000500000...	Tom_Bendiksen
9	9	34	gml_2338934f-9e0c-408b-8b24-36b10efbc5e1	01030000A0C00B0000010000000500000...	Tom_Bendiksen
10	10	33	gml_84d1eb02-36f8-4111-88c8-a47f06635b1a	01030000A0C00B0000010000000500000...	Tom_Bendiksen
11	11	33	gml_6ac5582d-15c4-409a-bf10-757e4b0918f6	01030000A0C00B0000010000000500000...	Tom_Bendiksen
12	12	26	gml_2fd46406-1213-4f63-8fcc-b4a0d69b9911	01030000A0C00B0000010000000500000...	Tom_Bendiksen



(a) (b)

Figure 6.4: (a) Resulting cityobject table after import. Undervisningshuset was imported first, then Multihuset. (b) Envelope definition in 3DCityDB (Source: 3DCityDB-docs).

Undervisningshuset LOD 2 was represented by rows 1-11 and both building solid (*objectclass_id* 26) and surfaces (*objectclass_id* 33, 34 and 35) were stored. Since LOD 1 only contains a solid representation of the building, Multihuset LOD 1 was stored in row 12 without any surfaces attached to it.

The building objects (Row 1 and 12 in Figure 6.4a) were also written to the *building* table (Figure 6.5). No geometries were stored here, only metadata and IDs used to connect to other tables. *Id*, *objectclass_id*, *building_root_id* and *lod2_solid_id* were the attributes stored. *Id* and *objectclass_id* were stored similar to the *cityobject* table. A building in CityGML may be represented by several different building parts (Gröger et al. 2012). Because of this the *building_root_id* keeps track of the root, or base, part of the building and all the buildings parts must reference the root. However, none of the test data had any building parts, and therefore the *building_root_id* always referred to itself as the top level of the building root. *Lod2_solid_id* (1) and *lod1_solid_id* (33) were used as references for the solid geometries being stored in the *surface_geometry* table.

	id [PK] integer	objectclass_id integer	building_root_id integer	lod2_solid_id integer	lod1_solid_id integer
1	1	26	1	1	[null]
2	12	26	12	[null]	33

Figure 6.5: Resulting building table after import.

The geometries of both the building solid and surfaces were stored in the *surface_geometry* table (Figure 6.6). This table stored the IDs *id, gmlid, parent_id, root_id* and *cityobject_id* for each object stored. The IDs were stored similarly to the previously mentioned tables except for *cityobject_id* which referred to the same object stored in the *cityobject* table (Figure 6.4) and could be used to connect to them. For each solid registered an additional attribute *is_solid* is stored as 1, and for each surface registered the value is 0. This is used to differentiate the solids and surfaces. Lastly, the actual geometries were stored as either *solid_geometry* for the solid or *geometry* for the surfaces.

	id [PK] integer	gmlid character varying (256)	parent_id integer	root_id integer	is_solid numeric	is_composite numeric	is_xlink numeric	solid_geometry geometry	geometry geometry	cityobject_id integer
1	1	UUID_afc2a0cd-a551-4a5c-adde-72876726ee38	[null]	1	1	0	0	010F0000A0C00B00000A...		1
2	2	UUID_777e4997-b3d9-42e7-b3a5-aa5e3b6fcbb7	1	1	0	1	0			1
3	3	gml_d8c0deac-41df-4fd8-b76e-4df88dbd6a7e_polygon	2	1	0	0	1		01030000A0C00B0...	1
4	4	gml_e2ef0ed7-84ca-4942-b0cd-bf0f62f36ebe_polygon	2	1	0	0	1		01030000A0C00B0...	1
5	5	gml_316e7bd0-bf0b-4a7d-8204-f4e86db7c6e9_polygon	2	1	0	0	1		01030000A0C00B0...	1
6	6	gml_708662c3-d3e6-443c-99ec-23f35f70ee7b_polygon	2	1	0	0	1		01030000A0C00B0...	1
7	7	gml_2f6284ea-37d5-4fed-ba3d-37aaad24984d_polygon	2	1	0	0	1		01030000A0C00B0...	1
8	8	gml_ab2384d9-8caf-4ab2-8af3-fd9dfe7974af_polygon	2	1	0	0	1		01030000A0C00B0...	1
9	9	gml_2338934f-9e0c-408b-8b24-36b10efbc5e1_polygon	2	1	0	0	1		01030000A0C00B0...	1
10	10	gml_e87f14c2-7435-485f-8812-644f60896148_polygon	2	1	0	0	1		01030000A0C00B0...	1
11	11	gml_84d1eb02-36f8-4111-88c8-a47f06635b1a_polygon	2	1	0	0	1		01030000A0C00B0...	1
12	12	gml_6ac5582d-15c4-409a-bf10-757e4b0918f6_polygon	2	1	0	0	1		01030000A0C00B0...	1
13	13	gml_d8c0deac-41df-4fd8-b76e-4df88dbd6a7e	[null]	13	0	0	0			2
14	14	gml_d8c0deac-41df-4fd8-b76e-4df88dbd6a7e_polygon	13	13	0	0	1		01030000A0C00B0...	2
15	15	gml_e2ef0ed7-84ca-4942-b0cd-bf0f62f36ebe	[null]	15	0	0	0			3
16	16	gml_e2ef0ed7-84ca-4942-b0cd-bf0f62f36ebe_polygon	15	15	0	0	1		01030000A0C00B0...	3
17	17	gml_ab2384d9-8caf-4ab2-8af3-fd9dfe7974af	[null]	17	0	0	0			4
18	18	gml_ab2384d9-8caf-4ab2-8af3-fd9dfe7974af_polygon	17	17	0	0	1		01030000A0C00B0...	4
19	19	gml_316e7bd0-bf0b-4a7d-8204-f4e86db7c6e9	[null]	19	0	0	0			5
20	20	gml_316e7bd0-bf0b-4a7d-8204-f4e86db7c6e9_polygon	19	19	0	0	1		01030000A0C00B0...	5
21	21	gml_e87f14c2-7435-485f-8812-644f60896148	[null]	21	0	0	0			6
22	22	gml_e87f14c2-7435-485f-8812-644f60896148_polygon	21	21	0	0	1		01030000A0C00B0...	6
23	23	gml_2f6284ea-37d5-4fed-ba3d-37aaad24984d	[null]	23	0	0	0			7
24	24	gml_2f6284ea-37d5-4fed-ba3d-37aaad24984d_polygon	23	23	0	0	1		01030000A0C00B0...	7
25	25	gml_708662c3-d3e6-443c-99ec-23f35f70ee7b	[null]	25	0	0	0			8
26	26	gml_708662c3-d3e6-443c-99ec-23f35f70ee7b_polygon	25	25	0	0	1		01030000A0C00B0...	8
27	27	gml_2338934f-9e0c-408b-8b24-36b10efbc5e1	[null]	27	0	0	0			9
28	28	gml_2338934f-9e0c-408b-8b24-36b10efbc5e1_polygon	27	27	0	0	1		01030000A0C00B0...	9
29	29	gml_84d1eb02-36f8-4111-88c8-a47f06635b1a	[null]	29	0	0	0			10
30	30	gml_84d1eb02-36f8-4111-88c8-a47f06635b1a_polygon	29	29	0	0	1		01030000A0C00B0...	10
31	31	gml_6ac5582d-15c4-409a-bf10-757e4b0918f6	[null]	31	0	0	0			11
32	32	gml_6ac5582d-15c4-409a-bf10-757e4b0918f6_polygon	31	31	0	0	1		01030000A0C00B0...	11

Figure 6.6: Resulting Undervisningshuset LOD 2 import to surface_geometry table.

In Figure 6.6, row 1 stored the solid object of Undervisningshuset. Row 2 was a root object and with the *is_composite* attribute described that row 3-12 were surfaces that when combined described the solid. Row 13-32 stored the root surfaces and the geometries of the surfaces in pairs. After one root object was stored the next row contained the physical object with geometric values. The pairs can be identified using *cityobject_id* on the far-right side of Figure 6.6.

For each surface stored in *surface_geometry* additional information was also stored in the *thematic_surface* table (Figure 6.7). *Thematic_surface* stored *objectclass_id* for every surface to determine which feature type they belonged to.

	id [PK] integer	objectclass_id integer	building_id integer	lod2_multi_surface_id integer
1	2	35	1	13
2	3	35	1	15
3	4	34	1	17
4	5	34	1	19
5	6	34	1	21
6	7	34	1	23
7	8	34	1	25
8	9	34	1	27
9	10	33	1	29
10	11	33	1	31

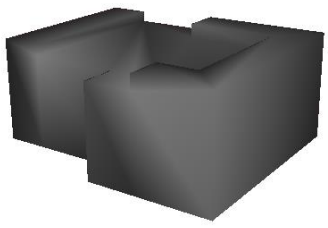
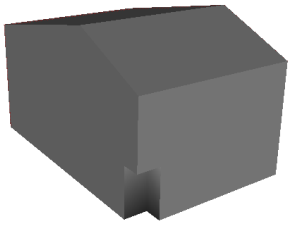
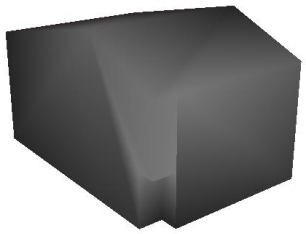
Figure 6.7: Feature types of Undervisningshuset imported to thematic_surface table.

Like *cityobject* the following variables were used: *Building*-26, *GroundSurface*-35, *WallSurface*-34, *RoofSurface*-33. *Lod2_multi_surface_id* referred to the previously mentioned root surfaces, and *building_id* referred to the objects reference in the *building* table.

6.3. Step C and D: Export to CityGML and CityJSON and Geometry validation

Two buildings were exported to CityGML: Multihuset CityGML LOD 1.3 and Undervisningshuset LOD 2.0. One building was exported to CityJSON: Undervisningshuset LOD 2.0. Kanaan's Café LOD 2.0 was also attempted to be exported, but because of the issues discussed in section 6.1 this was not possible because it was not properly transformed. Therefore, the only other viable option for LOD 2 was Undervisningshuset. For LOD 1 Multihuset was chosen as it was one of the most complex LOD 1 data set when considering the geometries. All three exported models were successfully validated with Val3dity. The source code for both models can be found in Appendix A1 and A2 (Undervisningshuset CityGML LOD 2) and B (Undervisningshuset CityJSON LOD 2).

Table 6.3: Exported CityGML and CityJSON models from 3DCityDB.

Export: Multihuset CityGML LOD 1.3	Export: Undervisningshuset CityGML LOD 2.0	Export: Undervisningshuset CityJSON LOD 2.0
		

An attempt was also made to export Kanaan's Café CityGML LOD 2, but because of the non-planar surfaces (explained in section 6.1) the export failed. Kanaan's Café already contained these errors when importing. Non-planar surfaces were supposed to be detected and repaired in step A when transforming NS building to CityGML, but the geometric validators in the script did not catch this error. Therefore, Kanaan's Café contained these errors during import, and the same problem persisted in the export process: the non-planar surfaces could not be repaired as they were not identified by the script. This led to the export process failing when attempting to

convert the geometries extracted from 3DCityDB to a face geometry. A face geometry is a planar area in 3D used to describe 3D surfaces[16].

6.5. Step E: Applications

The CityGML models Multihuset LOD 1.3 and Undervisningshuset LOD 2.0 were imported to FZKViewer and styled automatically by the software. Multihuset (Figure 6.8) was only styled with one colour (teal) as it was only represented by a building solid, and no surfaces. Undervisningshuset (Figure 6.9) were given different colours to differentiate the thematic surfaces. Red represented *RoofSurface*, light grey *WallSurface*, and dark grey *GroundSurface*.

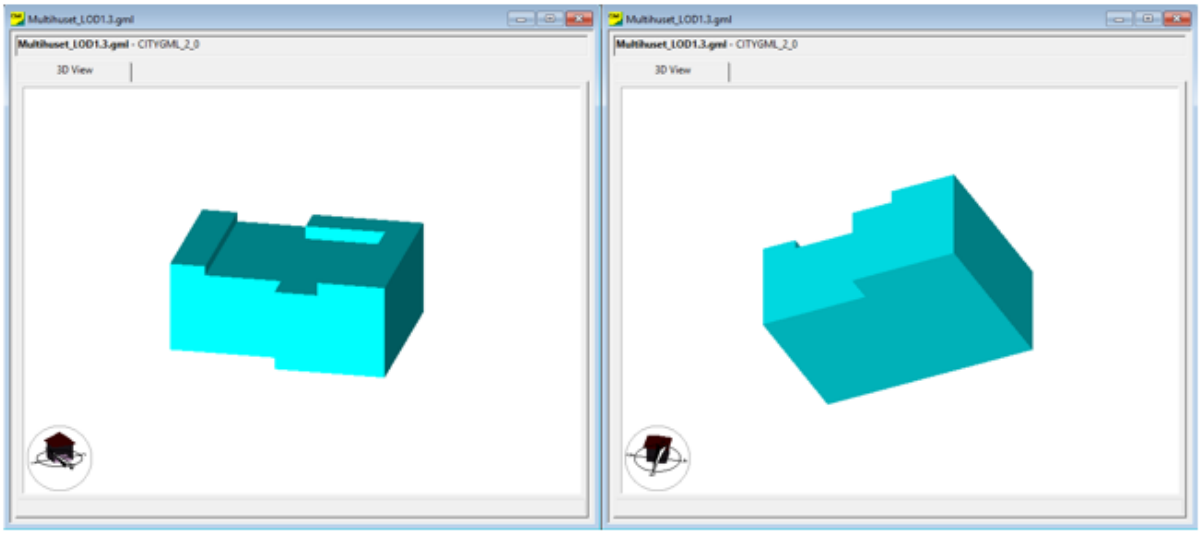


Figure 6.8: CityGML model Multihuset LOD 1.3 in FZKViewer. Styled with one colour as it is only represented by building solid.

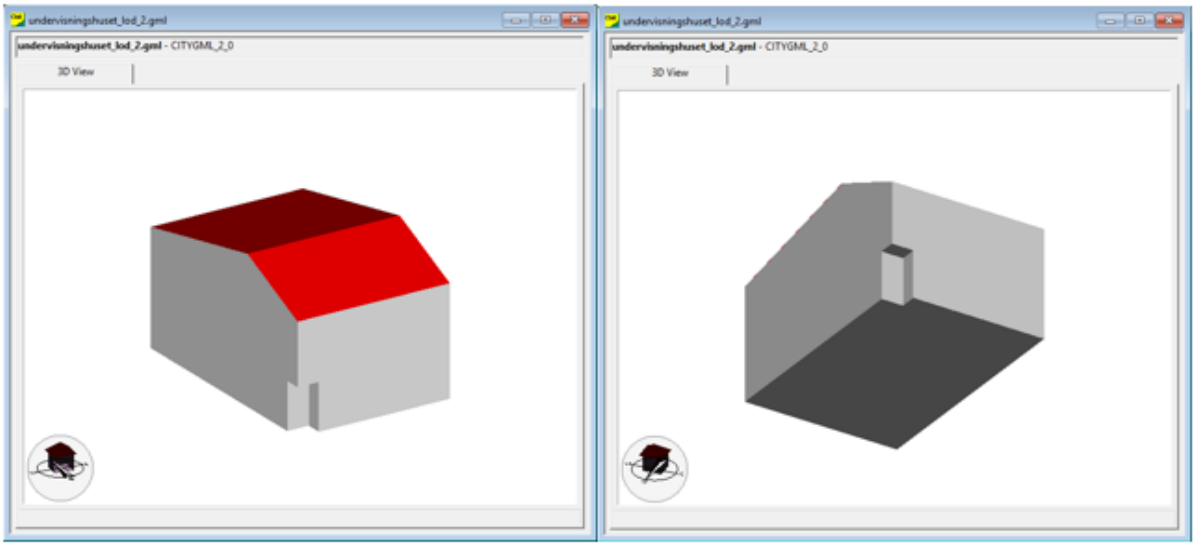


Figure 6.9: CityGML model Undervisningshuset LOD 2.0 in FZKViewer. Surfaces has been styled accordingly based on colour and thematic data.

The CityJSON model Undervisningshuset LOD 2 was loaded in the QGIS plugin *CityJSON loader*. The loader also had the ability to recognize and load semantic surfaces and style them

[16]http://docs.safe.com/fme/html/FME_Desktop_Documentation/FME_Transformers/Transformers/facereplacer.htm

automatically. Figure 6.10 shows Undervisningshuset CityJSON LOD 2 opened in the loader and surfaces styled according to their thematic information. Red was recognized as *RoofSurface*, grey/white as *WallSurface* and *GroundSurface* black.

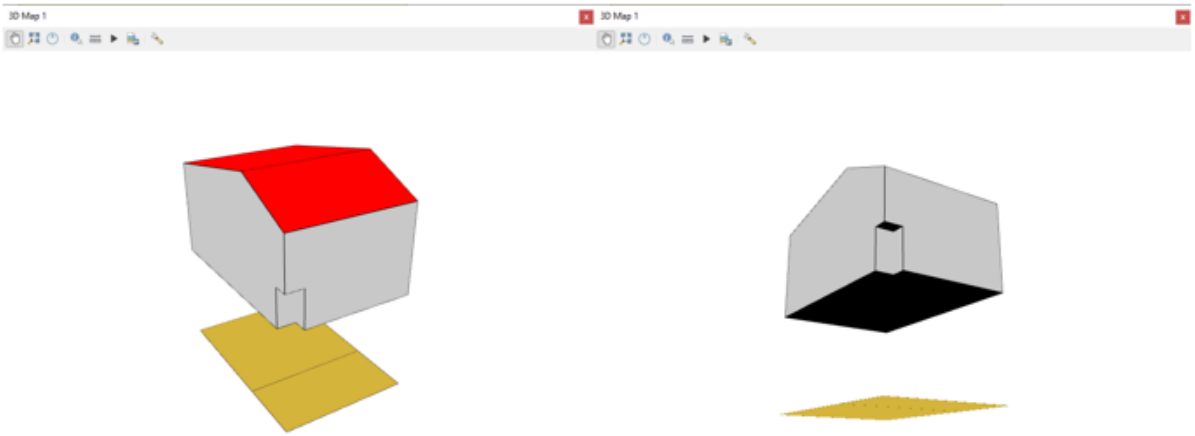


Figure 6.10: CityJSON model Undervisningshuset LOD 2.0 in QGIS CityJSON loader. Surfaces has been styled accordingly based on colour and thematic data.

7. Discussion

7.1. Transformation, import and export

The transformation of Kanaan's Café from NS LOD 2.0 to CityGML LOD 2 created a surface that was non-planar. This surface was discovered using Val3dity, where a tolerance of 0.01 is allowed, but the distance of the surface was 0.0101105. Any attempt to repair the surface in FME failed and Kanaan's Café was therefore imported to 3DCityDB with a non-planar surface. This issue reappeared when attempting to export the building. 3DCityDB stores geometries as hexagon-values and they had to be transformed during export. When attempting to transform these geometries several surfaces failed as FME also requires them to be non-planar. Because of this, Kanaan's Café was not successfully exported. Val3dity only found one non-planar surface, while FME discovered several.

The roof overhangs of Kanaan's Café were not properly imported in accordance to the CityGML standard when considering xlinks. All surfaces (*RoofSurface, WallSurface and GroundSurface*) that are part of the building solid shall be marked with an xlink tag in the database. This helps the database system identify which surfaces define the solid building, and which surfaces does not. Roof overhangs is part of a *RoofSurface*, but they do not help define the building solid, and therefore should not have any xlink tag in the database. When importing the *RoofSurface* in FME there were no way to differentiate roof overhangs from the rest of the roof, and both groups were imported with xlink tag for each surface. This means that the database environment does not have the ability to distinguish these groups after import, and a user must visualize the building to accomplish this. If the import script could be altered to split the roof into overhangs and rest of the roof this could be solved easily. CityGML can add user-defined attributes and one such attribute could be created for roof overhangs and then checked during import. This way the roof could be split into different workflows and correct xlink tags applied.

Another issue with the import process is that it only imports one building at a time. This is fine if the user only wishes to update a 3D city model with one building, e.g. a newly modelled building is added to an already existing city model. However, in most cases it might be better to import several buildings, but the import process does not support this. The import script should therefore be upgraded to handle data sets containing more than one building. The core components of the import script can still be used as it does a good job in splitting and structuring a CityGML building for import, but it must be upgraded so that after one building has been imported, the script restarts and repeats the process for the next building in a data set.

In the export process only two tables were accessed (*surface_geometry* and *cityobject*) even though there were two more tables which contained data (*building* and *thematic_surface*). The main reason for this is the simplicity of the imported buildings and that the two tables *surface_geometry* and *cityobject* were sufficient when exporting. A CityGML building may contain several *building parts* and will have a hierarchical relationship with the building it is part of. This relationship is described in the *building* table using a root ID. None of the buildings transformed contained any *building parts* and it was therefore not necessary to extract any data from the *building table,* although the building itself was required to be represented in the table. As for *thematic_surface* it was more suitable to access *cityobject* to retrieve thematic information as it had direct access to *gml_id* to identify the surfaces. If more advanced CityGML buildings, e.g., with *building parts*, are to be exported, it will be required to modify the script so that the additional information also can be accessed.

Size difference of the exported models

The exported formats, CityGML and CityJSON, had different size when comparing the data. Undervisningshuset was exported to both CityGML and CityJSON LOD 2.0. The building was 11kB in the CityGML format, and 3kB in the CityJSON format. This is a compression factor of 3.66 (CityGML/CityJSON). Ledoux et al. (2019) did a similar test by transforming CityGML files to CityJSON and comparing them. The study achieved a mean compression factor of around six, with results varying from 4.4 to 8.1. It should be noted that the CityGML files were trimmed down in size by removing unneeded spaces and tabs that serve no purpose for the software reading the files. No such trimming was done on the CityGML file in this master thesis. Ledoux et al. (2019) achieved a larger compression factor even though their CityGML files were trimmed. A possible explanation might be that the CityGML files created in this master thesis only contained basic attributes needed to comply to the standard. If more attributes, such as user defined attributes, were created the CityGML files might increase in size as such attributes in CityGML can be expansive (Ledoux et al. 2019). This might increase the compression factor. Another reason might be because the 3D buildings created in this master thesis only represent one building per file. If geometries are shared in CityJSON the vertices are combined. No such merging is done to the CityJSON file as it only contains one building. Larger 3D city models might contain many buildings that have shared geometries and therefore the compression factor might increase as a CityJSON file will be lower in size.

CityGML has a hierarchical data structure, while CityJSON is flattened out (see appendix A and B), and this is reflected in the data size. This will also have an effect if the 3D buildings were stored in a CityJSON structure instead of CityGML in the database. CityGML stores thematic surfaces (*RoofSurface, WallSurface and GroundSurface*) individually in the database in addition to the building solid, while CityJSON stores this information in the building solid

itself. Less tables, and space, will therefore be necessary if the 3D buildings were stored as CityJSON instead of CityGML.

7.2. Reference validation of 3D building models

Section 3.1 promoted two types of validation methods: geometric validation of a 3D building model and validation of how the 3D building model is generated. This thesis has only validated the geometries of 3D building models, and not focused on their accuracy compared to their real-life counterparts. Transformation of NS LOD data to CityGML and CityJSON has not been tested before and it is a challenge to obtain reference data. However, this can still be achieved by manually creating 3D building models from highly detailed 2D building footprints and photogrammetry or laser scanning. These models could then be used to compare the generated CityGML and CityJSON models. In return this would heighten the quality assessment of the created 3D building models.

7.3. Vision model and 3D database environment

This thesis attempted to create a 3D database environment in a municipality context. As mentioned in Section 2.5 the municipalities will be responsible for collecting and updating building data in accordance to NS building. If a municipality uses 3DCityDB for storing 3D city models it seems illogical to convert newly gathered building data to NS LOD and then transform it to CityGML for further storage in 3DCityDB (as is done in this master thesis). It would be more reasonable to transform the new building data to CityGML and import it to the database environment. From there, the municipality can export to NS building data and send the updated building to Lantmäteriet. Alternatively, the municipality can make the NS building data directly available from their own database solution. Either way, the municipality will be responsible to publish updated NS building data. Therefore, it seems more sensible to make NS building data available as an export process, and not import as this master thesis has done.

The vision workflow (Figure 1.2) introduced in section 1.1 described an environment for importing, storing, exporting, and using 3D building data. The environment is a result of several FME-scripts created to properly handle the information flow when importing and storing CityGML to 3DCityDB and exporting to CityGML or CityJSON. Undervisningshuset LOD 2.0 was exported to both CityGML and CityJSON and tested in two applications: FZKViewer for CityGML and the CityJSON loader in QGIS. Multihuset CityGML LOD 1.3 was also exported and tested in FZKViewer. The applications were able to properly identify semantic surfaces for both LOD 2 models and coloured them accordingly. The models went through several altering processes and validating the models after each such process were vital for the integrity of the vision model. The models were validated using Val3dity on two occasions: after transformation from NS building to CityGML and after being exported to CityGML and/or CityJSON. The results for the three models were 100% valid on both tests. These results, both from validation and testing in applications, indicate that the vision workflow can be a helpful tool for users who wish to start or continue using 3D building data. By offering two different export options users can choose which format that best suits their needs, and which software they can access. CityJSON can be used as an alternative for smaller municipalities, or other users, as both QGIS and the CityJSON launcher in QGIS are free software. It should be noted that the models have only been tested for visualization purposes, and no consideration has been taken to the analytical aspects of 3D building models. Also, more tests should be done on exporting and validating from the vision workflow as only three models were exported, while 13 models were transformed and imported. All models, except for Kanaan's Café LOD 2, were 100% valid

before being imported to 3DCityDB, but problems may still arise when going through the altering processes of import and export. The 3D buildings tested in this master thesis were basic when considering the geometries. Other users may have more complex 3D buildings which may increase the risk of improper transformations during import and export. This risk may be greater if the buildings have detailed roof structure as Kanaan's Café had.

The vision workflow uses NS LOD as a starting point for importing and updating the 3D database environment. This is fine for testing the transformation of NS LOD to CityGML, which was an important aim of this thesis, but might not be an ideal situation for all real-life applications. Another solution would be to connect the database to something that is updated regularly like a municipality base map or BIM models. The test data NS LOD was derived from BIM models so integrating this might be doable. However, the test data only contained geometric properties from BIM and no additional information describing the data which is the advantage of BIM. This means much work would have to be done to make this happen. A base map which is updated regularly by municipalities with building footprint data could be connected to the database environment. The database already uses footprints to transform some models to CityGML and using data from base maps would be preferable as the data is kept up to date. Alternatively, the database environment could be skipped completely, and a new workflow would transform directly from NS LOD, base maps or BIM to CityGML/CityJSON. This may be a viable solution for very small 3D city models, but as a model increases with more buildings this might lead to challenges. A database is very advantageous when considering data access. Large sized data can be stored in a structured manner and easily searched. If the database and 3D building models are integrated with additional information such as solar, wind and sound exposure, finding areas of a specific value would be possible. For instance, a city planner could search the database to find the best possible areas of a city to install solar panels. If one would store a large 3D city model without a database finding specific data can be a challenge.

3DCityDB has built-in functions which allows users to import and export CityGML datasets. It does not have any limitations considering the amount of buildings contained in a CityGML file, and even has the functionality to export to other relevant formats such as KML, COLLADA and glTF which can be used in Google Earth, ArcGIS etc. 3DCityDB does not have functionality which allows updating or versioning. Some commercial solutions exist for this purpose, but they can be expensive. Buildings can undergo changes in real life and keeping them up to date in a database environment is important for correct representation. Versioning a building can store changes over time and can be useful to show the history and development of a city. By integrating the 3D database environment created in this master thesis users can develop such solutions by using the available scripts as a foundation. The scripts can be expanded to handle different workflows that either updates or applies versioning details.

3CIM project

One goal of the 3CIM project is to integrate a 3D city model with an information model which allows it to perform analyses and simulations. The 3D city model will be tested with a rainfall analysis which are useful when analysing the effects of climate change. An aim of this master thesis was to aid the project so it could import CityGML data to 3DCityDB without using the built-in import function 3DCityDB has. This would allow 3CIM to control the information flow during the import. In addition, the processes used for importing the data can be further developed for future needs. This aim has been partially achieved as the import process has some

limitations that previously have been discussed. Weaknesses of the import process are issues regarding roof overhangs, limited attributes import options, only importing one building at a time and only being able to import buildings (not building parts or accessories).

8. Conclusions

The first aim of this master thesis was to evaluate if NS LOD data could be transformed to CityGML. The transformation process created in this master thesis confirms that NS LOD data (in accordance to the Swedish specification NS building) can be transformed to CityGML. However, consideration should be taken to detect and repair errors that occur during transformation. Non-planar surfaces, double points and small polygons have been a struggle during this thesis, as these sorts of errors appear individually and fixing them seems to be a case dependant issue. The findings also suggest that these issues should be resolved for the Swedish standard NS building to properly be used for visualization and analytical purposes.

The second and third aim attempted to create open-source processes that could handle the import of CityGML data to 3DCityDB and the export of CityGML and CityJSON data. The 3D database environment created in this master thesis can successfully import and export said 3D formats. The environment can be a useful tool for users wishing to get started with 3D building data, or for existing users wanting to enhance their solutions. If the database environment is to be adapted, it must undergo serious polishing to ensure that complex 3D buildings that goes beyond this master thesis can be properly imported. It can be concluded that as long as the 3D buildings are stored correctly in the 3D database environment, exporting them is an easier matter as it involves accessing tables already written to, and the structure is therefore already known.

The final aim was to assess the applicability of the vision model workflow. The vision model might not be structured in an optimal way as it uses NS LOD data as a starting point. The Swedish specification NS building is not fully developed, but NS LOD data will be stored in the national geodata platform and from there exported to CityGML. This might not be interoperable with the vision model as it begins with transforming NS LOD data. On the other hand, the models that went through the entire vision model workflow (transformed, imported and exported) were validated with success throughout the processes and finally visualized in open-source software. This implies that the workflow does a good job handling and structuring the information. More tests could have been done to verify this as only three models went through the entire workflow, while 13 models were transformed and imported. It can be argued that the relevance of the vision model is strong as it allows export to CityJSON which has a potential of being supported in more software applications (especially on the web), but to access the processes created, a user will still need to use FME which is a proprietary software solution.

References

Ates Aydar, S., J. Stoter, H. Ledoux, E. Demir Ozbek, and T. Yomralioglu. 2016. ESTABLISHING A NATIONAL 3D GEO-DATA MODEL FOR BUILDING DATA COMPLIANT TO CITYGML: CASE OF TURKEY. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences,* 41-B2**:** 79-86. DOI: 10.5194/isprs-archives-XLI-B2-79-2016

Autodesk. 2015. Planar and non-planar polygons. Retrieved 25.05.2021, from https://knowledge.autodesk.com/support/maya-lt/learn-explore/caas/CloudHelp/cloudhelp/2015/ENU/MayaLT/files/Polygons-overview-Planar-and-nonplanar-polygons-htm.html.

Awrangjeb, M., M. Ravanbakhsh, and C. S. Fraser. 2010. Automatic detection of residential buildings using LIDAR data and multispectral imagery. *ISPRS Journal of Photogrammetry and Remote Sensing,* 65**:** 457-467. DOI: https://doi.org/10.1016/j.isprsjprs.2010.06.001

Azhar, S. 2011. Building Information Modeling (BIM): Trends, Benefits, Risks, and Challenges for the AEC Industry. *Leadership and Management in Engineering* 11**:** 241-252. DOI: https://doi.org/10.1061/(ASCE)LM.1943-5630.0000127

Barnes, M., and E. L. Finch. 2008. COLLADA – Digital Asset Schema Release 1.5.0. Specification.

Bhatia, S., P. Cozzi, A. Knyazev, and T. Parisi. 2017. glTF Specification 2.0. .

Biljecki, F., K. Kumar, and C. Nagel. 2018. CityGML Application Domain Extension (ADE): overview of developments. *Open Geospatial Data, Software and Standards,* 3. DOI: https://doi.org/10.1186/s40965-018-0055-6

Biljecki, F., H. Ledoux, and J. Stoter. 2016. An improved LOD specification for 3D building models. *Computers, Environment and Urban Systems,* 59**:** 25-37. DOI: 10.1016/j.compenvurbsys.2016.04.005

Biljecki, F., J. Stoter, H. Ledoux, S. Zlatanova, and A. Çöltekin. 2015. Applications of 3D city models: state of the art review. *SPRS International Journal of Geo-Information,* **:** (4):2842-2889. DOI: https://doi.org/10.3390/ijgi4042842

Borkowski, A., M. Karpina, and P. Tymkow. 2016. 3D GIS FOR FLOOD MODELLING IN RIVER VALLEYS. DOI: 10.5194/isprs-archives-XLI-B8-175-201

Brink, L. v. d., J. Stoter, and S. Zlatanova. 2013. Establishing a national standard for 3D topographic data compliant to CityGML. *International Journal of Geographical Information Science***:** 27:21, 92-113. DOI: 10.1080/13658816.2012.667105

Burggraf, D. 2015. OGC KML 2.3, Version 1.0.

de Laat, R., and L. van Berlo. 2011. Integration of BIM and GIS: The Development of the CityGML GeoBIM Extension. *Advances in 3D Geo-Information Sciences***:** 211-225.

Desheng, L., and F. Xia. 2010. Assessing object-based classification: advantages and limitations. *Remote Sensing Letters,* 1**:** 187-194.

Eriksson, H., T. Johansson, P.-O. Olsson, M. Andersson, J. Engvall, I. Hast, and L. Harrie. 2020. Requirements, Development, and Evaluation of A National Building Standard—A Swedish Case Study *ISPRS International Journal of Geo-Information***:** 9(2):78 DOI: https://doi.org/10.3390/ijgi9020078

Floros, G., I. Pispidikis, and E. Dimopoulou. 2017. INVESTIGATING INTEGRATION CAPABILITIES BETWEEN IFC AND CITYGML LOD3 FOR 3D CITY MODELLING. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences,* 42-4/W7.

Gilbert, T., C. Rönsdorf, J. Plume, S. Simmons, N. Nisbet, H.-C. Gruler, T. H. Kolbe, L. v. Berlo, et al. 2020. Built environment data standards and their integration: an analysis of IFC, CityGML and LandInfra.

Gruber, U., J. Riecken, and M. Seifert. 2014. Germany on the Way to 3D-Cadastre. DOI: 10.12902/zfv-0028-2014
Gröger, G., T. H. Kolbe, C. Nagel, and K.-H. Häfele. 2012. OGC City Geography Markup Language (CityGML) Encoding Standard, Version 2.0.0. No. 12-019.
Gröger, G., and L. Plümer. 2012. CityGML – Interoperable semantic 3D city models. *ISPRS Journal of Photogrammetry and Remote Sensing* 71:12–33. DOI: 10.1016/j.isprsjprs.2012.04.004
Häufel, G., D. Bulatov, M. Liebelt, and P. Solbrig. 2015. Simulation of urban terrain models using VBS2, TerraTools and FZK Viewer. *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)***:** 4610-4613. DOI: 10.1109/IGARSS.2015.7326855
Julin, A., K. Jaalama, J.-P. Virtanen, M. Pouke, J. Ylipulli, M. Vaaja, J. Hyyppä, and H. Hyyppä. 2018. Characterizing 3D City Modeling Projects: Towards a Harmonized Interoperable System *International Journal of Geo-Information*, 7.
Kolbe, T. H., H. Nguyen Son, K. Chaturvedi, B. Willenborg, A. Donaubauer, C. Nagel, Z. Yao, H. Schulz, et al. 2019. 3D City Database for CityGML Version 4.2.
Konde, A., and S. Saran. 2017. Web enabled spatio-temporal semantic analysis of traffic noise using CityGML.
Kutzner, T., K. Chaturvedi, and T. H. Kolbe. 2020. CityGML 3.0: New Functions Open Up New Applications. *PFG 88***:** 43–61 (2020). DOI: https://doi.org/10.1007/s41064-020-00095-z
Laakso, M., and A. O. Kiviniemi. 2012. The IFC standard: A review of History, development, and standardization, Information Technology. *The IFC standard: A review of History, development, and standardization, Information Technology*, 17**:** 134-161.
Lantmäteriet. 2018. Mätanvisningar Geometrisk representation vid utbyte Version 3.2

Lantmäteriet. 2021a. Byggnadsgeometrier-NS_mätningsanvisningar-3CIMap4.
Lantmäteriet. 2021b. NATIONELL INFORMATIONSSPECIFIKATION Byggnad Version 1.0 Test 1.
Ledoux, H. 2018. Val3dity: validation of 3D GIS primitives according to the international standards. *Open geospatial data, softw. stand.***:** 3, 1 (2018). DOI: https://doi.org/10.1186/s40965-018-0043-x
Ledoux, H., K. A. Ohori, K. Kumar, B. Dukai, A. Labetski, and S. Vitalis. 2019. CityJSON: A compact and easy-to-use encoding of the CityGML data model. *Open Geospatial Data, Software and Standards, In Press.* DOI: http://dx.doi.org/10.1186/s40965-019-0064-0
Liu, X., X. Wang, G. Wright, J. C. P. Cheng, X. Li, and R. Liu. 2017. A State-of-the-Art Review on the Integration of Building Information Modeling (BIM) and Geographic Information System (GIS) *ISPRS Int. J. Geo-Inf.*, 6(2), 53. DOI: https://doi.org/10.3390/ijgi6020053
Löwner, M. O., and G. Gröger. 2016. Evaluation Critera for Recent LoD Proposals for CityGML Buildings. *Photogrammetrie - Fernerkundung - Geoinformation 2016***:** 31-43. DOI: 10.1127/pfg/2016/0282
Machl, T. 2013. Minutes of the International OGC, SIG 3D and TUM Workshop on Requirements for CityGML 3.0. 1-28.
Miettinen, R., and S. Paavola. 2014. Beyond the BIM utopia: Approaches to the development and implementation of building information modeling. *Automation in Construction*, 43**:** 84-91.

Noardo, F., K. Ohori, Arroyo., F. Biljecki, C. Ellul, L. Harrie, T. Krijnen, H. Eriksson, J. v. Liempt, et al. 2020. Reference study of CityGML software support: The GeoBIM benchmark 2019—Part II. DOI: https://doi.org/10.1111/tgis.12710

Nys, G.-A., F. Poux, and R. Billen. 2020. CityJSON Building Generation from Airborne LiDAR 3D Point Clouds *SPRS Int. J. Geo-Inf.*, 9(9), 521.

Santana, J. M., J. Wendel, A. Trujillo, J. P. Suárez, A. Simons, and A. Koch. 2017. Multimodal Location Based Services—Semantic 3D City Data as Virtual and Augmented Reality. DOI: 10.1007/978-3-319-47289-8_17

SIG3D. 2017. Modeling Guide for 3D Objects Part 2: Modeling of Buildings (LoD1, LoD2 and LoD3) Retrieved 11.05.2021, from https://en.wiki.quality.sig3d.org/index.php?title=Modeling_Guide_for_3D_Objects_-_Part_2:_Modeling_of_Buildings_(LoD1,_LoD2,_LoD3).

Soon, K. H., and V. H. Khoo. 2017. CITYGML MODELLING FOR SINGAPORE 3D NATIONAL MAPPING. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42-4/W7.

Stoter, J., J. Beetz, H. Ledoux, M. Reuvers, R. Klooster, P. Janssen, F. Penninga, S. Zlatanova, et al. 2013. Implementation of a National 3D Standard: Case of the Netherlands. *Progress and New Trends in 3D Geoinformation Sciences***:** 277-298.

Stoter, J., G. Vosselman, J. Goos, S. Zlatanova, E. Verbree, R. Klooster, and M. Reuvers. 2011. Towards a National 3D Spatial Data Infrastructure: Case of The Netherlands. *Photogrammetrie - Fernerkundung - Geoinformation 2011*, 6**:** 405-420. DOI: 10.1127/1432-8364/2011/0094

Sun, J., S. Mi, P.-O. Olsson, J. Paulsson, and L. Harrie. 2019. Utilizing BIM and GIS for Representation and Visualization of 3D Cadastre *SPRS Int. J. Geo-Inf.***:** 8(11), 503. DOI: https://doi.org/10.3390/ijgi8110503

Sun, J., P.-O. Olsson, H. Eriksson, and L. Harrie. 2020. Evaluating the geometric aspects of integrating BIM data into city models. *Journal of Spatial Science***:** 65:62, 235-255. DOI: 10.1080/14498596.2019.1636722

Wagner, A., N. Alam, M. Wewetzer, M. Pries, and V. Coors. 2015. METHODS FOR GEOMETRIC DATA VALIDATION OF 3D CITY MODELS

van Berlo, L., T. Dijkmans, and J. Stoter. 2013. EXPERIMENT FOR INTEGRATING DUTCH 3D SPATIAL PLANNING AND BIM FOR CHECKING BUILDING PERMITS. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2-2/W1.

Wate, P., and S. Saran. 2015. Implementation of CityGML energy application domain extension (ADE) for integration of urban solar potential indicators using object-oriented modelling approach. *Geocarto International*, 30:10**:** 1144-1162. DOI: 10.1080/10106049.2015.1034192

Wentz, E., and Q. Zhao. 2015. Assessing Validation Methods for Building Identification and Extraction. In *JURSE 2015*. Lausanne, Switzerland.

Vitalis, S., K. A. Ohori, and J. Stoter. 2020. CityJSON in QGIS: Development of an open-source plugin. *Transactions in GIS*, 24**:** 1147-1164. DOI: https://doi.org/10.1111/tgis.12657

Volk, R., J. Stengel, and F. Schultmann. 2014. Building Information Modeling (BIM) for existing buildings — Literature review and future needs. *Automation in Construction*, 38**:** 109-127. DOI: https://doi.org/10.1016/j.autcon.2013.10.023

Yao, Z., C. Nagel, F. Kunde, G. Hudra, P. Willkomm, A. Donaubauer, T. Adolphi, and T. H. Kolbe. 2018. 3DCityDB - a 3D geodatabase solution for the management, analysis, and visualization of semantic 3D city models based on CityGML. *Open Geospatial Data, Software and Standards***:** 3, Article number: 5.

Appendix

Appendix A contains the code for the exported CityGML data set Undervisningshuset LOD 2.

A: Undervisningshuset CityGML LOD 2 - 1/2

A: Undervisningshuset CityGML LOD 2 – 2/2

Appendix B contains the code for the exported CityJSON data set Undervisningshuset LOD 2.

B: Undervisningshuset CityJSON LOD 2


```
{
    "CityObjects": {
        "fid_d54735ff-0803-41d2-905c-2972490b6cf7": {
            "attributes": {},
            "geometry": [
                {
                    "boundaries": [
                        [[0,1,2,3,4,5,6,7]],
                        [[8,9,10,11]],
                        [[12,13,1,0,7,9,8,14]],
                        [[15,16,4,3,2,17]],
                        [[18,14,8,11,5,4,16]],
                        [[10,6,5,11]],
                        [[9,7,6,10]],
                        [[2,1,13,17]],
                        [[17,13,12,15]],
                        [[15,12,14,18,16]]
                    ],
                    "lod": 2.0,
                    "semantics": {
                        "surfaces": [
                            {
                                "type": "GroundSurface"
                            },
                            {
                                "type": "WallSurface"
                            },
                            {
                                "type": "RoofSurface"
                            }
                        ],
                        "values": [0,0,1,1,1,1,1,1,2,2]
                    },
                    "type": "MultiSurface"
                }
            ],
            "type": "Building"
        }
    },
    "metadata": {
        "geographicalExtent": [
            153976.781236116,6581761.660888468,19.94,
            154008.896156191, 6581803.872662367,43.218
        ]
    },
    "type": "CityJSON",
    "version": "1.0.1",
    "vertices": [
        [153981.511105176,6581774.394545195,19.94],
        [153976.781236116,6581799.095776715,19.94],
        [154001.728006718,6581803.872662367,19.94],
        [154006.457875778,6581779.171430846,19.94],
        [154008.896156191,6581766.43777412,19.94],
        [153987.455683662,6581762.332285388,19.94],
        [153986.952606137,6581764.959553552,19.94],
        [153983.446308064,6581764.288156632,19.94],
        [153983.949385589,6581761.660888468,27.54],
        [153983.446308064,6581764.288156632,27.54],
        [153986.952606137,6581764.959553552,27.54],
        [153987.455683662,6581762.332285388,27.54],
        [153981.511105176,6581774.394545195,43.218],
        [153976.781236116,6581799.095776715,38.375586249],
        [153983.949385589,6581761.660888468,37.518836668],
        [154006.457875778,6581779.171430846,43.218],
        [154008.896156191, 6581766.43777412,37.518836668],
        [154001.728006718,6581803.872662367,38.375586249],
        [153987.455683662,6581762.332285388,37.518836668]
    ]
}
```