Updating and Versioning of 3D City Models

Konstantinos Mastorakis
student #4844238

1st supervisor: Stelios Vitalis
2nd supervisor: Hugo Ledoux
On-site supervisor: Maarten Vermeij

January 5, 2020

Contents

Acronyms

3DCM 3D City Model. ii, 1–7, 9–14, 17

API Application Program Interface. 12

CLI Command Line Interface. 11

CVS Concurrent Versions System. 4, 10

GML Geography Markup Language. 2, 7, 8

GUI Graphical User Interface. 3, 4, 11

JSON JavaScript Object Notation. 3, 8

NGO Non-Governmental Organization. 7

OGC Open Geospatial Consortium. 2, 5, 7

OWM Oracle Workspace Management. ii, 10

RCS Revision Control System. 4

SCCS Source Code Control System. 4

UNIX Uniplexed Information and Computing Service. 4

VCS Versioning Control System. ii, 4, 8–11

1 Introduction

1.1 3DCMs in a nutshell

3DCMs are digital models of urban areas that represent terrain, surfaces, sites, buildings, vegetation, infrastructure and landscape elements in three-dimensional scale including related objects (city furniture) belonging to urban areas. (Wikipedia, 2019a)

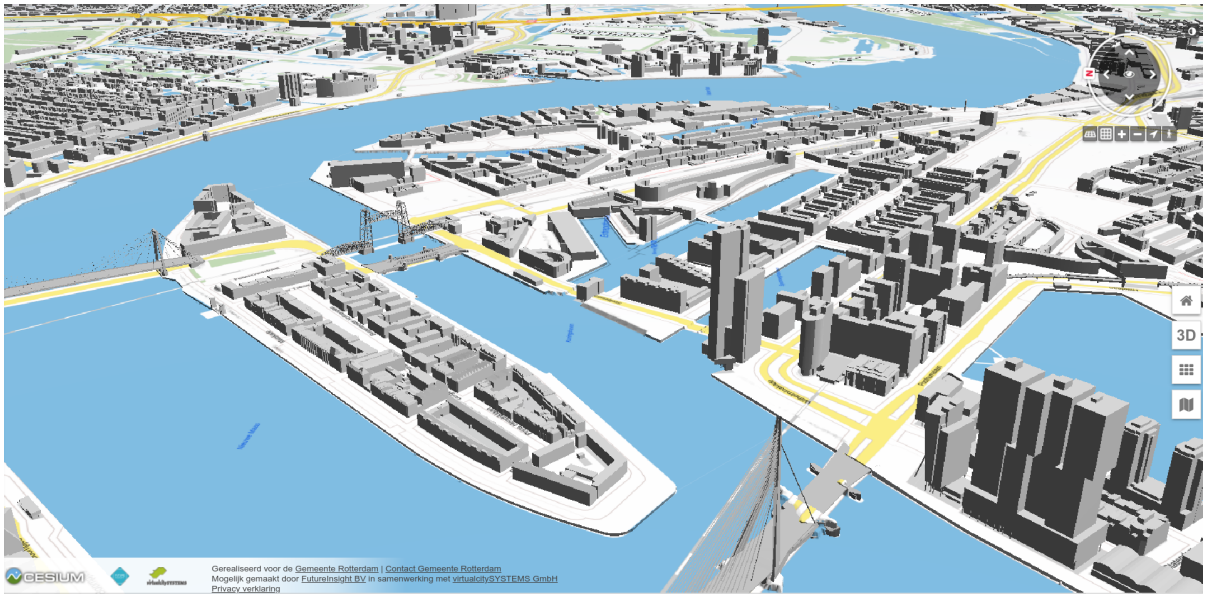


Figure 1: A snapshot of the publicly available 3DCM of the city of Rotterdam.
Source: `https://3drotterdam.nl/#/`

There are many factors, that make 3DCMs so popular for academia, industry and individual researchers. Four of the most important originate from the nature of these models:

i. They can contain and organize enormous amount of geo-information, representing potentially every entity that resides in urban environments such as buildings, road network, vegetation, underground utilities etc.

ii. They are schematically extendable, which means their internal structure can be augmented and customized to store additional entities, with completely different characteristics and attributes.

iii. They are stored mainly as open formats that are both human and machine readable, which allows more people to understand their structure and developers to create different software implementations for these models, for any purpose.

iv. They can associate geometries with attributes and semantic information.

There is no limitation in the nature or the amount of the (semantic) information that 3DCMs can incorporate and that is why there are utilized in a vast number of applications, from various domains and for completely different end goals. Namely, Biljecki et al. (2015) demonstrate -at least- 29 use cases of 3DCMs that comprise a subset of at least 100 applications. By storing semantic information the computer can 'understand' surfaces and objects based on that information. This allows further extended analysis apart from geometrical. For example, knowing that a surface is "roof" and not "wall" allows the solar capacity analysis of the model. The

computer can identify the potential surfaces for solar panels to be placed, which would be impossible lacking the semantic information, and carry out the analysis on them. The same idea applies for any other kind of analysis that is attempted on the model and is possible only by semantic information.

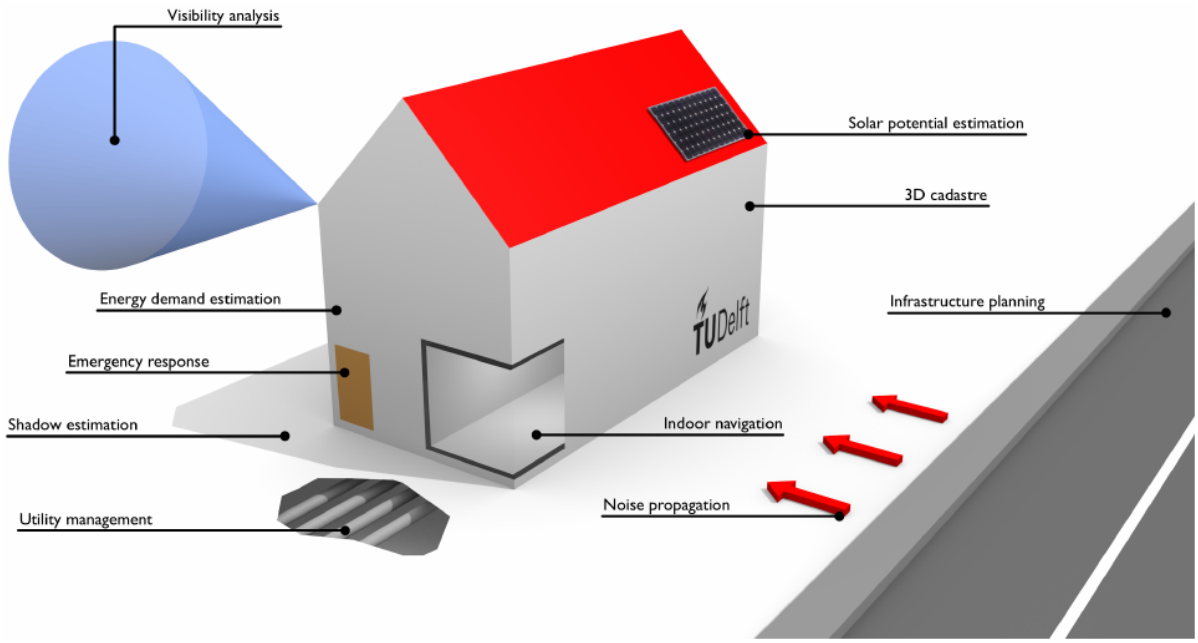


Figure 2: The applications of 3DCM in various domains.
Source: Biljecki et al. (2015)

1.2 Problem statement

Cities change amazingly fast. Having potentially all available urban information contained in a 3DCM -at a specific time instance-, that will ideally be utilized by a very wide range of users, creates an urgent need for an efficient managing solution. In other words, having obsolete data can render the 3DCM partially or completely useless. Changes and modifications should be done at a building level in order for model to be constantly up to day on a daily basis. There is currently no such solution that allows building-level or lower visual editing, updating and keeping track of the updated versions of a 3DCM. That can be partly attributed to the lack of a unique representation schema due to the heterogeneity of 3DCMs contents (Wikipedia, 2019a); which also allows the introduction and propagation of -very common- errors (Biljecki et al., 2016). Although an encoding standard of the schema has been already implemented by the Open Geospatial Consortium (OGC) (Gröger et al., 2012), in practice it is really difficult to implement any updating solution based on it. That is due to the the conceptual design of this standard, which allows more than one ways to store the same information and the verbosity of Geography Markup Language (GML), which is officially adopted by OGC as the standard's implementation language.

The lack of such a solution, makes 3DCMs owners, administrators, managers etc unable to have a day to day updated model that also keeps track of its past states. This leads to long periods of data obsolescence and massive updates every few years, not always clean of errors. This translates first and foremost into reduced functionality of the model itself, the need to outsource the updating task to third parties and a high financial cost. This is the case with the

municipality of Rotterdam as well, whose model gets updated once every few years, not necessarily at equal time intervals and always depending on their budget. Such a solution would address more than one problems at the same time. It would tackle all the issues mentioned, while it would give the capability of in-house data management, correcting errors that exist in the 3DCM at any given time and diminishing error propagation down the line. It would revolutionize the way practitioners think about 3DCM management and attract many more users with the guarantee of up to date, integer and consistent 3DCM data. Finally, having a tracking system of all the modifications in every updated version of a 3DCM means that more than one people can work on the model concurrently without messing with each others work, which is a big need in the spatial data industry at the moment.

2 Motivation

The end goal of this thesis is the proof of concept that a framework which allows manual object-level updating and versioning of 3DCMs through an intuitive Graphical User Interface (GUI) is feasible. It will be built on a alternative encoding of the *CityGML* data model based on the JavaScript Object Notation (JSON) format (**?**), *CityJSON* (Ledoux et al., 2019). Specifications of the *CityJSON* and why it was chosen will be thoroughly explained in section 5.1.3. The use case for prototyping this framework is the 3DCM of the municipality of Rotterdam. In section 4 the research scope of the project will be thoroughly analyzed under the frame of the use case.

The mere fact that multidisciplinary practitioners use 3DCMs, implies that there is different familiarization level regarding the 3DCM structure and its constraints, which lies underneath the -very easy to understand- 3D representation of it. Simply put, not everybody understands how a 3DCM is created, encoded and stored; and as a matter of fact they do not necessarily need to. Thus, updating and maintaining them in an intuitive way, that strongly focuses on the facilitation of small scale changes i.e. changing the attributes of a building or (part of) its geometry would be really useful. This will not only bring more value to 3DCM-related workflows, but it will establish the basis for a 'data at the source' approach. The same dataset will be continuously updated, via mechanisms that ensure data integrity, while allowing previous-state-browsing and branching capabilities. It will then be stored in one remote repository that could be accessed and cloned locally at will, by all interested parties. Pretty similar to what *Git* allows developers to do with their source code files.

Another important benefit of endorsing small scale changes in 3DCMs, is the potentiality of in-house maintaining. It is a fact that the volume of information 3DCMs may contain is massive. In addition, tools that enable small scale editing, as described in the previous paragraph are lacking. Eventually, the creation or the update of the models is in most cases carried out outside the organization that will be the owner of the model. That is because of the extremely high labor-intensity it requires to massively update such an amount of information and the lack of resources, leading to outsourcing. It is technically impossible to outsource minor object-level day-to-day updates of the model in real time. Regarding the municipality of Rotterdam, currently, this outsourcing cycle takes few to many years due to the cost of the venture itself, leaving the model in an obsolete state for long time spans. It becomes then clear that the impact of in-house small scale maintenance would free 3DCMs owners and maintainers from outsourcing, with all the benefits that it comes with. It would probably redefine the workflow of updating and maintaining 3DCMs, enabling instant updates rather than big stockpiled ones that cover large time periods.

Apart from the specific technical challenges, which are a strong motivation alone, there is also broader motivation. It originates from the obstacles imposed by proprietary software with undisclosed file formats, when the same data needs to be used in different industry domains by a versatile range of multidisciplinary practitioners. This results in the user getting trapped in limited software solutions -regulated by others-, while the exchange of information is severely crippled, resulting in lack of communication between neighboring industries (i.e. urban planners and architects); something conceptually similar to what data silos are for linked data.

Keeping that in mind all the existing software used and all software developed under the scope of this thesis will be free and open source software that one can use and modify at will. Specifically, for the visualization and editing of the 3DCMs, an add-on was created for *Blender* (Blender Foundation, 2019), a really powerful open source 3D modeling software with an intuitive GUI. One of the advantages of using Blender is that once a 3DCM is imported in it, it can be exported to any of the available export formats *Blender* offers. What is more, if *Blender* were to be extended to export in a new -previously unsupported- format, then it would automatically mean that the 3DCM could be exported to this format as well. The extensibility of *Blender* is of utmost importance in the effort to create an integrative platform for 3DCMs maintenance. Finally, it is and will be free for everybody to use and extend, adding lots of extra functionality -that could potentially be useful regarding 3DCMs- at zero cost.

3 3DCMs and VCSs

Versioning control systems is not something new. Early implementations of VCS go over four decades back with Source Code Control System (SCCS) (Glasser, 1978) being one of the very first, developed in 1972 by Uniplexed Information and Computing Service (UNIX) developers. It was later followed from the Revision Control System (RCS) and Concurrent Versions System (CVS) (Wikipedia, 2019b). The major problem all VCSs are designed to solve, is to keep track of the changes made in computers programs source code files (i.e. text files). They are essential for efficient and effective improvement of computer software especially when the source code is collaboratively developed.

There are plenty of different implementations of VCSs through the last forty years, both proprietary and open source, with some of them prevailing over others (Wikipedia, 2019c). In the software development domain the versioning problem can be considered solved, since the improvement of VCSs for almost half a century led to extremely robust and free VCS implementations, which correspond to the actual needs of today's programmers. One of the most powerful characteristics of most modern VCSs is branching. In simple words branching is the ability to create a copy of the actual code at any given time instance and work on that instance without affecting the original code. Changes to the original code can be made independently on a copy of it, which later -if needed- can be merged back to the main (original) branch. Arguably, the world's most dominant VCS today is *Git*[1]. This can be attributed to its fundamental differentiation from the traditional past approaches, i.e. the way it treats updated files (new versions) and the fact that is a distributed VCS (Chacon and Straub, 2019). A more in depth explanation on the mechanics of *Git* will be given in section 5.

Open encodings of 3DCMs files resemble to source code files as shown in figure 4, in the sense that they are both standardized syntax-wise. The syntax of both conforms to a predefined

[1] `https://git-scm.com/`

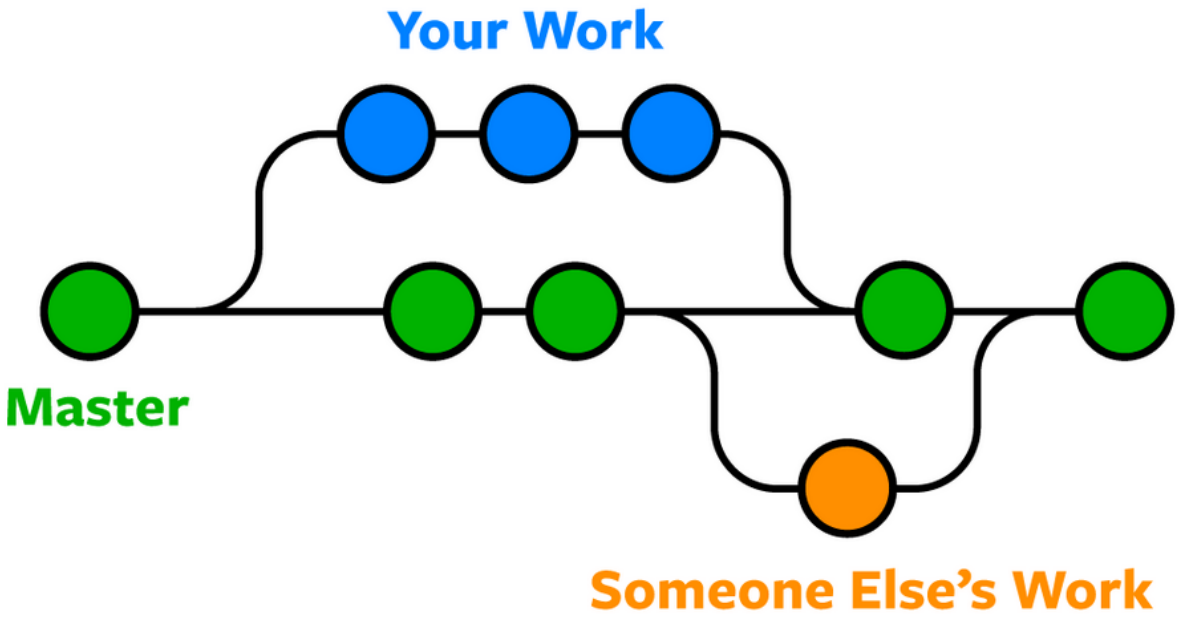


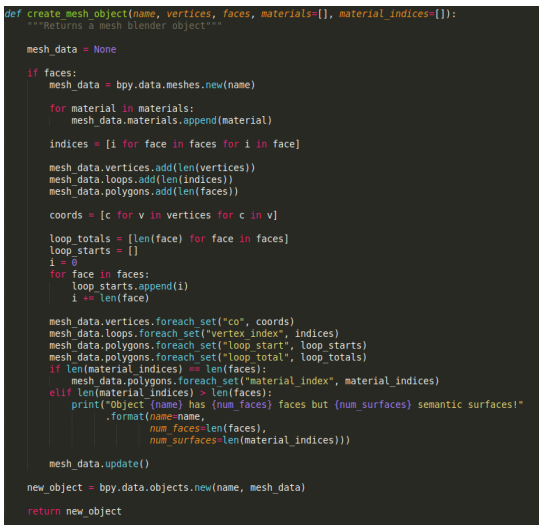
Figure 3: The concept of branching. A branch must have at least 1 commit (orange branch) or more (blue branch) stemming from different point (time instance) of the master branch and merged back to it on any point, without affecting it at all.
Source: `https://www.nobledesktop.com/learn/git/what-is-git`

schema, which is essentially a strictly predefined pattern to store information or write scripts respectively. Although it is comparatively harder than with source code, changes in the information contained in 3DCM files can also relatively easily be tracked, as long as the the file encoding is kept simple enough to parse and there is not redundancy of information or ways of storing that information. Being able to track all changes and store them alongside their metadata -such as who did the change, timestamp, elements changed etc- means that apart from having always an up to date model, the time dimension is integrated in the model and shuffling through different (time) instances of the 3DCM becomes possible. What is more, at any given time instance the branching principle mentioned in section 3 can be applied allowing for practitioners to test ideas and possible scenarios without interrupting the workflow of other maintainers.

There has been already a proposal about version management of 3DCMs, for datasets encoded under the *CityGML* data model (Chaturvedi et al., 2016), which will be incorporated into *CityGML*'s third version[2]. It is clear that the official authoring and maintaining institution of the whole *CityGML* initiative, OGC, attempts to tackle this challenge. Admittedly, from a technical point of view, it looks quite difficult if not impossible for this conceptual model that supports versioning, to be implemented in a functional way in practice. Although *CityGML* is quite useful as a data model, in the sense of standardizing the schema that 3DCM data should be stored; when implemented into a data exchange format, it becomes inefficient. According to Ledoux et al. (2019), most efforts have been focused on developing concepts, but not on how to implement them. More insight on *CityGML* and its versioning schema extension will be given in section 5.1.1 and 5.1.2 respectively.

[2] `https://github.com/opengeospatial/CityGML-3.0CM`

(a) A 3DCM encoded in *CityJSON*
Source: Municipality of Rotterdam (originally in CityGML)



(b) A python (version 3) script

Figure 4: A side to side comparison between a *CityJSON*-encoded 3DCM and a source code python script

Source: Author

4 Research scope

The research scope of this thesis is to investigate how a framework should be implemented, to enable graphical user interaction with the 3DCM -viewing and modifying- while keeping track of all the modifications via a versioning control system designed specifically for this purpose. As mentioned in section 1 the use case is the municipality of Rotterdam. The framework will be focused on their publicly available 3DCM, but it will be designed to be functional more or less with any other 3DCM -encoded with the *CityJSON* format- with little to none modification of the framework mechanisms themselves.

The core of the project can be divided in two major components. First, the technical aspect. This includes the creation of the software that implements the framework in practice. By the time of writing this thesis, there is no implemented solution to my knowledge, that manages 3DCMs. This implies that there many challenges to be faced and decisions to be made, which will have to be tested at a primitive level. So all software that will be developed, should be seen as prototype software and under no circumstances as a complete once-off solution to the 3DCMs updating and versioning problem. Ideally, this prototype software, should be further developed and maintained in the future by me, TU Delft, or any third parties, as the source code will be freely available online under an open source license.

The second component is the managerial aspect, that could be summarized as follows: Receiving explicit feedback from multidisciplinary practitioners inside the municipality of Rotterdam, who interact with the 3DCM, in order to understand and prioritize their needs, so the framework can be adapted around them, rendering it is as appealing and intuitive as possible for the user the interact with. In parallel with the technical aspect, practitioners will be interviewed, so as vital information on the current management procedures and shortcomings can be gathered. This information will then be evaluated and taken under consideration during the development of the framework's software so major features can be incorporated directly

to it, via an iterative process.

As mentioned in section 2, the global goal of this thesis is the proof of concept of the potentiality to create a highly functional integrative framework based on an alternative encoding of the *CityGML* data model, for updating and versioning 3DCMs at object level, using open source software. On a lower level, given the time frame of this thesis which is around 8 months more specific milestones can be defined:

1. Create a Blender extension (add-on) to import *CityJSON* files.
2. Add functionality to the add-on to export any scene of Blender in *CityJSON* encoding.
3. Elaborate on the integration of a versioning control system prototype (Vitalis et al., 2019) to the add-on.
4. Incorporate features received from practitioners' feedback into the extension.
5. Test framework and report results.

The extent to which functionalities that practitioners will suggest, will be introduced, as well as the testing of the framework itself, is highly dependent on the time frame of the project. It is related to the challenges that will have to be tackled during any of the above milestones.

5 Related work

There is a lot of activity in the 3DCM domain last years with respect to standardization and software developed for interaction with them. This section gives an overview of what has been done so far on a conceptual and practical level. Section 5.1 is about standardization of storing 3DCMs. Section 5.2 focus specifically on software. Some of the software focuses specifically on the *CityJSON* encoding. Some other implementations are about geo-versioning control systems.

5.1 Data models and encodings

In this section all the fundamental related work on which this thesis will be based on, is presented. The term fundamental is used to describe all that preliminary work that without it, it would be impossible for this project to exist. Those are the data model that was created to schematically dictate how 3DCM related information should be stored and a couple of implemented encodings that translate this data model into files.

5.1.1 CityGML

CityGML[3] can be a confusing term. That is because it either stands for the open data model definition of the schema in which 3DCMs information is stored, or the *CityGML* file exchange format that implements the schema utilizing a GML-based *.gml* format. It is designed, maintained and supported by the OGC and is the very popular among companies, Non-Governmental Organizations (NGOs), public agencies etc. It is already an international standard, which is reassuring for interested stakeholders should they adopt it or not.

[3] `https://www.opengeospatial.org/standards/citygml`

Arguably the biggest shortcoming of GML based formats is their verbosity. The same opening and closing tags must be used repetitively throughout the whole file length, increasing dramatically the size of it without adding extra information. Also there are more than one ways -redundancy- to store the same information, meaning that software implemented to parse it will not be robust and will often crash. In addition, there are no native GML parsers for programming languages so one needs to create their own ad-hok parser; which it not trivial. Finally, it has a rather hierarchical (nested) structure, which adds to the overall complexity and reduces its web compatibility.

5.1.2 CityGML versioning extension

As already introduced in section 3, there is a documented suggested approach from Chaturvedi et al. (2016) for the *CityGML* data model to be extended in order to support versioning. In spite of the end purpose of this extension, which is to lay the conceptual basis for (software) implementations of VCSs, I strongly believe that it will not be the case. The major reason in my opinion is the fact that it does not allow versioning down to the object level (buildings, road segments etc). This is of crucial importance as it goes in pair with the concept of small scale changes, which inevitably are object-level changes. Even the authors of this proposed extension recognize this drawback, mentioning that in the future this feature might be required. Implementing it will be impossible unless the GML specification is changed (Chaturvedi et al., 2016).

5.1.3 CityJSON

CityJSON[4] is an alternative encoding -of a subset- of the *CityGML* data model. It is developed and maintained by the *3D geoinformation group* at Delft's University of Technology. In comparison with the latter, it is based on the JSON format, keeping software developers in mind. It was designed as an attempt to address the shortcomings of the *CityGML* encoding. It is superior to the CityGML encoding in the way it stores data. Its structure is more flat, it stores no redundant information and the vertex storing approach simplifies a lot the topological reconstruction. The absence of opening and closing tags for each entity makes it six times more compact on average than *CityGML*. Last but not least, it can be parsed natively by every programming language that supports JSON files parsing, which is extremely convenient when one wants to implement software based on this file format. There is already free software that allows bi-directional conversion between *CityJSON* and *CityGML* encoding, so from an external, non-programming point of view, both files can be considered equivalent.

5.2 Software implementations

Except from the work presented above, there are also several software implementations related with the manipulation of geoinformation which are interesting under the research scope of this thesis. Some are related because they are developed around the *CityJSON* encoding, others because they implement a geo-VCS. Except those mentioned in section 5.2.5, all the others are open source non-proprietary ones.

[4] `https://www.cityjson.org/`

5.2.1 Git

Git[5] is the only software that is not related with geoinformation at all. It is a VCS created to maintain source code files and for many developers probably the most robust and reliable. It is also free and open source. It is a distributed VCS, which in simple words means that you don't need to be connected to a server to work and the whole database that stores the different versions is saved also locally.It was initially developed by *Linus Torvalds* in 2005 to maintain the *Linux* kernel. The fundamental difference of *Git* related to other VCSs is the way it keeps track of the different versions. While most VCSs just store the differences -often referred as 'deltas'- from a version to another, *Git* stores complete snapshots throughout the project's lifecycle, creating a tree structure in which nodes are references to these snapshots. For a better understanding of *Git's* internal architecture the Chacon and Straub (2019) manual will come really handy. *Git's* robust principles and simplistic design has tempted some geo-software developers to create a *Git*-like VCS wrapped around geodata files. Software presented in 5.2.2, 5.2.3, 5.2.4 and 5.2.5 are such examples.

5.2.2 GeoGig

GeoGig[6] is a versioning open source tool that implements the *Git* principles to manage the versioning of geospatial vector data. It is a distributed VCS which can be accessed as a datastore in *Geoserver*[7] (Franceschi et al., 2019). It currently supports *Shapefiles*, *PostGIS* and *SpatiaLite* data, which are imported into a *Git*-like repository where all changes are tracked. Internally, when a dataset is imported, *GeoGig* converts it to its own binary format that can handle and keep track. Then, the dataset is staged for commit and after it is committed, a new version of the dataset is created and stored to *GeoGig's* repository, which is a database. As with *Git* all the objects that comprise a version are stored, not only the deltas. In case an object is the same in two consecutive versions the new version will contain a reference to the previous version's object. In other words, the original dataset is stored only once. What is stored in the local repository might as well be also stored in a remote repository. *GeoGig* is conceptually almost identical to what this thesis is about. The difference is, geospatial vector files are not similar with 3DCM files and every successful VCSs should wrap around the file structure it attempts to keep track of, as firmly as possible. That is the same reason why *Git* as an implementation does not solve the problem for geospatial data out of the box, since it is designed to track changes in source code files.

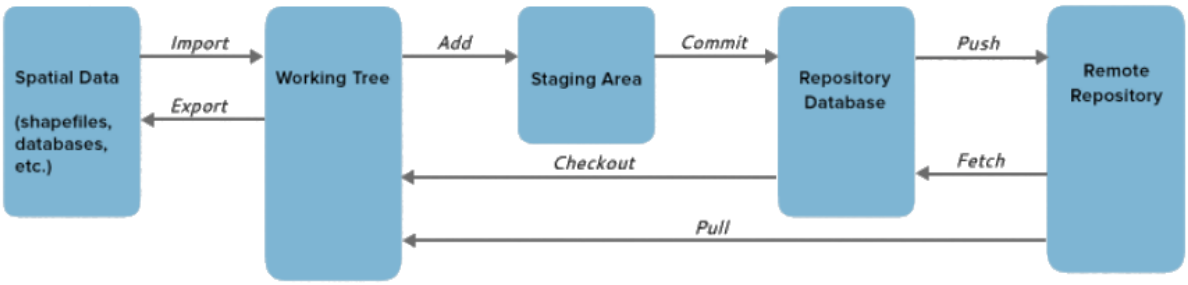


Figure 5: *GeoGig's* workflow
Source: `http://geogig.org/docs/start/introduction.html`

[5] urlhttps://git-scm.com/
[6] `http://geogig.org/`
[7] `http://geoserver.org/`

5.2.3 QGIS versioning plugin

The QGIS versioning plugin[8] developed by *Oslandia*[9] is another approach to achieve geospatial data history management. It is also a distributed VCS which uses the *PostGIS*[10] database schema in its core. According to the official website, this project started because none of the previous implementations fulfilled the creators' needs. Those were using a *PostGIS* as the main repository and the ability to work offline, i.e. having a distributed VCS. Although the approach is neat, similar to *GeoGig* it focuses on spatial vector and raster data, so it can not support versioning of 3DCM files.

5.2.4 PostGIS Versioning - pgVersion

pgVersion is another geo-VCS implementation which attempts to tackle the problem of concurrent editing of one *PostGIS* layer by more than one persons. It is a centralized VCS, which means there must always be an active connection to the server similar to how CVS or *Subversion*[11] operate. As in 5.2.2 and 5.2.3 this approach is also wrapped around a different data structure than the one used from 3DCMs file structure. Furthermore, the fact that it is a centralized VCS, probably would make it a non practical VCS for tracking 3DCMs due to their notably bigger file size; which translates to lots of data being exchanged between the server, where the database is stored and every client, which commits the updates.

5.2.5 OWM and ESRI ArcSDE

Both systems are database oriented and they operate in a very similar level conceptually. What changes is the difference in terminology. For example in OWM users work in *workspaces* which can be seen as individual rooms where users work, with all the changes being visible only to these users. The changes of the *workspace* are applied to the parent workspace via a merge transaction which makes the changes visible to the parent *workspace* as well (Oracle, 2013). To be noted that in the case of *Oracle* products, this approach does not apply only to geospatial data but in every kind of data as long as the database schema can support it. Similarly, the equivalent of *workspace* for *ESRI* is called *version*. *ESRI's* solutions are by default spatial-database oriented, although the principles do not differ so much from traditional non-spatial databases. The approach in this case is that every database has a *default* version which can not be deleted and is owned by the administrator. The different versions are stored in the geodatabase and regardless how many of them exist each dataset is only stored once. The technique used here is known as 'delta tables' meaning tables storing all the updates from version to version (Law, 2010). Both systems are commercial and proprietary, meaning that they can not be freely used or further developed by anyone except their owners. Lastly, the data types that these solutions handle are not the similar to 3DCM data encodings. So, even if the source code were publicly available it would not change much with respect to 3DCMs versioning.

[8] https://oslandia.com/en/2013/07/13/qgis-versioning-plugin/
[9] https://oslandia.com/en/
[10] https://postgis.net/
[11] https://subversion.apache.org/

5.2.6 Azul, CityJSON-Web-viewer, CityJSON-QGIS-plugin, cjio, 3dfier

CityJSON viewers

Azul[12] is a fast viewer for *CityJSON* datasets that operates on *macOS*, while there is a cross-platform web viewer[13] for *CityJSON* files. Lastly, a *QGIS* plugin[14] has been also developed for visualizing 3DCMs encoded in *CityJSON* into the *QGIS* environment.

CityJSON editors, manipulators and generators

cjio[15] is an editing and manipulating software, designed specifically for *CityJSON*-encoded files. Through the Command Line Interface (CLI) the user can extensively edit a *CityJSON* file. It also allows chain processing (pipelines) for multiple operations to be applied directly in one go.

3dfier[16] is a generator of *CityJSON* files. It takes as input 2D GIS datasets and it extrudes the 2D objects in 3D with the use of point clouds. It also considers the semantics of each entity, so it is properly lifted -for example water should always be a flat surface- while rooftops not necessarily.

All the above software has been developed and maintained by TU Delft's *3D geoinformation group*. They are all free to use and open source, so they can be further developed, combined with other software or one could use the design concepts this software was created with to further develop 3DCM-related software.

6 Research questions

Having everything mentioned in the previous sections in mind, eventually lays the foundation for the research questions:

i. To what extent is it possible to implement an integrative framework for geometric and semantic updating and versioning of *CityJSON*-encoded 3DCMs?

ii. Which are the small scale day to day editing needs of a 3DCM?

iii. Is the *Git*-like VCS approach appropriate to address the needs of versioning 3DCMs files encoded in *CityJSON*?

iv. How could the current updating process of 3DCMs be improved by such a framework?

7 Methodology, tools and datasets used

To answer the research questions as they are defined in section 6, a specific methodology must be introduced. First and foremost, the feasibility of an integrative framework, as it is mentioned in section 6 for managing 3DCMs, must be examined. For this reason, a GUI tool will be created, that will allow the visualization of a 3DCM and graphical editing of the information it contains, geometrical, descriptive or semantic. For this purpose, *Blender* a free and open

[12] `https://apps.apple.com/nl/app/azul/id1173239678?mt=12`
[13] `https://viewer.cityjson.org/`
[14] `https://github.com/tudelft3d/cityjson-qgis-plugin`
[15] `https://github.com/cityjson/cjio`
[16] `https://github.com/tudelft3d/3dfier`

source 3D creation suite, Blender Foundation (2019) is chosen. From a technical point of view, that is mainly for three reasons. First, it offers a very intuitive Application Program Interface (API) in python, thus it can be easily extended. Second, it is free and open source so anybody can use it free of charge and understand how it works. Third it is very versatile and powerful 3D suite that gets developed and maintained by a very big community.

Once the tool is created that successfully imports a CityJSON-encoded 3DCM in *Blender* with all its attributes and semantic information and allows editing, then the updated 3DCM should be exported back into the *CityJSON* format, in its updated version. So, an exporter should be implemented to convert the 3DCM from the structure *Blender* uses to handle it internally, to *CityJSON* encoding. Doing so establishes a complete workflow of visualizing a 3DCMs, graphically editing it and creating new -updated- versions of it.

With more than one versions of a 3DCM, the versioning aspect takes over. At this stage a mechanism is needed to be in place to manage the different versions efficiently creating all the necessary links and relations among them. In the framework of this project, an already implemented solution for versioning *CityJSON* files as described by Vitalis et al. (2019), will be used[17]. This solution attempts to port *Git's* robust principles into a prototype versioning control system for *CityJSON* files. It will be attempted to adapt this solution, so it can be incorporated into the *Blender*'s suggested 'update' workflow and extend it to a 'complete maintenance' workflow, meaning both updating and versioning 3DCMs

In parallel with the technical aspect of this thesis, i.e. creating or adapting software to establish a graphical workflow for maintaining 3DCMs, great importance will be given in understanding *Rotterdam's Municipality* 3DCM management process and needs. Analyzing through the findings will lead to incorporating features and principles in the proposed framework, that will better suit the demands of *Rotterdam's Municipality* with respect to maintaining its 3DCM. Collecting information will happen through interviews, with active multi-disciplinary practitioners on the geoinformation field from inside the municipality. In an ideal scenario, all the necessary features should be implemented in the framework. In practice, most probably that will not be the case; nevertheless, even if not all the demanded features will be implemented, all the findings will be reported for future reference.

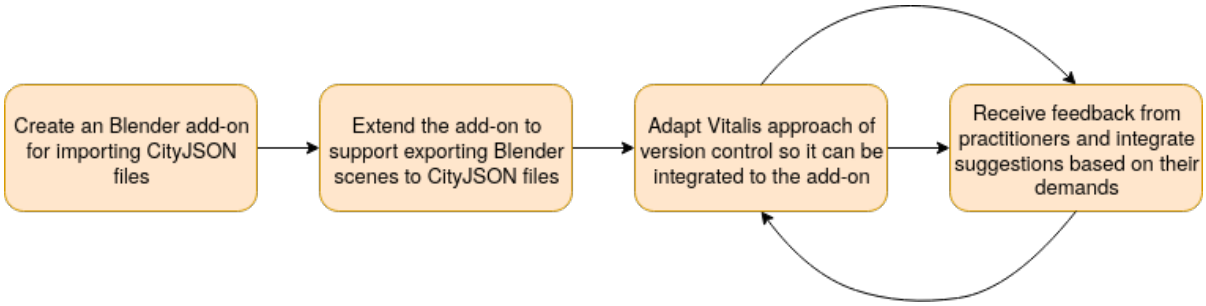


Figure 6: The methodology explained in a graph as conceptual steps and their transition.

[17]More details on this mechanism in section 9.2

8 Time planning

In figure 7 an idea is given on the -approximate- time that will be dedicated on every task.

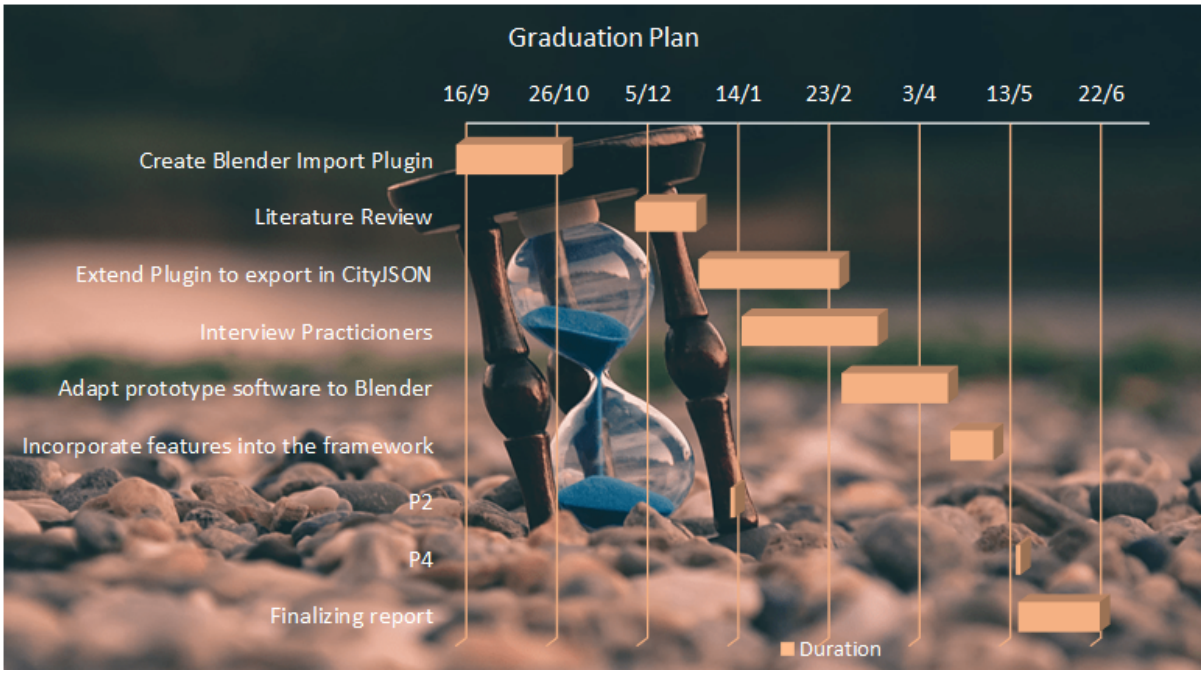


Figure 7: Time allocation of the master thesis tasks.

9 Datasets and tools

9.1 Datasets

For software developing and testing certain datasets are used. Those are the 3DCM of Rotterdam, which is publicly available, free to use and was provided directly by the municipality and free *CityJSON* encoded datasets that can be found on *CityJSON's* official website. The latter ones cover more or less every possible variation a *CityJSON* files can have -with respect to representation of the information- and that is why they were also chosen for testing.

9.2 Tools

The tools that will be used are:

i. The *Blender* 3D suite as mentioned above, that will be extended to handle *CityJSON* files.

ii. The *CityJSON-Blender* plugin[18] that imports *CityJSON* 3DCMs files into Blender and allows to graphically edit the objects' geometry, attributes or semantics using *Blender's* functionality (see figure 8). It has been initially developed by me under the frame of TU Delft's course *'GEO5010 - Research Assignment'* and it is currently further developed by the *3D geoinformation group* of TU Delft and myself under the official *CityJSON*[19] repository.

[18] `https://github.com/cityjson/Blender-CityJSON-Plugin`
[19] `https://github.com/cityjson`

iii. Vitalis et al. (2019) versioning prototype[20] for *CityJSON* files. This software implementation ports *Git's* robust principles to the 3DCM domain. More specifically the *CityJSON*-encoded 3DCMs files are treated by this prototype implementation as source code files are treated by *Git*. Once versioning for a 3DCM file is initiated it augments the structure of the *CityJSON* by adding one extra entry/tag/dictionary key into it, which is called *'versioning'* and contains all versions of the file, branches, tags etc as its children. For a better understanding and a visual comparison between a versioned and a non-versioned *CityJSON* file see figure 9.

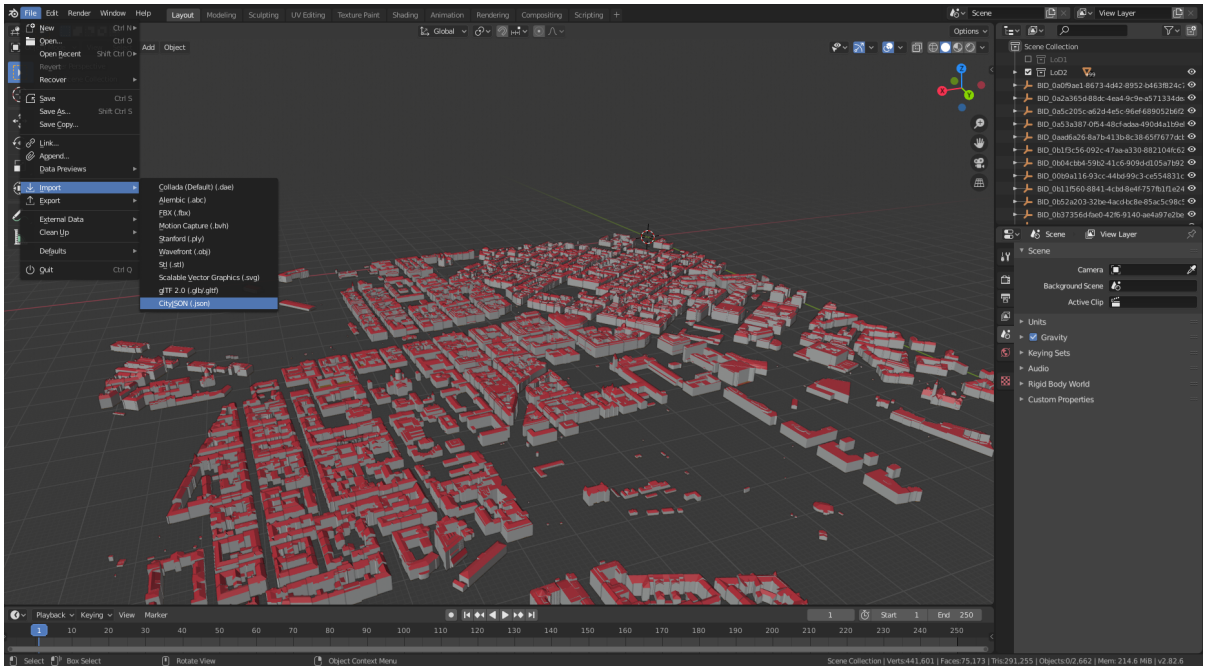


Figure 8: The plugin that imports *CityJSON* files into *Blender*.

Figure 9: The augmented structure of a *vCityJSON* (left) file compared to an ordinary *CityJSON* (right) file. Under the versioning tag all the version-related information are stored.

[20] `https://github.com/tudelft3d/cityjson-versioning-prototype`

References

F. Biljecki, J. Stoter, H. Ledoux, S. Zlatanova, and A. Çöltekin. Applications of 3D City Models: State of the Art Review. *ISPRS International Journal of Geo-Information*, 4(4):2842–2889, dec 2015. doi: 10.3390/ijgi4042842.

F. Biljecki, H. Ledoux, X. Du, J. Stoter, K. H. Soon, and V. H. S. Khoo. The most common geometric and semantic errors in CityGML datasets. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2/W1:13–22, oct 2016. doi: 10.5194/isprs-annals-iv-2-w1-13-2016.

Blender Foundation. Blender 2.82 reference manual, 2019. URL `https://www.blender.org/`.

S. Chacon and B. Straub. *Pro Git*. Apress, 2019. URL `https://git-scm.com/book/en/v2`.

K. Chaturvedi, C. S. Smyth, G. Gesquière, T. Kutzner, and T. H. Kolbe. Managing Versions and History Within Semantic 3D City Models for the Next Generation of CityGML. In *Advances in 3D Geoinformation*, pages 191–206. Springer International Publishing, oct 2016. doi: 10.1007/978-3-319-25691-7_11.

S. Franceschi, K. Adoch, H. K. Kang, C. Hupy, S. Coetzee, and M. A. Brovelli. OSGEO UN Committee Educational Challenge: A use case of sharing software and experience from all over the world. *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-4/W14:49–55, aug 2019. doi: 10.5194/isprs-archives-xlii-4-w14-49-2019.

A. L. Glasser. The evolution of a Source Code Control System. *ACM SIGSOFT Software Engineering Notes*, 3(5):122–125, nov 1978. doi: 10.1145/953579.811111.

G. Gröger, T. H. Kolbe, C. Nagel, and K.-H. Häfele. *OGC City Geography Markup Language (CityGML) Encoding Standard*. Open Geospatial Consortium, 2.0.0 edition, 2012.

D. Law. Versioning 101: Essential information about ArcSDE geodatabases. 2010. URL `https://www.esri.com/news/arcuser/0110/versioning101.html`.

H. Ledoux, K. Arroyo Ohori, K. Kumar, B. Dukai, A. Labetski, and S. Vitalis. CityJSON: a compact and easy-to-use encoding of the CityGML data model. *Open Geospatial Data, Software and Standards*, 4(1):4, June 2019. ISSN 2363-7501. URL `https://doi.org/10.1186/s40965-019-0064-0`.

Oracle. Oracle Database 12c: Workspace Manager. 2013. URL `https://download.oracle.com/otndocs/products/workspace_manager/pdf/workspace_manager_12c_twp.pdf`.

S. Vitalis, A. Labetski, K. Arroyo Ohori, H. Ledoux, and J. Stoter. A data structure to incorporate versioning in 3D city models. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-4/W8:123–130, 2019. doi: 10.5194/isprs-annals-IV-4-W8-123-2019. URL `https://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/IV-4-W8/123/2019/`.

Wikipedia. 3D city models. *Wikipedia*, 2019a. URL `https://en.wikipedia.org/wiki/3D_city_models`.

Wikipedia. Source Code Control System. *Wikipedia*, 2019b. URL `https://en.wikipedia.org/wiki/Source_Code_Control_System`.

Wikipedia. Comparison of version-control software. *Wikipedia,* 2019c. URL https://en.wikipedia.org/wiki/Comparison_of_version-control_software#History_and_adoption.

List of Figures