The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLVIII-4/W4-2022
17th 3D GeoInfo Conference, 19–21 October 2022, Sydney, Australia

APPLYING VERSIONING TO MULTI-LOD 3D CITY MODELS

S. Vitalis[1],* K. Arroyo Ohori[1], J. Stoter[1]

[1] 3D geoinformation group, TU Delft, the Netherlands - (s.vitalis, k.ohori, j.e.stoter)@tudelft.nl

Commission IV, WG IV/9

KEY WORDS: Versioning, CityJSON, CityGML, 3D city models, level of detail (LoD).

ABSTRACT:

Level of Detail (LoD) is a well known concept in 3D city models, used to designate different geometric detail that can be used in different applications. Nevertheless, multi-LoD datasets are hard to maintain and manage because of their intrinsic complexity. Versioning is a solution that aids in the storage and management of big and complex dataset, with its main goal being to facilitate the tracking of changes and collaboration. In this paper, we investigate the effects of utilising versioning and, more specifically, the concept of branches as a way to manage the evolution of multi-LoD datasets. We propose a framework according to which every LoD is stored in its own branch and can be extracted and updated independently. We tested this framework on a tile from 3D BAG, a dataset of 3D buildings for the whole of the Netherlands containing four LoDs (namely, LoD0, LoD1.2, LoD1.3 and LoD2.2). Our results suggest that there are certain benefits from this solution, such as the efficient tracking of changes for individual LoDs and the ability to extract and update the model using one LoD at a time. Nevertheless, there is a lot of complexity added to the process as a set of rules needs to be enforced when managing the model.

1. INTRODUCTION

3D city models are useful in a number of applications, such as solar potential estimation and CFD simulations (Biljecki et al., 2015; García-Sánchez et al., 2021). However, different applications need different geometric detail, which is why the concept of Level of Detail (LoD) has been early adopted in 3D city models. LoDs are defined in the CityGML data model specification (Open Geospatial Concortium, 2021) and their refined proposition by Biljecki et al. (2016a) has been incorporated in CityJSON (Ledoux et al., 2019). Despite the existing mechanism being available for years, there is still a scarcity of multi-LoD datasets. This has mostly attributed to the difficulties that arise from managing multiple LoDs in one file, which has lead researchers into investigating alternative ways of storing and utilizing multiscale datasets (Arroyo Ohori et al., 2015).

In Vitalis et al. (2019) we have proposed a data structure to incorporate versioning in 3D city models stored in CityJSON, mostly focusing on retaining the evolution of a dataset. One of the concepts introduced in this solution is branches, which allows for different lines of evolution to be handled independently, while still allowing some interaction between them (e.g. merging). Branches have been proposed to be used in applications such as storing different scenario outcomes for simulation. However, the utilization of branches for handling individual lineages of LoDs has not been investigated before.

In this research, we propose a methodology to store and manipulate multi-LoD datasets in a versioned CityJSON file. Our intention is to evaluate the feasibility and usefulness of storing different LoDs as different branches, so that lineage of semantics and attributes is separated from that of geometries; and individual LoDs hold their own line of evolution for their geometry. We use 3D BAG (Peters et al., 2022) as a use case; 3D BAG is a dataset which contains 3D models of all buildings in the Netherlands in three LoDs: LoD1.2, LoD1.3 and LoD2.2.

* Corresponding author

2. RELATED WORK

2.1 3D City models

3D city models have been standardized since the introduction of CityGML, which is now in its third iteration (Kolbe et al., 2021). CityGML defines an object-oriented data model to represent features of a city. It, also, describes a GML-based data format for storing and exchanging such data. Ledoux et al. (2019) introduced CityJSON as a JSON encoding of the CityGML data model, in order to overcome some of the shortcomings of the GML nature of the CityGML data format (e.g. its verbosity and complexity).

2.1.1 Level of Detail (LoD) Level of Detail (LoD) is a concept which stems from 3D graphics, used to efficiently render 3D models of high detail (Luebke et al., 2002). CityGML has in-corporated LoDs as a way to represent scale in 3D city models. The standard defines five LoDs based on their overall details: LoD0, LoD1, LoD2, LoD3 and LoD4. Biljecki et al. (2016a) argued that the originally proposed LoDs can be further sub-divided based on the horizontal and vertical scale and proposed a more advanced LoD scheme (Figure 1). In addition, Löwner et al. (2016) proposed a more flexible mechanism of LoDs, that allows authors of 3D city models to define their own LoDs ac-cording to their needs.

LoDs in 3D city model are particularly important due to the complexity of 3D city models and the wide range of applications that can be related to them. While intuitively geometries of higher detail might be considered always superior to lower detailed ones, there is evidence that they do not always contribute to more accurate results (Biljecki et al., 2017). In addition, geometries of higher LoDs might present more validation issues (Biljecki et al., 2016b).

Multi-LoD datasets are relatively scarce and are considered hard to maintain. According to Biljecki et al. (2014), this is related to the lack of consistency, due to the use of different acquisition, modelling and storage techniques.

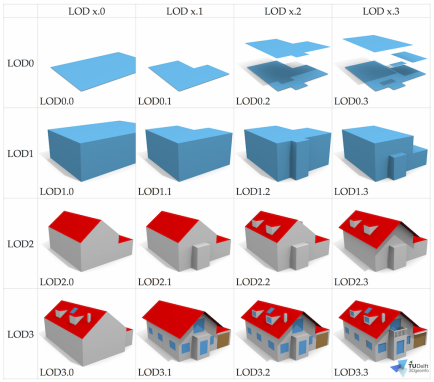

Figure 1. The proposed concept of dual-level LoD as proposed by Biljecki (2017) (Figure from Biljecki (2017)).

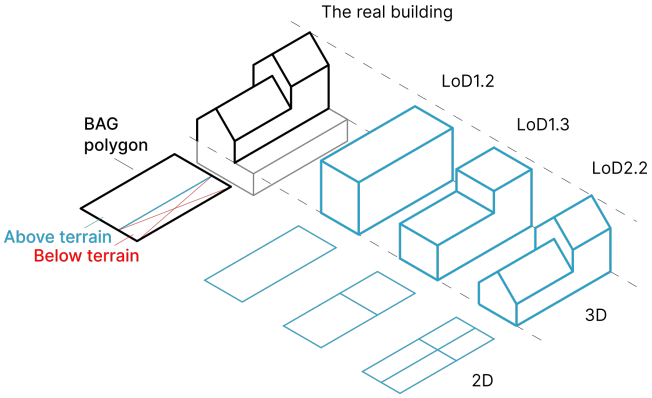

Figure 2. The three LoDs that are automatically reconstructed in 3D BAG (figure from `http://3dbag.nl`).

2.2 3D BAG

3D BAG is a dataset of all buildings in the Netherlands in 3D (Peters et al., 2022). The buildings are created based on two open datasets of the country: the footprints of buildings from the "Basisregistraties Adressen en Gebouwen" (BAG), or the Building and Address register of the Netherlands; and LiDAR point data from the "Actueel Hoogtebestand Nederland" (AHN), the national height model of the Netherlands. The reconstruction produces three different LoDs per building (Figure 2): LoD1.2 (i.e. prismatic volumes of the footprint), LoD1.3 (i.e. prismatic volumes based on a subdivision of the footprints to better much the building's vertical profile) and LoD2.2 (i.e. volumes with actual roof shapes).

The dataset is available through different data formats, one of which is a set of CityJSON files; every file constitutes a tile according to the subdivision used by the reconstruction method. Every CityJSON file contains all information available per building as attributes; four geometries are available per building: the footprint, as LoD0, and the 3D geometries of the three LoDs mentioned before. It is important to clarify that one footprint of a building does not always correspond to one geometry. This is because, as depicted in Figure 2, a footprint in BAG might contain areas with no elevation (e.g. a yard) or constructions below the ground level, which cannot be reconstructed by the methodology used in 3D BAG. Therefore, in CityJSON a building city object contains only the attributes for the whole building and its footprint geometry as LoD0. Then, the three LoDs produced by the reconstruction are represented as multi-LoD building parts which are the building's children. There can be as many building parts per building as the individual volumes produced during the reconstruction.

2.3 Versioning of 3D city models

Versioning of GIS data has been investigated, mostly by practi-tioners with software such as GeoGig[1] and the QGIS versioning plugin[2]. Due to the complex nature of 3D city models, though, these solutions do not apply.

CityGML 3.0 contains a proposal for a versioning module based on Chaturvedi et al. (2017). The proposal defines mechanisms to store both temporal information for city objects (e.g. the 'cre-ation' and 'termination' date and time for a feature) and inform-ation related to their lineage (e.g. through the use of concepts such as feature transitions between different versions). In our opinion, the CityGML versioning module conflates the concept of versioning with the idea of life-cycle modelling. In addi-tion, it aims to store information related to changes between versions of features. This complicates the proposed solution (e.g. by modelling versions, transitions and transactions) and introduces redundancy (e.g. linking features from both versions and transactions) which result in a delicate data model.

2.3.1 Proposed CityJSON versioning In Vitalis et al. (2019) we proposed a data structure to incorporate versioning of 3D city models stored in CityJSON. Our approach aimed to incor-porate the successful characteristics of Git's data structure in the context of CityJSON. Therefore, it focused only on solv-ing the problem of storing the versions of multiple city objects efficiently in CityJSON.

The solution suggests that a CityJSON file can contain all ver-sions of city objects ever existed; and versions can be composed by linking to the individual city objects that are contained in it. This file (called a "versioned CityJSON file") can be considered to be the equivalent of a repository in Git. Every version is, therefore, just metadata containing information about it (e.g. the author, a message and a timestamp) and links to the list of objects that belong to it. In addition, every version holds a link to its parent (i.e. the previous version). Therefore, the lineage of the dataset can be tracked by traversing the list of version from the latest to the earliest.

The system can contain branches and tags similarly to Git, by associating them to a specific version. Therefore, a branch is just a link to the latest version of its history.

To test our proposed system, we have implemented the afore-mentioned functionality in a prototype software (named `cjv`[3]). The software supports all basic commands expected by a com-plete versioning system:

- `commit`, that creates a new version based on a given CityJSON model.
- `checkout`, that extracts a CityJSON model from the versioned dataset.
- `branch`, that creates a new branch based on a given version.

[1] `http://geogig.org/`
[2] `https://github.com/Oslandia/qgis-versioning`
[3] `https://github.com/tudelft3d/cityjson-versioning-prototype`

- `merge`, that creates a new version based on two branches.
- `log`, that shows the history of a given set of branches.
- `diff`, that shows the differences between two versions (at the city object level).

3. METHODOLOGY

A multi-LoD dataset can be stored in a versioned CityJSON file (from now on, called a "repo"). The repo will contain one branch which tracks the evolution of the city objects attributes (`main`). Then, each LoD of the objects is stored in one branch (`lod/`x, where x is the name of the LoD).

Figure 3 shows an example of this approach for a dataset with two LoDs: LoD1 and LoD2[4]. `main` stores just the attributes at the city object level and contains no geometry. Branches `lod/1` and `lod/2` contain the same city objects with only the respective LoD geometry. When attributes of city objects are updated, the LoD branches can be updated by merging from `main`.

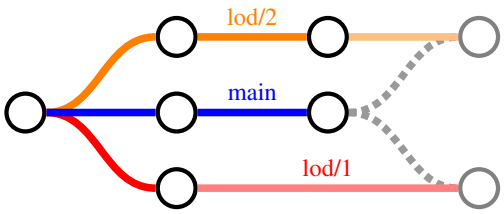

Figure 3. An overview of the evolution of a repo containing a model with two LoDs. Three branches are used: `main` (blue line) for the semantics and attributes of the objects; `lod/1` (red line) for LoD1; and `lod/2` (orange line) for LoD2.

3.1 Creation of the dataset

First, a repo is created using a CityJSON model without geometries, so only the city objects with their attributes are contained; this is stored in branch `main`. Then, we create one branch per LoD. This is done by using a CityJSON file with the same city object hierarchy as in `main`, but also containing one geometry of one LoD at a time to make a commit in each of the branches. Therefore, a city model with only LoD1 geometries is committed to the model and the new commit creates branch `lod/1`. Then, similarly, branch `lod/2` is created. This results in a file with three branches as shown in Figure 4a.

3.2 Updating the model

When one wants to update the semantic aspect of the model, they need to check out from `main`, edit the respective information and, finally, make a new commit to `main` (Figure 4b). Then for each individual LoD branch, a merge needs to occur so that semantics are updated in their lineage (Figure 4c).

When a change in the geometry of an LoD is required, then one just checks out the respective LoD branch and makes the change to the file. Then this file is committed back to its original branch.

[4] While we encourage the use of advance LoDs (e.g. LoD1.2) in real datasets, we use simple LoDs for simplicity in this example.

3.2.1 Merging without conflicts

Conflicts can occur during a merge operation between two branches, when both branches have changed the same *piece of information (PoI)*. PoI defines what is the minimum possible change that a system can identify. For instance, in our previous implementation of a versioning prototype we considered every city object as PoI. This means that, before, when a city object was changed in both branches that were being merged, this was considered to be a conflict even if the changes inside the object were not affecting the same properties (e.g. two different attributes being changed). To resolve this, a more comprehensive system was implemented for this research, so that it treats as PoI every individual property of every city object.

Assuming such a system in place, the proposed solution should not trigger any conflicts. This is because every branch updates different properties of the city objects: the LoD branches would only update the `"geometry"` property and `main` would update everything else except for it.

3.2.2 Adding a new city object

Adding a new city object can add some complexity to the process. This is because under normal circumstances an object would be created alongside its geometries. To resolve this, one can create the city object in a CityJSON file with all its geometries, then create individual versions of the files by filtering out the geometries (e.g. using a tool like cjio[5]) and submit to the individual branches.

3.3 Extracting one or more LoDs

It is possible to extract one LoD of the dataset by simply checking out the specific LoD branch. Before checking out an LoD branch, it is important to ensure that the latest version of `main` has been merged so that the attributes of the model are up-to-date.

To extract a multi-LoD dataset, one can merge the individual LoDs together in a different branch. Given that a merge can only occur between two branches, this has to be conducted in incremental steps. For example, assuming a dataset with three LoDs (1, 2 and 3), the process can be done as follows:

1. A new branch is created based on any of the existing LoD branches (e.g. on `lod/1`). We can name this `multilod/1/2/3` to denote that this is used to combine all three LoDs.
2. One of the other two branches (e.g. `lod/2`) is merged to `multilod/1/2/3`.
3. The last branch is merged to `multilod/1/2/3`.

4. TESTING WITH 3D BAG

To test our approach, we use a CityJSON tile[6] from 3D BAG. As mentioned in Section 2.2, every tile contains four LoDs: LoD0, LoD1.2, LoD1.3 and LoD2.2. The LoDs are assigned to buildings and building parts, so that every building contains all attributes and the LoD0 geometry (i.e. the building's footprint) and it has one building part with LoD1.2, LoD1.3 and LoD2.2 geometries for every derived volume from the reconstruction (Figure 5).

[5] https://github.com/cityjson/cjio
[6] We randomly chose tile no 3693.

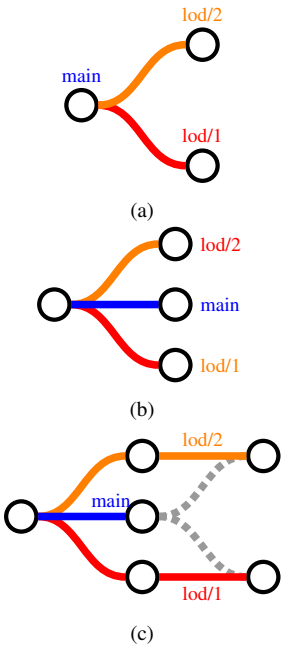

Figure 4. Evolution of the repo using our methodology. In (a), after the repo is created, the `main` branch contains the initial commit with geometry-less city objects and the LoD branches derive from the initial commit contain the respective geometries. In (b), a new commit is introduced in `main` containing the updated semantics of the model. Finally, in (c) a merge occurs towards every `lod` branch to update them with the semantic changes.

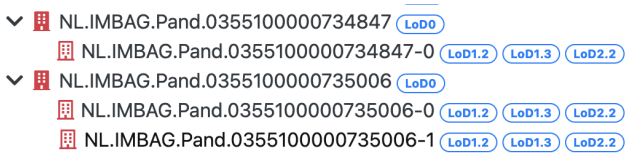

Figure 5. Example of the hierarchy of two buildings from a 3D BAG tile, as shown in ninja (`https://ninja.cityjson.org`). Object `NL.IMBAG.Pand.0355100000734847` contains only one volume, while `NL.IMBAG.Pand.0355100000735006` contains two.

4.1 Initializing the dataset

To prepare the data we used `cjio` to create individual files: one without geometries, and one for every other LoD. We initialized a repo using `cjv` (our prototype software described in Section 2.3.1) and committed the file without geometries. Then, we created one branch for every LoD and committed every filtered file to the respective branch.

4.2 Updating the model

To test the update of one attribute we updated one property of a city object in the geometry-less file and committed it to `main`. Then, we merged the respective change to every branch. Figure 6 shows the output of the history from `cjv` and the diff that was created by the merge. We finally extracted the updated LoD0 model by checking out `lod/0` and inspected it in ninja, so we validated that the change has actually applied.

We also tested the effect of updating a geometry, by altering one vertex in the LoD2.2 geometry of an object then merging

```
Found 6 versions.

*   version 1eb46c720f76396bfae5ecc362847d26bdacd370 (lod/0)
|\  Author: Stelios Vitalis
| |   Date: 2022-07-06 23:56:03.669804
| |   Message:
| |
| |     Merge main to lod/0
| |
| *   version b90f6f370df94336a68b1038f784551091c302e9
| |   Author: Stelios Vitalis
| |   Date: 2022-07-06 23:21:12.343635
| |   Message:
| |
| |     Add all LoD0 geomtries
| |
* |   version f30ca22dd3896dc8c67e1a0fd38c791fa6d05761 (main)
| |   Author: Stelios Vitalis
| |   Date: 2022-07-06 23:14:06.534114
| |   Message:
| |
| |     Update attribute
| |
|/
*   version 12af8d7cb45922f3450af34b172bcbf820342754
|   Author: Stelios Vitalis
|   Date: 2022-07-05 18:49:00.806456
|   Message:
|
|       Initial commit
|
```

(a) The history of `main` and `lod/0`.

```
This is the diff between f30ca22dd3896dc8c67e1a0fd38c791fa6d05761 and 12af8d7cb45922f3450af34b172bcbf820342754
Changes:
        changed: NL.IMBAG.Pand.0355100000828769 (28e7abca698932b.. -> 51d921db265da01..)
2895 objects not changed.
```

(b) The diff shown by `cjv` on the `lod/0` branch after the change from `main` was merged.

Figure 6. The status of the repo after updating an attribute in `main` and merging the change to `lod/0`.

`main` with the updated attribute. Figure 7 shows the history of `lod/2-2` after this operation is completed.

4.3 Extracting multiple LoDs

To test the extraction of multiple LoDs we decided to export a CityJSON file containing LoD0 and LoD2.2 geometries. We first created a branch named `multilod/0/2-2` based on `lod/0`. Then we merged `lod/2-2` to `multilod/0/2-2`. Finally, we checked out `multilod/0/2-2` to a CityJSON file and inspected its content with ninja. Figure 8 shows the resulting model containing just the two LoDs.

5. DISCUSSION

From our experimentation we concluded that `cjv`, despite being just a prototype, has been proven quite reliable and relatively straightforward to use, in order for the proposed workflow to be applied. We believe that this highlights the robustness of the versioning solution, which can be attributed to its simplicity. The data structure that is designed to work on versions and branches, as inspired by Git, is hard to break as there are no redundancies in the model; and Git, using a similar data structure, has been used successfully in large source codebases which share similar characteristics to large geographic datasets.

As mentioned before, certain functionality had to be implemented in `cjv` to support the proposed workflow. Most notably, a more comprehensive diffing and merging mechanism had to be put in place, so that changes between attributes and geometries conducted in different branches would not raise a conflict and could be handled by the system itself.

```
Found 7 versions.

*   version 3fcbbb682abcdce072f492e80b4414866f046381 (lod/2-2)
|\  Author: Stelios Vitalis
| |   Date: 2022-07-08 15:42:59.282702
| |   Message:
| |
| |     Merge main to lod/2-2
| |
| *   version 8cd1805a77c7e8a197227b9336080f77d6c9b1c1
| |   Author: Stelios Vitalis
| |   Date: 2022-07-06 18:16:08.820911
| |   Message:
| |
| |     Alter one vertex in LoD 2.2
| |
| *   version 69caa0923a4e1b82e17876b931a571a5358d4b09
| |   Author: Stelios Vitalis
| |   Date: 2022-07-05 18:52:17.541829
| |   Message:
| |
| |     Add LoD 2.2 geometries
| |
* |   version f30ca22dd3896dc8c67e1a0fd38c791fa6d05761 (main)
| |   Author: Stelios Vitalis
| |   Date: 2022-07-06 23:14:06.534114
| |   Message:
| |
| |     Update attribute
| |
|/
*   version 12af8d7cb45922f3450af34b172bcbf820342754
|   Author: Stelios Vitalis
|   Date: 2022-07-05 18:49:00.806456
|   Message:
|
|       Initial commit
```

Figure 7. The history of `main` and `lod/2-2` after updating one attribute on the first and one geometry on the latter.

By experimenting on a complex dataset, such as 3D BAG, we concluded that there are certain interesting nuances related to the content of a city model that could be used to the advantage of the proposed mechanism. For example, in 3D BAG all building part objects do not contain any attributes, therefore their version in `main` is quite "hollow" (i.e. they are basically empty objects, with just the `"parents"` attribute). If it wasn't for the existence of `"parents"` in the city objects, these would be empty objects which would, eventually, have the same hash; and, as such, they could all be stored as one instance in the data structure to save a lot of space. We believe that this is an interesting scenario and one that highlights that it might be beneficial to investigate ways of optimizing how city objects are stored, so that they do not store unnecessary information in a versioned CityJSON file. In this case, `"parents"` could be easily omitted without affecting the integrity of the file; and, if necessary, a small script could recover the property after a CityJSON file is checked out from the repo.

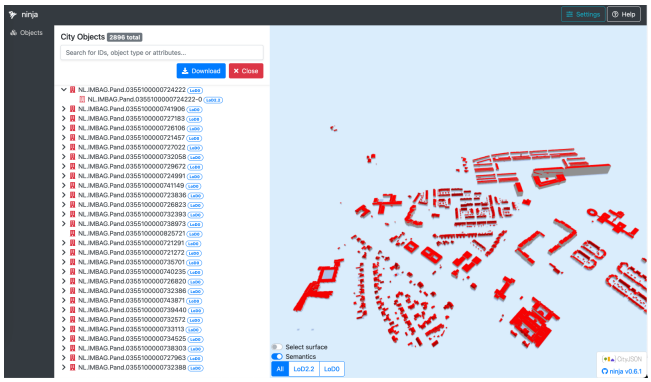

Figure 8. The dataset containing only LoD0 and LoD2.2 visualized in ninja, after updated and extracted from the repo.

While our experience with the versioning mechanism was quite straightforward (assuming certain familiarity with the software), we noticed that a big number of intermediate files had to be created in the process of implementing this workflow. This includes: (a) the different versions of the original tile, after filtering the different LoDs; and (b) the different files that had to be checked out and committed again. We believe that a more user-friendly software could help with improving this aspect. For the first problem, a specific script that filters the geometries and commits them on the fly could be used; and for the second, a software that allows to visualize and edit versions directly from the versioned CityJSON file would simplify the process.

6. CONCLUSIONS

The proposed methodology provides a solution for separating LoDs from each other, so eventually every LoD tracks its own lineage. This provides a framework for maintaining a model where the individual LoDs can be considered to be relatively independent. The solution provides a better way or tracking changes for individual LoDs and extracting subsets of the dataset containing only the required LoD for an application.

Nevertheless, this approach has its own limitations and caveats. First, using a versioning itself adds an overhead of complexity with respect to maintenance, compared to traditional multi-LoD datasets; this is less prominent, though, in cases where versioning is already utilized. Second, such a framework requires a very structured workflow regarding updating the model. Semantics and geometries have to be updated individually and committed to separate branches, otherwise some post-processing (e.g. filtering of geometries) needs to be conducted to separate the changes from one another.

The biggest weakness in the solution, regarding its usability, is the requirement for updating the attributes using files without geometries. While this is, in theory, possible and often updating the semantic aspect of a model is done without the need to visually examine its geometry, the coexistence of semantics and geometries in a 3D city model is a key component and this solution undermines this.

At the current stage, we can consider this methodology more of an exercise that demonstrates versioning's strengths and weaknesses, than a mature solution to be used in practice. This does not rule out the possibility that such a solution can be used as an underlying mechanism in the future, as soon as a comprehensive interface is built on top of it to allow users to interact with the versioned dataset at a higher level. For example, a user-friendly application with a graphical user interface can allow the user to easily extract specific LoDs and commit them back, while the software would take care of separating the changes to multiple files and committing them to the individual branches.

Through our experimentation we were able to further test our initially proposed versioning data structure and further clarify certain concepts and processes related to it. More specifically, we had to emphasize on the complexity of merging branches and its repercussions. Merging is a process heavily dependent on identifying changes between objects and, therefore, we had to implement a robust, yet efficient, diffing solution for city objects. In addition, we had to further define the concept of conflict between two branches. Such a notion is not straightforward and requires a more comprehensive analysis. We concluded that one of the key challenges related to diffing and merging is to define the concept of an object's "identity". At the

city object level this is not as much of a problem, as one can assume that the object's name (in the form of an "id") can be used to derive its identity. But at a lower level, for instance, when multiple geometries are in place this is not always a possibility (in CityGML geometries are not forced to have ids, and in CityJSON this is not possible at all). This raises questions in cases such as when two geometries have been added in two branches; there is no deterministic approach to derive if these are two separate geometries added in the object (hence, resulting in two additions after the merge) or if they concern the same geometry added in both branches (which constitutes a conflict or should result in a combined version of the two geometries).

Based on the above, and for this specific solution, we decided to identify geometries based on their LoD. This strategy should be in line with the expectations of the system, as it was designed. Nevertheless, this solution cannot be considered universal, as in CityJSON it is possible to have multiple geometries of LoDs and in other versioning use cases this might be utilized. This leads us to conclude that, especially with respect to diffing (and, therefore, merging) the specific application for which the versioning system is designed can majorly impact the decisions made. In other words, a versioning system cannot be "unopinionated" and universally designed.

Another important aspect that arose from our research is that the concept of LoD cannot be considered necessarily independent of geometry. This is a major aspect of GIS information, in the sense that generalization of features can affect both their semantic and geometric aspect. 3D BAG's structuring of city objects highlights this as well; the dataset uses different city objects at different levels to store LoD0 separate from LoD1.2, LoD1.3, and LoD2.2. Our methodology was able to handle this designation successfully, but depending on another applications this might complicate the workflow furthermore.

References

Arroyo Ohori, K., Ledoux, H., Biljecki, F., Stoter, J., 2015. Modeling a 3D City Model and Its Levels of Detail as a True 4D Model. *ISPRS International Journal of Geo-Information*, 4(3), 1055–1075.

Biljecki, F., 2017. Level of detail in 3D city models. PhD thesis, Delft University of Technology.

Biljecki, F., Ledoux, H., Stoter, J., 2014. Improving the consistency of multi-LOD CityGML datasets by removing redundancy. *Lecture Notes in Geoinformation and Cartography*, Springer International Publishing, 1–17.

Biljecki, F., Ledoux, H., Stoter, J., 2016a. An improved LOD specification for 3D building models. *Computers, Environment and Urban Systems*, 59, 25–37.

Biljecki, F., Ledoux, H., Stoter, J., 2017. Does a finer level of detail of a 3d city model bring an improvement for estimating shadows? *Advances in 3D Geoinformation*, Springer.

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

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

Chaturvedi, K., Smyth, C. S., Gesquière, G., Kutzner, T., Kolbe, T. H., 2017. Managing versions and history within semantic 3d city models for the next generation of citygml. *Advances in 3D Geoinformation*, Springer, 191–206.

García-Sánchez, C., Vitalis, S., Paden, I., Stoter, J., 2021. THE IMPACT OF LEVEL OF DETAIL IN 3D CITY MODELS FOR CFD-BASED WIND FLOW SIMULATIONS. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLVI-4/W4-2021, 67–72.

Kolbe, T. H., Kutzner, T., Smyth, C. S., Nagel, C., Roensdorf, C., Heazel, C., 2021. *OGC City Geography Markup Language (CityGML) Version 3.0 Part 1: Conceptual Model Standard*. Open Geospatial Consortium. International Standard.

Ledoux, H., Ohori, K. A., Kumar, K., Dukai, B., Labetski, A., Vitalis, S., 2019. CityJSON: a compact and easy-to-use encoding of the CityGML data model. *Open Geospatial Data, Software and Standards*, 4(4).

Löwner, M.-O., Gröger, G., Benner, J., Biljecki, F., Nagel, C., 2016. PROPOSAL FOR A NEW LOD AND MULTI-REPRESENTATION CONCEPT FOR CITYGML. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2/W1, 3–12.

Luebke, D., Watson, B., Cohen, J. D., Reddy, M., Varshney, A., 2002. *Level of Detail for 3D Graphics*. Elsevier Science Inc., New York, NY, USA.

Open Geospatial Concortium, 2021. OGC City Geography Markup Language (CityGML) Part 1: Conceptual Model Standard.

Peters, R., Dukai, B., Vitalis, S., van Liempt, J., Stoter, J., 2022. Automated 3D Reconstruction of LoD2 and LoD1 Models for All 10 Million Buildings of the Netherlands. *Photogrammetric Engineering & Remote Sensing*, 88(3), 165–170.

Vitalis, S., Labetski, A., Ohori, K. A., Ledoux, H., Stoter, J., 2019. A DATA STRUCTURE TO INCORPORATE VERSIONING IN 3D CITY MODELS. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-4/W8, 123–130.