# 3D CHANGE DETECTION FOR SEMI-AUTOMATIC UPDATE OF BUILDINGS IN 3D CITY MODELS

A. Tamort[1], A. Kharroubi[2], R. Hajji[1], and R. Billen[2]

[1] College of Geomatic Sciences and Surveying Engineering, Institute of Agronomy and Veterinary Medicine, Rabat, Morocco
tamortabdelghani@iav.ac.ma - r.hajji@iav.ac.ma
[2] UR SPHERE, Geomatics Unit, University of Liège, Liège, Belgium
akharroubi@uliege.be - rbillen@uliege.be

**KEY WORDS:** 3D City Model, Change Detection, Update, Reconstruction, CityJSON, Buildings.

**ABSTRACT:**

Automatic update of 3D city models has become a crucial operation in the context of urban digital twins. It commonly relies on a reconstruction from ALS point clouds. However, frequent reconstructions due to a dynamic urban environment are resource-intensive and lack change information about the scene. In this paper, we present a novel framework which aims to combine an instance change detection approach based on a distance-computing algorithm, geometric features and thresholding with a building reconstruction algorithm to ensure an efficient geometric, semantic and thematic (change labeling) update of an existing 3D city model. This approach comprises three stages spanning from data preparation to the integration of change results in the updated model. First, we prepare our input data. Next, we assess the changes between the two epochs. This process involves two stages. New and lost buildings are extracted in the first, and the changed and unchanged in the other. The results of the change detection are evaluated using standard evaluation metrics. The evaluation results are encouraging considering the various sources of errors. Finally, unchanged buildings are kept in the model, while the changed and new ones are reconstructed using Geoflow3D. The final model is semantically augmented using a change attribute. Since the 3D city model undergoes an update rather than complete reconstruction, the tracking of both geometric and semantic changes of some buildings can be made possible through a versioning system. The change information can be leveraged in multiple applications like 3D cadastre, urban inventory, urban planning ...

## 1. INTRODUCTION

Three-dimensional (3D) city models are digital representations of our world, enabling a wide range of intelligent applications in the domain of geospatial sciences (Biljecki et al., 2015). These models not only store geometric information but also incorporate the semantic labels and attributes of the urban objects. The creation process of these models is often referred to as 3D reconstruction. Point Clouds (PC) acquired from Airborne Laser Scanners (ALS) have proved to be the most used and efficient data source for the reconstruction of large-scale 3D city models. In light of the current increasing availability of point cloud data in the built environment, information encompassed within the models quickly become outdated with each new acquisition. A generic solution for updating 3D city models would be to reconstruct a new model with each point cloud acquisition. However, this solution does not consider the history of changes within the urban scene, nor it is computationally optimized. Reconstructing highly detailed building models remains challenging given the current level of automation in this field (Arroyo Ohori et al., 2022). Furthermore, a significant number of buildings do not undergo any considerable changes between short time acquisitions. To completely reconstruct them is time consuming and computationally demanding.

Recent advancements in the automatic registration, denoising and semantic segmentation of point clouds, have provided the research community with benchmark of semantically segmented multitemporal point cloud open data of large-scale urban environments. This has played a crucial role in the increasing tendency of the number of scientific papers addressing the 3D change detection (CD) topic and its applications (Kharroubi et al., 2022). Most of the state-of-the-art point cloud based change detection methods that operate on buildings embed the change information on the pixel level, patch-level or point-level, but rarely on the building instance level. These results are usually binary (changed/unchanged) (Stilla and Xu, 2023). Furthermore, methods that tackle multi-class (e.g., changed, new, demolished) change detection on buildings often confuse building instances (i.e., continuous rows of buildings are sometimes considered as a single building instance). Another example would be when a part of a building is demolished, it is often labeled as a whole building that was lost. This is often the case when using traditional methods (methods which are based on difference of distance or height), where no a priori labeling has been done to individualize each building instance.

Learning-based change detection methods provide the best state-of-the-art results but are not well suited for an end-to-end algorithmic approach that can be up-scaled to wider urban extents. They require a substantial amount of labeled training data, they're context-bound (i.e., they adapt to the training data and sometimes over-fit) and computationally demanding. Furthermore, multi-class public change detection datasets are not very common (de Gélis et al., 2023). Conversely, distance-based methods are computationally efficient (fast and less resource consuming) (de Gélis et al., 2021). Moreover, they have not yet been adequately tested on urban environments, especially with a provided instance segmentation information. The latter paves the way for an automatic end-to-end algorithmic fast-computing change detection method, which efficiently optimizes the reconstruction process.

To the best of our knowledge, there is no method (standard or learning-based) that transfers a change information extracted

from point clouds to an existing city model, or leverages it to optimize its geometric and semantic update.

To efficiently update 3D city models, we have implemented a change detection process that utilizes semantically segmented point clouds and two-dimensional (2D) vector files (i.e., building footprints). These building footprints are deployed as instance segmentation to individualize each building. Our aim is to alleviate the computational cost of the reconstruction algorithm (especially when dealing with an urban scene of considerable extent) by focusing only on buildings that are new or those that have undergone a significant change. The demolished buildings will simply be deleted from the existing model and the unchanged ones will be kept. Each building object in the final model will see a change label automatically added to its attributes, providing a history of change between the two epochs and a visual interpretation of change when using a 3D city model viewer.

In order to update an existing 3D city model in the CityJSON encoding (Ledoux et al., 2019), we first provide a semi-automatic point-based instance change detection method . Then, the new and changed buildings are reconstructed, and the updated model is parsed to add the change type as an attribute to each Building or BuildingPart object (2). Next, we present the pipeline's results and we assess the change detection process (3). We finally provide several perspectives for future work (4).

## 2. METHOD

In this section, we first present the input datasets specificities (2.1). Then we showcase how we prepared the input data (2.2). Next, we enumerate the change types we defined (2.3). Afterwards, we present our semi-automatic building instance change detection workflow (2.4) & (2.5). Finally, we showcase how the model is updated using the change information we just extracted (2.6) & (2.7).

### 2.1 Input data specificities

To apply our distance computation change detection process, we must have two point clouds, one of each epoch. Epoch1 point cloud must represent exactly the same urban objects as the existing 3D model (i.e., both refer to the same date). In most cases, the same point cloud that served to reconstruct the existing model is available. Epoch2 point cloud is the newer acquisition. The point clouds should satisfy the following conditions :

- A prior denoising and outlier removal
- A prior semantic segmentation that efficiently discriminates the building class from the other classes
- Both aligned to the same reference frame

The AHN (Actueel Hoogtebestand Nederland, 2023) dataset was chosen due to it being multi-epoch (different acquisitions over time), semantically segmented and having the same spatial partitions in all the acquisitions (e.g., the same tiling schema for AHN3 and AHN4).

The 3DBAG (Peters et al., 2021) is an up-to-date dataset comprising detailed 3D building models across the Netherlands. It is openly accessible data containing 3D models at various levels of detail, produced through automated integration of two open datasets: building data from the BAG (Kadaster, 2023) (stands for "Register of Buildings and Addresses", which is an open dataset of building footprints provided by the Dutch cadastre) and elevation data from the AHN.

We chose to apply our updating methodology on the 21.09.8 version of the 3DBAG for the following reasons:

- It covers the same extent as the AHN datasets
- It was reconstructed using AHN3, which means that they both contain the same building instances (same epoch)
- It was reconstructed using the same algorithm as the one we use in our methodology
- The instance segmentation to individualize each building was also done using precise footprints provided by the cadastre

To separate our buildings into instances to assess the change and to reconstruct on the individual building level, footprints of each epoch must be provided as vector files. Epoch1 footprints are up-to-date with the 3D model, since they can be directly extracted from it using a tool like CityJSON Loader (Vitalis, 2022) which can be found in QGIS (QGIS Development Team, 2009).

Epoch2 footprints need to be up-to-date with their point cloud counterpart, and provided from the same source as Epoch1 ones. Thus, we are guaranteed to have the same building identification system, and the same logic for defining building extents.

The BAG is used to provide reliable building footprints that were generated following the same logic as the ones present in the existing 3D model. This would ensure that if buildings remain untouched in between the two epochs, they ought to have the same footprints.

### 2.2 Data preparation

We prepare our raw data in order to limit their extent to that of the existing model, while focusing on the building objects and alleviating the sources of errors. The following figure summarizes the data preparation process:

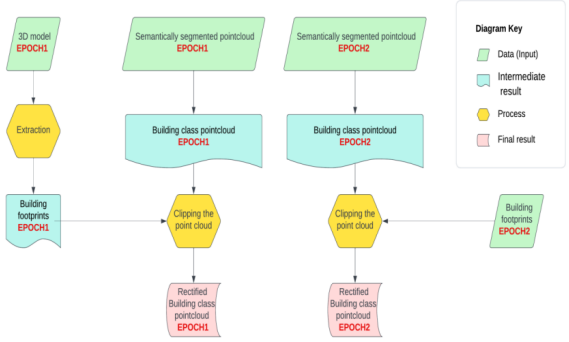



Figure 1. Data preparation diagram.

### 2.3 Change types

We defined 4 change types to serve our methodology. For each one, we have put in place some assumptions to which it has to adhere:

- **New building**: All the points/footprints labeled as "new building" are supposed to refer to building instances that are absent from epoch1, and present in epoch2. These are buildings that were constructed between the two epochs.
- **Lost building**: Points/footprints designated as "lost building" should represent instances of buildings that are present in epoch1 but absent in epoch2. These are the buildings that were demolished between the two epochs.
- **Changed building**: A building instance is regarded as "changed" when a considerable vertical distance has been computed between the epoch1 building and its epoch2 counterpart. The chosen distance is a combination of the vertical accuracies of each of the point clouds, and an additional empirical threshold to eliminate some false changes (e.g., windows that are open/closed in sloping roofs, small temporary objects on the roof etc...). The chosen point candidates are subject to the following conditions:
  - They need to have a high degree of normality along the +Z axis. This is done in order to eliminate the façade points since a considerable number respects the vertical distance criteria. We tend to eliminate building façade points simply because in ALS point clouds they are usually occluded. The same details of the façade might exist in the two epochs, but the point cloud does not necessarily represent this information
  - They need to have a moderate to high level of planarity and a low to moderate level of linearity (to eliminate the remaining façade points)
  - They need to fit into a relatively big cluster of points (to avoid small insignificant objects and noisy points)
- **Unchanged building**: All the building instances that do not fit any of the aforementioned criteria are considered "unchanged".

### 2.4 Extraction of New and Lost buildings

We apply vector intersections between the epoch1 footprints and the epoch2 ones. If a polygon (building instance) from epoch1 does not intersect with any polygon from epoch2, it is added to a new vector file called "lost buildings", if the opposite is true it is added to a new vector file called "epoch1 rest". Conversely, if a polygon from epoch2 does not intersect with any polygon from epoch1, it is added to "new buildings" while the rest is added to "epoch2 rest".

Subsequently, the "new buildings" vector file is used to clip the pre-processed epoch2 point cloud, to create a point cloud containing the points encompassed within the new building instances. The latter will serve as an input in the reconstruction step. In the same way, the "lost buildings" vector file is utilized to clip the pre-processed epoch1 point cloud, to generate a point cloud comprising the points enclosed within the lost building instances. Conversely, the lost building point cloud will not serve in the reconstruction process. It will only be used to extract samples for the validation step.

The clipping and vector intersections were applied using python scripts that we developed, which leverage the Geopandas, Shapely and Laspy modules.

### 2.5 Extraction of Changed and Unchanged buildings

Using the "epoch1 rest" and "epoch2 rest" vector files, we clip the pre-processed "epoch1" and "epoch2" point clouds respectively, to generate two new point clouds where no building instance has been completely demolished or constructed.

The newly generated point clouds are compared using a distance-based change detection tool named Cloud-to-Cloud distance (C2C) (Girardeau-Montaut et al., 2005). This method is the simplest and fastest direct 3D comparison method of point clouds as it does not require gridding or meshing of the data, nor calculation of surface normals. Its simplest version is called "nearest neighbor distance", it identifies the nearest point in the reference cloud for every point in the compared cloud, and then calculates the Euclidean distance using a Hausdorff distance computation algorithm. It is relevant to point that before computing the distance, the point clouds are divided into cells using an octree data structure.

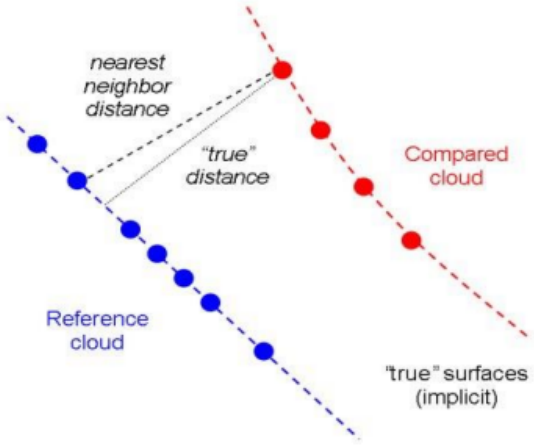



Figure 2. Nearest neighbor distance principle
(Girardeau-Montaut et al., 2015)

Results of C2C are displayed as scalar fields on the compared point cloud which refers in this case to the "epoch2 rest" one.

We also compute the normality along the +Z axis. Normal computation is the process of calculating the normal vector for each point in a point cloud. The normal vector is a vector that points in the direction of the local surface at that point.

Our particular focus lies on the C2C positive distance along the Z axis, which means that buildings that have undergone a reduction in height are not considered. This assumption is made to facilitate the process, since buildings are usually changed by adding floors or parts, by complete demolition and rarely by individual floor demolition.

A threshold is applied to extract changed candidate points. The latter comprises the vertical accuracies of each of the point clouds, along with an empirical threshold to remove false changes. Then we eliminate points having a low value of the Nz scalar field (normality along the +Z axis). It is done in order to keep the

roof points (in theory having a high value of Nz) and eliminate the façade points that lead to false changes.

Geometric features like planarity and linearity are computed on the remaining candidate points. Linearity is a geometric feature that measures how linear a surface is at a given point. A value of 1 indicates that the surface is perfectly linear, while a value of 0 indicates that the surface is not linear at all. Planarity is defined as the degree to which the normal vectors of a local neighborhood of points are aligned. In other words, a point is considered planar if the variance of the normal vectors in its neighborhood is low. A combination of high degree of linearity and low degree of planarity is used to filter out the remaining façade points.

Finally, a clustering algorithm such as ”Label Connected Components” (Girardeau-Montaut, 2015) is used to keep only the compact clusters that contain a considerable number of points. The chosen points are added to a point cloud labeled as “changed buildings”. These points are intersected with the “epoch2 rest” vector file. For each polygon, we count the number of points that are encompassed within it (points are projected to the XY 2D plane) and apply the same threshold as in the clustering step. Polygons that respect this condition are added to a new vector file labeled “changed buildings”. The others are generated as “unchanged buildings” vector file. Both resulting vector files are used to clip the “epoch2 rest” point cloud and produce two new point clouds. The “changed buildings” point cloud is used in the reconstruction and validation step while the “unchanged buildings” one is only used in validation.

The distance computation, the thresholding, the scalar fields creation (normality and geometric features) and the clustering are executed using CloudCompare (Girardeau-Montaut et al., 2015).

The figure 3 sums up the change detection process.

### 2.6 Reconstruction and building matching

For optimization purposes, we must differentiate between what must be kept in the existing city model and what must be reconstructed. A significant proportion of the buildings in the urban landscape remain unchanged over short periods of time. One of the main reasons of our change detection step is to point out these instances to leave them untouched in the existing model. When dealing with the update of 3D city models on a country-scale for instance, this filtering step (i.e., pinpointing unchanged buildings) can greatly optimize the reconstruction process in both time and resource consumption.

In order to achieve this, we match the identification number “ID” of building instances in the “unchanged buildings” vector with the “ID” of the Building and BuildingPart objects in the existing CityJSON. The matched objects are kept in the model while all the other objects are deleted. Lost buildings are also extracted via matching between model and vector file. However, we need to extract only their LoD0 footprint as a CityJSON file. The matching is done using a python script that we developed, while the LoD0 extraction is made possible using cjio (Ledoux, 2021).

Next, we reconstruct the new buildings and changed buildings using Geoflow3D. The latter is a framework that performs an automatic reconstruction of 3D building models by leveraging 2D building polygons and an ALS point cloud. The process


Figure 3. Change detection process diagram

generates models at various LoDs, all derived from a unified data source. A key focus of the workflow is to ensure robustness, allowing for easy iteration in response to algorithmic enhancements or the availability of new input data. The quality of the reconstructed data is closely monitored at multiple stages throughout the process, acknowledging that the outcome is heavily influenced by the quality of the input data.

For each of these two models, we enter the required footprints and point cloud to run the tool. The output is two CityJSON 3D city models named “New” and “Changed” respectively.

### 2.7 Semantic augmentation

Now that we have a CityJSON model for each building change type, we use python scripts that we developed, which leverage the straightforward and easy parsing of the CityJSON format, to add the change type as an attribute to each Building or BuildingPart object in each of the aforementioned models. Finally, we merge the models using cjio.

Figure 4 summarizes the update step.

## 3. RESULTS

For visualisation purposes, we inspect the original model in LoD2.2 using ninja (Vitalis et al., 2020) (Figure 5).

When inspecting the updated model, we apply conditional formatting, which consists in attributing to each Building or BuildingPart a distinct colour based on the values of an attribute. We choose the attribute “change_type”, and we define a color for each value (Figure 6).

In order to validate the proposed methodology and the framework implemented in this paper, we used some standard classi-

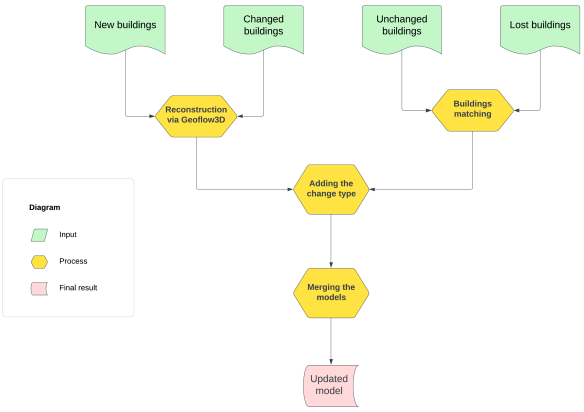



Figure 4. Summary of the update process

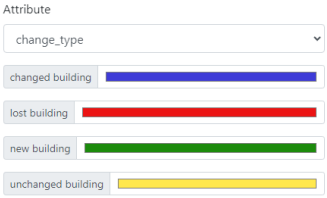
Figure 5. Top-view of the original model.

fication evaluation metrics present in the scikit-learn (Pedregosa et al., 2011) python library :

- Overall Accuracy (OA)
- Precision
- Recall
- F1 score
- Intersection over Union (IoU)

We manually prepared our validation dataset using orthoimages available in geotiles. We identified some buildings within the extent of our scene, then carefully and scrupulously annotated their points with their corresponding change type. Thus, for the chosen validation set, each point has two scalar fields:

- "prediction": which is the scalar field containing the detected changes using our pipeline. It contains 4 values:
  - 1: New
  - 2: Lost
  - 3: Changed
  - 4: Unchanged
- "ground _truth": which contains the change types that we manually annotated. The same codification as "prediction" is used.



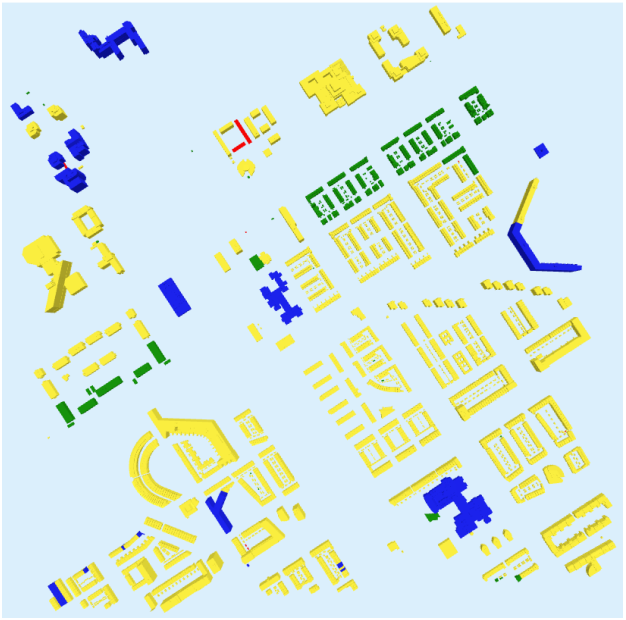



Figure 6. Final updated model

We regrouped the evaluation metrics results in table 1.

| Evaluation metric | Accuracy | Precision | Recall | F1 score | IoU |
|---|---|---|---|---|---|
| Value | 0.866 | 0.911 | 0.866 | 0.868 | 0.777 |

Table 1. Values of the evaluation metrics

The resulting normalized confusion matrix is illustrated in figure 7.

We notice that a considerable number of points has been labeled as "Changed" while the ground truth data shows that they should belong to the "Unchanged" class. This is due to several reasons; we enumerate the ones that are recurrent in our scene:

- Windows on slanted roofs are detected as change when they are closed in one epoch and open in the other
- Large temporary objects on roof terraces (e.g., vehicles if the building roof serves as a parking)
- Inconsistent point sampling. The same buildings are detailedly sampled in epoch2 while they have a low density of points in epoch1

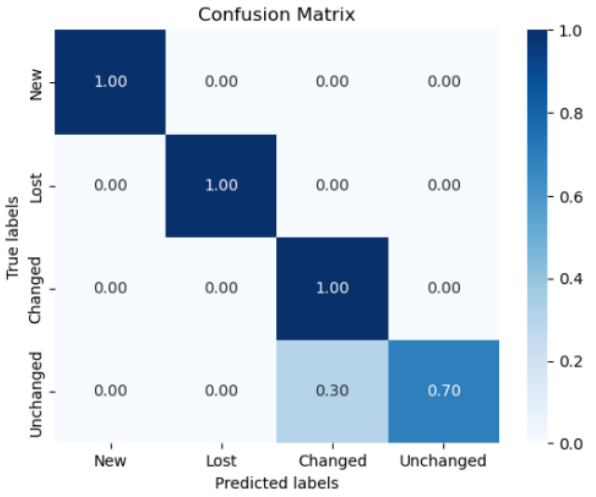



Figure 7. Resulting normalized confusion matrix

- Occlusions. Gaps in either point cloud might lead to a false change when computing vertical distance
- Remaining façade points that could not be filtered out with the assumptions we used

## 4. DISCUSSION

Our paper proposes a novel approach to deal with the updating of building objects in a 3D city model. With our proposed methodology, we managed to apply a point-based distance computation to assess the changes in building instances of a 3D city model, while updating it with newly reconstructed building objects and thematic change information.

Optimizing the reconstruction process by only reconstructing selected buildings from a prior change detection step, is not a common research topic. In most of the current state-of-the-art 3D change detection methods, the instance segmentation is not an important step, since the objective is usually to display inter-class change on the point-level. While in our case, the change information on the point-level is not enough to update the existing model, since no explicit affiliation information exists between each point and the building object to which it belongs. Geoflow3D makes use of the 2D vector building footprints to establish this affiliation.

The only framework that we found which has similar objectives as our method is the third version (2023.08.09 – beta) of the 3DBAG, in which a new attribute that serves as a change information has been added. The said attribute is named "b3_mutatie_ahn3_ahn4", it outputs the value "true" for each building object for which the point cloud has undergone a significant change between two epochs (AHN3 and AHN4). In other words, it is a binary change attribute. The reconstruction is also improved in a qualitative way. When it is detected that no significant change has been observed, and that the AHN3 data is denser and covers the roof surface better than the AHN4, the building object is reconstructed using the AHN3 and not the AHN4.

Based on our proposed methodology's limitations and new questions raised during our experiments, we would like to explore and recommend continuing this research in the following directions:

- Finding more rigid and robust assumptions to filter out false changes, which automatically adapt to the specificities of each urban scene.
- Considering an ALS acquisition simulator like Helios++ (Winiwarter et al., 2022) to provide an epoch1 point cloud having the same acquisition parameters as the epoch2, and thus getting rid of occlusions.
- Coming up with a learning pipeline, which leverages the local neighborhood of each point to assess the change at the building instance level without the need to provide epoch2 building footprints. The pipeline should be able to learn to define the same instances for the changed and unchanged buildings as the ones found in the existing model, and define new ones for the newly constructed buildings. It should also be robust against differences in point density, occlusions, noise and prior semantic segmentation discrepancies between the two epochs.
- Explore the generalization process of the reconstruction algorithm (the reconstructed model does not accurately represent the same geometrical aspect as the point cloud), to pinpoint exactly what details are worth considering as a change (since only those are going to be clearly visible in the reconstructed model). Thus, making the "model to cloud comparison" possible without having to work with the underlying point cloud from epoch1.
- Generalize the pipeline to not only cover one tile of the 3D city model, but also scale it to the extent of a whole region or country for instance. This is made possible using indexing and tiling techniques. An example of this scaling approach is detailed in (Yarroudh, 2023).

## 5. CONCLUSION

In this work, we proposed a comprehensive approach in which we investigated the robustness of semi-automatic change detection using distance-based methods on urban ALS point clouds, which have undergone instance segmentation using 2D building footprints. We also automatically detected the concerned buildings in the model: we kept the unchanged ones, deleted the lost ones and left their LoD0 footprints for inspection purposes, and reconstructed the new and changed ones. Then each building object received its respective change type as an attribute. Depending on factors like the input point clouds' vertical and horizontal accuracy, point density and thresholds of vertical distance and geometric features, our methodology can be applied to various applications. For the chosen values of these parameters, our methodology can be applied, for instance, to detect violations of urban planning laws. Buildings which have undergone an illegal construction of floors or installations (huts or walls on the rooftop) can be identified. By producing python scripts, we emphasize our commitment to the use of open-source software. We aimed to ensure that our method can be reproducible, verifiable and built-upon by students, researchers and practitioners. The python scripts that were developed can be accessed via github.

## REFERENCES

Actueel Hoogtebestand Nederland, 2023. https://www.ahn.nl/.

Arroyo Ohori, K., Ledoux, H., Peters, R., 2022. *3D modelling of the built environment*. https://3d.bk.tudelft.nl/courses/geo1004/data/3dbook.pdf.

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

de Gélis, I., Lefèvre, S., Corpetti, T., 2021. Change Detection in Urban Point Clouds: An Experimental Comparison with Simulated 3D Datasets. *Remote Sensing*, 13(13). https://www.mdpi.com/2072-4292/13/13/2629.

de Gélis, I., Lefèvre, S., Corpetti, T., 2023. Siamese KP-Conv: 3D multiple change detection from raw point clouds using deep learning. *ISPRS Journal of Photogrammetry and Remote Sensing*, 197, 274-291. https://www.sciencedirect.com/science/article/pii/S0924271623000394.

Girardeau-Montaut, D., 2015. Cloudcompare documentation. `https://www.cloudcompare.org/doc/wiki/index.php/Label_Connected_Components`.

Girardeau-Montaut, D. et al., 2015. CloudCompare: 3D point cloud and mesh processing software. *Open Source Project*, 197.

Girardeau-Montaut, D., Roux, M., Marc, R., Thibault, G., 2005. Change detection on point cloud data acquired with a ground laser scanner. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36.

Kadaster, 2023. Basisregistratie adressen en gebouwen. `https://www.kadaster.nl/zakelijk/registraties/basisregistraties/bag`.

Kharroubi, A., Poux, F., Ballouch, Z., Hajji, R., Billen, R., 2022. Three Dimensional Change Detection Using Point Clouds: A Review. *Geomatics*, 2(4), 457–485. https://www.mdpi.com/2673-7418/2/4/25.

Ledoux, H., Arroyo Ohori, K., 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(1), 4. https://doi.org/10.1186/s40965-019-0064-0.

Ledoux, H. e. a., 2021. cjio, a python cli to process and manipulate cityjson files. `https://github.com/cityjson/cjio`.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E., 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

Peters, R., Dukai, B., Vitalis, S., van Liempt, J., Stoter, J., 2021. Automated 3d reconstruction of lod2 and lod1 models for all 10 million buildings of the netherlands.

QGIS Development Team, 2009. QGIS Geographic Information System. Open Source Geospatial Foundation.

Stilla, U., Xu, Y., 2023. Change detection of urban objects using 3D point clouds: A review. *ISPRS Journal of Photogrammetry and Remote Sensing*, 197, 228-255. https://www.sciencedirect.com/science/article/pii/S0924271623000163.

Vitalis, S., 2022. Cityjson loader qgis plugin. `https://github.com/cityjson/cityjson-qgis-plugin`.

Vitalis, S., Labetski, A., Boersma, F., Dahle, F., Li, X., Arroyo Ohori, K., Ledoux, H., Stoter, J., 2020. CITYJSON + WEB = NINJA. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, VI-4/W1-2020, 167–173. https://isprs-annals.copernicus.org/articles/VI-4-W1-2020/167/2020/.

Winiwarter, L., Esmorís Pena, A. M., Weiser, H., Anders, K., Martínez Sánchez, J., Searle, M., Höfle, B., 2022. Virtual laser scanning with HELIOS++: A novel take on ray tracing-based simulation of topographic full-waveform 3D laser scanning. *Remote Sensing of Environment*, 269, 112772. https://www.sciencedirect.com/science/article/pii/S0034425721004922.

Yarroudh, A., 2023. Optim3d: Optimized reconstruction of large-scale 3d building models. GitHub Repository.