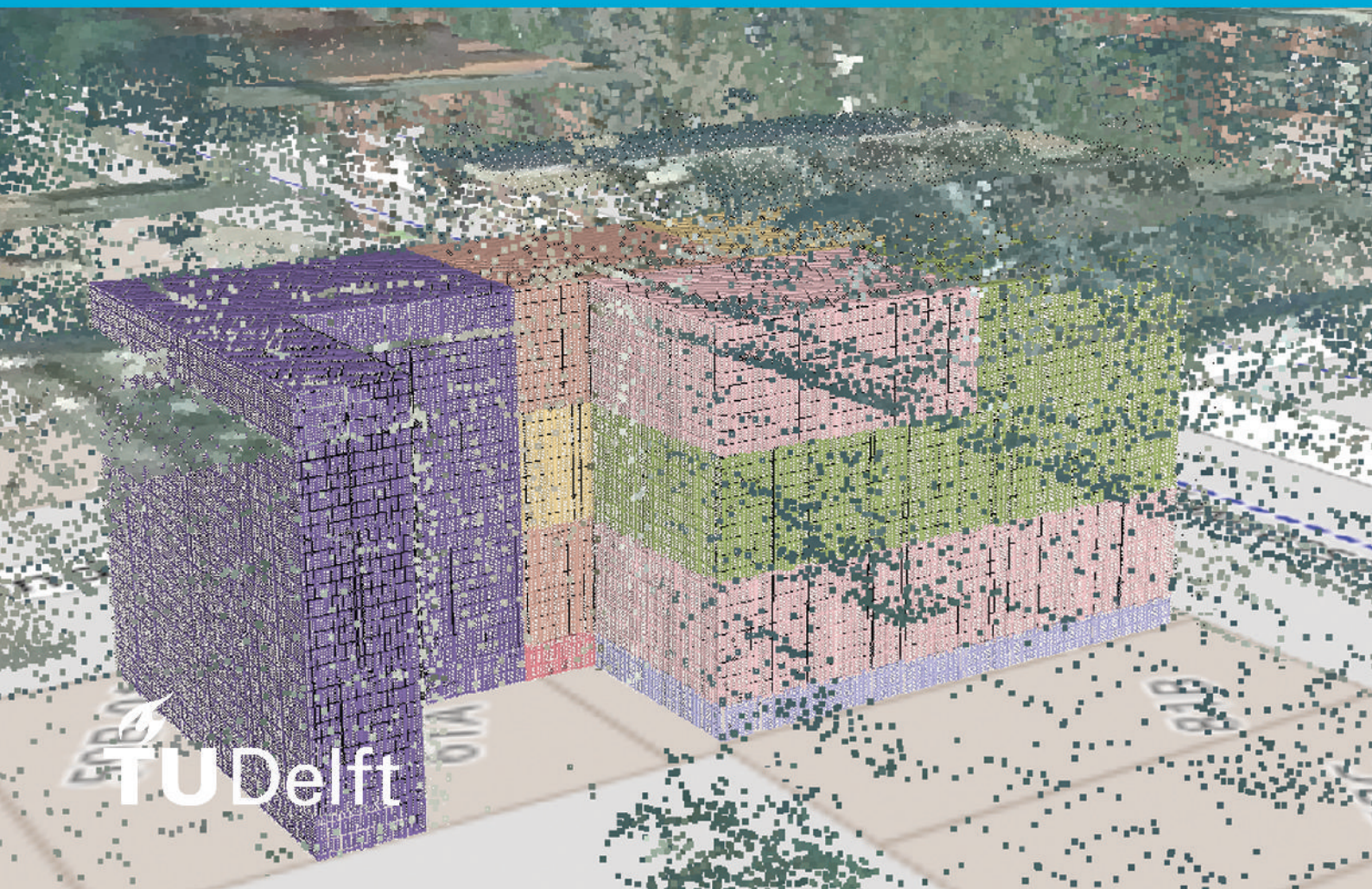
MSc thesis in Geomatics

Point Cloud for 3D Land Administration System (LAS)

Citra Andinasari
2025



MSc thesis in Geomatics

Point Cloud for 3D Land Administration System (LAS)

Citra Andinasari

July 2025

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

The work in this thesis was carried out in the:

Geo-Database Management Centre
Delft University of Technology

Supervisors:	Prof. Peter van Oosteroom
	Ir. E. Verbree
Co-reader:	dr. M.N. Koeva

Abstract

As cities grow denser and more and more in vertical directions, Land Administration Systems (LAS) must evolve to represent complex, multi-level property ownership, particularly in apartment buildings. While Building Information Models (BIM) are commonly used for 3D representation, their availability remains limited for many buildings. This research explores the use of point clouds as an alternative means to represent 3D spatial units in LAS, focusing on the integration of cadastral floor plans and the airborne Lidar point cloud datasets (in our case Actueel Hoogtebestand Nederland (AHN)).

Three apartment cadastral drawings from different years in Rotterdam serve as case studies. The proposed methodology involves five main steps: (1) parsing the scanned image of the floor plans using image processing to extract cadastral room boundary polygons; (2) segmenting AHN point cloud; (3) generating synthetic point clouds by extruding floor plan polygons and aligning them with AHN; (4) storing these 3D spatial units in a PostgreSQL-based database following the ISO 19152:2024 Land Administration Domain Model (LADM); and (5) developing a web-based 3D LAS using Vue.js, Cesium, and FastAPI for visualization and interaction.

Results show that unit boundaries can be extracted from cadastral drawings and converted into 3D point clouds for integration into a cadastral database. The synthetic point clouds include room-level attributes and spatial identifiers, enabling interactive visualization and LADM information through a web interface that can be accessed by the public and stakeholders. However, challenges such as misalignment due to occlusion in AHN data and inconsistent quality in older floor plan drawings affect the accuracy and automation of the process.

This research demonstrates that point clouds can effectively serve as final 3D representations in land administration, providing a scalable solution in the absence of BIM models and minimizing the need for additional field surveys. It also enables a seamless integration with AHN, offering a representation of real-world features such as building facades, walls, and fences, which often delineate cadastral boundaries.

The code for this project is available in GitHub, while the website can be accessed in gist.bk.tudelft.nl/apps/LADMPointCloud/.

Acknowledgements

I would like to express my deepest gratitude to all those who helped me finish this thesis. First, I am sincerely thankful to my supervisors, Peter van Oosterom and Edward Verbee, for their continuous support from the beginning to the end of this journey. Their valuable feedback and insight guide me to write and improve my research, and their understanding and patience encourage me to face all these difficulties and self-doubt along the way. I would also like to thank my co-reader, Milla Koeva, for her feedback and suggestions to refine my research. In addition, I would like to thank Bastiaan van Loenen for his support in hosting the presentation during my graduation procedure, and Martijn Meijers for his valuable insights and support in hosting the web server.

Although we are separated by more than a dozen thousand kilometers away, I am so grateful from the bottom of my heart for the endless love from my parent that has always accompanied throughout these days. Thank you to my father, who has always been the best father figure I can imagine. Without you, I would never have dreamed of studying abroad. Thank you to my mother, who always takes care of me, gives me warmth, and never tires of listening to my whining and crying. Their support always encourages me to stand, and their prayer always protects me to move forward. Thanks also to Waro, our chunky-fatty-lazy cat, who is supposed to be my mental support.

I would like to thank my Geomatics classmates who accompanied and helped me during the entire time of my master's studies. Thank you also to all of my friends who have become my second family here, with whom I can travel around exploring the world, with whom I can share all of the hardship that I endure during this study, and with whom I have first met at the departure from Indonesian airport to our countless departures together.

Finally, I would also like to thank LPDP for giving me the scholarship and the opportunity to study my master's here.

I dedicate this to my dearest Papa and Mama

Contents

Acronyms

API	Application Programming Interface
CSF	Cloth Simulation Filter
RANSAC	RANdom SAmple Consensus
AHN	Actueel Hoogtebestand Nederland
LA	Land Administration
LAS	Land Administration System
LADM	Land Administration Domain Model
PDAL	Point Data Abstraction Library
ICP	Iterative Closest Point
GIS	Geographical Information System
LiDAR	Light Detection and Ranging)
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
RRRs	Right, Restrictions, and/or Responsibilities
BIM	Building Information Modelling
IFC	Industry Foundation Classes
BAUnit	Basic Administrative Unit
SU	Spatial Unit
DT	Digital Twin
ALS	Airborne Laser Scanning
TLS	Terrestrial Laser Scanning
MLS	Mobile Laser Scanning
HT	Hough Transform
MBR	Minimum Bounding Rectangle
UML	Unified Modeling Language
OCR	Optical Character Recognition
CRS	Coordinate Reference System

HTML	Hypertext Markup Language
CSS	Cascading Style Sheets
RMSE	Root Mean Square Error
JSON	JavaScript Object Notation
SQL	Structured Query Language

1. Introduction

This chapter begins with a brief explanation of the problem statement of the thesis topic. To address the identified problems, a main research question is formulated, followed by a set of sub-questions. The scope of the research is defined to clarify the boundaries and limitations of the study, indicating what is covered and what is not. Finally, an overview of the thesis report is provided, outlining the structure and content of the subsequent chapters.

1.1. Introduction

Rapid growth in urban areas has led to an increasing number of apartment buildings. This growth requires a Land Administration System (LAS) capable of optimally storing and visualizing the legal status of these structures, ideally with 3D representation. LAS is a system formed by land administration and land registration, which maps the land parcels and registers their Right, Restrictions, and/or Responsibilities (RRR). Building Information Modeling (BIM) has been widely used in various studies as 3D representation in digital twin [Nguyen and Adhikari, 2023; Alonso et al., 2019], demonstrating great potential to represent LAS [Mao et al., 2024; Meulmeester, 2019]. However, since not all buildings have BIM data available, it raises the question of how to address this limitation.

Recent studies have used point clouds as the basis for creating digital twins. Baauw [2021] studies that the Actueel Hoogtebestand Nederland (AHN) point cloud is capable of fulfilling the basic requirements of a digital twin as it provides a realistic 3D visual representation and, through segmentation and classification, the semantic information can be derived, allowing direct interaction. Using point cloud, historical or previous epoch data can be easily compared for change detection and integrated with temporal attributes. However, as the point cloud from ALS can only capture the exterior building envelope, an additional method to model the walls and slabs for property boundaries needs to be explored. In the Netherlands, providing notarial deeds to the Cadastre government is obligatory, including floor plans to register the apartment rights. As the land administration system required a real-world presentation, this study attempted to visualize the 3D LAS by directly using the point cloud, enriching its semantics, and representing the 3D spatial unit derived from the floor plan.

1.2. Research Objectives

The main research question of this thesis is:

To what extent can point clouds represent and visualize 3D spatial units?

To address this question, the following sub-questions are posed:

1. What is the suitable method to parse the floor plan?
2. How can apartment spatial units be represented when exterior wall points are unavailable due to occlusion?

3. What approach can be used to represent wall and slab points for apartment spatial units inside the apartment building?
4. What approach can be used to accurately represent apartment spatial units and their boundaries using point cloud?
5. How can point clouds be stored in the Land Administration Domain Model (LADM) database for multi-spatial-unit apartments, including living units, storage areas, and parking spaces, as a single basic administrative unit?
6. Which web architecture is suitable for representing and visualizing the resulting 3D LAS?

1.3. Scope

This research aims to enrich the point cloud with spatial units derived from the floor plan, mapping it into the LADM database to develop 3DLAS and visualize it on the web. It focuses on an apartment building as the case study. The study provides a semi-automated pipeline to parse the floor plan and enrich the point cloud. The scope of this study does not involve any indoor laser scanning observation by Terrestrial Laser Scanning (TLS) and Mobile Laser Scanning (MLS), as the input data for the point cloud is only acquired from the AHN dataset.

1.4. Thesis Outline

This thesis is organized into six chapters. The first chapter is the introduction to the research and the objectives of the research, which includes the question and the scope. Chapter 2 describes the relevant theoretical background and other related research about point cloud, floor plan, LADM, and visualization of the 3D land administration system. Chapter 3 explains the methodology of the research, which is divided into 4 (four) main segments: parsing the floor plan, processing point cloud, generating a synthetic point cloud, storing the point cloud in LADM, and visualizing 3D LAS. The entire process of the pipeline is described in detail, step by step. Chapter 4 specifies the required tools to conduct the research, including the software and library. The dataset used as a sample for the study is also explained. Chapter 5 discusses the results of the experiments from the proposed pipeline. The evaluation and used parameters are also elaborated in this chapter. Chapter 6 concludes the study by answering the research question and identifying what improvements can be made for future studies.

2. Related Work

This chapter provides a comprehensive review of the literature relevant to this thesis. It begins by introducing the main topics, including Point Cloud, Floor Plan, Land Administration Domain Model (LADM) and Industry Foundation Classes (IFC), and 3D Land Administration System (LAS) Visualization. Subsequently, it examines previous research on each topic, focusing on the methodologies that are applicable to this study. The chapter also explores the interconnections between these topics, supported by relevant references.

2.1. Point Cloud

Point cloud is a set of 3D data points that can be organized to capture geometric information of the entire 3D object, and also can contain attributes like semantic information (e.g. classification) and RGB color. The two methods to acquire 3D points are the ranging-based principle, using laser scanning, and the image-based principle, using multiview stereo vision from cameras. Light Detection and Ranging) (LiDAR) is a surveying technology used in range-based methods. It employs laser pulses to measure distances to the surrounding objects from a LiDAR scanner by calculating the travel time of the laser signal. The intensity of laser light reflected by an object is influenced by object characteristics, including the material and color of its surface [Tiberius et al., 2022]. Laser scanners have three types: Airborne Laser Scanning (ALS), Terrestrial Laser Scanning (TLS), and Mobile Laser Scanning (MLS).

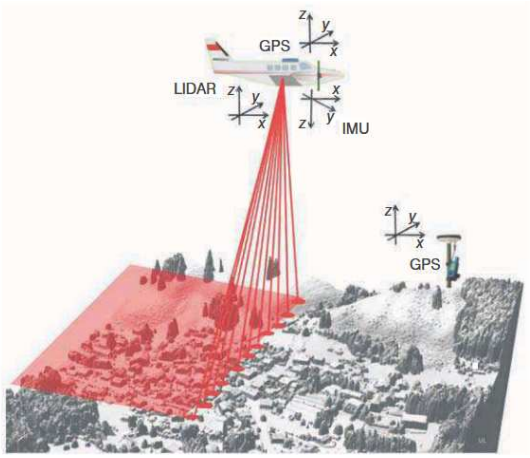


Figure 2.1.: ALS [Vosselman, 2011]

ALS is carried out using a fixed-wing aircraft, a helicopter, a drone or even a satellite with two main components: a laser scanner system that employs LiDAR technology and a GPS/IMU combination to determine its position and orientation, as shown in Figure 2.1. As TLS and MLS are able to acquire more accurate precision, ALS can measure point clouds for larger areas and has been used for efficiently capturing precise and highly detailed spatial

data of urban environments, including terrain elevation, vegetation, etc. Although ALS enhances the automation of accurate and efficient 3D urban model reconstruction, it also suffers from occlusion. Occlusion is a phenomenon where data is quite sparse or missing in certain point cloud regions because those areas were hidden from the sensor's view during data acquisition [Manders, 2023], as illustrated in Figure 2.2. Incomplete data remains a challenge, as key structures like vertical walls of buildings are often missing in airborne LiDAR point clouds due to the limited positioning and movement trajectories of airborne scanners [Huang et al., 2022].

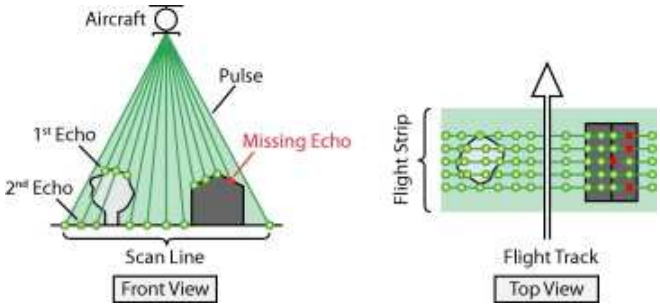


Figure 2.2.: Occlusion [Hinks et al., 2015]

a. Actueel Hoogtebestand Nederland (AHN)

AHN is a Dutch national dataset acquired using ALS technology containing a point cloud, digital terrain model, and digital elevation model. Since its first measurement in 1996, AHN has been updated for a period of time and produced 5 (five) data series. AHN has a height accuracy of no more than 5 (five) cm with a point density between 6 and 10 points per square meter for AHN2 and AHN3 and between 10 and 14 points per square meter for AHN4. The planimetric accuracy of AHN versions 2 to 4 is roughly 5 cm random error and 8 cm systematic error [AHN, 2020]. Since AHN3, the classification has been provided, such as ground, vegetation, building, and water [AHN, 2020]; however, the current AHN5 still does not have a sufficient classification for building class as it only classifies the roof, not the entire building as done in AHN3 and AHN4.

b. Point Cloud Segmentation

Segmentation is grouping several homogeneous points based on their common features. There are three most popular methods for segmentation.

1. Model fitting, which uses a mathematical representation.
   - Hough Transform (HT) uses a voting mechanism to identify shapes by transforming the data to parameter space and forming clusters according to their geometric features in the point cloud with the highest number of votes
   - RANdom SAmple Consensus (RANSAC) defines a model parameter from a minimum sample of random points. Iteratively checking their neighboring points, and a consensus set is formed if they are a match
2. The Region Growing method is performed by growing the segments from seed points and then expanding the segments iteratively by checking their neighbors that satisfy

similarity criteria (normal or distance).

3. Unsupervised Clustering-based methods
    - K-means starts by random points as centroids with a given number of clusters (k) and assigns the closest points to the corresponding centroid to form a cluster.
    - Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a density-based algorithm that identifies core points with a sufficient number of neighbors and expands their clusters.

c. Occlusion Correction

To densify the sparse point cloud and missing data in some regions due to occlusion, an occlusion correction technique can be employed. Balado et al. [2019] proposed the technique to generate points in highly colluded sidewalk areas of the point cloud dataset acquired by MLS. The point cloud is rasterized with a grid size of twice the distance between points, allowing all pixels to be filled with points and avoiding empty pixels. Three binary images are created: the sidewalks, the rest of the elements, and the global. The global binary image and the other elements' image are integrated to produce a mask image. The sidewalk image is expanded iteratively using morphological dilation, and then the pixels that overlap with the mask image are removed, ensuring the growth of the sidewalk image does not go beyond the external limit of the point cloud. The resulting sidewalk image is then subtracted from the initial sidewalk image to generate the occlusion image, where the only large occlusions are kept. To individualize occlusions, connected missing points are identified and grouped together, then are slightly expanded to clearly define their boundaries. All points along the edges of each occlusion are identified as boundary points to create a polygon outlining the occluded area. New random points are generated inside the polygon with the same density as the original point cloud to fill the empty space, where their Z coordinates are calculated through a multiple linear regression model and the existing boundary points as the sample. The new points with complete XYZ coordinates are put back into the point cloud, filling the missing areas. The algorithm for this entire process is depicted in Figure 2.3 while the result is in Figure 2.4.

d. Point cloud for 3D Land Administration

Various studies have explored the use of point clouds for 3D Land Administration. Koeva et al. [2019] demonstrated the ability to automatically detect changes in building geometry over time by utilizing point clouds linked to the LADM. The study showed promising results, as relevant changes for 3D Land Administration (e.g., walls and rooms) could be differentiated from temporary changes (e.g., people and furniture) and were connected to spatial subdivisions. This approach enables the Land Administration database to be updated based on the detected changes. However, the study was conducted in a university building, which does not fully represent the full range of real-world scenarios involving private units.

Another study by Bydłosz et al. [2021], in Cracow Country, Poland, proposed a 3D LAS using 3D Building Information Modelling (BIM) reconstructed from TLS point clouds. While point clouds offer a more realistic representation and can accommodate differences from the original architectural design, preparing a 3D model using TLS involves significant costs in terms of both time and money, which is not always possible, especially for large-scale.

Algorithm 1 (*Occlusion correction*).

Inputs: Refined_Point_Cloud {Pr}, Labels {L}, grid_size *gs*
Outputs: Regenereted_Point_Cloud {R}
Raster_image {I} ← Raster (P,L,gs)
Sidewalk_image {IS}, Rest_of_Elements_image {IE} ← binary (I)
Limit_image {IL} ← fill (I)
Mask_image {IM} ← complement (IL) + IE
Sidewalk_image_prev {ISp} ← zeros (IS)
Sidewalk_image_new {ISn} ← IS
While ISp ≠ ISn **do**
  ISp ← ISn
  Sidewalk_image_dilated {ISd} ← dilate (ISn)
  ISn ← ISd – IM
End_While
Occlusion_image {IO} ← morphological_opening (ISn - IS)
CC_oclusion_image {IOcc} ← ConnectedComponents (IO)
R ← Pr
For each IOcc(i)
  IOcc_dilated { IOccd} ← dilate (IOcc (i))
  IOcc_dilated_Sidewalk { IOccdS} ← IOccd - IE
  Sidewalk_Points {PS} ← {P: P ∈ IOccdS}
  Polygon {Pol} ← polygon{PS}
  New_Sidewalk_Points_XY {PSnXY} ← random (2*gs,inpolygon(Pol))
  New_Sidewalk_Points_Z {PSnZ} ← linear_regresion_model (PSnXY, PS)
  R ← Add (PSnXY, PSnZ)
End_For
Return {R}

Figure 2.3.: Occlusion Correction Algorithm [Balado et al., 2019]

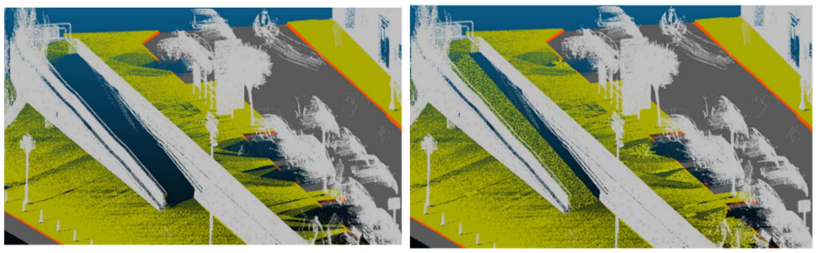


Figure 2.4.: Before and After Occlusion Correction Application [Balado et al., 2019]

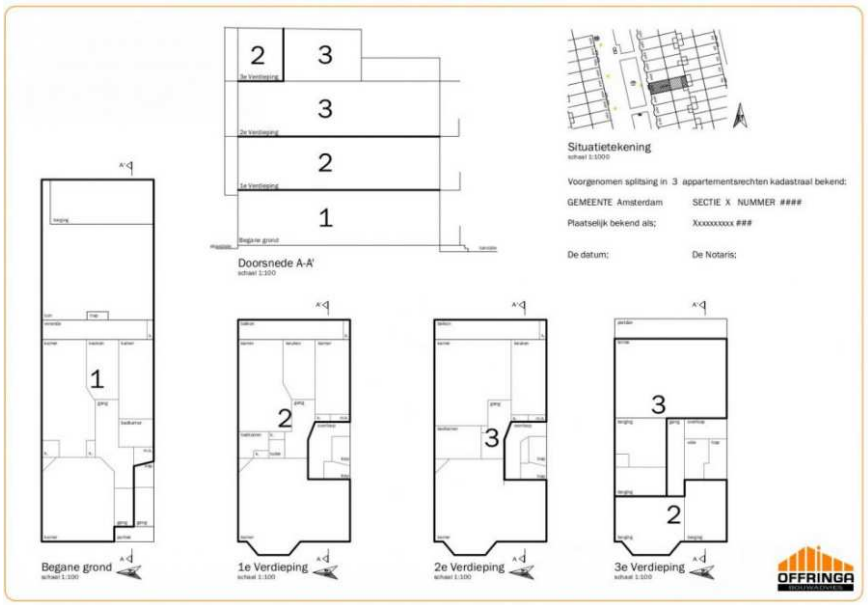


Figure 2.5.: 2D Floor plan example in notarial deed [Meulmeester, 2019]

2.2. Floor Plan

When registering land, the boundaries of a parcel must be clearly drawn to represent the exact division between it and neighboring properties. In the case of apartment buildings, the boundaries become more complex, as they must be represented not only on horizontal planes but also on vertical planes. Additionally, apartment rights include not only private units but also common areas, which are shared spaces that all owners in the apartment building are entitled to use based on their respective Right, Restrictions, and/or Responsibilities (RRRs). Therefore, as stated in Articles 5 and 6 of the Implementation Regulation of the Land Registry Act 1994, a detailed drawing must be included when registering an apartment unit in a notarial deed. This drawing should depict the boundaries of the land, as well as a floor plan that clearly illustrates the division of private and common areas on both the ground floor and upper floors of the building [Koninkrijksrelaties]. Figure 2.5 illustrates that property boundaries are outlined with thick black lines [Meulmeester, 2019], which are more prominent compared to normal walls. Since floor plans are a requirement for obtaining a building permit, municipalities also maintain floor plan datasets for buildings within their respective cities, but without the thick black lines.

Various studies incorporate floor plans as input to append indoor structures to 3D building models. By integrating 2D floor plans with 3D BAG data, Kippers et al. [2021] automatically reconstructed 3D building models, including their interiors, through two main processes. First, the floor plan was transformed into vectors with semantic attributes, where walls, openings, and room labels were classified separately using deep learning methods: U-Net, SSD with MobileNet v1 FPN, and RetinaNet50. The training dataset, sourced from CubiCasa, consisted of 5,000 floor plan images and annotated vector files. Since the floor plans lacked orientation, to merge the 3D BAG and floor plan, the 3D BAG was rotated based on the nearest road to align with the floor plan. For each facade adjacent to a road, alignment was optimized by maximizing polygon overlap using trust region methods. The

study achieved satisfactory results, with an accuracy of 0.7673, but due to the homogeneous training data, it may face challenges with more complex floor plans.

a. Parsing Floor Plan

Yin et al. [2009] describes that to decipher layout information, parsing the floor plan is required, which involves four steps: (1) Noise removal: A scanned image often contains noise and irrelevant details that need to be removed through image cleaning in order to enhance the quality of graphics recognition; (2) Text extraction: The system identifies and separates text from other graphical elements to facilitate further analysis; (3) Vectorization: To transform image pixels to the geometric primitive traditionally includes two steps. First, the raster bitmap is converted to a set of pixel chains with algorithms like parametric model fitting (HT), contouring, and skeletonization. After that, by implementing polygonal approximation or estimating curvature to determine key point segments, point chains can be segmented into sets of lines, polylines, and circular arcs; (4) Symbol recognition: After vectorization, it identifies and organizes architectural symbols or elements by using predefined constraints, thereby creating a structured representation of the building layout.

Thus, the floor plan would be preprocessed first, including cleaning the scanned image file, increasing the quality, and adjusting its scale and orientation for easier further processing. After georeferencing the raster file using QGIS, the information must be extracted for vectorization. Nottrot et al. [2023] utilized OpenCV to generate building outlines for each floor by identifying shape contours and drawing a convex hull from floor plans that contain multiple floors on a single page. The corresponding floor can be identified from the text using ACV. The resulting outline is then compared to the BAG polygons to match their scale and orientation, ensuring consistency with real-world representations. Another method can also be applied:

- FloorplantoBlender by Grebstew [2021] employed ORB (Oriented FAST and Rotated BRIEF) and image processing techniques to analyze and extract features from rasterized floor plans, see Figure 2.6. First, it prepares the input image by denoising, grayscaling, and rescaling it. It applied a threshold from the OpenCV library to the grayscale image to detect the wall, converting it into a binary format where the wall can be distinguished from the rest based on pixel intensity. The binary image is refined with morphological operations like dilation and erosion to remove small noise and close gaps in the walls. The walls are then extracted using a contour approach to retain only the most significant structures. By integrating contour closing and connected component labeling, rooms are identified as connected components enclosed by the previously detected walls, which are then refined by removing small gaps or noise. ORB is implemented to determine doors and windows by detecting key points and matching features between a template image and the floor plan. The detected features are transformed into a 3D representation that can be viewed in Blender.

- Liu et al. [2017] propose an algorithm using machine learning to convert a rasterized floor plan image into a vector-graphic representation, achieving around 90% precision. The method consists of three main steps as depicted in Figure 2.7. First, the CNN transforms the input floor plan image into a first set of junction maps and per-pixel semantic classification scores. Then, integer programming filters the candidates from junctions according to geometric and semantic constraints, producing a primitive layer. For example, it ensures that a bedroom is fully enclosed by walls that form a complete loop, with each wall correctly marked as belonging to the bedroom. Finally, the primitive layer is processed into the final vector format.

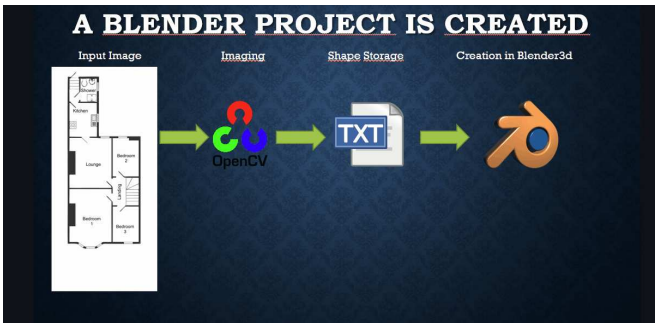


Figure 2.6.: FloorplantoBlender [Grebstew, 2021]

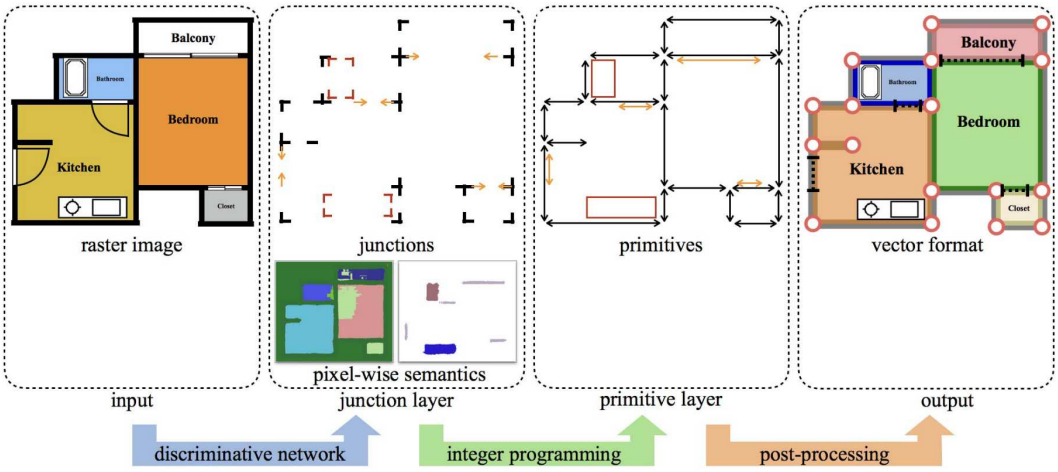


Figure 2.7.: Raster to Vector [Liu et al., 2017]

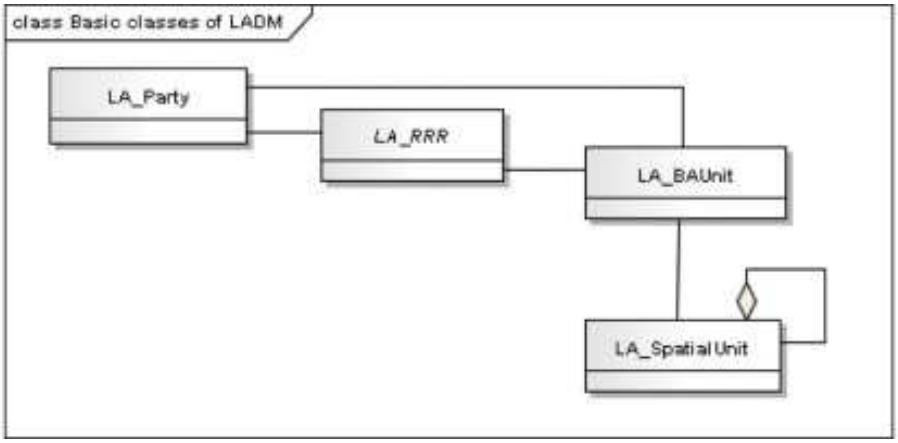


Figure 2.8.: LADM Basic Class [Lemmen et al., 2015]

2.3. LADM and IFC

LADM is a conceptual information model to support a land administration system initiated by the International Federation of Surveyors (FIG), documenting the relationship between people and land that concerns land tenure, land value, and land use plans as an International Standard (ISO 19152:2012). LADM Edition II integrates land value, land use, and land development into the first edition of LADM, which only contains a land tenure component for Land Administration. There are 6 parts in the second edition: Part 1 - Generic conceptual model, Part 2 - Land registration, Part 3 - Marine georegulation, Part 4 - Valuation information, Part 5 - Spatial plan information, Part 6 - Implementation aspects. Development of Part 6 is planned by the Open Geospatial Consortium (OGC). It accommodates the 3D Land Administration by referring to the boundary of the BIM/IFC model to ExtPhysicalBuildingUnit in the General Boundary Spatial Unit profile, preventing a mismatch between spatial unit representation in 2D and 3D. It efficiently supports the title and deed registration system, presenting the restrictions and responsibilities as rights, relationship ownership, and beneficiary entities. With the Valuation Information package, the correlation between land value and 3D aspects, such as a high floor with a better view and other factors like noise and routing influence, can be effectively modeled. Furthermore, the Spatial plan information packages represent RRRs derived from spatial planning or zoning plan, enabling the Spatial Development lifeCycle (SDC) that can maintain and store historical processes and information of an object, including financial, building/construction permits and occupancy from various databases [Kara et al., 2024]. As can be seen in Figure 2.8, basic classes of LADM are LA_Party, which specifies the property owner; LA_RRR, which outlines the ownership rights, restrictions and responsibilities; LA_BAU (Basic Administrative Unit), which represents the group of parcels with the same owner or party; and LA_SpatialUnit, which defines the spatial unit with its geometric boundary of the parcel.

To design 3D Land Administration system in the Netherlands, Meulmeester [2019] enriches IFC BIM model with legal space for apartment rights based on 2D floorplans. The study began by identifying a list of requirements for the 3D geometry representation of apartment rights derived from the Dutch Civil Code. It enriches the IFC BIM with legal information according to that requirement list. Ifcspace is a part of the spatial structure of IFC that represents a space or area within a building and can carry several attributes to

enhance the space with semantic information. The new user defines a property set named IfcPropertySet. After that, the information from the enriched IFC BIM with legal space is conceptually mapped to the LADM and then stored in an LADM-compliant database. The attributes from IfcPropertySet (apartment number and space type) are mapped to the LA_Right subclass that contains the type of (apartment) rights and LA_Basic Administrative Unit (BAUnit) class, which assigns the spaces associated with each apartment right number. The geometry and the id of ifcspace are mapped to the LA_LegalSpaceBuildingUnit class and also LA_BAUnit. During the storage process, the geometry of ifscpace is converted to polyhedral surfaces where the faces can be mapped to the LA_BoundaryFace class and become the 3D geometry representation for LA_LegalSpaceBuildingUnit. The ifcspace and its user-defined property set are extracted and merged as one entity to fill the LADM-compliant database's LA Right, LA BAUnit table, and LA LegalSpaceBuildingUnit. It is crucial to check if the stored data is valid semantically and geometrically. Finally, the 3D Land Administration system is visualized on the Cesium JS platform and QGIS. In QGIS, the system allows for splitting the apartment right by changing the apartment index number.

To associate point clouds and LADM, Koeva et al. [2019] stores 3D spatial units in GM_MultiSurface using Class LA_BoundaryFace. As GM_MultiSurface is not adequate for 3D spatial analysis and representation, enriched point clouds were employed as an external database for storing and representing 3D objects, enabling spatial attributes calculation for 3D Land Administration.

2.4. 3D LAS Visualization

Several essential components are identified by Kalogianni et al. [2020] to develop system architecture of a 3D Web-based LAS that integrates both spatial and non-spatial data: (1) datasets and datatypes availability; (2) data processing and validation; (3) data storage and management; (4) data visualization and manipulation. Van Oosterom [2013] specifies four challenge points to visualize 3D Land Administration as follows: (1) visualize dense 3D volumetric divisions where 3D spatial units often obscure each other which can be solved by using selections and wireframes or semi-transparent objects, displaying cross-sectional views or slices, or employing slide-out layers; (2) displaying open or unbounded parcels; (3) integrating the Earth's surface and reference objects like in CityGML for 3D Land Administration parcels; (4) presenting realistic depth perception for subsurface legal spaces associated with utilities by applying techniques such as stereoscopic imaging, perspective shifts, rotational views, or vertical markers connecting subsurface elements to the surface.

To ensure the 3D Land Administration visualization is able to address user preference when they are accessing the land administration information using a web client, Table 2.1 lists the requirements for 3D Land Administration and 3D web viewer identified by Cemellini et al. [2020], along with the other requirements from Mao et al. [2024] and new requirements proposed by the author. As both previous studies utilized IFC as their 3D model representation, a new requirement to facilitate the point cloud that will be used in this research, such as a Point cloud-based approach, needs to be considered as well.

Wish List	
For 3D Visualization of Land Administration Data	**For 3D Web Viewer**
Navigation tools and view controls	Platform and browser independence
Integrating topography and reference objects	Handling massive data and caching/tiling between server and client
Transparency	Layers control
Object selection	Database support
Object search	Support different models (vector/polyhedral, raster/voxel, point clouds)
Wireframe display	
Explode view	Support of basic 3D topographic visualization
Sliding	
Cross-section view	Support for georeferencing
Visualization cues	Ensure spatial validity (3D vector topology)
3D measurement tools	Underground view
3D buffer	Open source platform
Display partly unbounded objects and 'complex' geometries	Possibility for the platform to be extended
	2D overview map (orientation)
Party /RRRs visualization and selection	
Point cloud-based	

Table 2.1.: Wish List for 3D Visualization and Web Viewer

Various studies visualize 3D LAS from vector BIM using Cesium.JS. However, point cloud can also be transformed to Cesium 3D tile [Beil et al., 2021] to visualize 3D LAS from point cloud, enabling direct information of LADM with corresponding point cloud data. The overall method is illustrated in Figure 2.9.

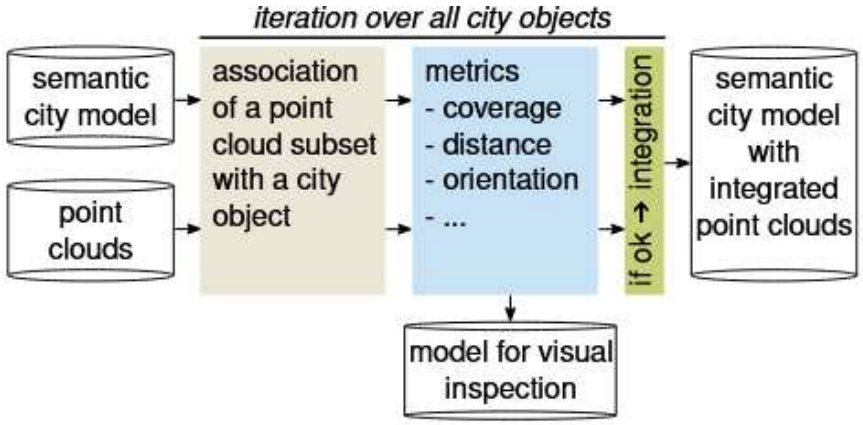


Figure 2.9.: Point Cloud and Semantic Models Integration [Beil et al., 2021]

Mao et al. [2024] utilized Cesium JS to create a novel 3D Land Administration website by linking LADM information to the BIM/IFC model represented as 3D spatial units of apartments through UML instance-level LADM diagrams to visualize the RRRs and parties, as can be seen in Figure 2.10. Any changes in information regarding parties and rights in the land administration database are also updated on the website, proposing a real-time

update capability. Some practical features are implemented as follows: (0) Switching on/off physical object and legal space; (1) By adopting 3DBAG, it displayed its surroundings with a determined Level of Detail (LoD) from the user; (2) floor visibility, allowing the user to examine each floor in more detail; (3) dynamic slicing view to help user inspect the internal structure and layout of building detail; (4) underground space view that can visualize the basements, garage or utilities as well; (5) sunlight simulations that are essential for real-estate marketing; (6) providing LADM information based on user interaction through owner and property inquiry.

Figure 2.10.: A digital twin based on Land Administration [Mao et al., 2024]

3. Methodology

This chapter outlines the methodology employed in this study, providing a step-by-step explanation of the entire process. The methodology consists of five main steps, each elaborated in detail within the respective sections: (1) Parsing the floor plan, (2) Segmenting AHN Point Cloud, (3) Generating Synthetic Point Cloud, (4) Storing Point Cloud to LADM, and (5) Visualizing 3D LAS. The technical implementation, including the algorithms, parameters, and code, is thoroughly described in the subsections. The complete pipeline of this research is illustrated in Figure 3.1.

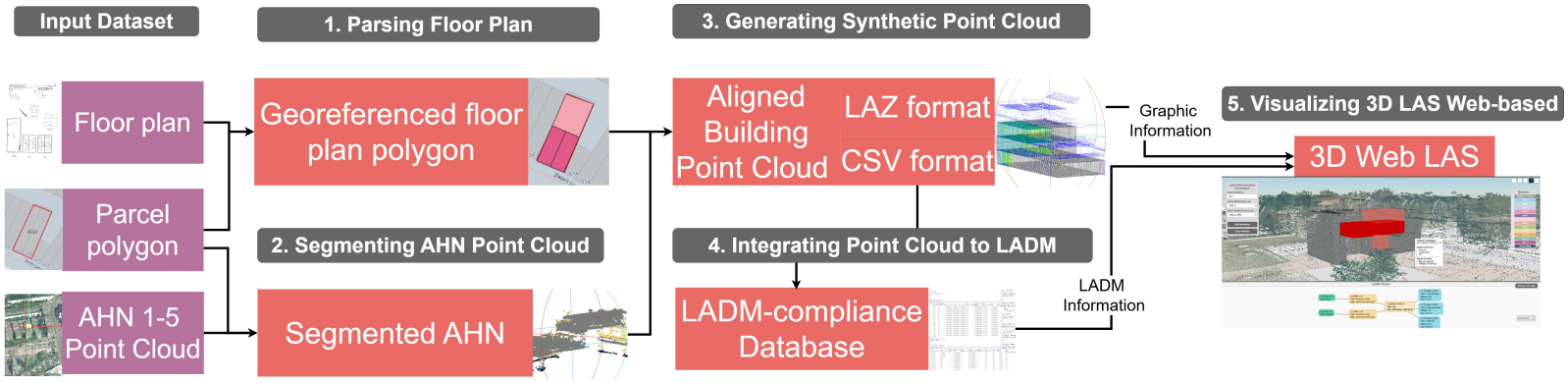


Figure 3.1.: The Overview of Methodology

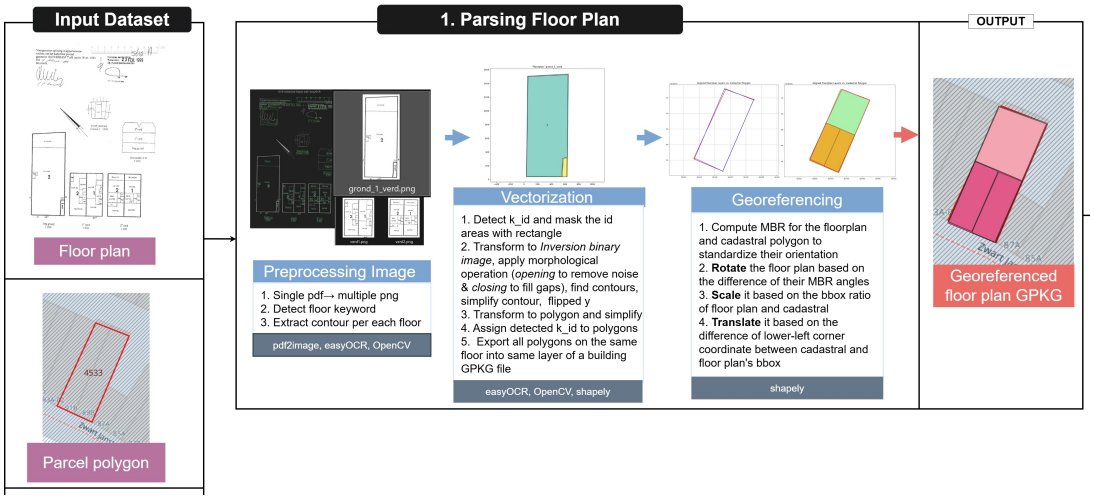


Figure 3.2.: Parsing the Floor plan procedure

3.1. Parsing the floor plan

Parsing the floor plan involves multiple steps with the floor plan and parcel polygon as the input, as illustrated in Figure 3.2. It starts with Preprocessing Cadastral Drawing PDF file, Vectorization, then Georeference to position the floor plan into the real world coordinates, same with the parcel polygon. The output of this process is the georeferenced floor plan polygon.

a. Preprocessing Cadastral Drawing PDF file

Parsing the floor plan starts by preprocessing a single PDF of the cadastral apartment that contains multiple floor plans into multiple PNGs for each floor using pdf2image with 300DPI and transforming to grayscale using an image processing python library, OpenCV. A resolution of 300 DPI is preferred as it offers an optimal balance between text readability and file size: higher resolutions, such as 600 DPI, do not improve text recognition, while 1200 DPI introduces unnecessary detail that reduces text clarity and hinders recognition. On the contrary, a lower resolution, such as 150 DPI, results in insufficient image quality, making it difficult to extract text information effectively. To locate the floor plan, Optical Character Recognition (OCR) is implemented to read keywords that are associated with ground or floor in Dutch, such as ”begane” and ”verdieping”. It isolates the individual floor plan block with GaussianBlur and Canny Edge detection to outline the floor plan frames or edges, and findContours to extract all the contours from the outlined edges. For each detected OCR text from before, it searched for all the closest identified contours. If a matching contour was found above the label, it crops the image using the bounding box of the contour and exports it into a PNG image. The entire procedure of preprocessing the floor plan is explained in Algorithm 3.1.

Algorithm 3.1: Preprocessing Floor plan

Input: `pdf_path` (architectural drawing), `output_dir` (where cropped images are saved)
Output: Labeled cropped floor plan PNGs saved per detected section

```
// Step 1: Convert PDF to high-resolution image
image ← convert first page of pdf_path to image at 300 DPI;
gray ← convert image to grayscale;
horizontal_threshold ← 30% of image width;
// Step 2: Run OCR to detect all text blocks
ocr_data ← pytesseract OCR output with bounding boxes;
// Step 3: Detect contours from drawing
blurred ← GaussianBlur(gray);
edges ← Canny(blurred);
contours ← findContours(edges);
// Step 4: Loop over OCR texts and find labels
processed_labels ← empty set;
for i ← 0 to length of ocr_data do
    full_text ← concatenate ocr_data[i:i+3] as lowercase;
    if full_text contains any of ["begane", "verdieping", "grond", "kelder", "verd"] then
        // Step 4a: Extract bounding box
        (x, y, w, h) ← bounding box of OCR label;
        label_text ← clean and format full_text (e.g., begane_grond);
        if label_text in processed_labels then
            continue;
        // Step 4b: Find closest contour above the label
        closest_contour ← None;
        min_distance ← ∞;
        foreach contour in contours do
            (x_cnt, y_cnt, w_cnt, h_cnt) ← boundingRect(contour);
            if contour lies directly above label then
                distance ← vertical gap between contour and label;
                if distance < min_distance then
                    closest_contour ← bounding box;
                    min_distance ← distance;
        // Step 4c: Crop and save based on closest contour
        if closest_contour exists then
            (x1, y1, x2, y2) ← bounding box ± margin, clipped to image bounds;
            cropped ← crop image within (x1, y1, x2, y2);
            add label_text to processed_labels;
        else
            fallback_crop ← large rectangle above label;
```

b. Vectorization

Algorithm 3.2 depicts the vectorization process. Initially, each PNG image must first be converted into grayscale. Using OpenCV, it applies thresholding to create an inverted binary

image with cv2.-THRESH_BINARY_INV, where pixel values greater than the threshold become 0 (black), while pixels with lower values (darker) become 255 (white). The threshold is determined automatically from the image histogram with cv.THRESH_OTSU. Followed by morphological operations with their corresponding kernels; morphological opening, remove white pixels near edges first (erosion) then add white pixels around edges (dilation) to remove noise, and morphological closing, add white pixels near edges first (dilation) then remove pixels around edges (erosion) to fill small gaps to ensure full contours of walls are formed. The kernel size affects the result, where a higher kernel size in the open kernel removes more and larger noise. In contrast, in the close kernel, it connects larger gaps, which also influences thicker lines to be generated instead of thinner ones. Thus, images with different resolutions may require different kernel sizes. Afterwards, it retrieves all contours with cv2.findContours, where it applies RETR_TREE to retrieve all the contours along their full hierarchy list and also cv2.CHAIN_APPROX_SIMPLE to ensure contour shape without redundant vertices. The tiny areas, less than 500 pixels, are removed. This minimum area, however, also relied on the resolution of input image. Before the remaining contours are converted into a polygon using Shapely, their Y coordinates are flipped to conform to the standard Cartesian coordinate system used in Geographical Information System (GIS) platforms and in mathematics, and any wiggles in the remaining contours are cleared through cv2.approxPolyDP with a certain epsilon. A higher epsilon results in simpler contours with fewer vertices, while a lower epsilon retains more detail but may introduce jagged or overly complex geometries. The polygons are simplified again with the simplify and buffer method from Shapely and rectangularized, ensuring the symmetric shape of the polygons and removing redundant vertices. The parameters used in these Shapely functions have a similar effect to the epsilon parameter in cv2.approxPolyDP, controlling the balance between simplification and shape fidelity. OCR reads the room numbers in the image and assigns the room number if it is inside the room polygon. All room polygons in the same image or floor are extracted into each layer of a geopackage file per building.

Algorithm 3.2: Floor plan Vectorization

Input: `input_dir` (folder with floor plan PNGs), `output_gpkg` (path to output GeoPackage)

foreach *file in* `input_dir` **do**
  gray ← convert image to grayscale;
  height ← image height;
  `// Step 1: OCR Digit Detection using EasyOCR`
  ocr_results ← detect digits in gray using EasyOCR;
  ocr_points ← [ ], ocr_texts ← [ ], ocr_masks ← [ ];
  foreach *(bbox, text, confidence) in ocr_results* **do**
    if *text is a single digit* **then**
      add center point of bbox to ocr_points;
      add text to ocr_texts;
      add bbox to ocr_masks;
  `// Step 2: Mask OCR-labeled regions`
  foreach *bbox in ocr_masks* **do**
    fill bbox region in gray with white (255);
  `// Step 3: Preprocess for Room Detection`
  thresh ← threshold(gray, 0, 255, inverse binary, cv2.THRESH$_O TSU$)
  opened ← morphological open(thresh, kernel=2×2);
  closed ← morphological close(opened, kernel=5×5);
  `// Step 4: Contour Detection`
  contours ← find external + internal contours from closed;
  room_polygons ← [ ], room_ids ← [ ];
  foreach *contour in contours* **do**
    if *area of contour* $<$ 500 **then**
      continue;
    simplified ← approximate polygon from contour;
    flipped ← flip Y-axis of simplified;
    poly ← Polygon(flipped);
    if *poly is invalid* **then**
      continue;
    simplified_poly ← simplify polygon with tolerance;
    cleaned_poly ← simplify using buffer(5).buffer(-5);
    rectified_poly ← apply rectangularization to cleaned_poly;
    if *rectified_poly is invalid or empty* **then**
      continue;
    `// Step 5: Assign Room ID from OCR`
    assigned_id ← None;
    foreach *(pt, txt) in (ocr_points, ocr_texts)* **do**
      flipped_pt ← flip Y-axis of pt;
      if *flipped_pt inside rectified_poly* **then**
        assigned_id ← txt;
        break;
    add rectified_poly to room_polygons;
    add assigned_id to room_ids;
  `// Step 6: Save to GPKG`
  gdf ← GeoDataFrame(room_polygons, attributes = room_ids);
  save gdf to `output_gpkg` with layer name = image filename;

c. Georeference

The vectorized polygons are georeferenced based on the parcel polygon downloaded from PDOK.nl. Coordinate transformation typically involves three fundamental steps: scaling, rotation, and translation [Wolf et al., 2014]. As outlined in Algorithm 3.3, it is initialized by matching the Coordinate Reference System (CRS), then computing the orientation of the floor plan and parcel polygon using their respective Minimum Bounding Rectangle (MBR). The computation involves creating a convex hull to simplify geometry, where each pair of consecutive vertices is extracted from its exterior coordinates to form an edge. For each edge, its x and y differences (dx,dy) between the pair of vertices and their arc-tangent are calculated to determine its angle or direction towards the horizontal line (x-axis). The original polygon is virtually rotated by the negative of that angle to align that particular edge horizontally. After each rotation, the area of the bounding box is calculated, which varies depending on the angle of rotation. The angle with the smallest bounding box area is chosen, representing the most compact and efficient rectangular representation of the shape. The difference between the MBR angle of the cadastral and the floor plan reveals how much the floor plan must be rotated to align with the cadastral. After the floor plan is rotated at this angle, the bounding box of the rotated floor plan and the parcel polygon are recalculated to measure the x and y scale factor by comparing their width and height extent. Following this, translation offsets are computed by aligning the lower-left corners of the latest bounding boxes to shift the floor plan exactly at the cadastral footprint.

Algorithm 3.3: Georeferencing Floor plan Layers Using MBR Alignment

Input: Input GeoPackage `input_gpkg`, Output GeoPackage `output_gpkg`, Cadastral Geometry `cadastral_gdf`
Output: All layers are transformed to align with the cadastral reference

layers ← list of layers in `input_gpkg`;
$floorplan\_gdf$ ← load first layer from `input_gpkg`;
$floorplan\_merged$ ← unary union of $floorplan\_gdf$;
cadastral_polygon ← first geometry in $cadastral\_gdf$;
Function `GetMBRAngle`(*polygon*)**:**
    hull ← convex hull of *polygon*;
    coords ← boundary coordinates of *hull*;
    min_area ← ∞, *best_angle* ← 0;
    for $i \leftarrow 1$ **to** $|coords| - 1$ **do**
        Compute edge angle θ from $coords[i]$ and $coords[i+1]$;
        Rotate polygon by $-\theta$;
        Compute bounding box area;
        if *area* < *min_area* **then**
            min_area ← area;
            best_angle ← θ;
    return *best_angle*;
`// Step 1: Compute MBR for Floorplan and parcel polygon`
$floorplan\_angle$ ← `GetMBRAngle`($floorplan\_merged$);
cadastral_angle ← `GetMBRAngle`(*cadastral_polygon*);
`// Step 2: Rotation angle`
rotation_angle ← *cadastral_angle* − $floorplan\_angle$;
`// Step 3: Compute Scale factor`
$scale_x$ ← width of *cadastral_bounds* ÷ width of $floorplan\_bounds$;
$scale_y$ ← height of *cadastral_bounds* ÷ height of $floorplan\_bounds$;
`// Step 4: Compute Translation factor`
$translation_x$ ← *cadastral_bounds.minx* − $floorplan\_bounds.minx \times scale_x$;
$translation_y$ ← *cadastral_bounds.miny* − $floorplan\_bounds.miny \times scale_y$;
$flip\_center$ ← center of bounding box of $floorplan\_gdf$;
centroid ← centroid of $floorplan\_merged$;
Function `TransformGeometry`(*geom, apply_flip*)**:**
    Rotate *geom* by *rotation_angle* around *centroid*;
    if $apply\_flip$ **then**
        Flip *geom* around $flip\_center$;
    Scale *geom* using $scale_x$, $scale_y$;
    Translate *geom* using $translation_x$, $translation_y$;
    return transformed geometry;
`// Step 5: Apply Transformation parameters`
foreach *layer_name in layers* **do**
    gdf ← load layer from `input_gpkg`;
    Transform gdf to match `cadastral_gdf.crs`;
    foreach *geometry in* gdf **do**
        geometry ← `TransformGeometry`(*geometry, True*);
    Save gdf to `output_gpkg` under *layer_name*;

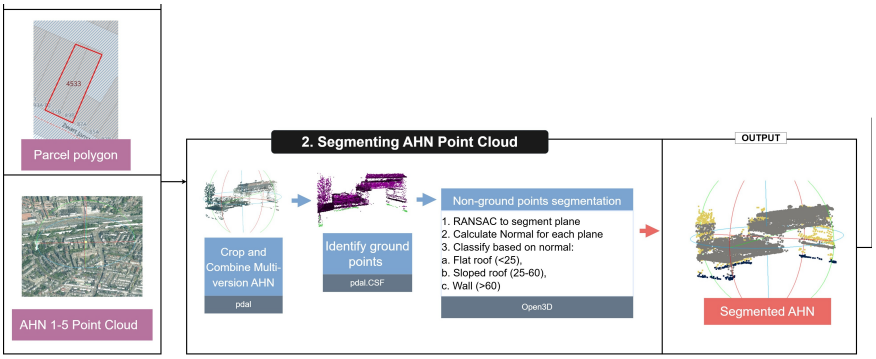
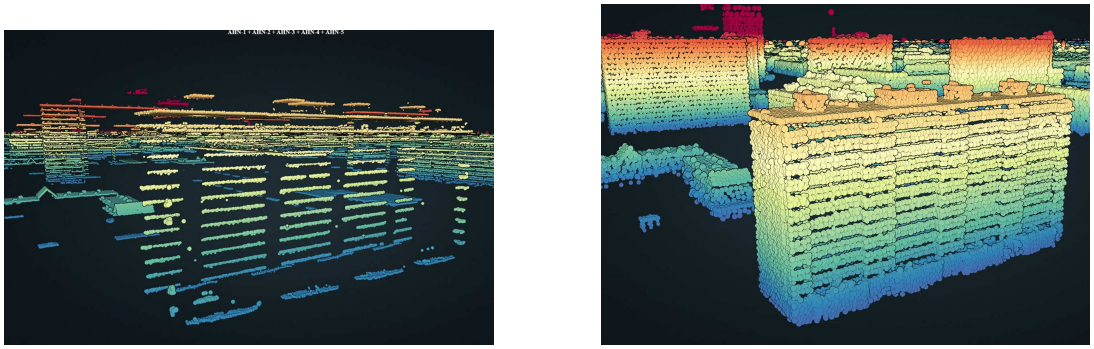


Figure 3.3.: Segmenting AHN Point Cloud

3.2. Segmenting AHN Point Cloud

To segment the AHN point cloud, multiple version of AHN dataset along the parcel polygon is required. Figure 3.3 illustrates that AHN Multi-version Combination is performed first, followed by Ground Classification to differentiate ground and non-ground points, then Point Cloud Segmentation to categorize wall and roof points.

a. AHN Multi-version Combination



(a) AHN5 (b) AHN 1–5

Figure 3.4.: Comparison of AHN datasets: (a) AHN5 and (b) AHN 1–5

Combining multiple versions of AHN can overcome the occlusion of the latest version of AHN data as demonstrated in Figure 3.4. In this example, missing sections of a building in one version are completed by corresponding parts from another version. This improvement is due to the variations in the flight path during AHN laser scanning, resulting in different parts of the building being scanned. Therefore, the AHN from multiple versions, from AHN 1 to AHN 5, would be combined together and cropped with 1-meter buffer towards the building footprint obtained from pdok.nl to retrieve the building points. A segmentation process should be implemented to distinguish outside walls and roofs of building points.

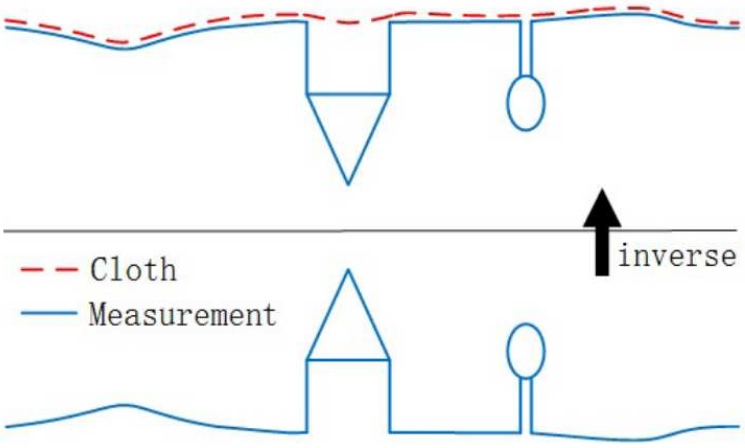


Figure 3.5.: CSF Illustration to classify ground points [Zhang et al., 2016]

b. Ground Classification using Cloth Simulation Filter (CSF)

Ground points are filtered using the CSF through filters.csf feature in Point Data Abstraction Library (PDAL). CSF is based on simulating a simple physical process to extract ground points from LiDAR points. It inverted the original point cloud, and then a rigid grid called cloth was dropped onto the inverted surface from above, as illustrated in Figure 3.5. The interactions between the nodes of the cloth and the corresponding point clouds can determine the final shape of the cloth to distinguish point clouds into ground and non-ground points. There are user-defined parameters: resolution, which represents grid resolution or cell size; step or time step, which adjusts the translation of points due to gravity during each iteration; and rigidness, which determines the stiffness of the cloth, where a higher value is preferred for flat terrain while a lower value is suggested for steep slopes [Zhang et al., 2016].

c. Point Cloud Segmentation

Segmentation is grouping several homogeneous points based on their common features. RANdom SAmple Consensus (RANSAC) is a model-fitting method that uses a mathematical representation. It defines a model parameter from a minimum sample of random points. Iteratively checking their neighboring points, then a consensus set is formed if they are a match. The overall algorithm can be seen in Figure 3.6. The non-ground points identified by the CSF filter are iteratively segmented into individual planar patches using RANSAC, implemented via the segment_plane function from Open3D, with a distance threshold of 0.3 meters and a minimum of 3 points. Normal for each point in the plane is estimated through Open3D's estimate_normal function and converted as a numpy array with Numpy's asarray to compute the angle between the normal vector and the vertical Z-axis using the inverse cosine (arccos). By calculating the angle of arccos degree, the points can be classified into Flat Roof if the angle of normal is below 25°, Sloped Roof if the angle is between 25 and 60°, and Wall if the normal is more than 60 °.

Algorithm 7: The RANSAC algorithm

Input: An input point cloud P, the error threshold ϵ, the minimal number of points needed to uniquely construct the shape of interest n, and the number of iterations k

Output: the detected shape instance $\mathcal{I}_{best}$

```
s_best ← 0;
I_best ← nil;
for i ← 0 ... k do
    M ← n randomly selected points from P;
    I ← shape instance constructed from M;
    C ← ∅ ;
    for all p ∈ P \ M do
        d ← distance(p, I);
        if d < ε then
            add p to C;
    s ← score(C);
    if s > s_best then
        s_best ← s ;
        I_best ← I;
```

Figure 3.6.: RANSAC Algorithm [Ledoux et al., 2023]

3.3. Generating Synthetic Point Cloud

Figure 3.7 shows that to generate a synthetic point cloud, the outputs from the previous steps are required, the georeferenced floorplan and segmented AHN. The synthetic point cloud is then aligned with Iterative Closest Point (ICP) before it is combined with the AHN. This produces an aligned building point cloud in two different formats, LAZ and CSV.

a. Generate Synthetic Point Cloud

To construct a synthetic building point cloud, it initially load a floor plan layer from a Geopackage input file and iterates through each room polygon, searching the boundary using polygon.exterior and calculate its boundary to estimate how many points are needed to be created based on the user-defined point density, where 20 points per square meter as point density means 0.05 m spacing between points. Following that, it iterates over the number of points and, by using linestring.interpolate on the boundary line, it returns a point with 0.05 spacing, generating points with even spacing on the boundary line of every room polygon. Each point is then iterated and numpy.arrange(start, stop, step) is implemented to generate points along the z-axis from the floor to the ceiling with the height of the floor as start, the height of the ceiling plus the spacing as stop, and the spacing as step. The 3D coordinates along their attributes are stored as well inside this loop, creating the wall points that are generated vertically along the boundaries of the polygon from the floor to the ceiling. The height for the ground floor is calculated based on the ground point from AHN, while the height for the ceiling is computed based on the average height of the roof point in AHN divided by the total number of floors above the ground.

The bounding box for every room polygon is computed to generate a grid of points within

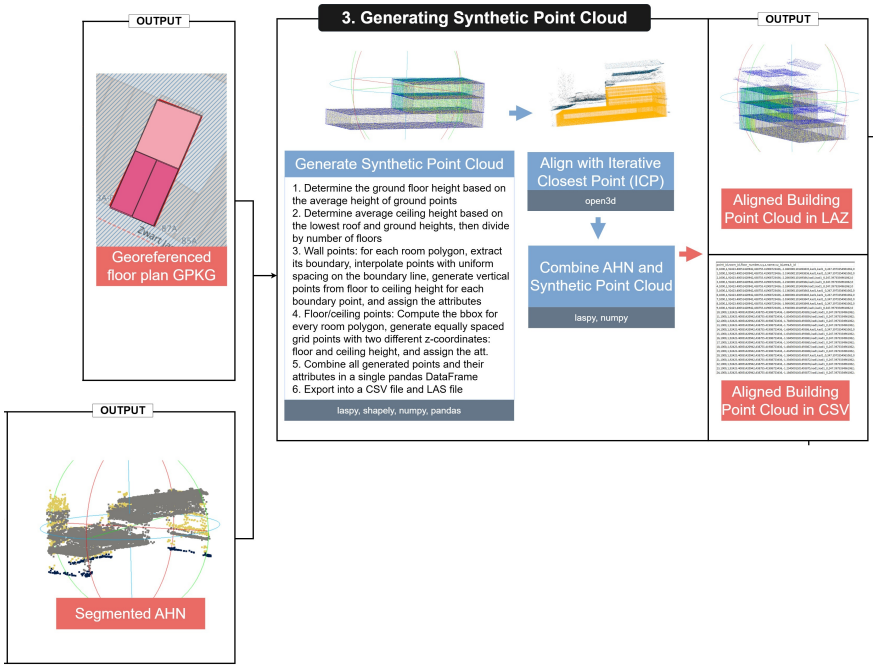


Figure 3.7.: Generating Synthetic Point Cloud

the interior of the polygons, creating a floor and a ceiling with different z coordinates. After it iterates through all floor layers, all the points are combined into a new pandas.DataFrame along their attributes, then exported into a CSV and LAZ 1.4 point cloud file with the same CRS as the AHN file. Each point is assigned a unique ID and attributes, namely point_id, room_id, floor_number, x, y, z, name, k_id, and su_id as summarized in Table 3.1. The room_id is generated by multiplying the floor number by 1000 and adding the polygon index, while the su_id is the concatenation of name, room_id cadastral number, where name means the building name. The value of floor_number is start from 1 for groundfloor while -1 for underground without 0 value to avoid leading zero that is not allowed in integers or float. Since classification only has a range from 0-255, k_id is used as classification for the LAZ file. The entire process of this step is outlined in Algorithm 3.4.

Attribute	Description
point_id	unique id to identify point
room_id	unique id to identify room
floor_number	number of floor
k_id	id in cadastral drawing
su_id	unique id to identify spatial unit
x	longitude
y	latitude
z	height

Table 3.1.: Attribute for point cloud semantic

Algorithm 3.4: Synthesis Point Cloud from Floor plan Reconstruction

Input: `floor_shapefiles` list of (gpkg_path, layer_name, floor_number)
`ground_points` AHN classified ground points
`planes_info` RANSAC-derived planes (roof, wall)

```
// Step 1: Estimate average ground height
ground_points_df ← DataFrame of ground points [x, y, z];
global_ground_height ← 5th percentile of ground_points[:, 2];
// Step 2: Select valid roof height above walls
max_wall_top_z ← highest max_z from wall planes;
highest_valid_roof_min_z ← min_z of highest roof plane above walls;
if none found then
    fallback to mean height of all roof planes;
lowest_roof_height ← selected min_z of highest valid roof plane;
n_floors ← number of above-ground floor shapefiles;
calculated_floor_height ← (lowest_roof_height − global_ground_height) / n_floors;
// Step 3: Generate floor point clouds
foreach (gpkg_path, layer_name, floor_number) in floor_shapefiles do
    buildings ← load layer from GPKG;
    foreach polygon in buildings do
        room_id ← floor_number × 1000± index;
        base_height
          ← global_ground_height + (floor_number − 1) × calculated_floor_height;
        floor_z ← base_height;
        ceiling_z ← base_height + calculated_floor_height;
        // 3.1: Generate wall points along polygon boundary
        foreach ring in exterior and interior rings do
            interpolate points along perimeter;
            foreach height z from floor_z to ceiling_z do
                record (x, y, z) with room_id, floor_number, attributes;
        // 3.2: Generate floor points inside polygon
        foreach (x, y) in bounding grid with spacing do
            if (x, y) inside polygon then
                record (x, y, floor_z);
        // 3.3: Generate ceiling points
        foreach (x, y) in same grid do
            if (x, y) inside polygon then
                record (x, y, ceiling_z);
// Step 4: Concatenate attributes for all floors
attributes_df ← concat of all per-floor DataFrames;
fill missing k_id with 0;
// Step 5: Finalize and export CSV
attributes_df ← select columns [point_id, room_id, floor_number, x, y, z, name,
  su_id, area, k_id];
save attributes_df to csv_output;
// Step 6: Export LAS
las ← create LAS file (version 1.4, format 6);
las.x, las.y, las.z ← combined x, y, z arrays;
las.classification ← k_id;
ahn_crs ← read CRS from AHN header;
save LAS file ;
```

b. Iterative Closest Point (ICP)

After the synthetic point cloud is generated, it will be aligned to AHN using ICP. ICP is a popular spatial registration-based method to align two point cloud datasets. The algorithm operates over two main steps: first, it initially finds correspondences between the target point cloud and the source point cloud by finding the nearest neighbor in Euclidean space; second, given these correspondences, it iteratively estimates the optimal rigid transformation that best aligns the source to the target by minimizing cost function (the sum of squared distance between matched pairs) until convergence or the value is less than the threshold. This algorithm is known as point-to-point ICP with equation as follow:

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \left((\mathbf{p} - \mathbf{T}\mathbf{q}) \cdot \mathbf{n}_\mathbf{p}\right)^2 \tag{3.1}$$

The other ICP variant, point-to-plane ICP, uses the intersection of the normal point in both datasets to determine the corresponding points. To increase convergence speed, the cost function is improved by replacing point-to-point distances with point-to-plane distances, which minimizes the distance between the source point and the tangent plane of the corresponding target point [Wang and Zhao, 2017]. The formula for this method:

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \|\mathbf{p} - \mathbf{T}\mathbf{q}\|^2 \tag{3.2}$$

After the alignment, both point cloud, AHN and synthetic floor plan point cloud will be combined into one LAS file with the same header as the latter to preserve the generated attributes. Since AHN is the envelope of the building, the SU ID for AHN would be the same as the outer wall in synthetic points, which is the name plus 0, as there is no cadastral number.

3.4. Storing Point Cloud to LADM

LAZ is a compressed version of standardized format for point cloud data. However, it lacks support for concepts of semantic object structures like hierarchies or aggregation. To represent the point cloud as 3D LAS, the point cloud data needs to be integrated with structured knowledge and semantics by accommodating hierarchically structured and topologically connected representations of objects with multiple attributes [Poux, 2019]. Using PostgreSQL, as illustrated in Figure 3.8, the synthetic point clouds are stored in a Land Administration Domain Model (LADM) compliance database based on ISO 19152-2024, where five tables are created: Point cloud, LA_SpatialUnit, LA_BAUnit, LA_RRR, and LA_Party. The Unified Modeling Language (UML) model for the LADM is illustrated in Appendix B.

a. Point Cloud Table

The point cloud and its attributes are stored in PostgreSQL using the CSV output from the previous process with an import query as follows:

```
COPY synth_pc (point_id, room_id, floor_number, x, y, z, name, su_id, area, k_id)
FROM 'D:/TUDELFT/Thesis/Thesisprep/New folder/kad1.csv'
DELIMITER ',' CSV HEADER;
```

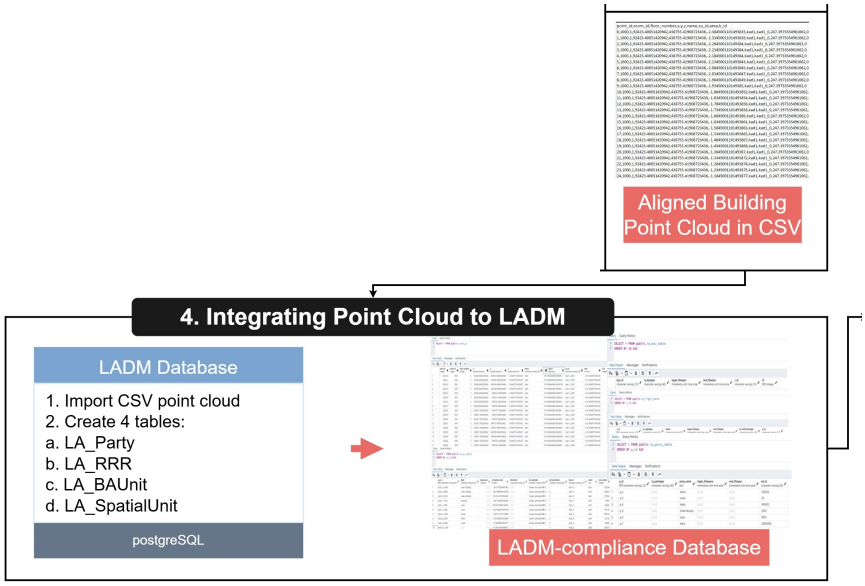


Figure 3.8.: Storing Point Cloud to LADM

The geometry for each point is created using *ST_MakePoint* based on the coordinate columns originally in the Dutch national coordinate system EPSG:28992. Since Cesium uses WGS84, *ST_Transform* is applied to transform the coordinate system from EPSG:28992 into EPSG:4326.

```
SET geom = ST_Transform(ST_SetSRID(ST_MakePoint(x, y, z), 28992),4326)
```

b. LA_SpatialUnit Table

Spatial unit is a general term for the land parcel that can be tied to formal, informal, or customary rights. LA_SpatialUnit can belong to zero or more LA_BAUnit. This means that one may have a land registry without any mapped spatial unit, as not all land is formally surveyed, and one may have multiple spatial units at one BAUnit, which usually happens in an apartment unit that contains a living unit, a parking unit, and a storage room. One parcel may also be mapped but not legally owned, or have multiple legal interests [Lemmen et al., 2025]. Spatial unit contains columns as listed in Table 3.2.

```
INSERT INTO la_su_table (su_id, computed_area, bau_id)
SELECT name || '_' || k_id || '_' || room_id AS su_id,
    area AS computed_area, name || '_' || k_id AS bau_id
FROM synth_pc GROUP BY name, k_id, room_id, area;
```

c. LA_BAUnit Table

BAUnit is created to group all spatial units under the same rights. Each LA_RRR can only link to exactly one LA_BAU, while LA_BAU must have at least one LA_RRR. This means that one land registry requires one BAUnit to be registered under that right, ensuring BAUnit is the legal container for rights. If there are multiple parcels that are owned together, then

	Definition
su_id	spatial unit identifier
label	short description of the spatial unit
legal_area	area written in the legal document
computed_area	area computed after conversion
ext_addressid	external address of the spatial unit
surfacerelation	indicates underground or above ground
begin_lifespan	start date of spatial unit
end_lifespan	end date of spatial unit
bau_id	to link the SU table with the BAU table based on their id

Table 3.2.: Attributes in LA_SpatialUnit class

either combine them into a single BAUnit that includes multiple spatial units, like the apartment unit case, or if the condition is not possible, they may have separate RRRs that will be linked to different BAUnits [Lemmen et al., 2025]. This class has attributes as listed in Table 3.3.

	Definition
bau_id	BAUnit identifier
la_bautype	type of bau: apartment / condominium unit or a land consolidation area
begin_lifespan	start date of BAUnit
end_lifespan	end date of BAUnit
r_id	to link BAU table with Right table based on their id
id	primary key for bau and right row

Table 3.3.: Attributes in LA_BAUnit class

d. LA_Right Table

RRRs consist of rights, restrictions, and responsibilities. This study only focuses on the right class as it can be interpreted as the general RRRs in most countries. One right can link to zero or one party, while LA_Party can have zero or more rights. One without any right means the person is involved in the transactions without property rights. A right without a party means the right is purely related to the land, such as servitude. If rights are linked to several parties, the sum of shares in a right must equal one. The share field is filled when one right is shared for multiple parties [Lemmen et al., 2025]. Attributes stored in this table are listed in Table 3.4

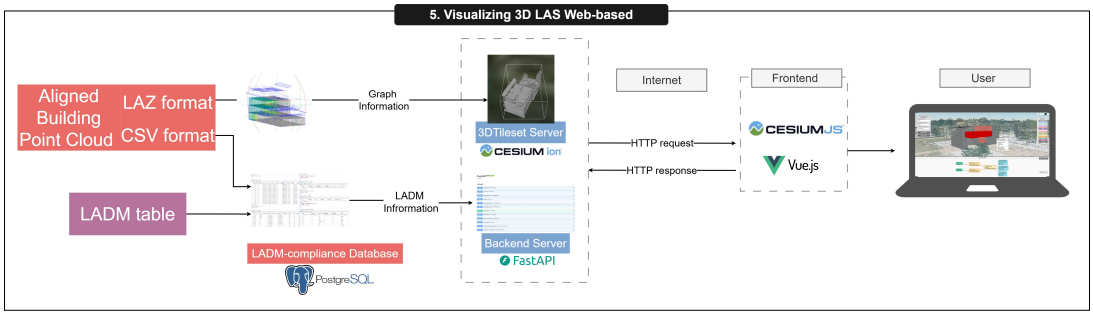


Figure 3.9.: Generating Synthetic Point Cloud

	Definition
r_id	right identifier
la_righttype	right type: ownership, lease, occupation, usufruct, tenancy
share	a share in an instance of a subclass of a LA_RRR
begin_lifespan	start date of the right
end_lifespan	end date of the right
p_id	to link Right table with Party table

Table 3.4.: Attributes in LA_Right class

e. LA_Party Table

A party means a person or organisation that is involved in any land administration process. As a party can also refer to any stakeholder, such as a surveyor, notary, or registrar, in this study, the role of party only focuses on the holder of a right on a land parcel. This class contains columns as listed in Table 3.5.

	Definition
p_id	party identifier
la_partytype	type of party
party_name	name of the person or institution
begin_lifespan	start date of party
end_lifespan	end date of party
ext_id	national/external identifier of party

Table 3.5.: Attributes in LA_Party class

3.5. Visualizing 3D LAS

A land administration dissemination system is developed with Vue.js, Cesium, and FastAPI to represent and visualize 3D spatial units, allowing the user access and modify land administration information, including owner, right, and spatial unit. Initially, the resulting point cloud datasets from the previous step, Generating Synthetic Point Cloud, are uploaded to Cesium.ion to render the point cloud tileset from Cesium server. The web application front-

end is built using the Vue.js framework based on Hypertext Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript to manage user interaction. It integrates with CesiumJS to enable 3D geospatial visualization for the web. The front-end and PostgreSQL database server are connected through a RESTful Application Programming Interface (API) using FastAPI, a web framework for developing HTTP-based service APIs in Python. It manages how users retrieve and update data from the database server. The code for Vue.js or a front-end application is mostly written in a main file called App.vue with Options API style, while the back-end application is in a Python file named pgquery.py. The summary of this step is depicted in Figure 3.9.

a. LADM Information Retrieval

To access the LADM, there are three steps required (Address → BAUnit → Spatial Unit). First, the user needs to type the address and then select the name/address of the building from the options. The v-for directive will generate option elements within the select dropdown from addressOptions that fetches a list of addresses from the database server. A two-way data binding directive, selectedAddress, is stated as in v-model, and two event listeners are: onAddressSelected, and fetchBauIds. onAddress is defined in the Vue method to load the Cesium Ion asset ID based on the selected name (stored in selectedAddress) via Cesium-3DTileset. The corresponding building will be loaded into the scene, and the camera will zoom into the building. The fetchBauIds function fetches a list of bau_id from the database server via the defined localhost in the fast API backend module and stores it as bauOptions, with query :

```
SELECT DISTINCT bau_id FROM la_su_table WHERE name = %s ORDER BY bau_id
```

Similar to onAddress, the bauOptions stated in v-for in the Select BAUnit dropdown will generate an array bau_id as options, the user can see how many BAUnits are available in the building, and select one of them. The fetchSuIds event listener triggers the FAST API to fetch a list of su_id for that BAUnit (selectedBauId variable in v-model directive) from the database server and stores it as suOptions through a basic query:

```
SELECT DISTINCT su_id FROM la_su_table WHERE name = %s and bau_id = %s ORDER BY su_id
```

Following that, the user can select one of the spatial units (from suOptions), and the loadLinkedData method is triggered to fetch detailed information for the specified spatial unit (selectedSuId). The web will zoom and highlight all the points of the selectedSuId and show the information graph, which will be explained in the following subsection. The sequence diagram of entire mechanism is illustrated in Figure 3.10.

b. Tooltip or Click Point Attribute

Although retrieving object attributes in Cesium with a point cloud is more challenging, as per-point data is stored only on the GPU. To accommodate the tooltips features, the pickPosition function from Cesium and the backend database server are used together. First, as the user clicks on a point in the unit, it triggers the onMouseClick() method in the Vue component that calls scene.pickPosition() function in Cesium to retrieve the clicked point's Cartesian 3D coordinate. It then converts the coordinates into geographic coordinates and fetches data from the backend endpoint, `/nearest-attribute`. It searches the nearest point from the point cloud table with the following SQL query:

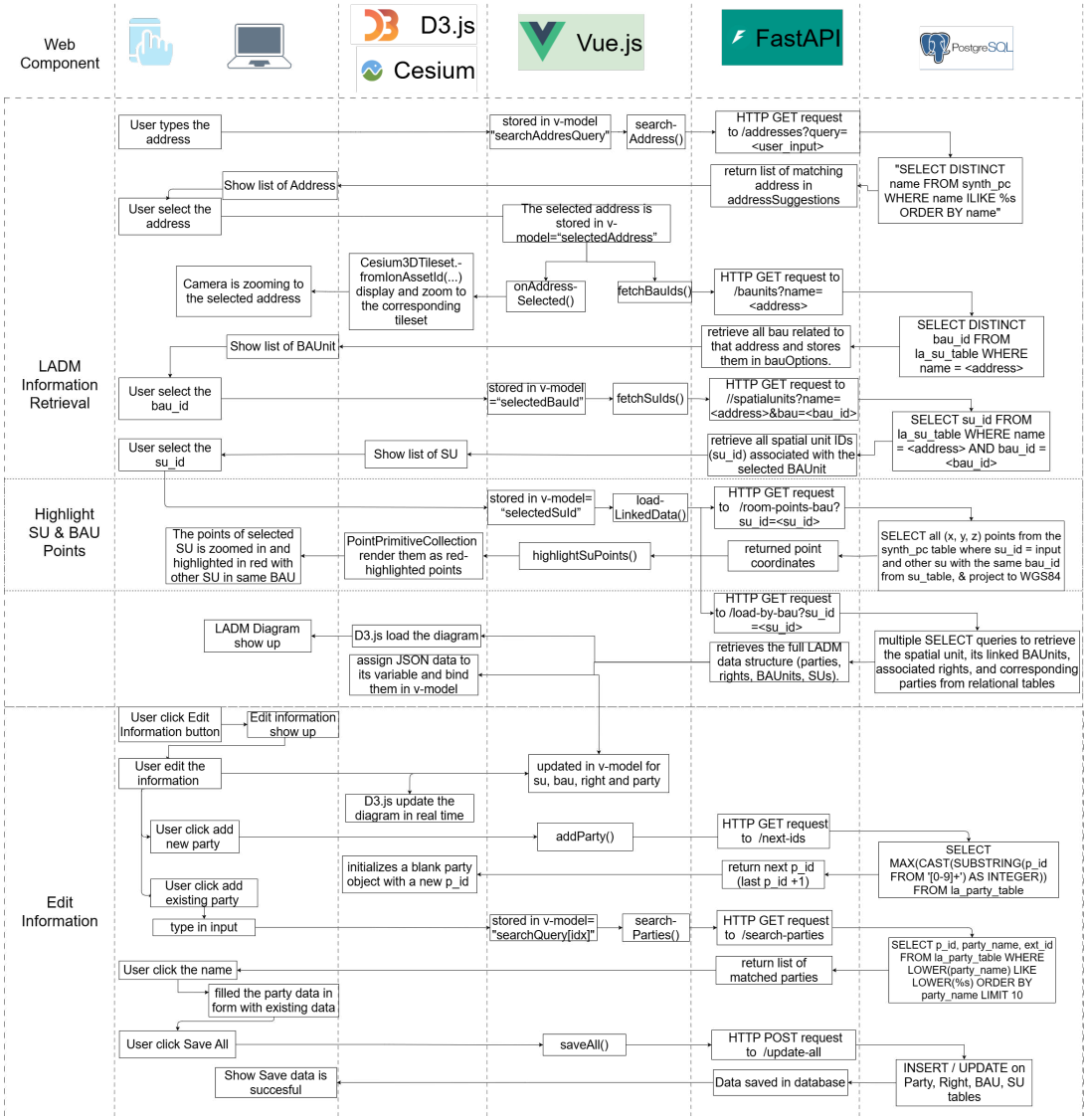


Figure 3.10.: Sequence Diagram of LADM Information Retrieval, Highlight SU and BAUnit Points and Edit information

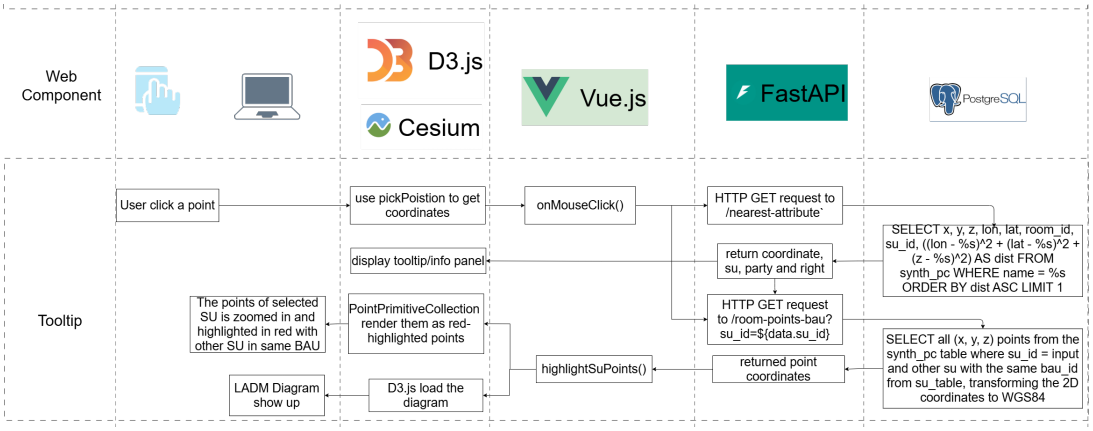


Figure 3.11.: Sequence Diagram of Tooltip

```
SELECT x, y, z, lon, lat, room_id, su_id FROM (
SELECT x, y, z, lon, lat, room_id, su_id, ((lon - %s)^2 + (lat - %s)^2 + (z - %s)^2)
AS dist FROM synth_pc WHERE name = %s
ORDER BY dist ASC LIMIT 1) AS nearest_point
```

As the matching point is found from the table, it calls another SQL query to gather the land administration information from the party, right, BAUnit, and spatial unit table. Consequently, the JSON response is sent to the front end to display the information in the tooltip panel. This also triggers the highlighting features, highlighting the spatial units and BAUnit where the user clicks a point and LADM instance level diagram that will be explained in the next subsections. The mechanism of this feature is demonstrated in Figure 3.11.

c. BAU Visualization Checkbox

To allow the user to interact with the spatial unit in the same BAUnit, the cadastral ID is stored as classification in the LAS files. Retrieving the available class from 3D Tiles in Cesium is challenging, due to the per-point data is stored only on the GPU when the tileset is loaded, not on the CPU. As the classification can only be parsed and applied during rendering with the tileset.style that works at the GPU level, the label for each classification is defined manually inside the data component in App.vue, along with their colours. After selecting the address/name of the building in main dropdown menu, all points will be colorized based on their BAUnit by default by applying thePointCloudClassificationColors () function. If one or more units are unchecked in the checkbox, the Vue's reactive binding will update the selectedClassification array where the unchecked units are removed from the list and in updateClassificationVisibility() function that stated inside @change check box as event handler, will update the classification colors based on the list while the unlisted units will be styled using Cesium3DTileStyle with gray color and one pixel or smaller size, as the cesium style does not allow styling points as entirely transparent.

d. Highlighting

Since storing semantics into a point cloud is challenging and the classification of the LAS file already refers to the bau_id, to allow the user to highlight the spatial unit within BAUnit, the

website retrieves all points of the selected spatial unit and presents them from PostgreSQL instead of 3Dtiles. The highlighting process is triggered when the user selects a spatial unit from the dropdown menu. This invokes the loadLinkedData function, defined in the Vue.js component. This function sends a request to the backend using the `/room-points-bau` API endpoint to retrieve the coordinates of all points belonging to the selected spatial unit.

```
SELECT ST_X(geom_wgs), ST_Y(geom_wgs), z
FROM ( SELECT ST_Transform(ST_SetSRID(ST_MakePoint(x, y), 28992), 4326)
AS geom_wgs, z FROM synth_pc WHERE su_id = %s ) AS transformed
```

Followed by another query to collect other spatial units within the same BAUnit.

```
SELECT ST_X(geom_wgs), ST_Y(geom_wgs), z
FROM (SELECT ST_Transform(ST_SetSRID(ST_MakePoint(x, y), 28992), 4326) AS geom_wgs, z
FROM synth_pc WHERE su_id IN ( SELECT su_id FROM la_su_table
WHERE bau_id = %s AND su_id != %s)) AS transformed
```

The result is returned as a JSON array of 3D coordinates and passed to the `highlightSuPoints` function on the client side. This function uses Cesium's `PointPrimitiveCollection` to render the points and applies a semi-transparent red color for visual emphasis. A vertical offset of 43.5 meters is added to the original z-coordinate to account for a discrepancy between the coordinate reference systems. This is due to Cesium reprojects all geometry into EPSG:4326 and measures height relative to the WGS84 ellipsoid, while the source point cloud in EPSG:7415 uses NAP (Normaal Amsterdams Peil), which measures elevation relative to Amsterdam normal sea level. Finally, the camera is automatically zoomed into the highlighted spatial unit with flyToBoundingSphere. The mechanism of this feature is depicted in Figure 3.10.

e. LADM Istance Level Diagram

In the Select Spatial Unit dropdown, the loadLinkedData() function retrieves the associated LADM information by calling the `/load-by-bau` API endpoint, using the selected su_id as a query parameter. The resulting data, which includes parties, rights, basic administrative units (BAUnits), and spatial unit details, is then stored in the component's reactive state variables for display and editing. However, to understand its connection with other spatial units that share the same bau_id, it will find the spatial unit by su_id from la_su_table along other spatial units under the same bau_id, then search for all rows with the same bau_id in the la_bau_table, then retrieve their corresponding r_id and p_id from la_right_table and la_party_table. The LADM Diagram will show up in the bottom panel of the website. D3.js is implemented to create a graph to visualize the relationship between party, right, BAUnit and SpatialUnit. Four groups of nodes are created in order based on the LADM table: Party → Right → BAUnit → Spatial Unit. To fit the size, their scale and vertical spacing are calculated based on the number of retrieved objects. Each node contains all attributes of its corresponding table and is colorized based on them, which are specified in the drawGraph function. Spatial Units are grouped by bau_id and sorted to make their position align in order with their corresponding bau_id. The nodes will be linked based on their id: Party and Right are linked with p_id and r_id, right and BAUnit are linked with r_id and bau_id, and BAUnit and SpatialUnit are linked with bau_id and su_id. D3 elements such as zoom and transform are implemented to allow the user to pan and zoom on the LADM graph. The user is also able to click two of the boxes: party and spatial unit, which can trigger different results. This click interaction is enabled by using event listeners directly tied to D3 elements.

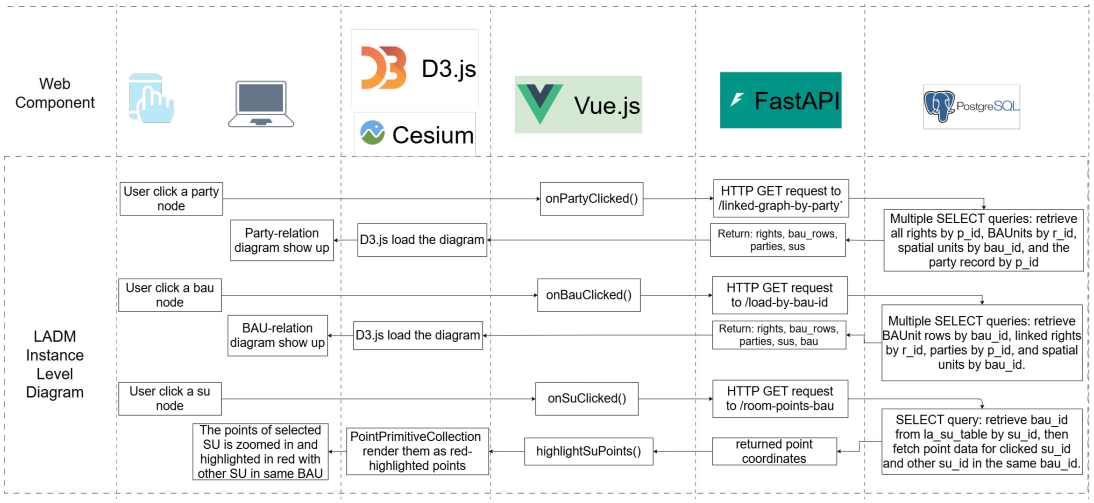


Figure 3.12.: Sequence Diagram of LADM Instance Level Diagram

The event binding is added inside the drawGraph function with `.on('click', ..)` method, then it listens to events that are defined in the Vue component $<$ *LinkedGraph/* $>$ template and the click event handler function that is declared within the methods block of a Vue component.

There are two types of relations that the graph provides. First, the LADM information of the specified BAUnit, which is an initial graph that will show all the rights and parties in the selected spatial units along with the other spatial units within the same bau_id. During the LADM Information Retrieval, the data from the GET request is also passed to the LinkedGraph.vue component to render the graph. If the user clicks on one of the spatial unit boxes, the website will zoom and highlight the spatial unit points using highlightSuPoints function.

The second type is the LADM information of the specified Party. If the user clicks on one of the party boxes, it will show another kind of graph that shows how many rights the selected party has, along with their corresponding BAUnit and spatial units. This feature sends a GET request to the backend server, where it first gathers all rights linked to the desired party, then collects all the BAUs and their corresponding Spatial Unit. The returned data is stored in partyFocusedData, which is passed to LinkedGraph to visualize the relations of one party with their available rights in LADM. The sequence diagram of this feature is illustrated in Figure 3.12.

f. LADM Edit Form

The edit form appears after the user clicks on the Edit Information button. By default, the form is filled with current LADM information that has been retrieved from the `/load-by-bau` API address as explained in the previous feature. Since one BAU unit can link to multiple rows of right and party, the user can add party and right in the form. When adding a party, the user can choose whether they want to add the new party data or select from available parties in the database. The add new party data option will fetch the latest id through `/next_ids` backend point

```
SELECT MAX(CAST(SUBSTRING(p_id FROM '[0-9]+') AS INTEGER)) FROM la_party_table
```

This will return the next value, while the available party option will send GET request with the typed name as a query parameter through `/search-parties` API endpoint with a query that will return the list of matched parties:

```
SELECT p_id, party_name, ext_id FROM la_party_table WHERE LOWER(party_name)
LIKE LOWER(%s) ORDER BY party_name LIMIT 10
```

To ensure the party links to the correct right, the user needs to fill in the corresponding p_id inside the right panel edit form. The graph also allows real-time updates that can be seen by the user to check whether the relations and the data are already coherent or not. The two-way binding directive, v-model, is used to store the user inputs. A collection of these reactive state variables (including user input for party, right, BAUnit, and spatial unit) is passed as a property called linkedData into the LinkedGraphcomponent, a D3-based module that renders the visual graph. In `LinkedGraph.vue`, the incoming linkedData is declared as a prop and watched using Vue's watch() function with deep and immediate parameter set as true. This ensures that every time the user modifies any field in the form, such as editing a party name or right type, the graph is redrawn immediately by calling drawGraph(), providing instant visual feedback on the updated relationships and attributes. After the user finishes editing the form and clicks the Save button, the saveAll event listener is triggered. This function gathers all user input data, formats specific fields (especially date objects) into the required string format, and constructs a complete payload in JSON. The payload is then sent to the backend using a POST request to the `/update-all` API endpoint, where it is parsed and stored in the PostgreSQL database. The server iterates over the list of party objects and executes an INSERT into the la_party table for each object. Using ON COFLICT, it checks whether the p_id already exists, if there is already the same p_id in the database, then with DO UPDATE SET, it will update the other fields with the new values instead with the following query:

```
INSERT INTO la_party_table (p_id, party_name, ext_id)
VALUES (%s, %s, %s) ON CONFLICT (p_id) DO UPDATE SET
party_name = EXCLUDED.party_name, ext_id = EXCLUDED.ext_id
```

This same process is applied to the la_right and la_su tables. Meanwhile, for the la_-acBAUnit table, it will check the record based on the combination of bau_id and r_id. If it already exists, then it updates the other fields based on the new value. Otherwise, it inserts a new entry. The sequence diagram of this feature is illustrated in Figure 3.10.

g. Underground View

If the user slides the underground toggle, the state in showUnderground is on, and it will trigger the toggleUnderground function to execute the GlobeTranslucency from Cesium to a transparent terrain surface. To control its camera dynamic, the NearFarScalar is set with 0.2 transparency for up to 100 meters and gradually increases until 300 meters, when it will be fully opaque.

4. Implementation

This chapter begins by outlining all the tools required for conducting this research, including Python libraries, web development frameworks, and supporting applications. The datasets used as inputs for the study are introduced in the final section.

4.1. Tools

The algorithm is mainly written in Python for data processing (Parsing the floor plan, Segmenting AHN Point Cloud, and Generating Synthetic Point Cloud) while other languages are also used for database (Storing Point Cloud to LADM) and web development (Visualizing 3D LAS). The full code can be accessed in GitHub. Additional software is also utilized to check the result after every process. More details are as follows:

a. Essential Python libraries and framework

1. OpenCV is a real-time computer vision and image processing library that provides high-level interfaces for capturing, processing, and presenting image data. It is used to process floor plan images
2. Shapely is a Python library based on the GEOS (Geometry Engine - Open Source) library for geometric operations, including manipulating and analyzing planar (2D) geometric objects. It is implemented during georeferencing, such as simplification, rotation, and scaling.
3. PDAL is a Python binding to the C++ PDAL library for manipulating point cloud data through JavaScript Object Notation (JSON)-defined processing pipelines. It is used for ground filtering and cropping
4. Open3D is an advanced 3D data processing library focusing on point cloud and 3D geometry applications, which support visualization, registration, reconstruction, and optimization. It is implemented for RANSAC segmentation and ICP
5. laspy is designed for accessing point clouds in LAS/LAZ format with basic features such as read, modify, and create. It is implemented to generate synthetic point clouds and their attributes.
6. easyOCR is an open source OCR library that applies for reading text in images to detect drawing per floor and number ID to recognize the cadastral ID.

b. Web Development

1. HTML to layout and structure user interface
2. CSS to style the web

3. Vue.js to manage user interaction
4. PostgreSQL to serve as a database
5. Cesium to visualize 3D geospatial object, which in this case point cloud
6. D3.js is a JavaScript library for interactive data visualization in web browser, used for LADM diagram
7. psycopg2 is a PostgreSQL database adapter to connect to a database from PostgreSQL and run Structured Query Language (SQL) queries in Python. It is implemented to retrieve LADM information on the website.
8. FastAPI is a web framework for building APIs in Python. It is applied to retrieve LADM information on the website with psycopg2 support.

c. Software

1. Paint to check the image and manually clean the stairs or noises
2. QGIS to check the geopackage/shapefile from the floor plan, and also to manually edit the attributes for SU ID
3. CloudCompare to check the point cloud and clean the noise

4.2. Datasets

1. Cadastrel drawing

   Three cadastral apartment drawings from Kadaster are used as samples for this research, drawn in different years: 1999 (Figure 4.1), 2002 (Figure 4.2), and 2019 (Figure 4.3), located in the Rotterdam municipality.

2. AHN Version 1-5

   The AHN datasets are downloaded from GeoTiles, covering all available versions (1 to 5), using the following Tile IDs: 37FZ1_18 for Samples 1 and 3, and 37GN2_05 for Sample 2.

3. Cadastral parcel

   The parcel polygons are obtained from Kadaster kaart (WFS) dataset in PDOK.nl. The map is part of the Basic Registration Land Registry (BRK) that includes records of properties and the legal rights attached to them, like ownership, leasehold, or easements.

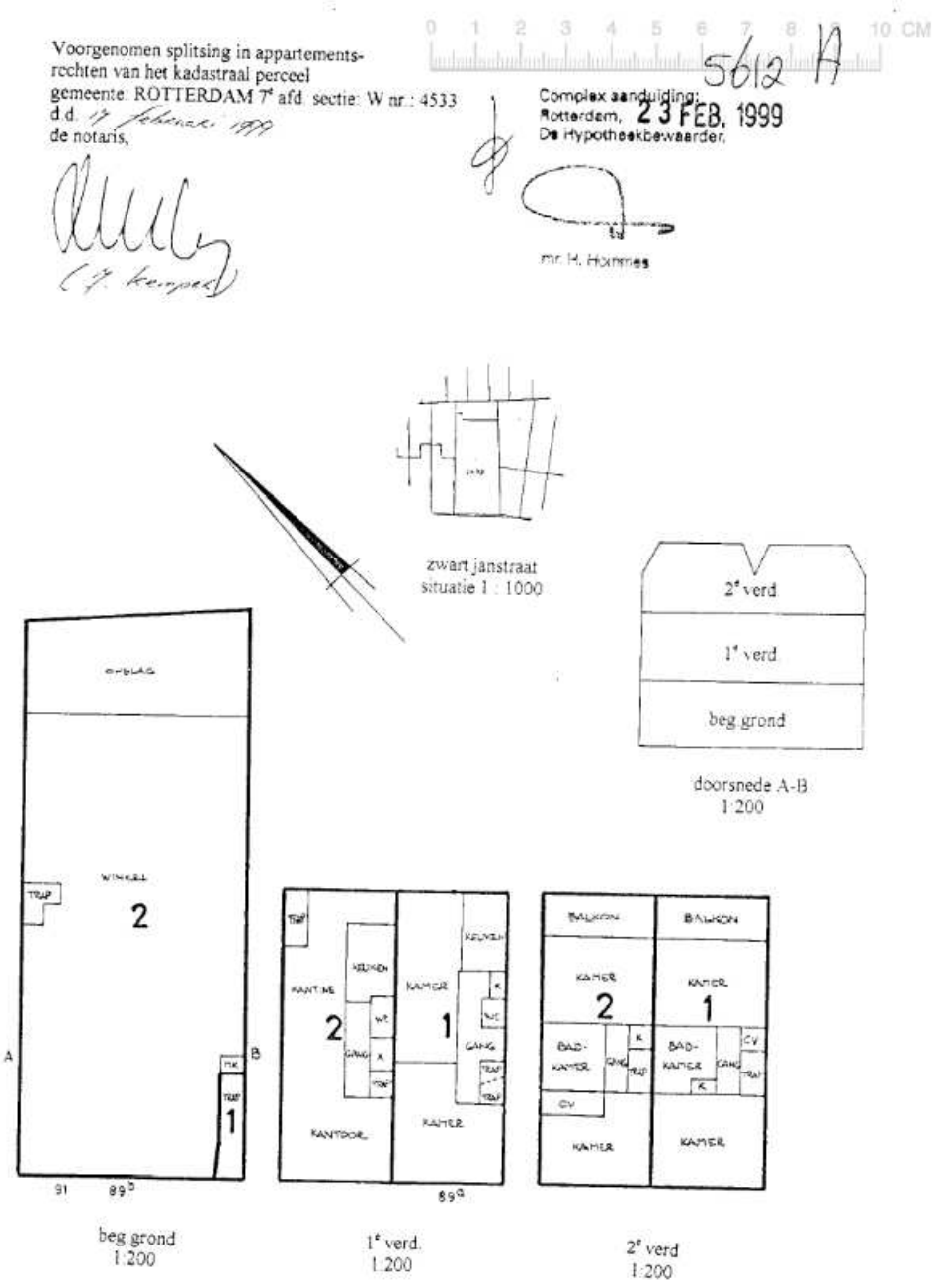


Figure 4.1.: Kadaster Sample 1

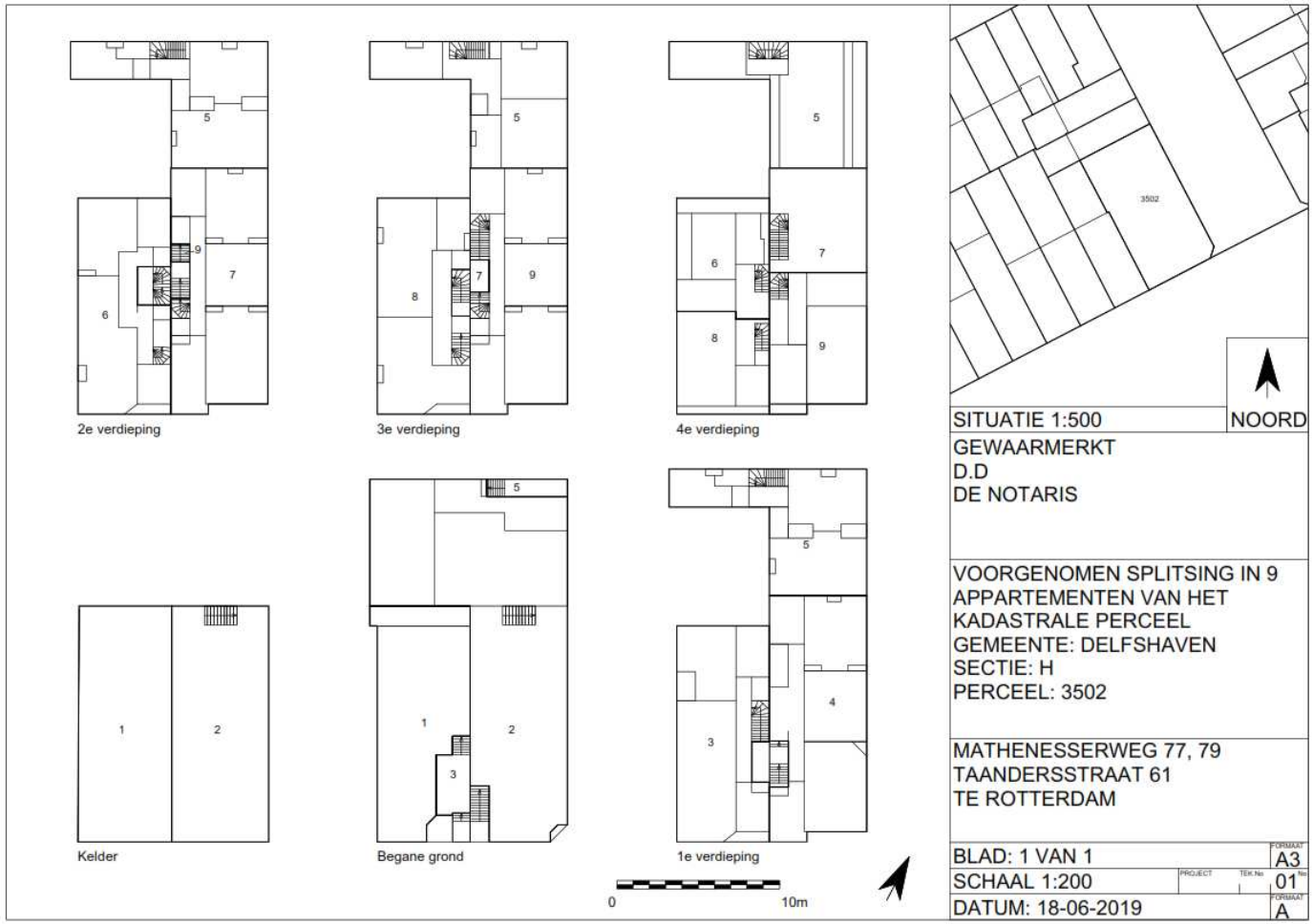


Figure 4.2.: Kadaster Sample 2

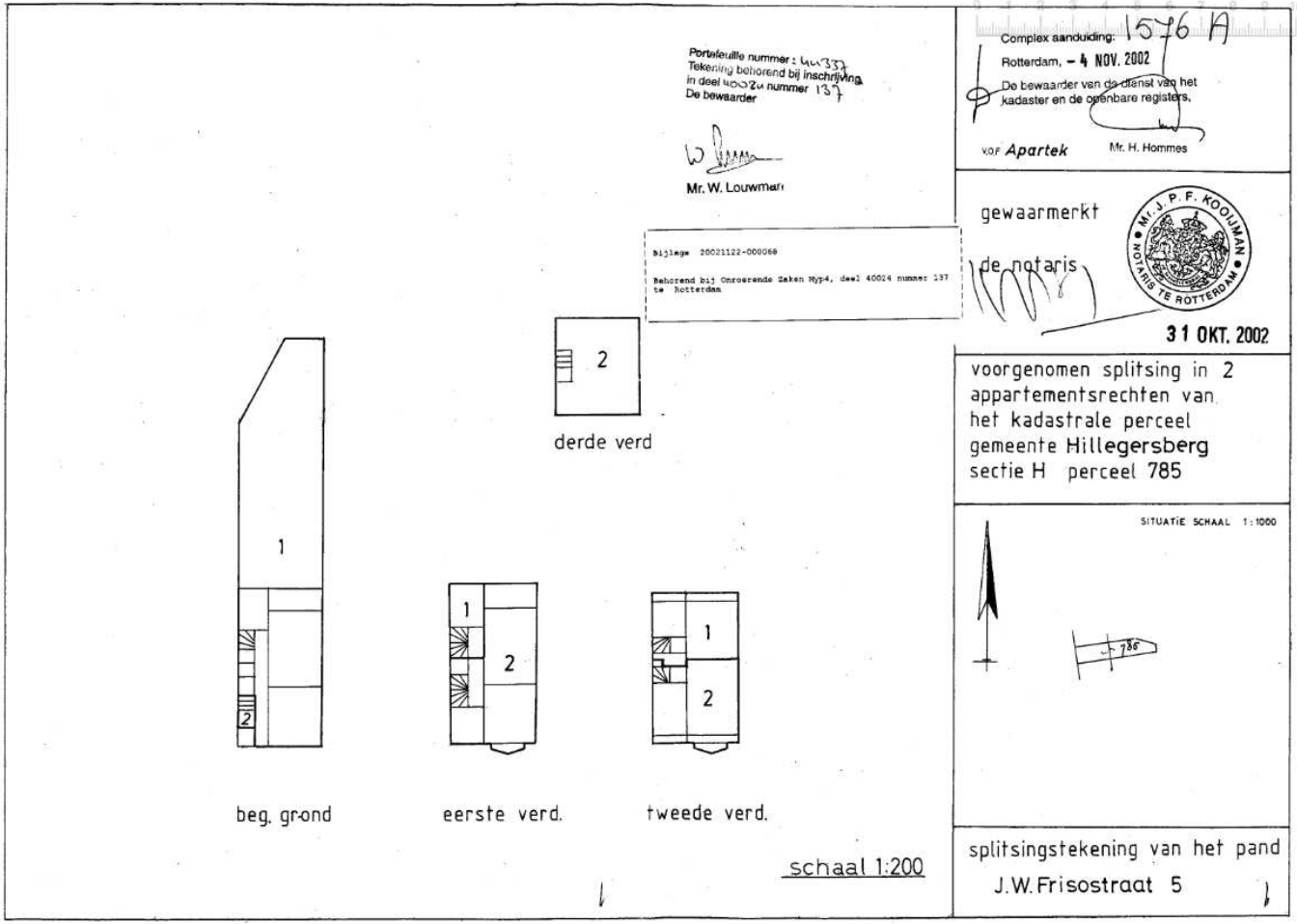


Figure 4.3.: Kadaster Sample 3

5. Results

This chapter presents and evaluates the results obtained at each stage of the research, including an assessment of accuracy and a discussion of the obstacles encountered during the process. In the final section, the chapter examines the overall workflow and reflects on the capability of the proposed pipeline to generate a 3D LAS dataset.

5.1. Parsing Floor Plan

a. From Image to Polygon

During the preprocessing of the image, OCR can read the floor label that is associated with keywords such as ”begane” and ”verdieping”; as a result, the contour block for each floor is able to be generated. However, during the vectorization, the OCR cannot read the room number in most cases as can be seen in Figure 5.1, Figure 5.2, and Figure 5.3; consequently, the spatial unit ID for most of the polygons is empty, and manual input based on the original floor plan needs to be executed in QGIS. Another user intervention is also needed to check and filter the output of the individual floor plan image, as some pictures may contain duplicate or wrong contour blocks.

The interior of the apartment can be detected and vectorized into geometric polygons. Cadastral apartment drawings depict cadastral boundaries with thicker lines and room segmentation with thinner lines. The algorithm is able to differentiate the thicker and thinner lines in the newest cadastral drawing, thus generating cadastral boundaries without room segmentation; on the contrary, for old cadastral drawings, Sample 1 and Sample 3, the polygons are generated from all room segmentation, not cadastral boundaries, as the thick lines are hard to distinguish even by eyesight. Some input images need to be cleaned manually using an image editor due to inconsistent lines or stair areas that cannot be detected during the automatic cleaning process. As Sample 1 does not require any manual cleaning, Sample 2 in Figure 5.4 shows that stairs and annotations in the drawing create noises, and inconsistent width boundaries lead to lines not being generated, while stairs in Sample 3 prevent room segmentation. The parameters must also be tuned for different drawing files, as each file may vary in resolution, style, and quality. The effects of adjusting each parameter have been previously discussed in Section 3.1 Parsing the floor plan. The specific parameter values used are listed in Table 5.1 below.

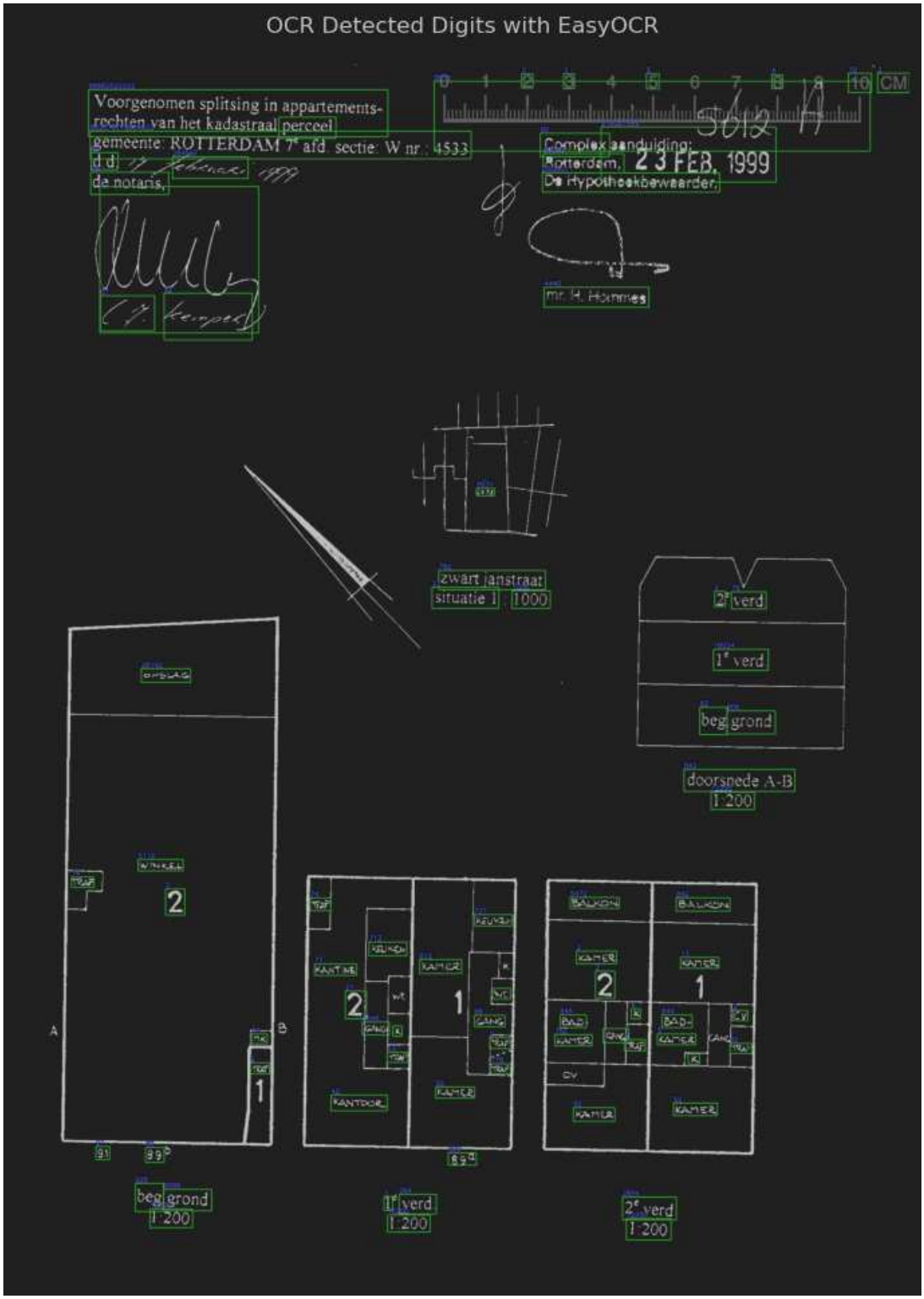


Figure 5.1.: Detected Texts in Sample 1

Figure 5.2.: Detected Texts in Sample 2

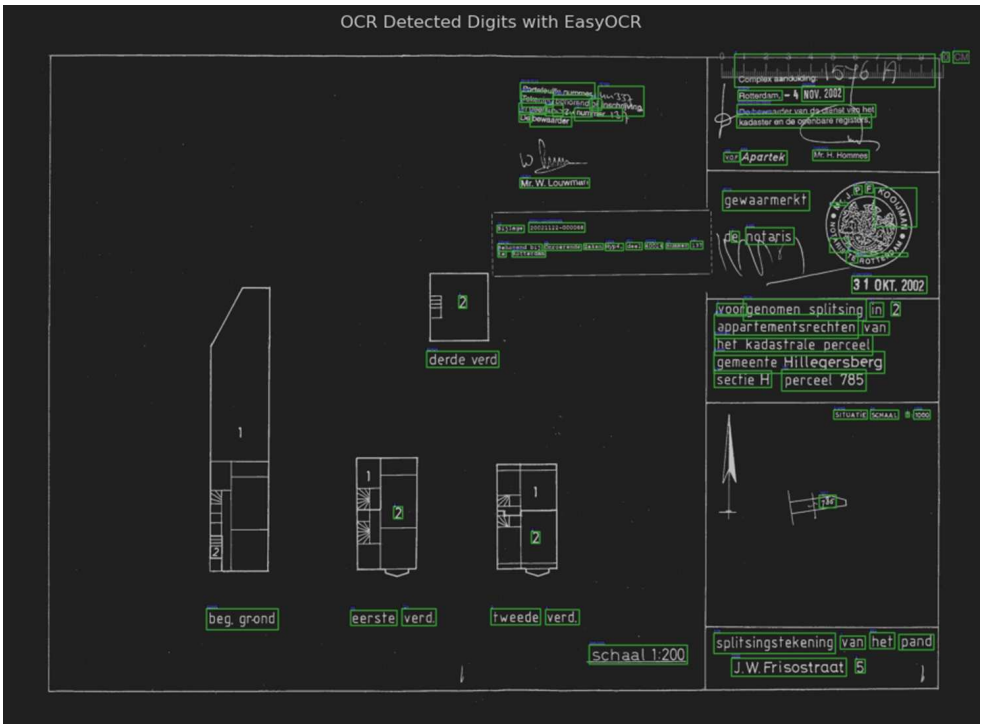
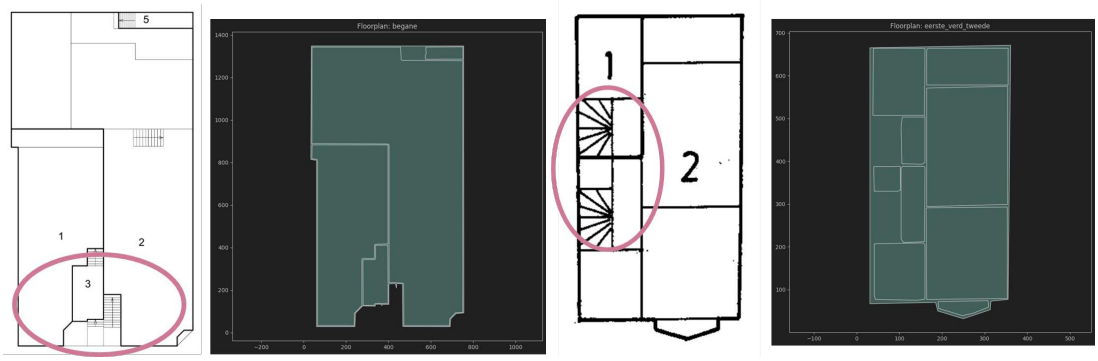


Figure 5.3.: Detected Texts in Sample 3



(a) Sample 2 (b) Sample 3

Figure 5.4.: Noises in Image that needed to be cleaned manually

Process	Parameter Sample 1	Sample 2	Sample 3
open kernel	3x3	2x2	2x2
open iteration	3	2	1
close kernel	20x20	10x10	5x5
close iteration	1	2	1
epsilon	0.013	0.001	0.005
simplify	10	1	9
buffer	8	5	5

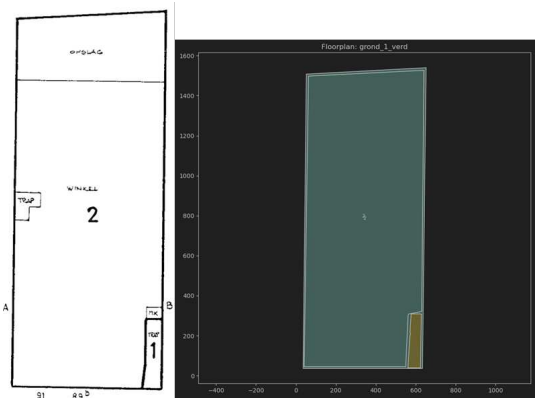
Table 5.1.: Parameter during Vectorization

b. Georeferencing

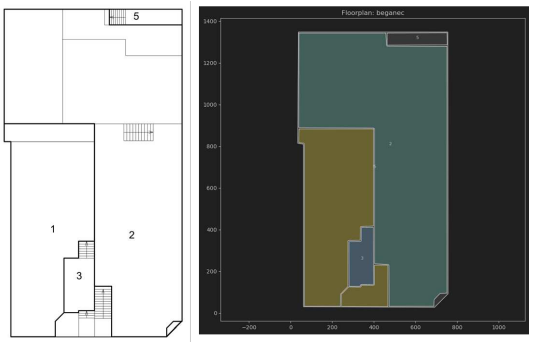
As described in Section 3.1 Parsing the floor plan, the georeferencing algorithm estimates transformation parameters, including rotation, scaling, and translation. For a more detailed explanation of the procedure, refer to Appendix C. Although the georefencing algorithm uses a simple calculation, it presents an adequate result where the polygon is located in the same place as the cadastral boundary, as illustrated in Figure 5.6. The Root Mean Square Error (RMSE) is also below half a meter, as shown in the Table 5.2. However, for certain cases, one must add additional rotations in the code input parameter, where the value of the input degree is tuned manually based on the orientation and shape of the vectorized polygon. For instance, since Sample 1 has a rectangular shape, which has rotational symmetry, it may need to be flipped.

	RMSE (cm)
Sample 1	32.18
Sample 2	18.25
Sample 3	25.71

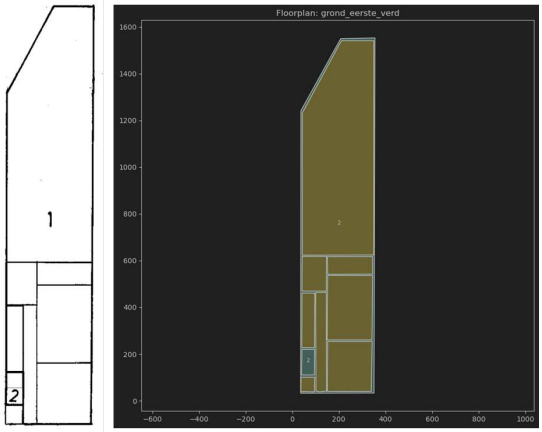
Table 5.2.: RMSE of Georeferencing



(a) Vectorization Result in Sample 1



(b) Vectorization Result in Sample 2



(c) Vectorization Result in Sample 3

Figure 5.5.: Vectorization Result in All Samples

5.2. Segmenting Point Cloud

Combining multiple AHN versions can overcome occlusion in the AHN as it provides more points for the building, as can be seen in Figure 5.7; however, wall points are still sparse and some parts still missing. Another problem is that the buildings are row houses, and they were located between other units as depicted in Figure 5.8; therefore, the surrounding walls, particularly the shared or common wall, were impossible to acquire by LiDAR scanning.

Ground points are effectively extracted from the complete building point clouds using the Cloth Simulation Filtering (CSF) algorithm, configured with default parameter values suitable for moderately flat terrain. The resulting ground points are visualized as blue-colored points in Figure 5.9. However, during subsequent segmentation, distinguishing non-ground points, specifically separating wall and roof components (shown in red and white, respectively), remains challenging. This is particularly evident in cases involving sparse wall points and sloped roofs, such as in Sample 3 (Figure 5.9c). Additionally, some outliers and vegetation points persist after classification, as seen in Sample 1 (Figure 5.9a), indicating limitations in the accuracy of the AHN-based classification.

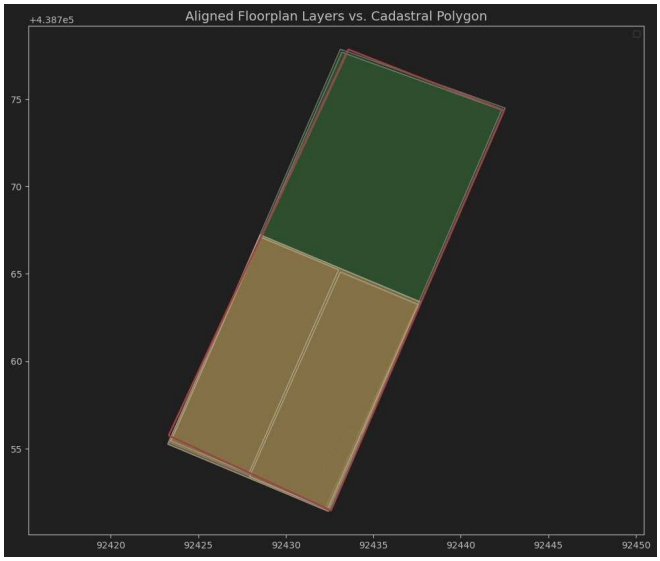
5.3. Synthetic Point Cloud Construction

The height of the ground floor fits the AHN as it uses the z value of ground points. However, the floor height does not seem to correctly conform to AHN, due to some misclassified roof points, as mentioned before in the previous step.

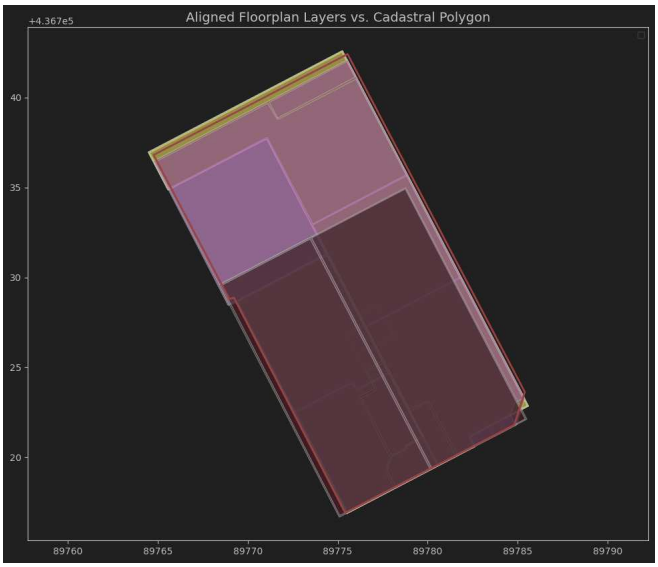
		Sample 1	Sample 2	Sample 3
Initial	Fitness	3.7e-05	6.92e-05	1.31e-04
	Inlier RMSE (cm)	1.406	1.478	1.418
	Correspondences	44	249	141
Point-to-Point ICP	Fitness	5.22e-05	8.73e-05	1.89e-04
	Inlier RMSE (cm)	1.480	1.480	1.459
	Correspondences	62	314	141
Point-to-Plane ICP	Fitness	0	7.43e-05	0
	Inlier RMSE (cm)	0	1.537	0
	Correspondences	0	267	0

Table 5.3.: RMSE of ICP

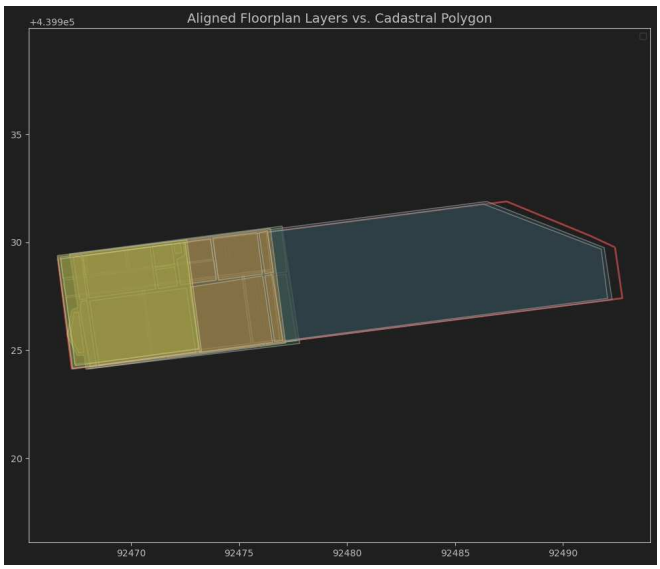
Although the floor plan polygons have been georeferenced based on the parcel polygon, and the generated point clouds from the floor plan are aligned with AHN through ICP, the highly accurate position is still hard to acquire, with an RMSE between 1.3 and 1.6 cm as can be seen in Table 5.3. This is due to sparse points in AHN that affect the performance of ICP. Point-to-point needs to find the corresponding point between the datasets; thus, it would be challenging if no matching points are available. Meanwhile, point-to-plane exploits normal calculation, which also becomes problematic if the surrounding neighbour points are not adequate to correctly calculate the normal for each point. For that reason, Point-to-Plane ICP only performs better for Sample 2 (Figure 5.11), which has more correspondence points, while it fails completely for Sample 1 and Sample 3 (Figure 5.10 and 5.12), which have fewer correspondence points. The algorithm will automatically use the ICP method that has lower RMSE and higher number of correspondences points. Point-to-Point ICP is preferred for



(a) All Georeferenced Floor plan in Sample 1

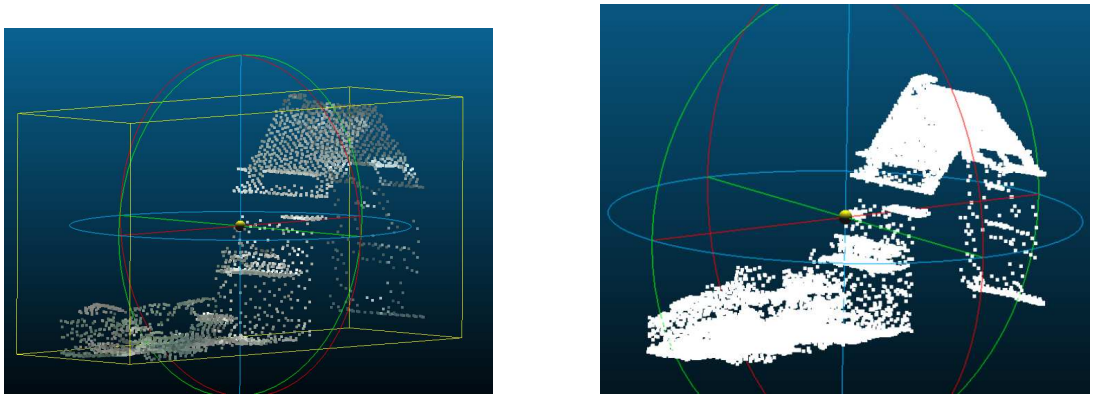


(b) All Georeferenced Floor plan in Sample 2



(c) All Georeferenced Floor plan in Sample 3

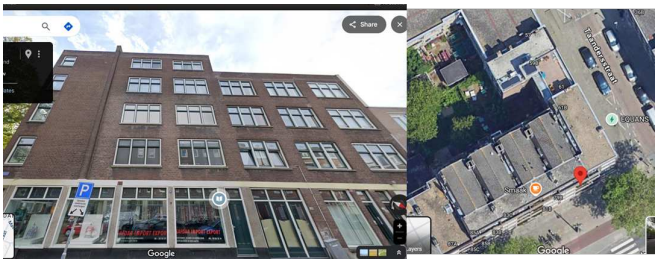
Figure 5.6.: Overview of Georeferencing Results



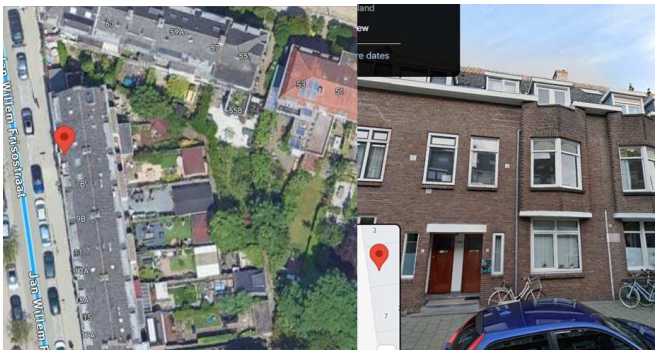
(a) AHN 5 (b) Combination of All Versions of AHN

Figure 5.7.: Comparison of AHN 5 and Combination in Sample 3

(a) Building location for Sample 1

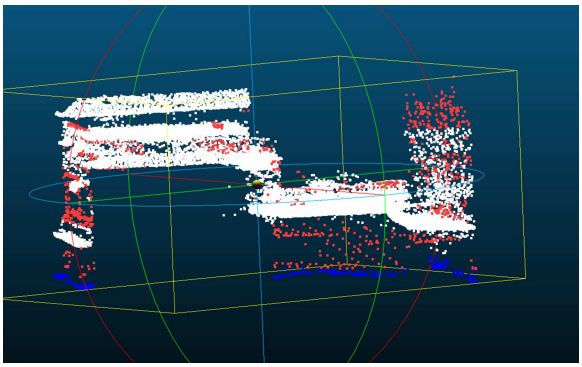


(b) Building location for Sample 2

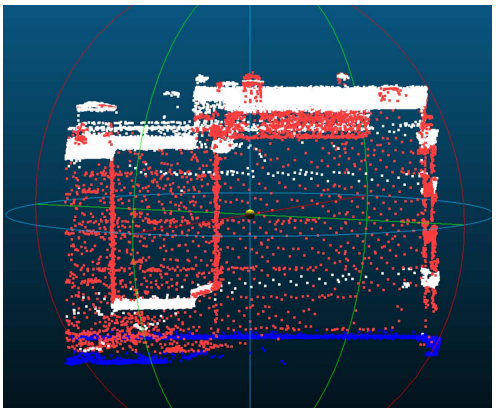


(c) Building location for Sample 3

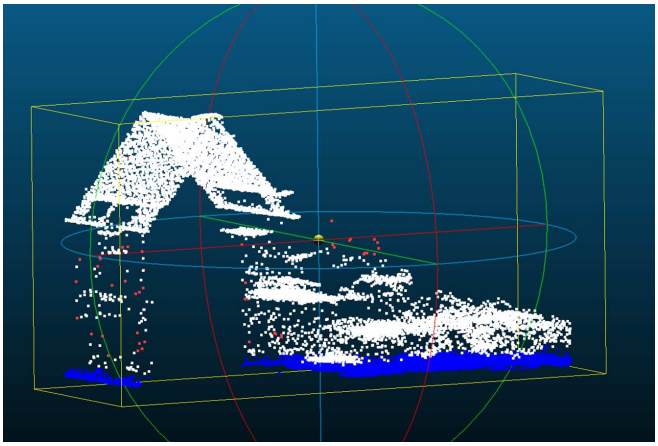
Figure 5.8.: Building location for All Samples



(a) Segmentation in Sample 1



(b) Segmentation in Sample 2



(c) Segmentation in Sample 3

Figure 5.9.: Result of Segmentation in All Samples

Sample 1 and Sample 3, and Point-to-Plane ICP is opted for Sample 2. Although all the resulting RMSEs are slightly higher or worse than the initial for the three cases, the values for correspondences increase, depicting a greater number of matched point pairs between the source and target after alignment. This also leads to a slightly higher fitness value, which means the proportion of total source points that matched within a threshold of 2 cm.

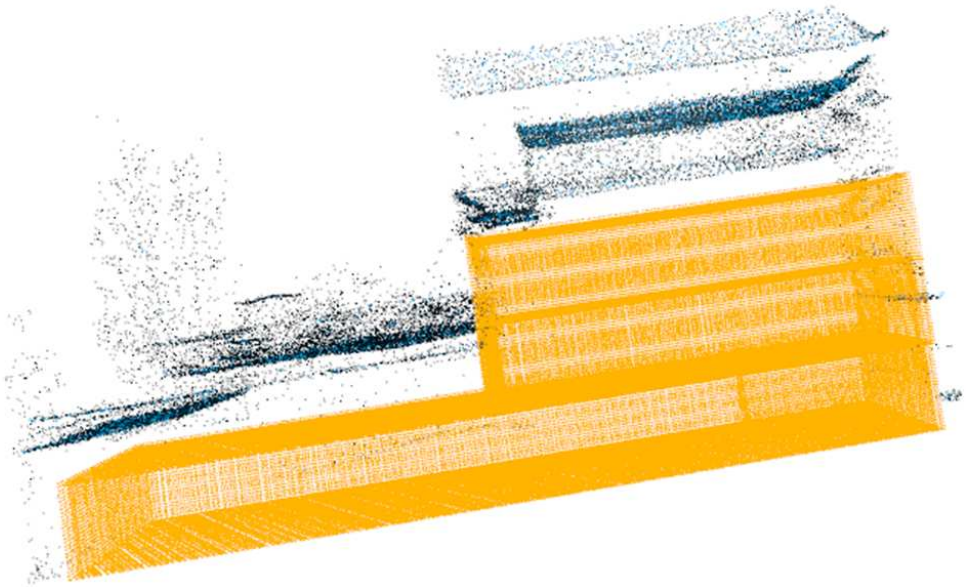


Figure 5.10.: ICP Result in Sample 1

After aligning the synthetic point cloud to AHN, both datasets are combined into one LAS file for each sample and uploaded into Cesium Ion.

5.4. Point Cloud to LADM Storage

To store the point cloud and link their LADM information, one needs to import the resulting CSV from Section 5.3 Synthetic Point Cloud Construction that contains 3D coordinates along with their attributes into PostgreSQL. Due to privacy concerns, the land administration information filled in this research is fictional. As mentioned in Section 3.4 Storing Point Cloud to LADM, five tables are created in PostgreSQL: (1) point cloud table in Figure 5.13; (2) spatial unit table in Figure 5.14; (3) BAUnit table in Figure 5.15; (4) Right table in Figure 5.16; (5) Party table in Figure 5.17. All of the LADM classes are derived from VersionedObject, a LADM special class that allows all information in the LADM database to be tracked historically through time stamps by providing attributes called begin_lifespan and end_lifespan.

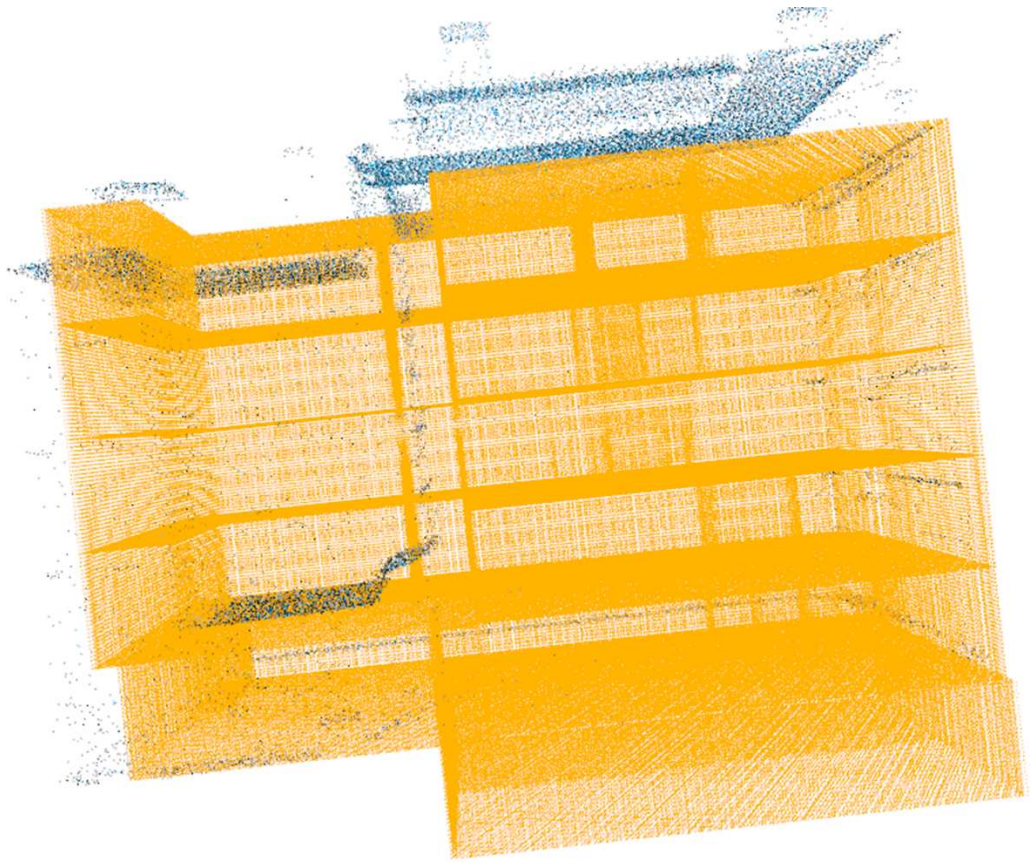


Figure 5.11.: ICP Result in Sample 2

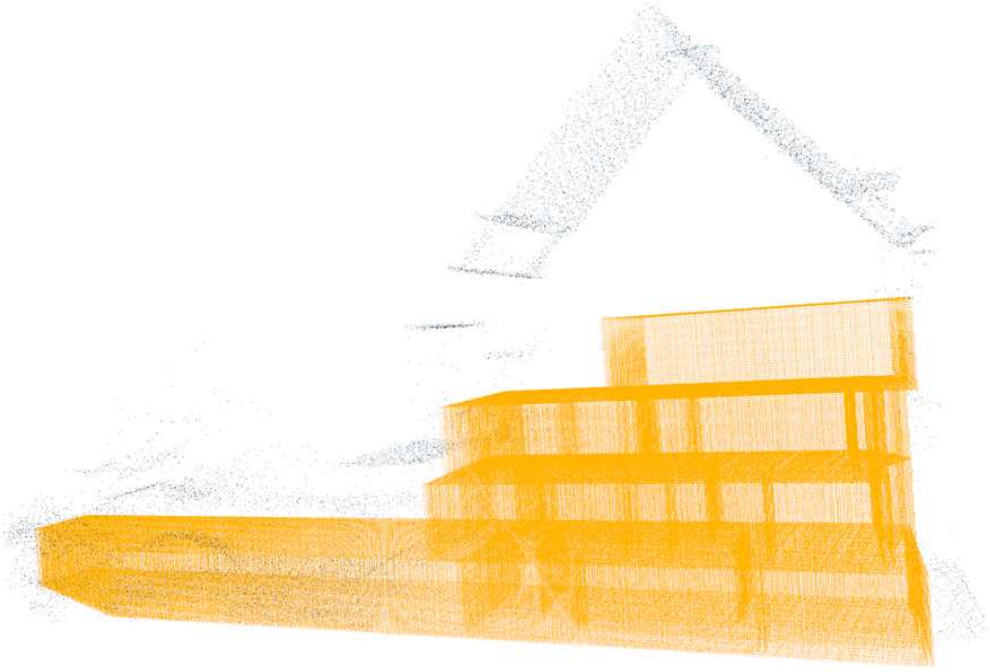


Figure 5.12.: ICP Result in Sample 3

Query | Query History

```
1 SELECT * FROM public.synth_pc
2
```

Data Output | Messages | Notifications

	point_id integer	room_id integer	floor_number integer	x double precision	y double precision	z double precision	name character varying (50)	geom geometry	su_id character varying (20)	area numeric	k_id integer
1	235210	3001	3	92435.06832355363	438763.94030429487	5.166037761051607	kad1	01010000A0E610000090...	kad1_1_3001	61.814638577645184	1
2	235211	3001	3	92435.06832355363	438763.99030429486	5.166037761051607	kad1	01010000A0E6100000816...	kad1_1_3001	61.814638577645184	1
3	235212	3001	3	92435.06832355363	438764.04030429485	5.166037761051607	kad1	01010000A0E6100000730...	kad1_1_3001	61.814638577645184	1
4	235213	3001	3	92435.06832355363	438764.09030429483	5.166037761051607	kad1	01010000A0E610000063...	kad1_1_3001	61.814638577645184	1
5	235214	3001	3	92435.06832355363	438764.1403042948	5.166037761051607	kad1	01010000A0E6100000515...	kad1_1_3001	61.814638577645184	1
6	235215	3001	3	92435.06832355363	438764.1903042948	5.166037761051607	kad1	01010000A0E6100000410...	kad1_1_3001	61.814638577645184	1
7	235216	3001	3	92435.06832355363	438764.2403042948	5.166037761051607	kad1	01010000A0E610000033...	kad1_1_3001	61.814638577645184	1
8	235217	3001	3	92435.06832355363	438764.2903042948	5.166037761051607	kad1	01010000A0E6100000225...	kad1_1_3001	61.814638577645184	1
9	235218	3001	3	92435.11832355363	438757.7903042963	5.166037761051607	kad1	01010000A0E6100000B1...	kad1_1_3001	61.814638577645184	1
10	235219	3001	3	92435.11832355363	438757.8403042963	5.166037761051607	kad1	01010000A0E6100000C7...	kad1_1_3001	61.814638577645184	1
11	235220	3001	3	92435.11832355363	438757.8903042963	5.166037761051607	kad1	01010000A0E6100000DF...	kad1_1_3001	61.814638577645184	1
12	235221	3001	3	92435.11832355363	438757.94030429627	5.166037761051607	kad1	01010000A0E6100000F36...	kad1_1_3001	61.814638577645184	1
13	235222	3001	3	92435.11832355363	438757.99030429625	5.166037761051607	kad1	01010000A0E6100000091...	kad1_1_3001	61.814638577645184	1
14	235223	3001	3	92435.11832355363	438758.04030429624	5.166037761051607	kad1	01010000A0E61000001E...	kad1_1_3001	61.814638577645184	1
15	235224	3001	3	92435.11832355363	438758.09030429623	5.166037761051607	kad1	01010000A0E6100000346...	kad1_1_3001	61.814638577645184	1
16	235225	3001	3	92435.11832355363	438758.1403042962	5.166037761051607	kad1	01010000A0E6100000481...	kad1_1_3001	61.814638577645184	1
17	235226	3001	3	92435.11832355363	438758.1903042962	5.166037761051607	kad1	01010000A0E61000005C...	kad1_1_3001	61.814638577645184	1

Figure 5.13.: Point Cloud table

Query Query History

```
SELECT * FROM public.la_su_table
ORDER BY su_id ASC
```

Data Output Messages Notifications

	su_id [PK] character varying (20)	label character varying (	legal_area numeric	computed_area numeric	dimension character varying (2)	ext_adressid character varying (255)	surfacerelation character varying (20)	bau_id character varying (10)	name text	num_points integer	begin_lifespan timestamp with time zone	end_lifespan timestamp with time zone
1	kad1_0_1000	outer building	[null]	247.397335496	[null]	Zwaart Janstraat 89B 9...	1	kad1_0	kad1	270256	[null]	[null]
2	kad1_0_2000	outer building	[null]	128.795824447	[null]	Zwaart Janstraat 89B 9...	2	kad1_0	kad1	150664	[null]	[null]
3	kad1_0_3000	outer building	[null]	130.518415938	[null]	Zwaart Janstraat 89B 9...	3	kad1_0	kad1	152354	[null]	[null]
4	kad1_1_1001	entrance	[null]	4.47130506905	[null]	Zwaart Janstraat 89B 9...	1	kad1_1	kad1	15064	[null]	[null]
5	kad1_1_2001	room	[null]	60.2620199981	[null]	Zwaart Janstraat 89B 9...	2	kad1_1	kad1	84606	[null]	[null]
6	kad1_1_3001	room	[null]	61.8146385776	[null]	Zwaart Janstraat 89B 9...	3	kad1_1	kad1	86118	[null]	[null]
7	kad1_2_1002	stores	[null]	233.971141971	[null]	Zwaart Janstraat 89B 9...	1	kad1_2	kad1	258106	[null]	[null]
8	kad1_2_2002	office	[null]	61.8318032563	[null]	Zwaart Janstraat 89B 9...	2	kad1_2	kad1	86126	[null]	[null]
9	kad1_2_3002	rooom	[null]	61.2879802786	[null]	Zwaart Janstraat 89B 9...	3	kad1_2	kad1	85484	[null]	[null]
10	kad3_0_-1000	[null]	[null]	170.445193498	[null]	[null]	-1	kad3_0	kad3	198310	[null]	[null]
11	kad3_0_1000	[null]	[null]	265.718367126	[null]	[null]	1	kad3_0	kad3	293111	[null]	[null]
12	kad3_0_1001	[null]	[null]	7.91454484913	[null]	[null]	1	kad3_0	kad3	22683	[null]	[null]
13	kad3_0_2000	[null]	[null]	226.408282304	[null]	[null]	2	kad3_0	kad3	277235	[null]	[null]
14	kad3_0_3000	outer building	[null]	225.927905749	[null]	[null]	3	kad3_0	kad3	276853	[null]	[null]
15	kad3_0_4000	[null]	[null]	226.305334925	[null]	[null]	4	kad3_0	kad3	277208	[null]	[null]
16	kad3_0_5000	[null]	[null]	226.144564984	[null]	[null]	5	kad3_0	kad3	277023	[null]	[null]
17	kad3_1_-1001	[null]	[null]	80.8847642597	[null]	[null]	-1	kad3_1	kad3	111908	[null]	[null]
18	kad3_1_1003	Room	[null]	68.2502931354	[null]	[null]	1	kad3_1	kad3	102142	[null]	[null]
19	kad3_2_-1002	[null]	[null]	84.7372905686	[null]	[null]	-1	kad3_2	kad3	115696	[null]	[null]
20	kad3_2_1004	[null]	[null]	167.619285229	[null]	[null]	1	kad3_2	kad3	214092	[null]	[null]

Figure 5.14.: Spatial Unit table

Query | Query History

```
1 SELECT * FROM public.la_bau_table
2 ORDER BY id ASC
```

Data Output | Messages | Notifications

	bau_id character varying (10)	la_bautype character varying (40)	begin_lifespan timestamp with time zone	end_lifespan timestamp with time zone	r_id character varying (10)	id [PK] integer
1	kad1_1	Apartment	2025-04-21 00:00:00+02	2025-05-11 00:00:00+02	r_2	3
2	kad1_2	Apt	2025-04-30 00:00:00+02	2025-05-30 00:00:00+02	r_3	4
3	kad1_2	Apt	2025-04-30 00:00:00+02	2025-05-30 00:00:00+02	r_4	5
4	kad1_1	Apartment	2025-04-21 00:00:00+02	2025-05-11 00:00:00+02	r_9	6
5	kad1_1	Apartment	2025-04-21 00:00:00+02	2025-05-11 00:00:00+02	r_10	7
6	kad5_1	Apartment	2025-05-01 00:00:00+02	2025-05-23 00:00:00+02	r_11	8
7	kad5_1	Apartment	2025-05-01 00:00:00+02	2025-05-23 00:00:00+02	r_12	9
8	kad3_1	Housing	2025-05-06 00:00:00+02	2025-05-15 00:00:00+02	r_13	10

Figure 5.15.: BAUnit table

Query Query History

```
1 SELECT * FROM public.la_right_table
2 ORDER BY r_id ASC
```

Data Output Messages Notifications

	r_id [PK] character varying (10)	la_righttype character varying (40)	share character varying (5)	begin_lifespan timestamp with time zone	end_lifespan timestamp with time zone	la_restrictiontype character varying (40)	p_id character varying (10)
1	r_10	Ownership	1/2	2025-04-30 00:00:00+02	2025-05-22 00:00:00+02	[null]	p_3
2	r_11	Ownership	1	2025-05-30 00:00:00+02	2025-05-27 00:00:00+02	[null]	p_6
3	r_12	Lease	1	2026-08-05 00:00:00+02	2025-05-16 00:00:00+02	[null]	p_4
4	r_13	Ownership	1/2	2025-04-30 00:00:00+02	2025-05-16 00:00:00+02	[null]	p_7
5	r_2	Lease	1	2025-04-22 00:00:00+02	2025-05-04 00:00:00+02	[null]	p_2
6	r_3	Ownership	1/2	2025-04-30 00:00:00+02	2025-05-14 00:00:00+02	[null]	p_3
7	r_4	Ownership	1/2	2025-04-30 00:00:00+02	2025-05-12 00:00:00+02	[null]	p_4
8	r_9	Ownership	1/2	2025-04-29 00:00:00+02	2025-05-28 00:00:00+02	[null]	p_5

Figure 5.16.: Right table

Query Query History

```
1  SELECT * FROM public.la_party_table
2  ORDER BY p_id ASC
```

Data Output Messages Notifications

	p_id [PK] character varying (10)	la_partytype character varying (40)	party_name text	begin_lifespane timestamp with time zone	end_lifespan timestamp with time zone	ext_id character varying (20)
1	p_2	[null]	Waroi	[null]	[null]	323223
2	p_3	[null]	oscar	[null]	[null]	32
3	p_4	[null]	lando	[null]	[null]	909032
4	p_5	[null]	Cilian Murphy	[null]	[null]	2332
5	p_6	[null]	Lana	[null]	[null]	8932
6	p_7	[null]	Keira	[null]	[null]	32900923

Figure 5.17.: Party table

5.5. 3D Land Administration System Visualization

The development of a 3D visualization website prioritizes optimal performance and user experience. The usability of the system for delivering LADM information is also considered to ensure the platform fulfills its intended main objective. To this end, several interactive features have been incorporated to facilitate user engagement and improve the effectiveness of the platform, including 3D Land Administration Visualization, BAUnit Visualization Checkbox, Tooltip, Selection and Highlight, LADM Instance Level Diagram, LADM Edit Form, and Underground View.

a. 3D Land Administration Visualization

Visualization in the Land Administration context focuses on the representation of ownership boundaries and their related legal information. With a 3D map, the visualization is upgraded to more complex 3D structures with a sense of depth that is closer to the real world representation [Pouliot et al., 2018]. A 3D parcel is the fundamental spatial unit in a LAS to which a unique and homogeneous set of rights, responsibilities, and restrictions (RRRs) is assigned. Homogeneous means that the same combination of RRRs applies uniformly to the entire 3D spatial unit. The 3D parcel is the largest spatial extent where this homogeneity holds; extending the parcel would introduce different RRRs, while subdividing it would create neighboring parcels with identical RRRs [Oosterom et al., 2011]. To deliver a real-world representation, integrating other datasets, including reference objects and a topography map, can offer a reference to interpret the parcel in terms of location and size [Cemellini, 2018; Kalogianni, 2016]. Since the 3D parcel is represented as a point cloud, the AHN dataset can serve as a reference object, enabling seamless integration of spatial data, as illustrated in Figure 5.18.

b. BAUnit Visualization Checkbox

The 3D point cloud building model is rendered as the user selects the address in the dropdown option, which will automatically zoom to the desired building. In the right panel, there is a checkbox list based on the Basic Administrative unit. By default, all color boxes are checked, showing all the BAUnit with different colors as shown in Figure 5.19. If the user unchecks, then the corresponding BAUnit points will be discolored and decreased in size as occurred in Figure 5.20. However, unlike in Potree, which has a built-in feature for color classification filter, in Cesium, this feature needs to be created manually, and due to restrictions on point styling, the points may not completely disappear, but still remain as small white dots. This tool enables users to identify the specific BAUnit by color and contrast certain BAUnits by their appearance, which can be selected via a checkbox. A study by Wang et al. [2012] shows that visual variables, including size and color, are suitable to represent bounded and partially bounded 3D legal units. Size and color are known as the most efficient visual variables for human perception. By leveraging the change and difference in size and color, it helps users to easily identify the BAUnit. Since a single BAUnit may contain multiple spatial units, this helps users to effortlessly recognize which units in the building are owned by the same or different person. Additionally, common spaces are easily distinguished by their distinct color, which is visually separate from other private spatial units.

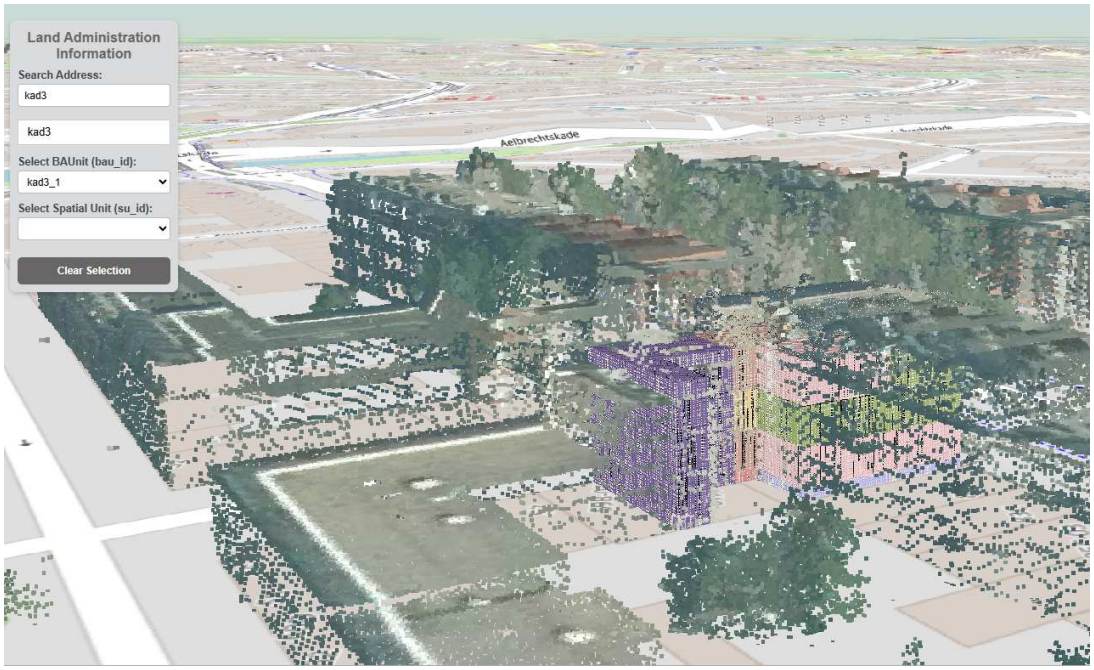


Figure 5.18.: 3D Land Administration Visualization

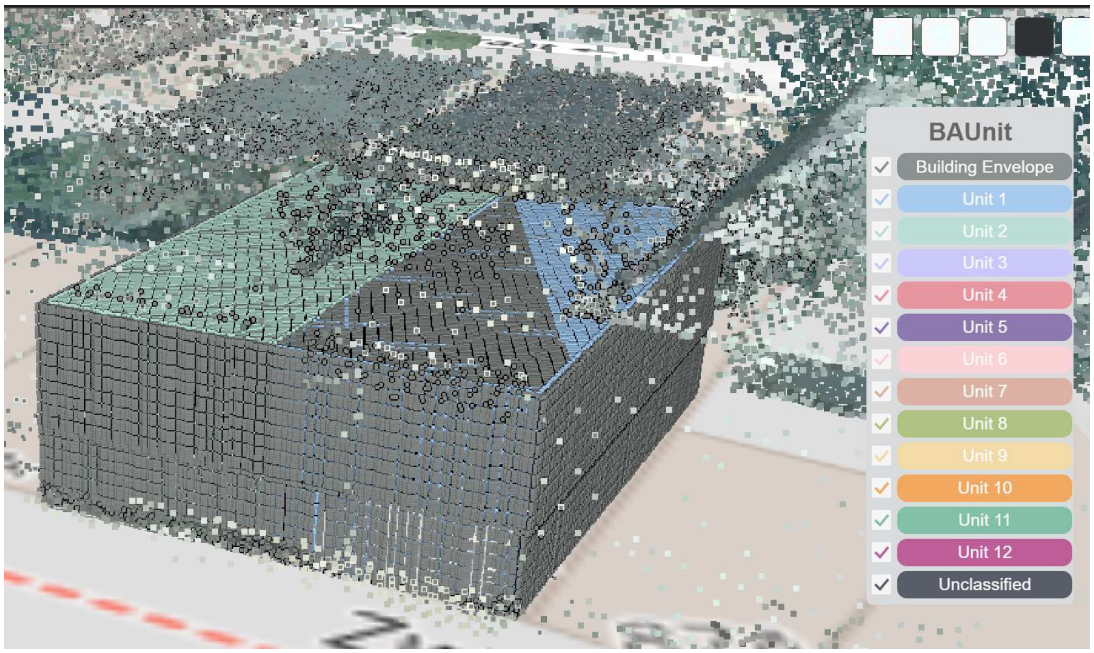


Figure 5.19.: All Checked Colorbox

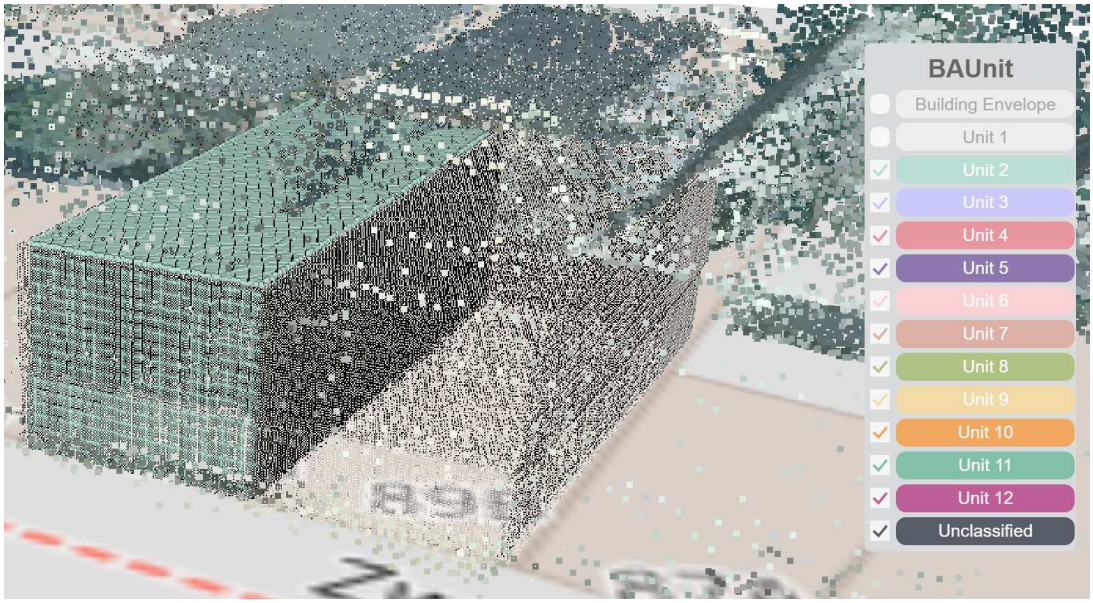


Figure 5.20.: Unchecked Colorbox for Building Envelope and Unit 1

c. Tooltip

Shojaei et al. [2013] includes a tooltip as one of the parameters for quick user recognition to improve the visualization utility. As a tool that is commonly used in GIS applications, its function is to identify the specified object and to provide the data attribute of that corresponding object. In this 3D LAS, see Figure 5.21, when the user clicks a point, it will select and highlight the points of the unit and offer brief information about the owner, right, and type of units of the corresponding unit. It displays the information of the selected spatial unit in bold, while the other spatial units within the same BAUnit are shown in regular font.

d. Selection and Highlight

Cemellini et al. [2020] identified object selection and highlighting as essential client-side tools for 3D LAS visualization. Selection is one of the fundamental universal tasks in 3D user interfaces, referring to the process of identifying and selecting one or more objects within a 3D environment [Steed, 2006; Bowman et al., 2012]. Highlighting, in turn, supports this interaction by enabling users to easily perceive the active or selected object. Typically, highlighting is achieved by altering the visual style or appearance of the selected object, thereby leveraging pre-attentive cognitive processing [Trapp et al., 2011]. Figure 5.22 illustrates two types of highlighting mechanisms. The first type highlights the BAUnit: when the user selects a BAUnit either via the toolbar or the LADM Graph, the camera automatically zooms to the selected BAUnit, and all points belonging to it are rendered in red. The second type highlights the spatial unit: when the user selects a spatial unit via the toolbar, clicks on a point, or selects a corresponding node in the LADM Graph, the application zooms in on the selected spatial unit and renders it in red, while other spatial units within the same BAUnit are displayed in a lighter shade of red. This feature delivers dynamic visualization to focus on the specified spatial unit for the user. However, since there is a discrepancy between

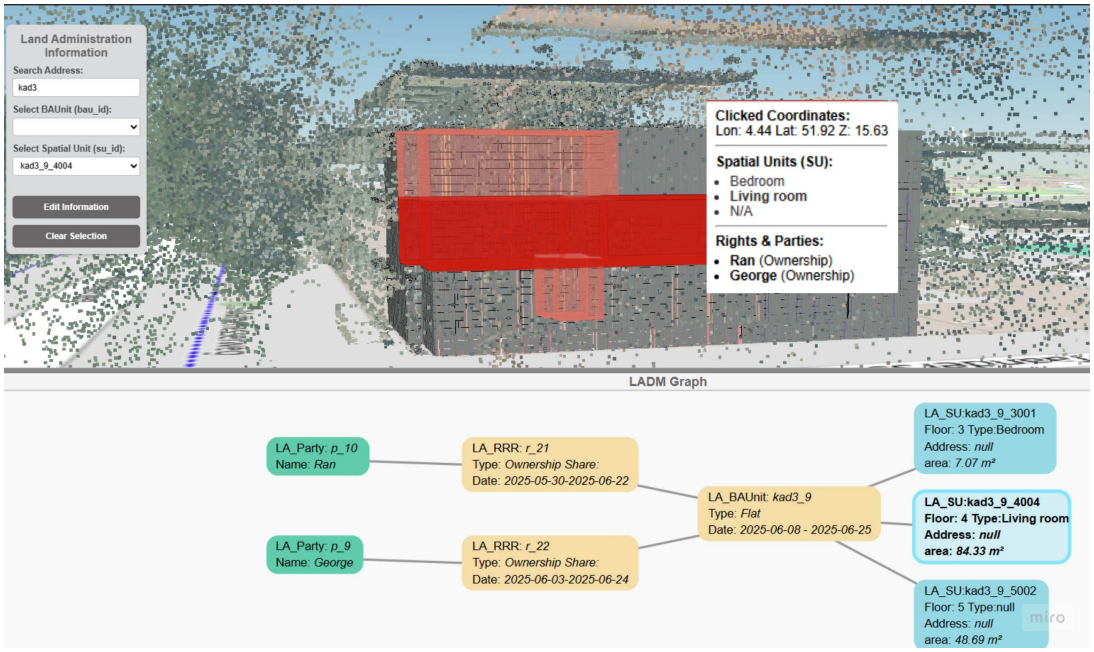


Figure 5.21.: Tooltip

Cesium and the Dutch national coordinate system for height measurements, as explained in Section 3.5, the highlight position may shift slightly from the rendered point cloud.

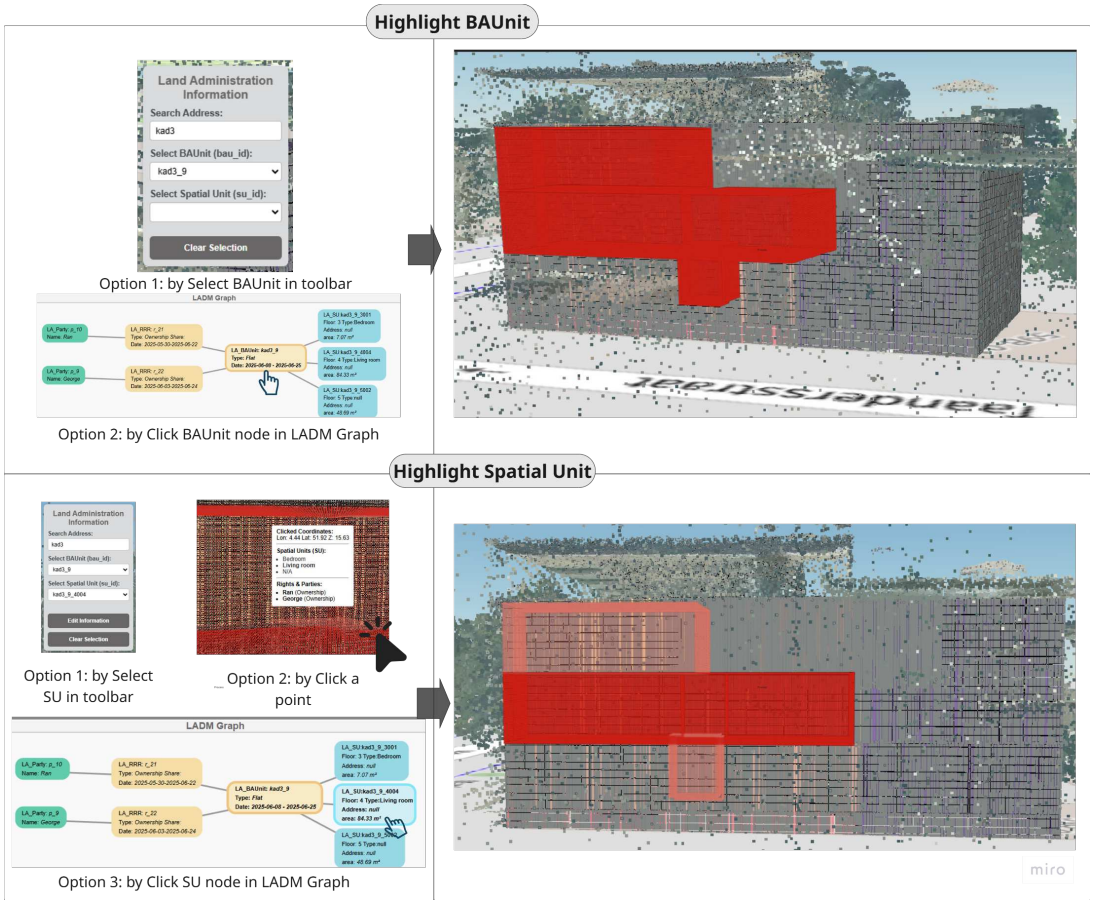


Figure 5.22.: Highlight Features

e. LADM Instance Level Diagram

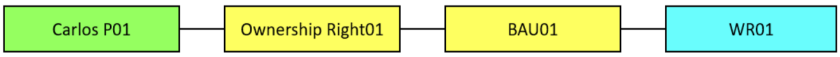


Figure 5.23.: Basic Instance Level Diagram [Lemmen et al., 2025]

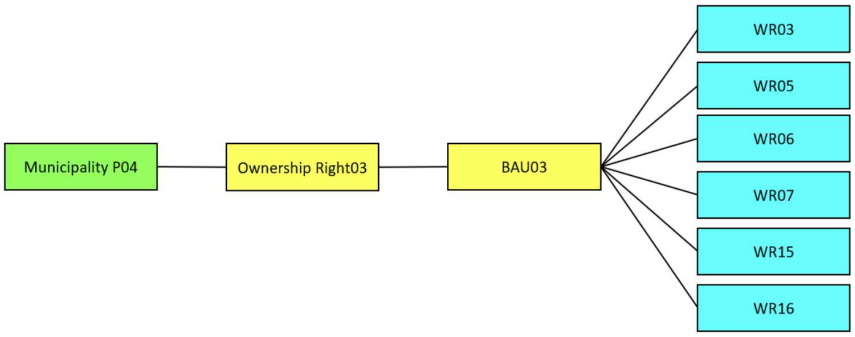


Figure 5.24.: BAUnit with multiple spatial unit [Lemmen et al., 2025]

To provide a comprehensive representation of LADM information to the user, an LADM instance-level diagram is introduced. Unlike a simple information table or panel, the diagram allows users to clearly perceive the relationships between parties, rights, BAUnits, and spatial units, as shown in Figure 5.23. It also enables the visualization of cases where a single BAUnit contains multiple spatial units, as demonstrated in Figure 5.24. In her land administration web platform, Mao et al. [2024] implements this instance-level diagram to facilitate user understanding. Two types of diagrams are presented on the web, as follows:

1. BAUnit relations diagram

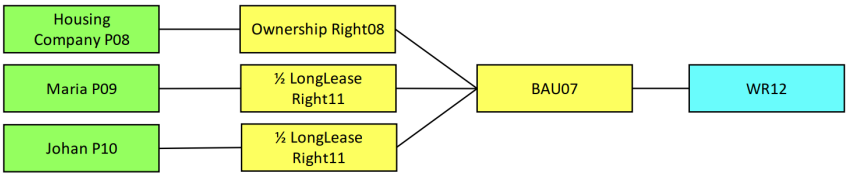


Figure 5.25.: Multiple shares in LADM diagram [Lemmen et al., 2025]

This diagram shows all the parties, right, BAUnit that are linked to the specified spatial unit along with the other spatial units under the same BAUnit, as illustrated in Figure 5.25. It allows users to comprehend the case when there are multiple shares in one property. It can occur when the property is leased, resulting in different types and records of right linked to the same BAUnit. Another case is if the property is owned or leased by multiple persons, such as a marriage partner, resulting in multiple rights linked to the same BAUnit which can be identified in the *share* column from LA_RRRs. This feature, see Figure 5.27.(a), appears as the user selects the spatial unit in the drop-down or clicks a point. To increase user experience and interaction, the graph can also be resized vertically and can be clicked. Although the web can zoom as the user selects the spatial unit in the drop-down, as this graph also shows the other spatial unit with the same BAU id, the user can also check where the other spatial unit is by clicking on the box in the diagram. This will trigger the website to zoom and highlight the spatial unit points as illustrated in Figure 5.22. As the diagram also shows, all the parties that own the property, the user can also click on one of the party boxes to see whether they own another property, which leads to the second diagram.

2. Party relations diagram

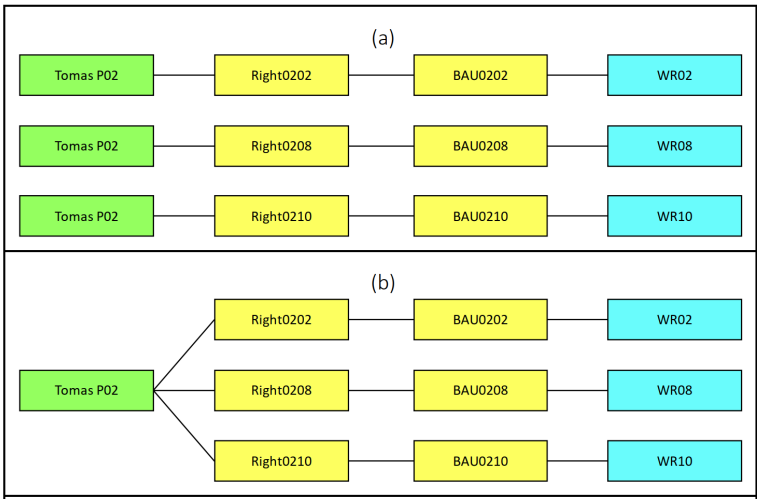


Figure 5.26.: People to land relations represented [Lemmen et al., 2025]

Lemmen et al. [2025] states that an LADM instance-level diagram can be used to reveal the party-to-land relationships in the LADM. In Figure 5.26, case (a) shows records of multiple spatial units, each with its corresponding BAUnit and right, connected separately to the same party. In contrast, case (b) resolves the repetitive layout by aggregating the party into a single node that branches out to multiple rights. By implementing this layout, it facilitates users to comprehend how many rights, BAUnits, and spatial units are associated with a specified party, as depicted in Figure 5.27.(b). To return to the previous graph, users can simply click the Back to Full Graph button located in the upper-right corner.

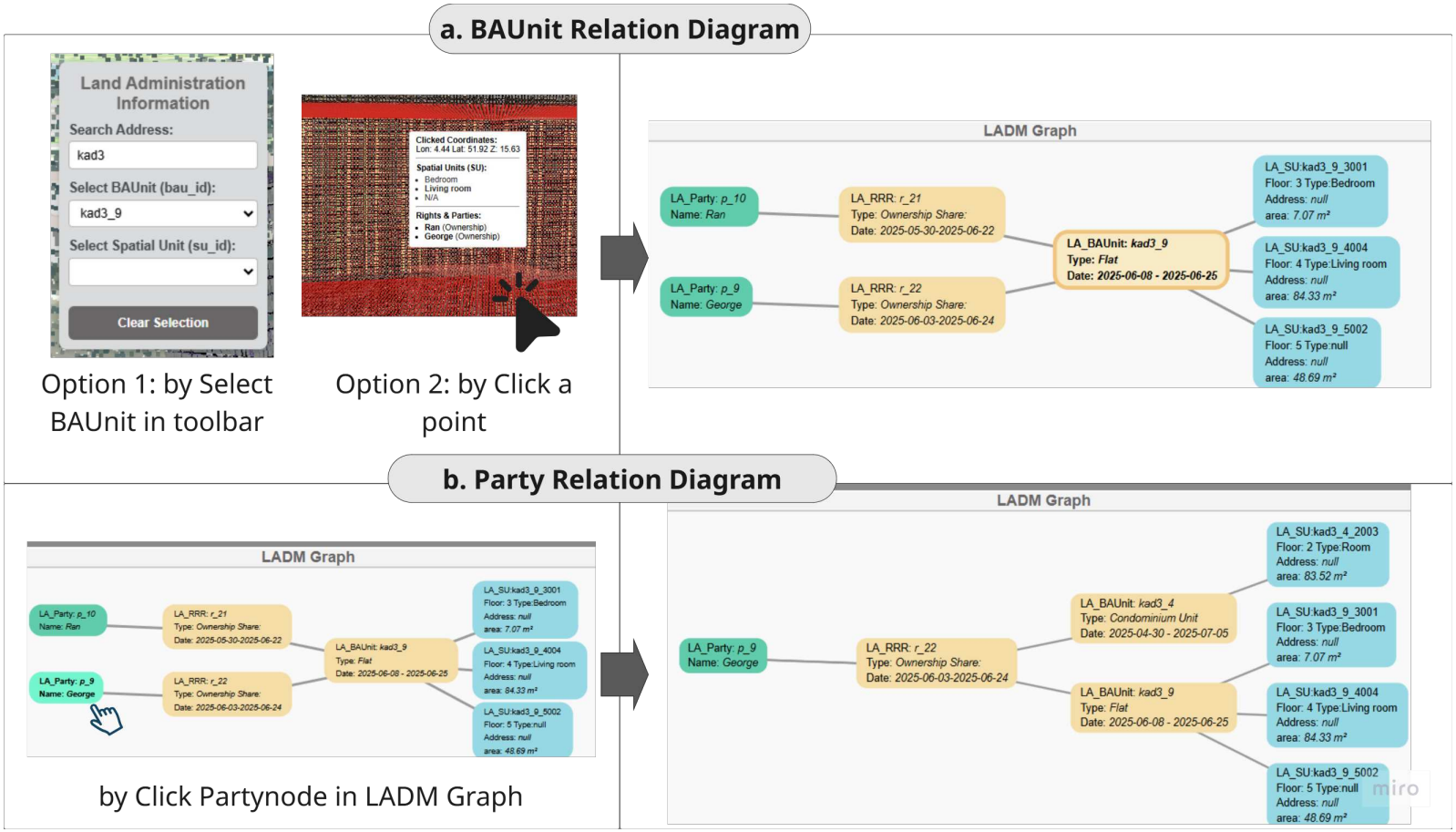


Figure 5.27.: BAUnit relation and Party relation in Instance Level Diagram

f. LADM Edit Form

To support the LAS maintenance, any update to the data needs to be facilitated for authorized users. As the land administration information records always change over time, the edit LADM form is incorporated into the 3D Web-based LAS prototype. This tool can be easily accessed by clicking the Edit Information button, like in Figure 5.28.a that appeared after the user finished selecting their desired Spatial Unit. The Edit LADM Form emerges when the user clicks the button. If the selected spatial unit already has filled data with the linked party and right, then all the available data appears as in Figure 5.28.b.2, and the user can modify the data, such as add more parties or remove an existing party. If not, then the empty LADM form is shown like in Figure 5.28b.1 and the user can add a new party and right with the Add button, and fill the BAU accordingly. Since one party may have multiple rights, the user can also select how they want to append a new party to the specified units, whether by adding a brand new party or searching for an available party, as illustrated in Figure 5.28.c. Every new party and new right will have generated a new ID that the user can use to link them together. During the editing process, any change can be seen in the LADM Graph immediately. The user simply clicks the Save All button to save any modified data, then any change is saved to the database server, Figure 5.28.e. It also allows users to remove a party or right in the corresponding BAUnit. With this tool, the troublesome process of filling the LADM information in PostgreSQL can be avoided.

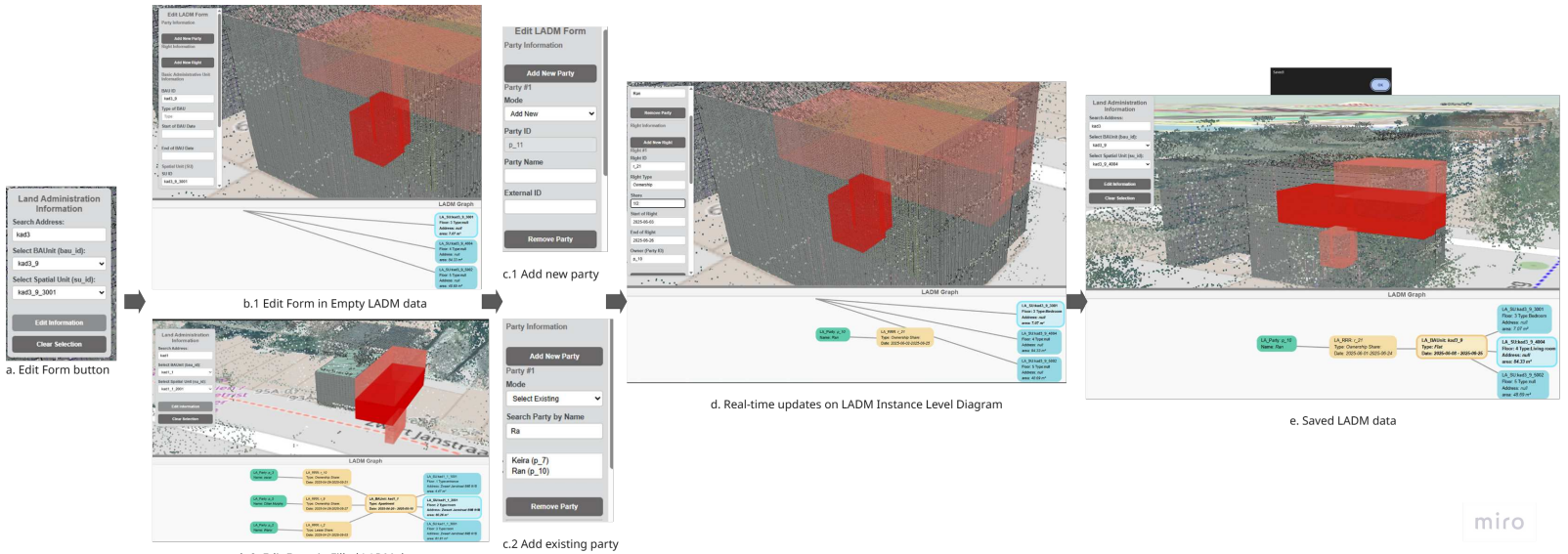


Figure 5.28.: Edit Feature Process

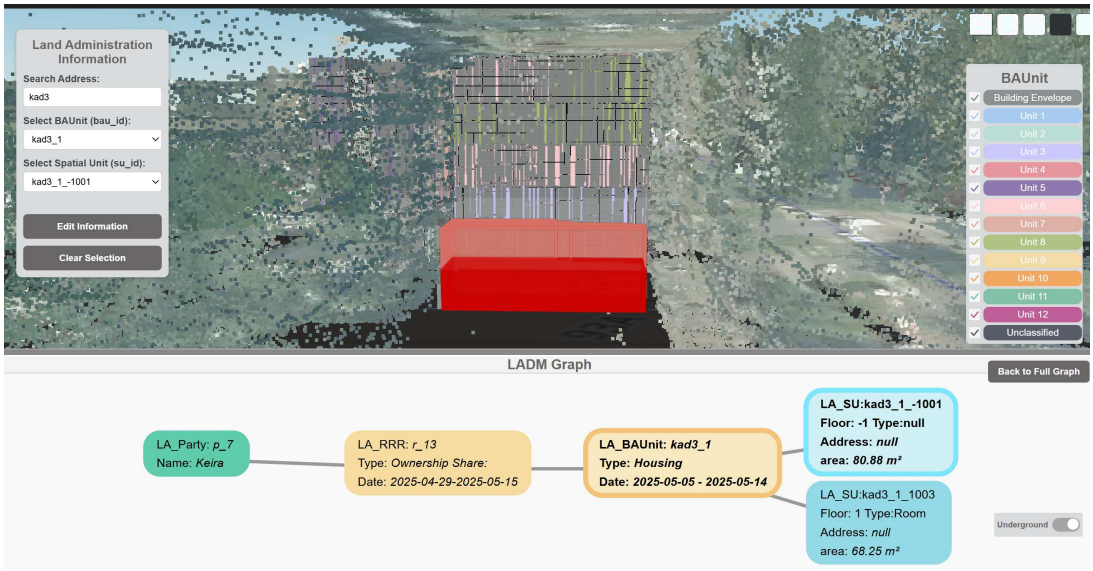


Figure 5.29.: Underground View

g. Underground View

As urban growth occurs not only above ground but also beneath the surface, several studies have emphasized that an underground view is an essential feature in 3D LAS visualization for revealing underground developments [Shojaei et al., 2013; Pouliot et al., 2018], such as utilities, basements, underground shopping malls, and subway stations. In the application, this feature can be activated by the user by toggling the underground mode via a control in the bottom-right corner of the interface. When enabled, the OpenStreetMap layer is hidden, allowing the user to explore the entire structure of the apartment building as shown in Figure 5.29, including all available spatial units located below ground level.

5.6. Reflection of Application

Geometry accuracy

The current cadastral map in the Netherlands has a graphic quality accuracy where the standard deviation of boundaries is 20 cm for urban areas and 40 cm for rural areas. Hence, the current map is inadequate to precisely determine the parcel's position on the ground [Hagemans, 2024]. These error measurements are close to the RMSEs for 2D positioning with errors between 18-32 cm. Furthermore, during 3D alignment, the RMSEs are decreased to around 1.5 cm for 62-298 matching points. However, there is still no standard about accuracy in 3D LAS, and in the cadastral drawing, the thick lines are implied as the boundaries, which is the representation of the wall of units. Thus, if one can consider the LiDAR building points as truth ground points, this accuracy, however, can be improved if the AHN or LiDAR point clouds are more abundant. It must be noted as well that planimetric accuracy of AHN versions 2 to 4 is around 5 cm [AHN, 2020].

3D Web-based LAS Prototype

After the research, not all 3D LAS requirements listed from the literature review in Ta-

ble 2.1 can be implemented in this research due to time limitations and the point cloud aspect, such as wireframe display, explode view, and sliding. Nevertheless, the majority of essential functionalities have been incorporated into the 3D LAS web prototype, as detailed in Table 5.4.

Digital twin is defined as a virtual representation of actual objects in real-world environments, synchronized through real-time data exchange [Park et al., 2023]. In this research, the point cloud serves as a virtual model of the built environment, while the LADM database provides the necessary data connections. Together, they constitute an instantiation of a digital twin for land administration purposes. In the context of the built environment, the concept of a digital twin refers to an autonomous system representation that facilitates synchronized and bidirectional updates, ensuring that any change in the virtual or physical world is reflected and acted upon in the other [Osama, 2024]. Edit Form web feature enables stakeholders to supply live data and update the LADM, implementing the lifecycle management and digital governance. However, as the current framework does not yet incorporate sensors, it can be classified as Descriptive Twin Level 1 based on five Digital Twin classifications by Autodesk [2024], see Figure 5.30. By incorporating integration of real-time sensors, change detection workflows, or lifecycle event tracking (construction, renovation, transaction updates), this 3D Web-based LAS prototype can evolve to a full Digital Twin.

For 3D Visualization of Land Administration Data	Implemented	**For 3D Web Viewer**	Implemented
Navigation tools and view controls	✓	Platform and browser independence	✓
Integrating topography and reference objects	✓	Handling massive data and caching/tiling between server and client	
Transparency		Layers control	✓
Object selection	✓	Database support	✓
Object search	✓	Support different models (vector/polyhedral, raster/voxel, point clouds)	✓
Wireframe display		Support of basic 3D topographic visualization	✓
Explode view		Support for georeferencing	✓
Sliding		Ensure spatial validity (3D vector topology)	
Cross-section view		Underground view	✓
Visualization cues	✓	Open source platform	✓
3D measurement tools		Possibility for the platform to be extended	✓
3D buffer		2D overview map (orientation)	✓
Display partly unbounded objects and 'complex' geometries	✓		
Party / RRRs visualization and selection	✓		
Point cloud-based	✓		

Table 5.4.: List of Implemented Features for 3D Visualization and Web Viewer

Point Cloud in 3D LAS

3D LAS based on BIM by Mao et al. [2024] is able to deliver nice visualization and many features that may be hard to fulfill in 3D LAS based on point cloud. However, not all buildings, particularly older buildings, have BIM, and some buildings may have different BIM formats that may also hinder the process. Point cloud and floor plan can be used as an alternative. Although the synthetic point cloud may deliver unusual visualization, its integration

Table 1
Digital twin levels.

Digital Twin	Description	Mechanism	Applications	Use Cases
Level 1	virtual replica of a built/to be built asset describes and simulates the properties and condition of a built/to be built asset and provides collaborative virtual environment supplied by live data generated by stakeholders	information flow in 2-way manner generated and communicated by stakeholders and simulated on digital twin aimed at communicating decisions/documenting changes/simulating information	Documentation/ Simulation Collaborative Planning, design, and construction Lifecycle Management Digital Governance	Helsinki 3D+ VU.CITY London [48,49]
Level 2	virtual replica of a built asset monitors, simulates and analyses the operational condition of a built asset supplied by live data generated by sensors	information flow in 2-way manner generated and communicated by sensors aimed at monitoring live asset condition, analysing operational performance, and adjusting operational behaviour	Simulation/Monitoring Performance Analysis Operational Analytics Facility Management	Virtual Singapore Shanghai Digital Twin [7, 50]
Level 3	virtual replica of a built asset observes, describes, and analyses the contextual conditions of a built asset in real-time supplied by live data generated by sensors or context users	information flow in 2-way manner describing unmeasured quantities aimed at analysing events, explaining use patterns, assessing performance, and detecting issues/unusual events	Simulation/Observation Contextual Analysis Use Pattern Analysis behaviour Analysis Fault Detection Facility Management	West Cambridge Digital Twin Herrenberg City Digital Twin [51, 52]
Level 4	virtual replica of a built asset predicts future scenarios and suggests actions/responses based on a cognitive learning supplied by live data generated by sensors, stakeholders and context users	information flow in 2-way manner describing and analysing and learning about measured and unmeasured conditions aimed at cognitive learning, future scenario prediction and proactive decision making	Proactive Planning Contextual Analysis Operational Analysis behaviour Analysis Cognitive Learning Future Scenario Simulation	N/A
Level 5	virtual replica of a built asset learn, analyses and recognises the changing events and conditions of a built asset, and therefore, react accordingly in an autonomous manner where these actions are actionable on the physical built environment	Information and actions flow in 2-way manner autonomously in a fully immersive physical-virtual environments that can learn and analyse its conditions and adjust itself autonomously	Decision Making Autonomous Operation of Built Environment Autonomous Design, Construction, and Facility Management Autonomous City Planning and Management	N/A

Figure 5.30.: Digital Twin Levels [Osama, 2024]

with more detailed LiDAR point clouds can yield a more realistic representation compared to 3D mesh models of buildings, as they can seamlessly blend together. In the context of land administration, the accurate visual representation of real-world features, such as walls and fences, captured through LiDAR scanning, is particularly valuable, as these features play a critical role in demarcating cadastral boundaries [Luo et al., 2016]. Additionally, as AHN offers precise XYZ measurement, the height of the building and the floor can be simply calculated based on the AHN [Luo et al., 2016]. The height measurements are important as a closed volume represents a legal space that is bounded by precise property boundaries and heights [Sun et al., 2019]. Despite point clouds are widely used as input data, not final data, they can provide land administration information by storing them in an LADM-compliant database. Most notably, since cadastral boundaries are determined by boundary points measured from control points [Çağdaş et al., 2023], a key advantage of using point clouds is their ability to preserve geometric representations that can be directly compared to these high-accuracy reference points. This allows for point-by-point analysis of geometric accuracy, aligning the point cloud data with precise cadastral measurements obtained via GNSS. Therefore, the integration of point clouds is crucial for achieving accurate 3D representations in land administration.

Further Implementation

The proposed pipeline requires approximately between 44 and 97 seconds per sample from processing the floor plan to generating the synthetic building point cloud, as detailed in Table 5.5. However, this estimate does not account for manual interventions that may be necessary in certain cases, such as image noise cleaning, parameter tuning, and missing cadastral number assignment. Despite these exceptions, the pipeline demonstrates sufficient efficiency for large-scale or nationwide implementation, provided that some manual input is accommodated when necessary.

Process	**Sample 1**	**Sample 2**	**Sample 3**
Preprocessing	7	7	8
Vectorize	8	22	10
Georeference	1	2	1
Crop AHN	13	15	14
Segment AHN	5	6	6
Construct PC	9	31	4
Align & combine AHN	2	14	1
Total	**45**	**97**	**44**

Table 5.5.: Processing times across samples (in seconds).

6. Conclusion

This final chapter provides a summary of the findings and conclusions of this study. The research questions presented in Section 1.2 Research Objectives are revisited to evaluate how the proposed pipeline addresses the primary research questions. Based on this reflection, the chapter also discusses the limitations of the study and suggests potential future works for refinement.

6.1. Conclusion of research question

This research studies to what extent point clouds can represent and visualize 3D spatial units. To address this main question, the answers to the sub-questions are as follows.

1. What is the suitable method to parse the floor plan?

   The analysis in Section 5.1 Parsing Floor Plan demonstrated that by using a combination of OpenCV for vectorization, easyOCR for text detection, and MBR calculation for Georeference, the cadastral drawings are able to be parsed and aligned with below 0.4 m RMSE. However, different styles of drawings may require different parameters that need to be adjusted by the user, as cadastral drawings drawn by different notaries in older times tend not to conform to the same format. As older drawings are mainly scanned without high-quality resolution, some text, especially the number of cadastral hard to detect by easyOCR. Thus, the method is able to parse the floor plan, but some manual involvement is still required.

2. How can apartment spatial units be represented when exterior wall points are unavailable due to occlusion?

   Combining multiple AHN versions often can lead to more points and more building facades provided by the variation of flight path in different versions of AHN as depicted in Section 3.2 Segmenting AHN Point Cloud. However, the result in Section 5.2 Segmenting Point Cloud shows that some parts are still occluded in the study case, either due to ALS flight elevation or the buildings located between other properties. Thus, appending the synthetic point cloud from the floor plan, which also captures the outer boundary, to AHN can help to visualize the building envelope.

3. What approach can be used to represent wall and slab points for apartment spatial units inside the apartment building?

   Result in Section 5.3 Synthetic Point Cloud Construction portrayed that generating point clouds from a floor plan demonstrates a sufficient visualization to represent spatial units. The interior geometry can be acquired without any further survey needed to be conducted. By integrating elevation data from AHN that has been segmented into wall, roof, and ground classes, the heights of both the ground floor and the roof can be estimated accurately. Using these heights, wall points can be generated by extruding the boundary polygons vertically. Similarly, slab points can be created by distributing points within the interior of the polygons at two distinct elevations, corresponding to the floor and roof heights obtained from AHN.

4. What approach can be used to accurately represent apartment spatial units and their boundaries using point cloud?

   The overall result reveals that integrating cadastral drawing, 2D parcel polygon, and AHN can model both the indoor and the outdoor of the building. As boundaries of spatial units are drawn by a thick line in the cadastral drawing, this offers a 3D representation of the cadastral boundary of the building, and also within the building. The pipeline delivers adequate-accuracy positioning, as the alignment process during 2D in Section 5.1 Parsing Floor Plan has RMSE between 18-32 cm compared to the 2D parcel polygon. During 3D in Section 5.3 Synthetic Point Cloud Construction, it improves to between 1.48-1.51 cm with 62-298 correspondence points found compared to AHN points. This accuracy, however, relied on the quality of AHN. In the Netherlands, the standard deviation for cadastral boundary positioning is 20 cm in urban areas and 40 cm in rural areas. The presented approach offers potential improvements over these standards, especially as the quality and density of AHN data improve, where the current version of AHN provides a planimetric accuracy of approximately 5 cm.

5. How can point clouds be stored in the LADM database for multi-spatial-unit apartments, including living units, storage areas, and parking spaces, as a single basic administrative unit?

   As explained in Section 5.4 Point Cloud to LADM Storage, the point cloud can be imported into PostgreSQL by using a CSV containing the 3D coordinates along with their attributes. By linking the point cloud table that has a geometry column with the other four main tables in LADM: party, right, BAUnit, and spatial unit, the points can represent 3D spatial unit along their attributes. Multiple spatial units within the same BAUnit can be associated by assigning them the same BAUnit identifier.

6. Which web architecture is suitable for representing and visualizing the resulting 3D LAS?

   Not only a building mesh, Cesium is also able to render a point cloud. Section 5.5 3D Land Administration System Visualization proves that integrating cesium, Vue.js, and FastAPI, the website offers 3D visualization and its LADM information. Some features are also available to enrich the user experience, including 3D Land Administration Visualization, BAUnit Visualization Checkbox, Tooltip, Selection and Highlight, LADM Instance Level Diagram, LADM Edit Form, and Underground View. However, several challenges have been encountered, particularly in styling and information representation of point clouds in Cesium, which are less feature-rich and harder to customize compared to the more straightforward options available for BIM models. Other 3D viewer, such as Potree, has built-in features like 3D tools measurement that allow users to directly measure points with other points and classification color filter, but in Cesium, there is no such feature available. Nevertheless, Cesium enables direct integration with OpenStreet Map and is easier to develop with Vue.js.

Main research question:

To what extent can point clouds represent and visualize 3D spatial units?

Instead of relying on an BIM model that may not be available for older buildings, this project proposes an alternative approach using point clouds as 3D spatial units in a land administration system. By combining cadastral drawings and a point cloud nationwide dataset, AHN, the buildings with their own spatial units can be generated in this framework without additional survey or an existing BIM model. This approach also enables the seamless

integration of real-world features provided from AHN such as building facades, walls, and fences, which often delineate cadastral boundaries. Another advantage is their ability to preserve geometric representation that can be directly compared to cadastral reference points measured with GNSS. Although point clouds are widely used as input data rather than final representation, by storing them in an LADM-compliant database and integrating them with 3D web-based LAS, this proposed framework is able to provide and visualize land administration information for the public and stakeholders, from identifying multi spatial unit apartment to updating the land registry. Moreover, the system is capable of generating a synthetic building point cloud in under two minutes per sample, indicating the feasibility of future nationwide implementation.

However, several limitations identified in this study warrant further refinement, including the need for manually tuning parameters and cleaning specific cadastral drawings because of inconsistencies in quality and style, and misalignment due to occlusion in AHN data.

Given that the consistency and quality of cadastral drawings and AHN data significantly influence the accuracy of the results, this study recommends to the relevant authorities that cadastral drawings issued by notaries be standardized in terms of format and resolution. Furthermore, the production of AHN data in urban areas should consider the use of higher-accuracy methods.

The code for this project is available in GitHub, while the website can be accessed in gist.bk.tudelft.nl/apps/LADMPointCloud/.

6.2. Future Work

Given the obstacles encountered and the new questions that emerged during this study, several refinements are suggested for future research to address these limitations and further improve the proposed pipeline:

- As the study only has a limited dataset, 3 apartment units, a larger-scale pilot (e.g., for an entire apartment block or city block) for future work could be conducted to improve the pipeline performance and robustness. More samples for the cadastral drawing can be utilized to develop a deep learning method, which has been popular for architectural floor plan parsing [Liu et al., 2017; Kippers et al., 2021]. Since the orientation of the floor plan also plays a big role in georeferencing, taking the front door and road along the north arrow during parsing into consideration would be important to explore. Implementing deep learning for AHN point cloud segmentation would also be considered to get a higher accuracy for roof and wall classification, which will affect more precise height calculation, as well as the height of the floor for the building. With more extensive training data, the pipeline could be extended to reconstruct more complex and realistic building geometries. Thus, new research questions can be asked, such as (1) how deep learning can parse the cadastral drawing that can identify the thick lines as cadastral boundary along with the building orientation based on the north arrow, the front door, and the road?; (2) how to segment sparse AHN point cloud with deep learning?; and (3) how to generate synthetic building point clouds that closely resemble real-world structures.

- Considering this study proposes an alternative approach to developing 3D LAS using point clouds instead of BIM, an additional research question arises: how does this method perform in terms of accuracy, completeness, and usability for LAS compared to the BIM-based approach?

- Given that the current study focuses solely on datasets from the Netherlands, both in terms of cadastral drawings and the availability of point cloud data such as AHN, future research could explore the applicability of this method in other countries. This includes utilizing alternative sources of point cloud data, such as drone-based ALS, TLS, LiDAR, or stereo matching. Furthermore, future studies could investigate how the algorithm performs when applied to various cadastral drawing styles from different countries, thereby enhancing its robustness.

- Since geometry accuracy is a critical factor for land administration, future work can focus on evaluating the accuracy for 3D LAS, as there is still limited research available on this topic. Due to time limitations, occlusion correction has not been applied to fill the occluded AHN point cloud, which plays a big role in point cloud alignment. As the current ICP method still offers moderate-accuracy positioning, other alignment algorithms may also need to be taken into consideration. Thus, it would be great to analyze how occlusion correction can improve the accuracy, compare various alignment methods that offer higher accuracy for point clouds that represent 3D LAS, and then analyze how geometry accuracy can be validated in 3D LAS. Integration with a higher resolution laser scanning point cloud may also be considered, such as drone-based LiDAR, TLS, or MLS, or photogrammetric point clouds.

- Sharing the same objective as digital twin, which is supporting the spatial development lifecycle, this 3D Web-based LAS prototype has the potential to evolve into a fully digital twin by incorporating integration of real-time sensors, change detection workflows, or lifecycle event tracking. As the current 3D LAS utilizes AHN that is regularly updated and has Edit feature that allows stakeholders to update data, new research can explore how a new version of AHN can be automatically incorporated into the current 3D LAS and how the LADM Edit form can not only exclusive for cadastral authority but also be expand into Crowdsourced sensors that allow public to report and monitor any parcel boundary validation and land rights violations. This opens the door to identifying which types of sensors and workflows can be implemented to detect real-time modifications in the context of the LADM and digital twin, which has been limited in recent studies.

- Future study can also focus on the visualization optimization, by comparing different 3D web viewers. Visualizing a point cloud dataset can become more efficient if one can render the points directly from PostgreSQL or use converted 3DTILES that can still store all the attributes in a features table to allow direct access for retrieving information on the website. Incorporating other web viewers, such as Potree, can also provide more features, as they have built-in point cloud measurements that allow users to measure the point accuracy which is essential for 3D LAS. Adaptive Level of Detail can also be implemented to give a smoother visualization.

A. Reproducibility self-assessment

A.1. Marks for each of the criteria

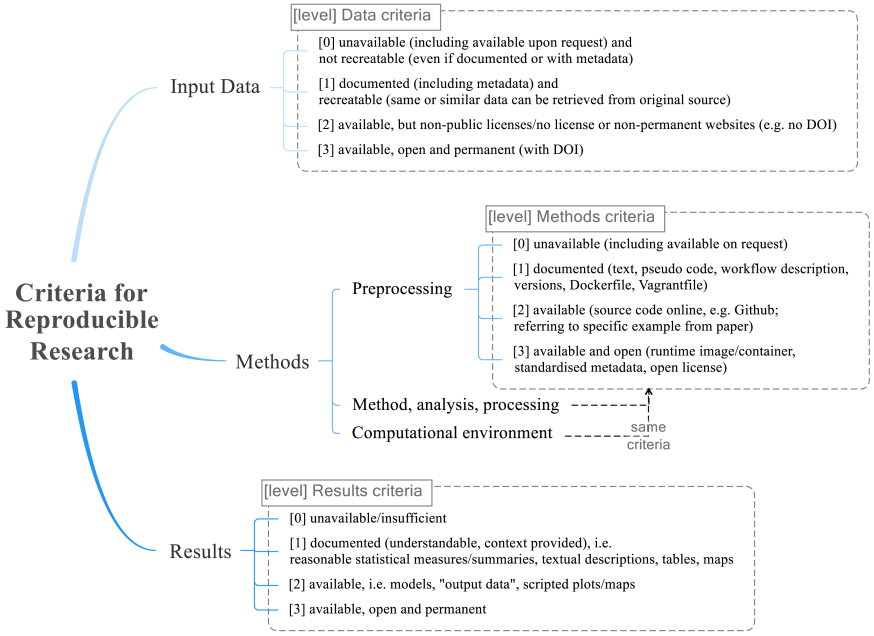


Figure A.1.: Reproducibility criteria to be assessed.

1. Input data: 0 for cadastral drawings but 3 for two other datasets (AHN and cadastral parcel)
2. Preprocessing: 2
3. Methods: 2
4. Computational environment: 2
5. Results: 2

A.2. Self-reflection

There are three datasets that are used in this research. First is the cadastral drawing that needed to be acquired with a request to Kadaster, while other datasets, AHN, and cadastral parcel are available as open data. The code for the entire process can be found at GitHub.

B. LADM UML Model

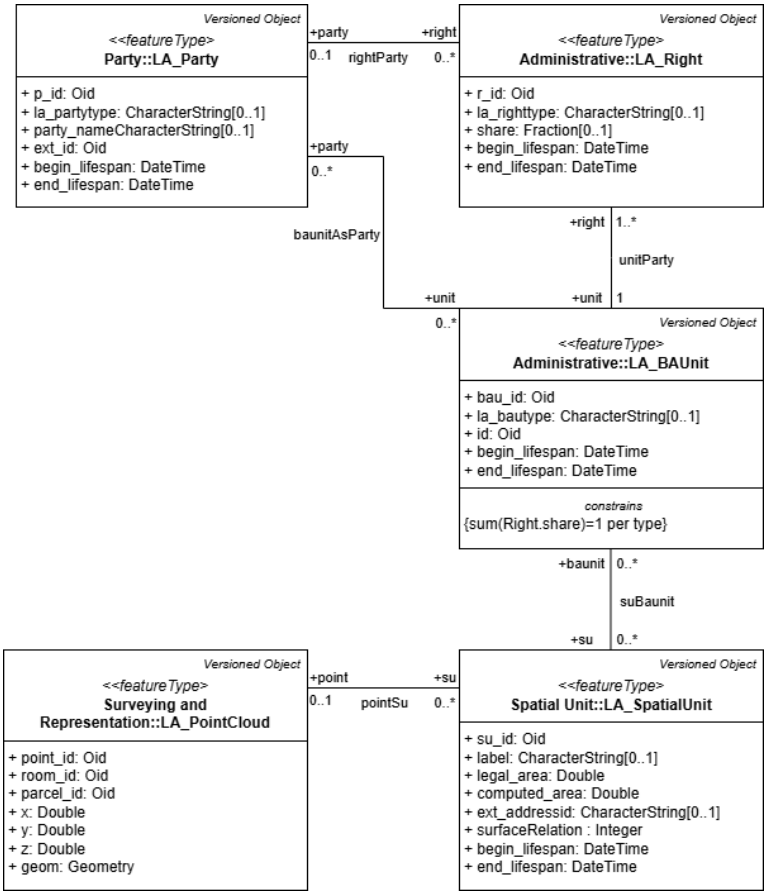


Figure B.1.: LADM UML Model

C. Georeference Method Explanation

First, it uses the MBR method to compute the orientation of the floor plan and parcel polygon. Figure C.1 illustrates how this MBR computation worked in one of the floor plans, Sample 3. It started by creating a convex hull, illustrated by a green dashed line, with the red area representing the original polygon. It will iterate depending on the number of edges on the convex hull, which in this floor plan polygon case is 11 (eleven). For each edge, the arctan is computed, where the degree is used to rotate the polygon counterclockwise towards the horizontal line. In the first iteration, the first edge consisted of two vertices (690.88, 27.00) and (61.12, 30.00), where their dx and dy are (-629.75, 3.00). The arctan is 179.73°, so the original polygon is rotated by minus 179.73° toward the x-axis, which is almost flipped vertically, as drawn with the blue line in the image. The bounding box of the rotated polygon, depicted with the orange dashed line, is created where the calculated area is 958412.81. It continues to the next edge for the next iteration with the same steps as before until all of the edges are computed. The angle with the minimum bounding box area is chosen as the floor plan orientation. The same method is also applied to the parcel polygon to get its polygon orientation. However, the number of iterations is different, as the polygon may be simpler than the floor plan polygon, which contains fewer vertices. The orientation of the floor plan polygon is -89.95° with a bounding box area of 953287.59, where the parcel polygon is -62.35° with a bounding box area of 274.92, as compared in Figure C.3. After standardizing the orientation of both polygons, the angular difference between the floor plan and parcel polygons defined the rotation angle for alignment. Thus, the floor plan is rotated by 27.61° to match the parcel polygon, as illustrated in Figure C.4. The second transformation parameter, scaling, is estimated by calculating a bounding box for the parcel polygon [89764.77, 436716.89, 89785.40, 436742.42] along with the floor plan after the rotation alignment [−223.87752112, −52.41846511, 995.22210699, 1436.90429306]. The x scale factor is derived by dividing the width of the cadastral bounding box by the width of the rotated floor plan bounding box, i.e., $\frac{89785.40-89764.77}{995.2221-(-223.8775)} = 0.0169$, while the y scale factor is derived by dividing the height of the cadastral bounding box by the height of the rotated floor plan bounding box, i.e., $\frac{436742.42-436716.89}{1436.9043-(-52.4185)} = 0.0171$. As the floor plan polygon has not been located in the real-world coordinate, after scaling the floor plan using both x and y scale factors, the difference of lower-left corner coordinate between the cadastral and scaled floor plan bounding box is computed: translation_x = 89764.77 - (-223.88 × 0.0169) = 89768.56 and translation_y = 436716.89 - (-52.42 × 0.0171) = 436717.79, and then used to translate the floor plan to the correct geographic location.

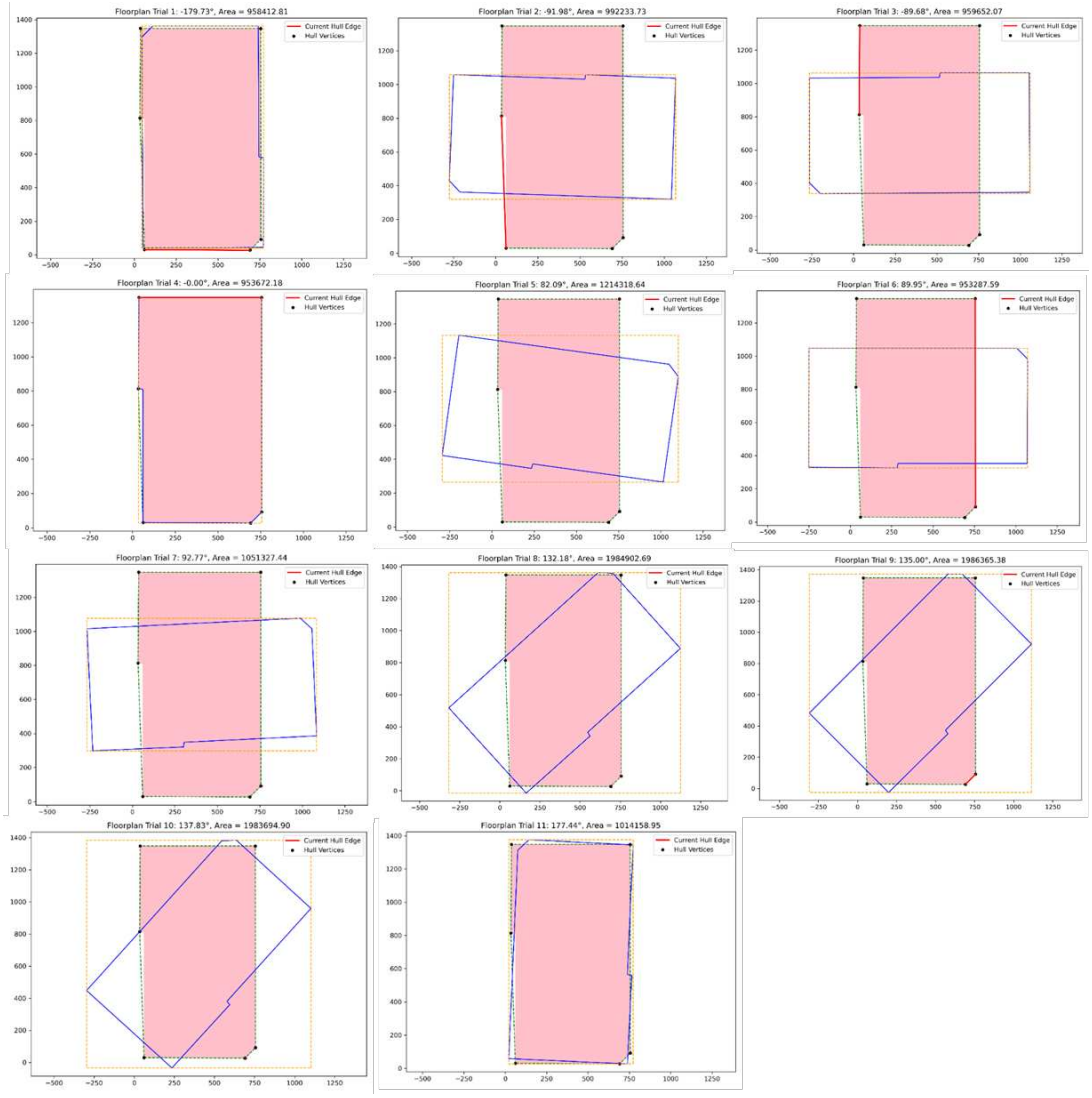


Figure C.1.: MBR Calculation in Floor plan polygon

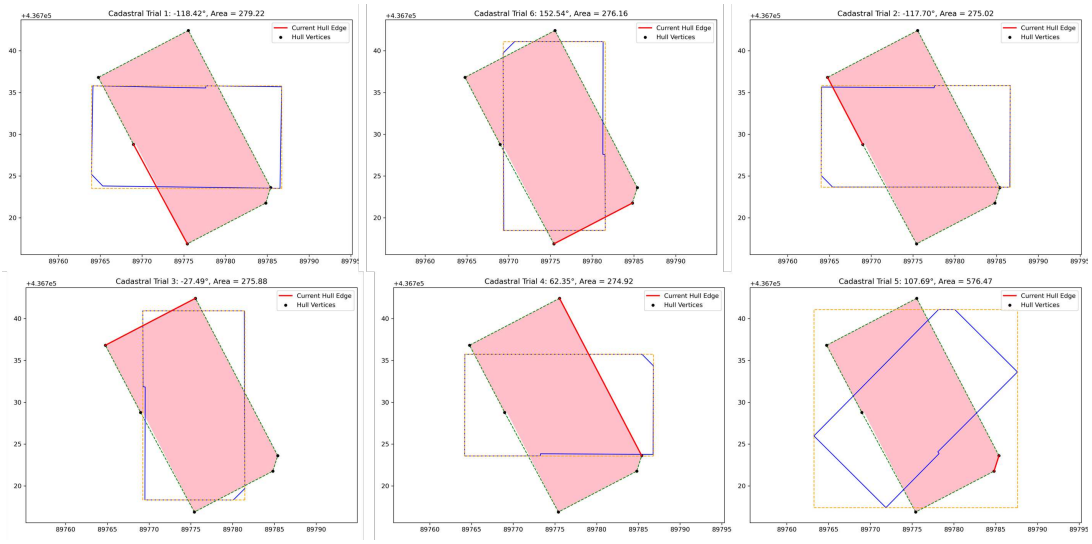


Figure C.2.: MBR Calculation in parcel polygon

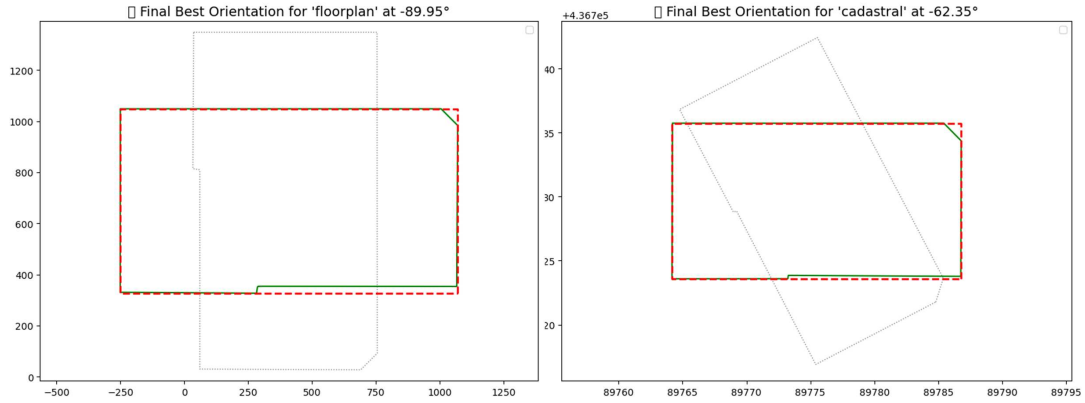


Figure C.3.: Comparison of Floor plan and parcel polygon orientation

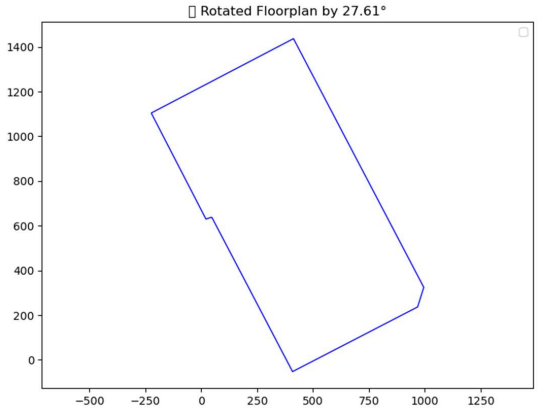


Figure C.4.: Rotated Floor plan

Bibliography

AHN. Kwaliteitsbeschrijving, February 2020. URL https://www.ahn.nl/kwaliteitsbeschrijving. Publisher: AHN.

Rubén Alonso, Mikel Borras, Rembrandt H. E. M. Koppelaar, Alessandro Lodigiani, Eduard Loscos, and Emre Yöntem. SPHERE: BIM Digital Twin Platform. In *Sustainable Places 2019*, page 9. MDPI, July 2019. doi: 10.3390/proceedings2019020009. URL https://www.mdpi.com/2504-3900/20/1/9.

Autodesk. What is a digital twin? Intelligent data models shape the built world, 2024. URL https://www.autodesk.com/design-make/articles/what-is-a-digital-twin.

Marc Baauw. Maintaining an up to date digital twin by direct use of point cloud data. Master's thesis, Delft University of Technology, 2021. URL https://gdmc.nl/publications/2021/MScThesisMarcBaauw.pdf.

Jesús Balado, Lucía Díaz-Vilariño, Pedro Arias, and Henrique Lorenzo. Point clouds for direct pedestrian pathfinding in urban environments. *ISPRS Journal of Photogrammetry and Remote Sensing*, 148:184–196, February 2019. ISSN 0924-2716. doi: 10.1016/j.isprsjprs.2019.01.004. URL https://www.sciencedirect.com/science/article/pii/S0924271619300048.

C. Beil, T. Kutzner, B. Schwab, B. Willenborg, A. Gawronski, and T. H. Kolbe. Integration of 3D Point Clouds with Semantic 3D City Models – Providing Semantic Information Beyond Classification. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, VIII-4-W2-2021:105–112, October 2021. ISSN 2194-9042. doi: 10.5194/isprs-annals-VIII-4-W2-2021-105-2021. URL https://isprs-annals.copernicus.org/articles/VIII-4-W2-2021/105/2021/isprs-annals-VIII-4-W2-2021-105-2021.html. Conference Name: ISPRS TC IV
16th 3D GeoInfo Conference 2021 - 11–14 October 2021, New York City, USA Publisher: Copernicus GmbH.

Doug A. Bowman, Ryan P. McMahan, and Eric D. Ragan. Questioning naturalism in 3D user interfaces. *Communications of the ACM*, 55(9):78–88, September 2012. ISSN 0001-0782, 1557-7317. doi: 10.1145/2330667.2330687. URL https://dl.acm.org/doi/10.1145/2330667.2330687.

Jarosław Bydłosz, Artur Warchoł, Monika Balawejder, and Agnieszka Bieda. Practical verification of Polish 3D cadastral model. 2021. doi: 10.4233/uuid:884b0c33-0d8e-40fd-bb88-669b21798a65. URL https://doi.org/10.4233/uuid:884b0c33-0d8e-40fd-bb88-669b21798a65.

Barbara Cemellini. Web-based visualization of 3D cadastre. Technical report, 2018.

Barbara Cemellini, Peter van Oosterom, Rod Thompson, and Marian de Vries. Design, development and usability testing of an LADM compliant 3D Cadastral prototype system. *Land Use Policy*, 98, November 2020. ISSN 02648377. doi: 10.1016/j.landusepol.2019.104418. Publisher: Elsevier Ltd.

Grebstew. grebtsew/FloorplanToBlender3d: Create 3d rooms in blender from floorplans., 2021. URL https://github.com/grebtsew/FloorplanToBlender3d.

Eric Hagemans. Development in Cadastral Surveying and Mapping in the Netherlands: About the Improved Cadastral Map of The Netherlands: Kadastrale Kaart Next. *Kart og Plan*, 117(2):230–240, August 2024. ISSN 0047-3278, 2535-6003. doi: 10.18261/kp.117.2.10. URL https://www.scup.com/doi/10.18261/kp.117.2.10.

Tommy Hinks, Hamish Carr, Hamid Gharibi, and Debra F. Laefer. Visualisation of urban airborne laser scanning data with occlusion images. *ISPRS Journal of Photogrammetry and Remote Sensing*, 104:77–87, June 2015. ISSN 0924-2716. doi: 10.1016/j.isprsjprs.2015.01.014. URL https://www.sciencedirect.com/science/article/pii/S0924271615000325.

Jin Huang, Jantien Stoter, Ravi Peters, and Liangliang Nan. City3D: Large-Scale Building Reconstruction from Airborne LiDAR Point Clouds. *Remote Sensing*, 14(9):2254, January 2022. ISSN 2072-4292. doi: 10.3390/rs14092254. URL https://www.mdpi.com/2072-4292/14/9/2254. Number: 9 Publisher: Multidisciplinary Digital Publishing Institute.

E Kalogianni. Linking the Legal with the Physical Reality of 3D Objects in the Context of Land Administration Domain Model (LADM). Master's thesis, Delft University of Technology Delft, The Netherlands, 2016.

Eftychia Kalogianni, Peter Van Oosterom, Efi Dimopoulou, and Christiaan Lemmen. 3D Land Administration: A Review and a Future Vision in the Context of the Spatial Development Lifecycle. *ISPRS International Journal of Geo-Information*, 9(2):107, February 2020. ISSN 2220-9964. doi: 10.3390/ijgi9020107. URL https://www.mdpi.com/2220-9964/9/2/107.

Abdullah Kara, Christiaan Lemmen, Peter Van Oosterom, Eftychia Kalogianni, Abdullah Alattas, and Agung Indrajit. Design of the new structure and capabilities of LADM edition II including 3D aspects. *Land Use Policy*, 137:107003, February 2024. ISSN 02648377. doi: 10.1016/j.landusepol.2023.107003. URL https://linkinghub.elsevier.com/retrieve/pii/S0264837723004696.

R. G. Kippers, M. Koeva, M. Van Keulen, and S. J. Oude Elberink. Automatic 3D Building Model Generation Using Deep Learning Methods Based on CityJSON and 2D Floor Plans. In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, volume 46, pages 49–54. International Society for Photogrammetry and Remote Sensing, October 2021. doi: 10.5194/isprs-archives-XLVI-4-W4-2021-49-2021. Issue: 4/W4-2021 ISSN: 16821750.

Mila Koeva, Shayan Nikoohemat, Sander Oude Elberink, Javier Morales, Christiaan Lemmen, and Jaap Zevenbergen. Towards 3D indoor cadastre based on change detection from point clouds. *Remote Sensing*, 11(17), 2019. ISSN 20724292. doi: 10.3390/rs11171972. Publisher: MDPI AG.

Ministerie van Binnenlandse Zaken en Koninkrijksrelaties. Uitvoeringsregeling Kadasterwet 1994. URL https://wetten.overheid.nl/BWBR0006596/2017-03-10/#Hoofdstuk2_Artikel6. Last Modified: 2024-02-24.

Hugo Ledoux, Ken Arroyo Ohori, Ravi Peters, and Maarten Pronk. Computational modelling of terrains. 2023.

CHJ Lemmen, MC Chipofya, A Da Silva Mano, Abdullah Kara, Dennis Ushiña Huera, Peter JM van Oosterom, Eftychia Kalogianni, Eva-Maria Morscher-Unger, JM Morales Guarin, Anthony Beck, and others. LADM in the Classroom. 2025. Publisher: International Federation of Surveyors (FIG).

Christiaan Lemmen, Peter Van Oosterom, and Rohan Bennett. The Land Administration Domain Model. *Land Use Policy*, 49:535–545, December 2015. ISSN 02648377. doi: 10.1016/j.landusepol.2015.01.014. URL https://linkinghub.elsevier.com/retrieve/pii/S0264837715000174.

Chen Liu, Jiajun Wu, Pushmeet Kohli, and Yasutaka Furukawa. Raster-to-Vector: Revisiting Floorplan Transformation. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2214–2222, October 2017. doi: 10.1109/ICCV.2017.241. URL https://ieeexplore.ieee.org/document/8237503. ISSN: 2380-7504.

Xianghuan Luo, Rohan Bennett, Mila Koeva, and Nathan Quadros. Cadastral boundaries from point clouds?: Towards semi-automated cadastral boundary extraction from ALS data. *GIM international*, 30:16–17, 2016. Publisher: GITC BV.

Niek Manders. Comparing AHN point clouds for their performance in representing 3D buildings in Zuid-Holland: A quantitative and qualitative performance review between AHN3 and AHN4, 2023. URL https://www.gdmc.nl/publications/2023/MScThesisNiekManders.pdf.

Ping Mao, Peter van Oosterom, and Azarakhsh Rafiee. A digital twin based on Land Administration. 2024.

Roeland Willem Erik Meulmeester. BIM Legal: Proposal for defining legal spaces for apartment rights in the Dutch cadastre using the IFC data model. 2019.

Tran Duong Nguyen and Sanjeev Adhikari. The Role of BIM in Integrating Digital Twin in Building Construction: A Literature Review. *Sustainability*, 15(13):10462, January 2023. ISSN 2071-1050. doi: 10.3390/su151310462. URL https://www.mdpi.com/2071-1050/15/13/10462. Number: 13 Publisher: Multidisciplinary Digital Publishing Institute.

Bas Nottrot, Erwin Folmer, Debraj Roy, Bob Scheer, and Peter Merx. Multi-unit building address geocoding: An approach without indoor location reference data. *Transactions in GIS*, 27(1):57–83, February 2023. ISSN 1361-1682, 1467-9671. doi: 10.1111/tgis.13017. URL https://onlinelibrary.wiley.com/doi/10.1111/tgis.13017.

Peter van Oosterom, Jantien Stoter, Hendrik Ploeger, Rod Thompson, and Sudarshan Karki. World-wide Inventory of the Status of 3D Cadastres in 2010 and Expectations for 2014. *Bridging the Gap between Cultures*, 2011.

Zaid Osama. The digital twin framework: A roadmap to the development of user- centred digital twin in the built environment. *Journal of Building Engineering*, 98:111081, December 2024. ISSN 23527102. doi: 10.1016/j.jobe.2024.111081. URL https://linkinghub.elsevier.com/retrieve/pii/S2352710224026494.

JungHo Park, WonGeun Choi, TaeYun Jeong, and JinJae Seo. Digital twins and land management in South Korea. *Land Use Policy*, 124:106442, January 2023. ISSN 02648377. doi: 10.1016/j.landusepol.2022.106442. URL https://linkinghub.elsevier.com/retrieve/pii/S0264837722004690.

Jacynthe Pouliot, Claire Ellul, Frédéric Hubert, Chen Wang, Abbas Rajabifard, Mohsen Kalantari, Davood Shojaei, Behnam Atazadeh, and Peter VAN Oosterom. 3D Cadastres Best Practices, Chapter 5: Visualization and New Opportunities. 2018.

Florent Poux. The Smart Point Cloud Model: Integration of point intelligence. December 2019. URL https://orbi.uliege.be/handle/2268/242190.

Davood Shojaei, Mohsen Kalantari, Ian D. Bishop, Abbas Rajabifard, and Ali Aien. Visualization requirements for 3D cadastral systems. *Computers, Environment and Urban Systems*, 41:39–54, September 2013. ISSN 01989715. doi: 10.1016/j.compenvurbsys.2013.04.003. URL https://linkinghub.elsevier.com/retrieve/pii/S0198971513000422.

A. Steed. Towards a General Model for Selection in Virtual Environments. In *3D User Interfaces (3DUI'06)*, pages 103–110, March 2006. doi: 10.1109/VR.2006.134. URL https://ieeexplore.ieee.org/document/1647515.

Jing Sun, Siying Mi, Per-ola Olsson, Jenny Paulsson, and Lars Harrie. Utilizing BIM and GIS for Representation and Visualization of 3D Cadastre. *ISPRS International Journal of Geo-Information*, 8(11):503, November 2019. ISSN 2220-9964. doi: 10.3390/ijgi8110503. URL https://www.mdpi.com/2220-9964/8/11/503. Number: 11 Publisher: Multidisciplinary Digital Publishing Institute.

Christian Tiberius, Hans Van Der Marel, Rene Reudink, and Freek Van Leijen. *Surveying and Mapping*. TU Delft OPEN Publishing, 2022. ISBN 978-94-6366-489-9. doi: 10.5074/T.2021.007. URL https://textbooks.open.tudelft.nl/textbooks/catalog/book/46.

Matthias Trapp, Christian Beesk, Sebastian Pasewaldt, and Jürgen Döllner. Interactive Rendering Techniques for Highlighting in 3D Geovirtual Environments. In Thomas H. Kolbe, Gerhard König, and Claus Nagel, editors, *Advances in 3D Geo-Information Sciences*, pages 197–210. Springer, Berlin, Heidelberg, 2011. ISBN 978-3-642-12670-3. doi: 10.1007/978-3-642-12670-3_12. URL https://doi.org/10.1007/978-3-642-12670-3_12.

Peter Van Oosterom. Research and development in 3D cadastres. *Computers, Environment and Urban Systems*, 40:1–6, July 2013. ISSN 01989715. doi: 10.1016/j.compenvurbsys.2013.01.002.

George Vosselman, editor. *Airborne and terrestrial laser scanning*. Whittles Publ, Caithness, repr edition, 2011. ISBN 978-1-904445-87-6 978-1-4398-2798-7.

Chen Wang, Jacynthe Pouliot, and Frédéric Hubert. Visualization Principles in 3D Cadastre: A First Assessment of Visual Variables. 2012.

Fang Wang and Zijian Zhao. A survey of iterative closest point algorithm. In *2017 Chinese Automation Congress (CAC)*, pages 4395–4399, October 2017. doi: 10.1109/CAC.2017.8243553. URL https://ieeexplore.ieee.org/abstract/document/8243553.

Paul R. Wolf, Bon A. Dewitt, and Benjamin E. Wilkinson. Coordinate Transformations. McGraw-Hill Education, New York, 4th edition edition, 2014. ISBN 978-0-07-176112-3. URL https://www.accessengineeringlibrary.com/content/book/9780071761123/back-matter/appendix3.

Xuetao Yin, Peter Wonka, and Anshuman Razdan. Generating 3D Building Models from Architectural Drawings: A Survey. *IEEE Computer Graphics and Applications*, 29(1):20–30, January 2009. ISSN 1558-1756. doi: 10.1109/MCG.2009.9. URL https://ieeexplore.

ieee.org/document/4736453/?arnumber=4736453. Conference Name: IEEE Computer Graphics and Applications.

Wuming Zhang, Jianbo Qi, Peng Wan, Hongtao Wang, Donghui Xie, Xiaoyan Wang, and Guangjian Yan. An Easy-to-Use Airborne LiDAR Data Filtering Method Based on Cloth Simulation. *Remote Sensing*, 8(6):501, June 2016. ISSN 2072-4292. doi: 10.3390/rs8060501. URL https://www.mdpi.com/2072-4292/8/6/501. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

Volkan Çağdaş, Abdullah Kara, Anka Lisec, Jesper M. Paasch, Jenny Paulsson, Tanja L. Skovsgaard, and Amalia Velasco. Determination of the property boundary – A review of selected civil law jurisdictions. *Land Use Policy*, 124:106445, January 2023. ISSN 02648377. doi: 10.1016/j.landusepol.2022.106445. URL https://linkinghub.elsevier.com/retrieve/pii/S0264837722004720.

Colophon

This document was typeset using LaTeX, using the KOMA-Script class `scrbook`. The main font is Palatino.