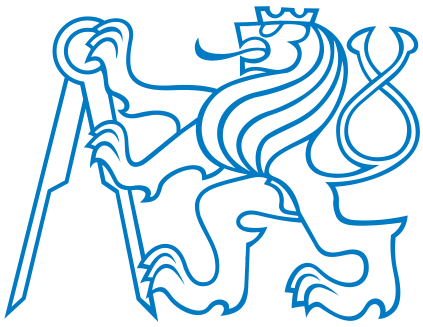
From Point Cloud to Digital Twin: Streamlining BIM Design Automation

Bc. Slávek Zbirovský

Supervisor: **doc. Ing. Václav Nežerka, Ph.D.**



Czech Technical University in Prague

Faculty of Civil Engineering

Department of Physics

Diploma thesis

January 2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Zbirovský** Jméno: **Slávek** Osobní číslo: **484477**

Fakulta/ústav: **Fakulta stavební**

Zadávající katedra/ústav: **Katedra fyziky**

Studijní program: **Budovy a prostředí**

Studijní obor: **Budovy a prostředí**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

From Point Cloud to Digital Twin: Streamlining BIM Design Automation

Název diplomové práce anglicky:

From Point Cloud to Digital Twin: Streamlining BIM Design Automation

Pokyny pro vypracování:

Tvorba BIM modelu z mračna bodů s využitím programovacího jazyka Python: 1) příprava mračna bodů (ředění, ořezání), 2) načtení a filtrování dat pomocí programovacího jazyka Python (distribuce hustoty bodů v prostoru), 3) segmentace mračna bodů a identifikace stavebních prvků, 4) tvorba IFC modelu

Seznam doporučené literatury:

John V. Guttag, 2016. Introduction to Computation and Programming Using Python, MIT Press Ltd

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Václav Nežerka, Ph.D. katedra fyziky FSv

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **25.09.2023** Termín odevzdání diplomové práce: **08.01.2024**

Platnost zadání diplomové práce: _____

doc. Ing. Václav Nežerka, Ph.D. podpis vedoucí(ho) práce	prof. Ing. Jiří Novák, Ph.D. podpis vedoucí(ho) ústavu/katedry	prof. Ing. Jiří Máca, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Declaration

I hereby declare that I have prepared my diploma thesis independently under the supervision of doc, Ing. Václav Nežerka, Ph.D., using the literature and sources listed in the appendix to this thesis.

In Prague, 8 January 2024

Bc. Slávek Zbirovský

Acknowledgments

I would like to express my sincere gratitude to Václav Nežerka, for his invaluable guidance and unwavering support throughout the process of completing this diploma thesis. His expertise, patience, and insightful feedback were instrumental in shaping this work. Additionally, I am truly thankful to Jan Valentin for the opportunity to contribute to the Reconmatic project, which provided me with valuable experiences and skills. Moreover, I would like to extend my appreciation to Cegra and Graphisoft for facilitating the provision of a complimentary ArchiCAD software license. Last but not least, I am grateful to my friends and family, especially my parents, for their unwavering encouragement and support during my studies. Their constant backing has been an essential source of motivation throughout this academic journey.

Funding This work was supported by the European Union's Horizon Europe Framework Programme (call HORIZON-CL4-2021-TWIN-TRANSITION-01-11) under grant agreement No. 101058580, project RECONMATIC (Automated solutions for sustainable and circular construction and demolition waste management), the Technology Agency of the Czech Republic, grant agreement No. S03010302 (Development of efficient tools to minimize production of construction and demolition waste, its monitoring and reuse), and the Czech Technical University in Prague, grant agreement No. SGS23/004/OHK1/1T/11 (Development and application of digitized and automated tools for multi-purpose applications in the construction sector).

Abstract

The aim of this diploma thesis is to design, develop, and test software that automates the Scan-to-BIM process. This process involves segmentation of point clouds and recognition of 3D entities. The proposed solution incorporates a segmentation algorithm based on point cloud density analysis, further enhanced by image and morphological operations. The developed solution is capable of extracting the geometry of building elements such as ceiling slabs, walls, windows, and doors, and generate output in the IFC format, thereby ensuring compatibility with other software tools. The motivation for automating the conversion of point cloud data into a 3D BIM model stems from the need for more straightforward and efficient decision-making at the end of a building's life cycle. Using digitized building models would lead to a more efficient decomposition and reuse of building elements/materials.

Keywords: Point Cloud, Automation, 3D BIM model, IFC, Python

Abstrakt

Cílem této diplomové práce je navrhnout, vyvinout a otestovat software, který automatizuje proces Scan-to-BIM. Při tomto procesu dochází k prokládání mračna bodů parametrizovatelnými 3D entitami. Navrhované řešení zahrnuje segmentační algoritmus založený na analýze hustoty mračna bodů, dále rozšířený o obrazové a morfologické operace. Vyvinuté řešení je schopno extrahovat geometrii stavebních prvků, jako jsou stropní desky, stěny, okna a dveře, a generovat výstup ve formátu IFC, čímž zajišťuje kompatibilitu s jinými nástroji. Motivace pro automatizaci převodu dat z mračna bodů do 3D BIM modelu pramení z potřeby přímočařejšího a efektivnějšího rozhodování na konci životního cyklu budovy. Tento přístup vede k efektivnějšímu využití stavebních prvků/materiálů při demolicích nebo dekonstrukci.

Klíčová slova: Mračno bodů, Automatizace, 3D BIM model, IFC, Python

Contents

Goals

The primary objective of this work was to develop a software tool enabling fully automatized conversion of point clouds to Building Information Models (BIMs). The specific goals were as follows:

- Optimize point cloud data quality through reduction and cropping.
- Utilize Python for data loading and filtering, including analysis of point density distributions in space.
- Develop algorithms for segmentation of the point clouds into individual building elements such as ceiling slabs, walls, and openings.
- Generate Industry Foundation Classes (IFC) model from the obtained geometry.

Chapter 1

Introduction

The development in the field of construction digitization has seen a sharp increase in the last decade, bringing forth a range of new technologies and methods for efficient management of construction projects. A frequently mentioned concept is Construction 4.0, which focuses on the integration of modern digital technologies and innovations into the construction industry to enhance the efficiency, quality, safety, and sustainability of construction projects [1].

Sustainability has become a highly debated topic in the context of buildings that are approaching the end of their lifespan. Nowadays, there is a growing emphasis on ecological responsibility and efforts to minimize the negative environmental impacts of construction. Structures built several decades ago often do not meet current sustainability and energy efficiency standards. Hence, the matter of dealing with aging structures that have reached the end of their operational life is steadily gaining significance. As per Eurostat [2], in the typical European nation, construction and demolition waste make up 36% of the total waste produced. There exist numerous possibilities for converting these edifices into structures that are not only more environmentally sustainable but also eco-friendly.

One option is renovation and modernization, which involves improving insulation, replacing windows and doors with more energy-efficient options, installing renewable energy sources, and taking other steps toward reducing CO_2 emissions and increasing energy efficiency [3]. Another option is deconstruction and material recycling from the original building. This approach helps minimize waste and reduces the need for mining new raw materials [4].

For both of the proposed solutions, there is often an issue with the existing documentation of old buildings, which is typically in inadequate condition or only available in paper form. This situation requires a careful and systematic approach to obtaining up-to-date and accurate data about the building. Once new digital data is available, it is essential to systematically document and store it for future use. Digital documentation can include a 3D model, point clouds, and the original documentation converted into electronic format [5, 6].

Ensuring up-to-date and accurate documentation is a crucial element for the successful execution of renovations, modernization, and sustainable measures in old buildings. It is essential not only for planning and implementing changes but also for compliance with building regulations and ensuring the future sustainability of these structures. One option to obtain precise digitized documentation of the as built state is to utilize point cloud technology through laser scanning or digital photogrammetry [7].

The use of point cloud data for the creation of a 3D model or documentation of the current state of a building has become a common practice in recent years. For instance, the renovation of heritage assets such as the Durham Cathedral extensively utilized point cloud technology to capture details of the cathedral's architecture [8]. Similarly, the digital documentation of The Engine House of Paços Reais employed point cloud data to accurately record the historical structure's architectural elements [9], another project describes the digitization of the state opera house in Hungary for the upcoming reconstruction of the building [10].

However, the creation process, also called Scan-to-BIM, is very lengthy and requires the skill of the user and advanced modeling software. For these reasons, semi-automatic [11, 12, 13, 14] and automatic solutions [15, 16] for segmenting point cloud data and obtaining the exact geometry of common building elements began to appear. Kwadjo et al. [17] implemented a robust methodology based on a RANSAC and 2D matrix template of wall representation, however multiple floors are not supported and the output is delivered in the nowadays obsolete version of the IFC2x3 format. Conversely, Romero Jaren [18] uses a clustering algorithm in combination with Delaunay triangulation to find surfaces. The output was 3D surfaces in vector format for planar surfaces and TIN format for non-planar elements. Chao Wang [19] implemented region growing plane segmentation algorithm in combination with an edge and boundary detection. This approach successfully converts the data into a gbXML format for energy simulations and demonstrates potential for broader applications in the AEC/FM domain. Overall, these papers demonstrate the ongoing efforts to develop fully automatic solutions for converting point clouds to BIM, aiming to improve efficiency and accuracy in the modeling process.

The most commonly segmented elements include wall, ceiling slab, windows and doors. Algorithms based on iterations were used in some cases to obtain accurate geometry. One of them is RANSAC, which can detect simple shapes like surfaces, spheres, cylinders [20, 21] or the Hough transformation [22, 23, 24]. Other algorithms used include region growing algorithms [19, 25]. By combining the right methods for different building elements, an efficient tool can be created to convert point cloud data into an accurate, easily parameterizable 3D model filled with non-graphical information. This model can help in deciding what will happen to the building at the end of its life cycle, it can also serve as a basis for the upcoming reconstruction, or to determine the amount and types of demolition waste that will be generated at the end of the building's life.

Chapter 2

Theoretical Background

2.1 Building Information Modeling

Building Information Modeling (BIM) is a comprehensive approach for modeling, constructing, and managing buildings and infrastructure. It is a way in which building information is integrated, created, and digitally shared throughout the entire lifecycle of a building. BIM transforms how professionals in the construction industry work and collaborate, providing a range of benefits and opportunities. One of the key elements is the digital model of a building or infrastructure that encompasses information about the geometry, materials, properties, and functions of the construction object.

Most sources suggest that the concept of BIM originated with Professor Charles Eastman [26] as early as 1975. In his article, he presented ideas that today form the foundation of every methodology for implementing BIM processes. These included the 3D representation of objects, collision detection, and a database of objects used in the project. However, the BIM concept is not the brainchild of a single individual; rather, it is the result of several decades of innovation and collaboration. This evolution began with the development of the first 2D CAD programs and progressed through the advent of 3D visualization tools, culminating in software that enables the comprehensive capture of all project data.

Nowadays, BIM is mainly used for the design and implementation of construction works, however, in recent years there have been cases of use in End of Life cycle of building management [27, 28]. The core of all analyzes is the Building Information Model (BIM). It is a 3D geometric model, which is filled with non-graphical information such as textures, materials, physical properties of materials, documents related to construction, revision protocols, etc. In addition, compared to traditional documentation, it is supplemented with a time dimension (4D) enabling construction planning, financial costs (5D) related to the cost analysis before the start of the construction phase, sustainability (6D) focusing on the analysis of energy consumption and the assessment of the overall serviceability of the building. A model filled with this information is called an n-Dimensional model [29].

2.1.1 Digital Twin

It is a data-driven replica of a physical asset or system, including real-time data and simulation capabilities. The concept that would evolve into the digital twin (DT) first emerged in a University of Michigan presentation to industry in 2002, under the name Product Lifecycle Management. Its primary purpose was to monitor the life cycle of a product, and it possessed foundational features of what would become known as a Digital Twin. The principal similarity was the flow of data between virtual replica and real product [30]. In 2012, NASA introduced its DT concept to create virtual replicas of physical assets, including satellites, spacecraft, and lunar modules. These models were designed to contain simulations based on real data obtained from the physical assets and their sensors [31]. Digital twin is utilized in the construction industry during the design phase [32]. It enables the project team to collaborate in a coordinated manner, thereby facilitating better decision-making and planning. Additionally, it proves useful in the field of computational simulation over the long term. During the building management phase, a digital twin can function as a product database, linked to information about revisions and technology service predictions [33]. Creating a 3D model of the actual state of the building using point cloud technology is a step towards creating a digital twin, but it may not be a full digital twin in itself.

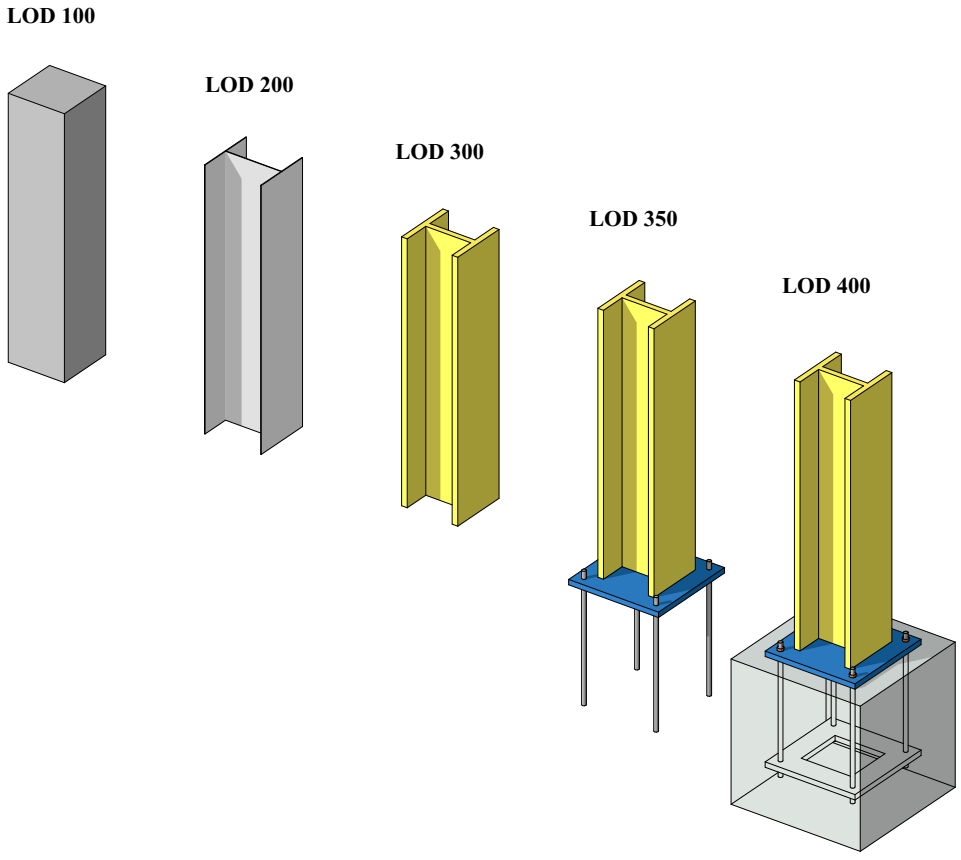


Figure 2.1: Level of Development of steel profile column. Author's own illustration based on [34].

2.1.2 Level of Development

Level of Development (LOD) is a concept used in BIM to specify the level of detail and accuracy of information within a BIM model at different stages of a project's lifecycle. It helps in standardizing and communicating the content and reliability of BIM data. Five levels starting with simplier LOD-100 to most detailed LOD-500 are defined, which specify the amount, accuracy and trustworthiness of the information transmitted within the process. The image 2.1 graphically shows the detail of individual LOD levels on the example of a steel profile column. It's also important to note that from the image shown in Figure 2.1, it might appear that the focus is solely on geometric accuracy, but this can be quite misleading. The various levels differ significantly, particularly in terms of the non-geometric information.

2.2 Industry Foundation Classes

The Industry foundation classes (IFC) standard was created for the effective sharing and interoperability of data between different software and systems in the field of construction and industry. The history of IFC dates back to the 1990s, the first version of IFC was published by the International Alliance for Interoperability (IAI), which later adopted the name BuildingSMART International. Since then, the IFC has been continuously updated and expanded to reflect technological developments and the needs of the construction industry. The IFC ADD2 TC1 standard and its schema an exchange file format for Building Information Model (BIM) data is described in ISO 16739-1:2018 [35]. IFC is not limited to geometrical building information. It also includes other aspects such as materials, schedule, cost, performance, energy efficiency and other key data that are important for project management and building operations.

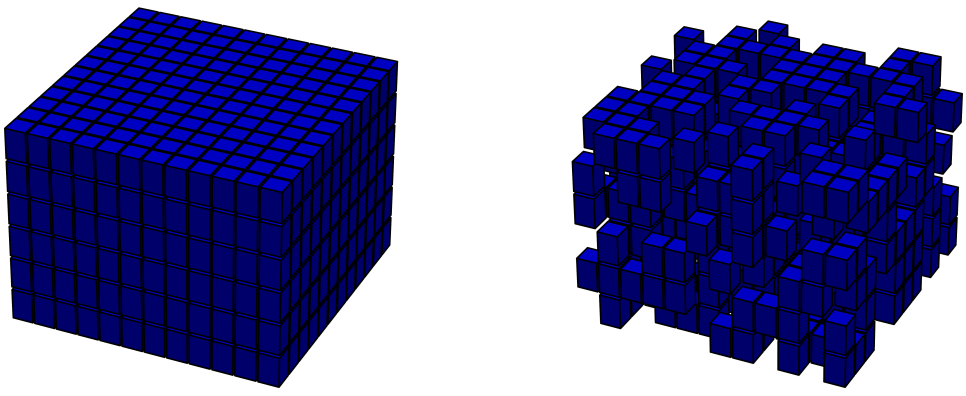


Figure 2.2: A graphical representation of MVD as a subset of IFC.
Author's own illustration based on [36].

2.2.1 IFC File Formats

Saving the BIM model is possible in different file formats, which are designed to facilitate the exchange and sharing of information between different software applications and construction professionals.

IFC: Standard binary format, written in STEP physical file structure according to ISO 10303-21 [37]. It's commonly used for storing and exchanging BIM data.

IFC-XML: Represents an XML-based version of IFC files, capable of direct generation by transmitting applications, adhering to the ISO 10303-28 structure [38], also recognized as STEP-XML. In comparison to IFC file formats, IFC-XML typically exhibits a size increase of about 300-400%.

IFC-ZIP: A compressed version of IFC or IFC-XML. This compression reduce .ifc file size by 60-80% and .ifc XML file by 90-90%. Is possible to unzip an IFC-ZIP file using the built-in Windows unzipper utility [39].

2.2.2 Model View Definition

Since each data transfer through IFC varies depending on its purpose, the Model View Definition (MVD), a subset of the complete IFC, was introduced. MVD aims to ensure standardized data and geometry exchange between software applications. Figure 2.2 illustrates the concept of MVD as a subset of IFC. Table 2.1 describes the most frequently used MVDs for IFC 4.0 ADD TC1 and IFC 3x2 schema.

Table 2.1: IFC model view definitions. Based on [40]

MVD Name	Schema	Summary
Reference View	IFC 4	Streamlined geometric and relational portrayal of spatial and physical elements for referencing model data in design coordination.
Design Transfer View	IFC 4	Improved geometric and relational representation of spatial and physical components to smoothly share model information between different tools. It's a high-quality one-way transfer of data and responsibility, not a "round-trip" transfer.
Coordination View	IFC 3x2	Components related to space and physical aspects for coordinating designs among architectural, structural, and building services (MEP) domains.
Structural Analysis View	IFC 3x2	Exchange of information related to structural analysis, including geometry, material properties, loads, and boundary conditions.

2.2.3 STEP Standard Format

In Section 2.2.1, the most commonly used format for transfer was mentioned. IFC standard is defined by the STEP physical file standard, ISO 10303-21 [37]. The file structure is divided into two main parts. The first, named HEADER (see Code listing 2.1), contains essential information about the file, such as the IFC schema version, model view definition, author, and more. At the beginning of this part, the structure of ISO 10303-21 is also referenced. The second part, DATA (see Code listing 2.2), includes information about the project. Each object in this section is assigned a unique identifier, which consists of a number and the # sign. Attributes that are not mandatory in the used classes can be substituted with the *$* character [41].

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('ViewDefinition [DesignTransferView_V1.0]'),'2;1');
FILE_NAME('output_IFC/output-2.ifc','2023-10-04T17:50:59.335336',('Slavek
    Zbirovsky'),('CTU in Prague'),'IfcOpenShell v0.7.0-476ab506d','Cloud2BIM
    ','None');
FILE_SCHEMA(('IFC4'));
ENDSEC;
```

Code Listing 2.1: IFC STEP physical file format Header part.

```
DATA;
#1=IFCSIUNIT(*,.LENGTHUNIT.,$,.METRE.);
#2=IFCSIUNIT(*,.AREAUNIT.,$,.SQUARE_METRE.);
#3=IFCSIUNIT(*,.VOLUMEUNIT.,$,.CUBIC_METRE.);
...
#6=IFCUNITASSIGNMENT((#1,#2,#3,#4,#5));
#7=IFCCARTESIANPOINT((0.,0.,0.));
#8=IFCDIRECTION((0.,0.,1.));
#9=IFCDIRECTION((1.,0.,0.));
#10=IFCAXIS2PLACEMENT3D(#7,#8,#9);
#11=IFCGEOMETRICREPRESENTATIONCONTEXT('Body','Model',3,0.0001,#10,$);
#14=IFCPROJECT('2Vtc6PDQLC8Bxh9U3OQjoc',$,'Sample project','Hotel Opatov','
    Hotel','Reconstruction','Deconstruction of non-load-bearing elements',$
    ,#6);
#15=IFCORGANIZATION($,'CTU in Prague',$,$,$);
#16=IFCAPPLICATION(#15,'version 1.0','Sample project','MY_IFC_APP');
#17=IFCPERSON($,'Zbirovsky','Slavek',$,$,$,$,$);
#18=IFCPERSONANDORGANIZATION(#17,#15,$);
#19=IFCOWNERHISTORY(#18,#16,$,.NOTDEFINED.,$,$,1699363081);
#20=IFCSITE('3LnRPF6M50EhjGwuWNUIPA',#19,'Site',$,$,$,$,$,.ELEMENT
    .,(50,5,0),(4,22,0),356.,$,$);
#21=IFCRELAGGREGATES('0Jat8oVN0Hxh9uL0NRwf9l',#19,'$','$',#14,(#20));
ENDSEC;
END-ISO-10303-21;
```

Code Listing 2.2: IFC STEP physical file format Data part.

Chapter 3

Point Cloud Processing

3.1 Point Cloud Data

Point cloud is the name for a group of 3D referenced points obtained by laser scanning or digital photogrammetry, which allows to display the surfaces of objects thanks to a dense network of points that represent the scanned surface. There are several reasons for using point cloud data, the most important of which include minimal measurement error, speed of data collection and overall capture of the scanned area and surroundings. There are many options for storing point cloud data in a file, they vary based on the type of information they may contain, and usually also depend on the type of equipment used to collect the data.Common contents of a point cloud file include: spatial coordinates (x, y, z), color (RGB), and intensity. The following table 3.1 lists the most commonly used formats for point cloud data transfer. The difference between these formats is in the way the data is written, readability for the user, and complexity.

Table 3.1: Point cloud data formats

Type	File extension	Characteristics
ASCII	.asc, .txt, .xyz, .pts	Plain text format that stores the values of XYZ coordinates in a tabular format. Each line corresponds to one point and may also include colors and intensities of points.
LAS	.las, .laz	Commonly used exchange format for LiDAR point cloud data; binary format. Large files can be compressed into LAZ format.
E57	.e57	Standard format for 3D imaging data exchange, including data from laser scans and 2D images.
PCG	.pcg	Autodesk's proprietary point cloud format used in its 2014 software suite.
PLY	.ply	3D geometry format (cloud or mesh) is made up of a header, a list of vertices, and a list of polygons.

The most common techniques used for data collection are laser scanning (LiDAR) or digital photogrammetry. The principle of laser scanning is to measure the distance between the sensor and the scanned object in space. The distance is calculated from the time it takes the beam to travel the distance between the scanner, the object to be scanned and the path back after reflection. The following formula is used for the calculation:

$$d = \frac{ct}{2}\,, \tag{3.1}$$

where d equals the distance, c is the speed of light and t is the time it took the beam to travel the distance. Geodesy describes this calculation of point coordinates as the spatial polar method. For the calculation, you need to know the following parameters: the scanner coordinates, the scanner rotation and the beam deflection angle in the scanner [42]. Figure 3.1 shows the principle of laser scanning. There are also other principles of how laser scanners can work, for example the principle of active triangulation, where the place of transmission and reception of the beam is separated and thus creates a triangle with the scanned object to calculate the exact location of points on the scanned surface. Another method is passive triangulation, in which there is no auxiliary light source, usually a laser. but the natural brightness of the scanned surface is used. A pair of devices with CCD or CMOS chips is most often used. The principle of stereovision, sometimes referred to as photogrammetry, is primarily employed due to its simplicity and compatibility with drones. Scans obtained through this method automatically capture the colors of the points [43, 44].

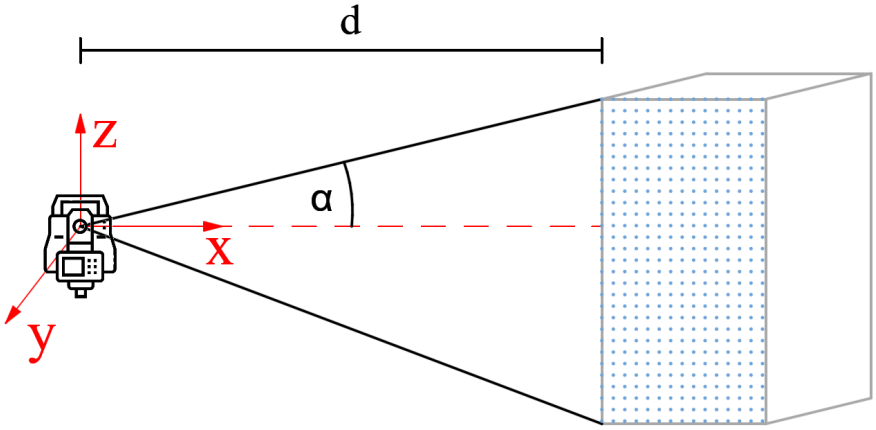


Figure 3.1: Principle of Laser Scanning. "α" represents the elevation angle, and "d" denotes the horizontal distance.

3.2 Delaunay Triangulation

Delaunay triangulation is one of the most widely used algorithms for creating triangular networks, mainly due to its wide range of applications and mathematical properties. The triangulation method, which maximizes the minimum angle in a triangular network, was introduced by Boris Delaunay in 1934 [45]. This algorithm is well documented and has wide support in various programming languages and libraries, making it easy to implement and use in different applications. In practice, it is used, for example, to create digital terrain models [46], fingerprint recognition [47], erosion modelling [48] or in combination with machine learning techniques to reconstruct surfaces from point cloud data [49]. The fundamental criterion of this triangulation specifies that a triangular network is Delaunay if no point in the set of points connected by the triangles formed falls within the circle/sphere circumscribed around that triangle. Figure 3.2 shows a triangular network created using the Delaunay triangulation algorithm, which is part of the SciPy Python library [50].

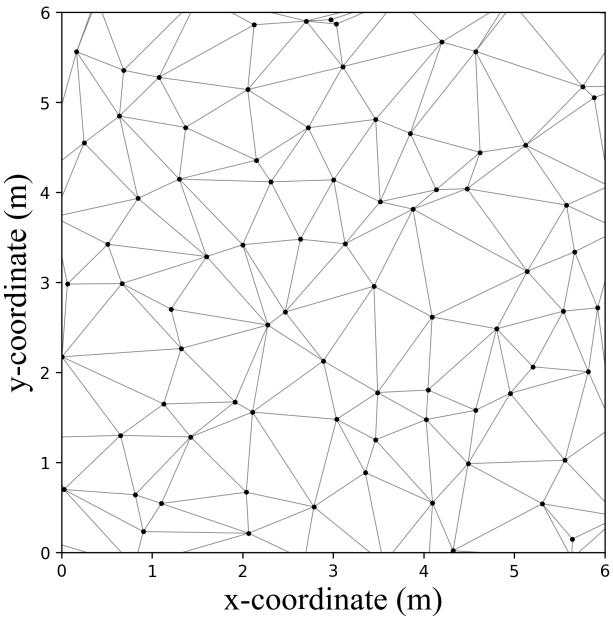


Figure 3.2: Delaunay triangulation for a mesh of points created using the SciPy library [50].

3.3 Alphashape

α-shape is a simple algorithm created by H. Edelsbrunner and P. Mücke [51]. This algorithm can be used to analyze shapes in a data set that is mostly represented by points in a two- or three-dimensional coordinate system. By applying the algorithm and determining the alpha parameter (α), the resulting shape of the envelope of these points can be determined. In general, a higher α value shows a more concave shape. For $\alpha = 0$, this alpha shape is equal to the convex hull of the figure. The change of the shape depending on the change of the parameter α can be seen in Figure 3.3 shapes were generated using the Python library of functions Alpha Shape Toolbox [52]. The algorithm removes edges that are longer than twice the value of α, while keeping shorter edges. This phenomenon was used in the scientific paper [53], where the roof shape envelope is extracted from LiDAR data using the α-shape.

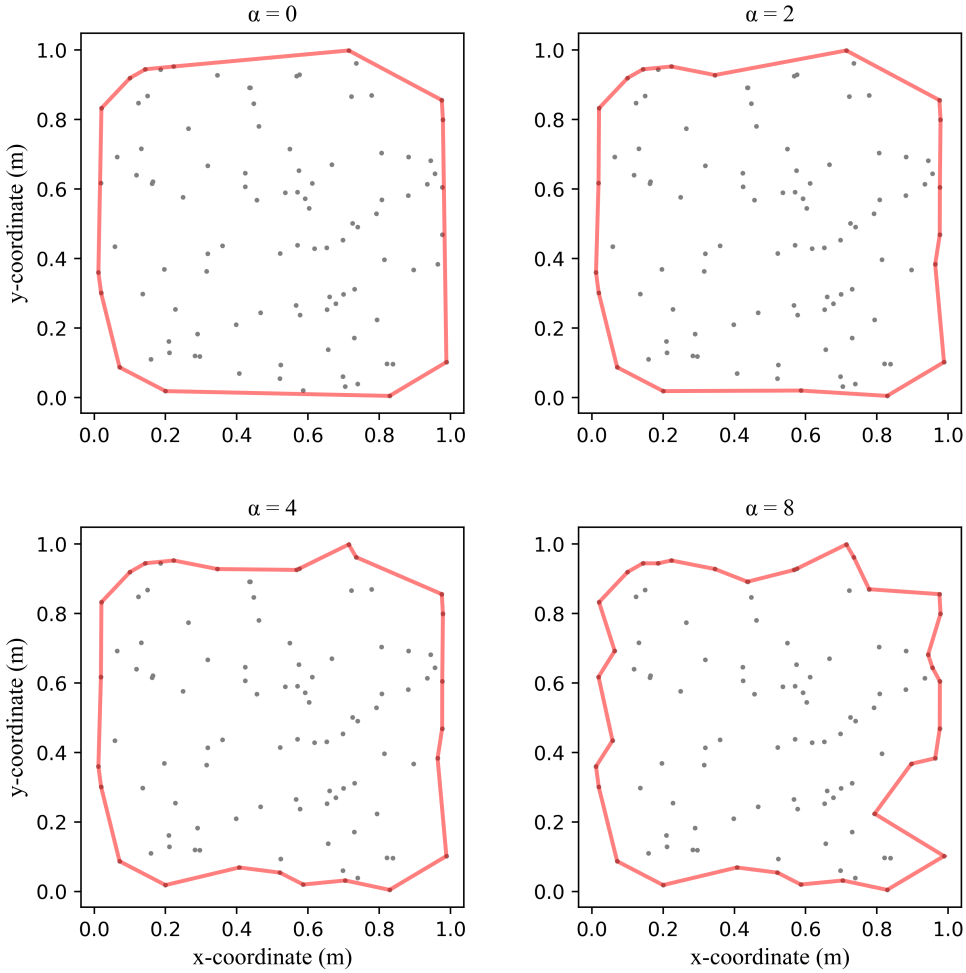


Figure 3.3: Changes of the α-shape depending on the change of the parameter α for a set of points. Created with Alpha Shape Toolbox [52].

3.4 Image Processing and Morphology Operations

Morphology operations are a set of image processing techniques used to process and manipulate the shapes and structures within binary and grayscale images. These operations are commonly employed in computer vision [54, 55], image analysis [56], and pattern recognition [57] to enhance or extract important features from images. The basic morphological operations include erosion, dilation, opening, and closing.

Binarization
Binarization is the process of converting an image (or any other data) into a binary (two-level) format. In the context of images, this means converting an image into one that has only two colors, typically black and white. Thresholding is one method to achieve binarization in images it is often used to simplify an image or highlight important parts. There are various types of thresholding methods according [58], broadly categorized as follows:

- Histogram shape-based methodologies involve the analysis of characteristics such as histogram peaks and valleys.
- Clustering-based approaches focus on the segmentation of gray-level samples into foreground and background components.
- Entropy-driven techniques rely on entropy measures for threshold determination.
- Object attribute-based methods quantify the similarity between gray-level and binarized images.
- Spatial methodologies take into account higher-order probability distributions and pixel correlations in the thresholding process.

Figure 3.4 shows the output of manual binarization, where a specifically selected threshold is used to transform the original image into binary form. This particular example uses the Numpy *np.where* function [59], which assigns a value of 255 to pixels in the original image that exceed a specified threshold, while pixels that are below the threshold are assigned a value of 0.

(a) Original image (b) Binarized image

Figure 3.4: Image binarization based on manual setting threshold.

Closing binary image
Closing is a morphological operation that consists of two steps: dilation followed by erosion. The dilation step expands the white regions in the binary image using the defined structuring element. It is a process of growing the white regions by including nearby white pixels. The erosion step then shrinks the white regions using the same structuring element. The combination of dilation and erosion with a square structuring element helps to close small holes and gaps in the binary image, making the regions more connected and continuous. The following Figure 3.5 shows the application of the *"closing"* morphological operation to a binary image. In this case, a 5x5 pixel square was used as the operational element. The function was applied in three iterations.

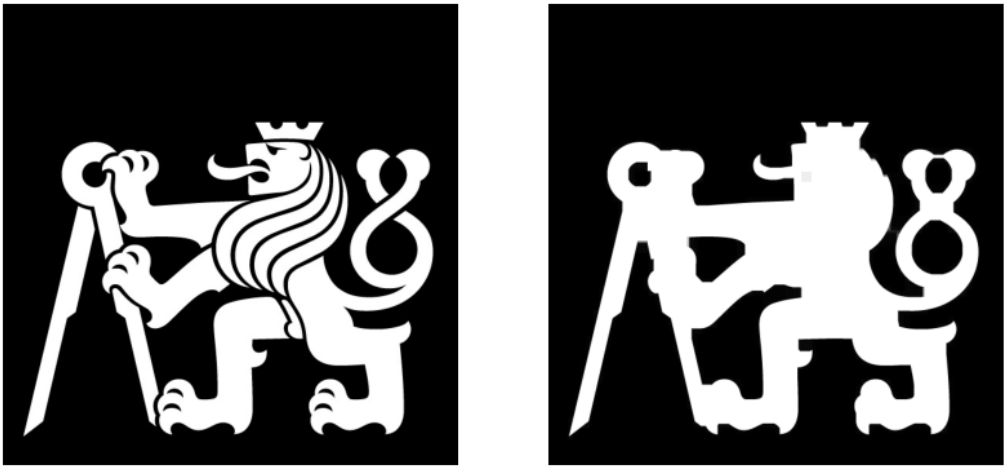


Figure 3.5: Application of closing morphology operation on binary image.

Contour detection
The *cv2.findContours* function is a fundamental image processing function provided by the OpenCV library (Open Source Computer Vision) [60]. It is used to detect and extract contours from binary images. Contours are continuous curves that form the boundaries of objects or regions in an image. This function is widely used in computer vision and image processing for various tasks, including object detection, shape analysis, and region identification.

The function has three variables, in addition to the input image, these are the parameters affecting the output. The first one retrieval mode determines the relationship and hierarchy of the contours found in the input image. There are several retrieval modes according [60]:

- *"cv2.RETR_LIST"* retrieves all the contours and doesn't create any hierarchy among them. Each contour is independent.
- *"cv2.RETR_EXTERNAL"* retrieves only the extreme outer contours.
- *"cv2.RETR_CCOMP"* retrieves all of the contours and organizes them into a two-level hierarchy. The top-level contours are the boundaries of the components, while the second-level contours are the holes within the components.
- *"cv2.RETR_TREE"* This mode retrieves all of the contours and reconstructs a full hierarchy of nested contours. Each contour is linked to its parent and child contours.

Second parameter method specifies the contour approximation. It controls how detailed the contour representation is. The possible values for this parameter are:

- *"cv2.CHAIN_APPROX_NONE"* stores all the contour points and doesn't approximate them. The resulting contours will have high accuracy.
- *"cv2.CHAIN_APPROX_SIMPLE"* compresses horizontal, vertical, and diagonal segments and leaves only their end points. For example, if a contour forms a straight line, this method will reduce it to just two end points.
- *"cv2.CHAIN_APPROX_TC89_L1"* This is the contour approximation method using Teh-Chin chain approximation algorithm. It calculates distances using the "L1" (Manhattan) norm, which is the sum of absolute differences in x and y coordinates [61]. It approximates the contour as a series of edges with a minimum distance between the original and the approximated contour.
- *"cv2.CHAIN_APPROX_TC89_KCOS"* It uses the Teh-Chin chain approximation algorithm but incorporates additional techniques for curve fitting and preserving the shape of the contour.

The following Figure 3.6 illustrates the application of the *cv2.findContours* function on a binary image. The detected contours are displayed as green polylines. The used retrieval mode was *cv2.RETR_EXTERNAL*, and the approximation method was *cv2.CHAIN_APPROX_SIMPLE*.

Figure 3.6: Image before (on the left) and after (on the right) contour detection, created using the OpenCV package [60]. The detected contours are displayed in green.

Douglas–Peucker curve approximation

The Douglas-Peucker algorithm is a method for simplifying or approximating a curve. It was developed by David Douglas and Thomas Peucker in 1973 and is commonly used in computer graphics [62], Geographic Information Systems (GIS) [63], and image processing. The primary purpose of this algorithm is to reduce the number of points used to represent a curve while preserving its basic shape and characteristics [64].

The algorithm recursively divides the polyline into segments. Initially, it considers all points between the first and last points. It automatically marks the first and last points as essential to preserve. Subsequently, the algorithm identifies the point that is furthest from the segment formed by the first and last points. This point represents the greatest deviation from the approximate straight line. If the distance of the point is less than epsilon from the line, then any points not yet marked for retention can be discarded without significantly affecting the accuracy of the simplified curve. However, if the distance of the furthest point from the approximation is more than epsilon, this point must be retained. The calculation of epsilon parameter is a critical step in approximating a contour with a simpler polygon. Epsilon is a parameter that controls the level of approximation, and it's based on the original contour's perimeter. In Figure 3.7, an approximation with parameter epsilon set to value 0.9.

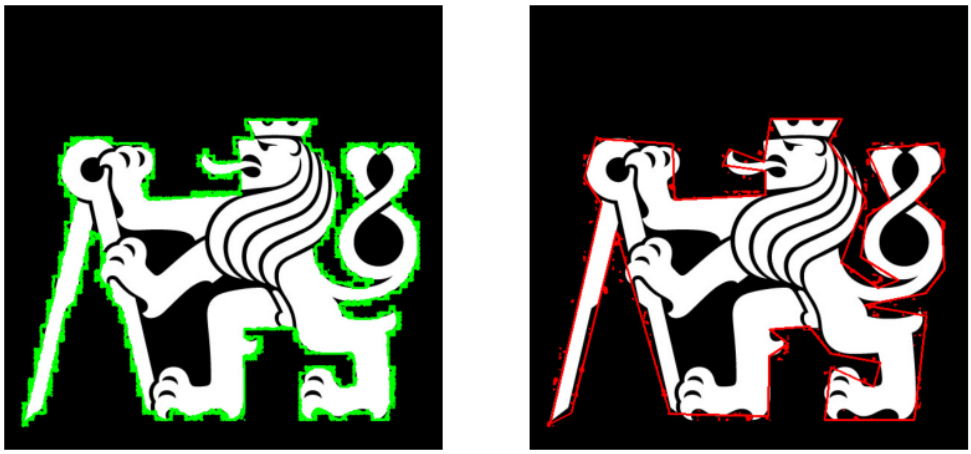


Figure 3.7: Original curve (on the left) simplified with Douglas-Peucker algorithm (on the right)[64].

Chapter 4

Implementation and Analysis

In the subsequent chapter "Implementation and Analysis," knowledge from Chapters 2 and 3 will be applied to describe the development of software, written in the Python programming language, for segmenting building elements from point cloud data into a 3D BIM model. Table 4.1 presents a summary of the specific software tools employed in the development, testing, and data preparation phases.

Table 4.1: Tools Utilized for Thesis Development

Software/Library	**Version**	**License**
Python	3.10	PSF license
CloudCompare	2.13.aplha	General public license
ArchiCAD	25 (4013)	Full license (128-18091115)
pye57	0.4.1	MIT license
pandas	2.1.3	BSD 3-Clause license
open3d	0.17.0	MIT license
alphashape	1.3.1	MIT license
matplotlib	3.8.1	PSF license
tqdm	4.66.1	MIT licence
numpy	1.26.2	BSD license
scikit-image	0.22.0	BSD 3-Clause license
opencv-python	4.8.1.78	MIT license
scipy	1.11.3	BSD 3-Clause license
ifcopenshell	0.7.0.231018	GNU lesser general public license

The complete code is available on the GitHub repository.

4.1 Data Preparation

For the segmentation of building elements, it was necessary to prepare a high-quality point cloud. This preparation involved two steps noise removal and point cloud subsampling. Subsampling the point cloud is a standard preprocessing operation that reduces the number of points, making the workflow more efficient. The CloudCompare program was used for subsampling data. This open-source program was originally developed for monitoring changes in point cloud data, but nowadays it is used for simple operations with point clouds, such as trimming and thinning. For point cloud subsampling, the program offers three methods listed in table 4.2. A subsampled point cloud retains the characteristics found in its original point cloud, including scalar fields, colors, and normals. Figure 4.1 illustrates the parameters employed to reduce the point count within a single test room. A minimum point distance mode was utilized, with the distance set to 1 cm, resulting in a reduction in the number of points from the original 5,008,195 to 245,544 points. The difference in appearance before and after the operation can be seen in Figure 4.2. To test the proposed algorithms, one room was cut out of the scanned object of the former hotel. This operation was performed again in the cloud Compare program using the Interactive Segmentation tool.

Table 4.2: Subsampling methods in CloudCompare software. Based on [65]

Method	**Summary**
Random	In "Random" mode, CloudCompare will simply select a specified number of points at random.
Space	The "Spatial" mode in CloudCompare works by allowing users to set a minimum distance between points. CloudCompare then selects points from the original point cloud in a way that ensures no two points in the resulting point cloud are closer to each other than the specified distance.
Octree-based	"Octree-based" subsampling reduces the number of points in a point cloud by dividing the space into smaller cubes and selecting representative points from each cube, simplifying the data while retaining its essential structure.

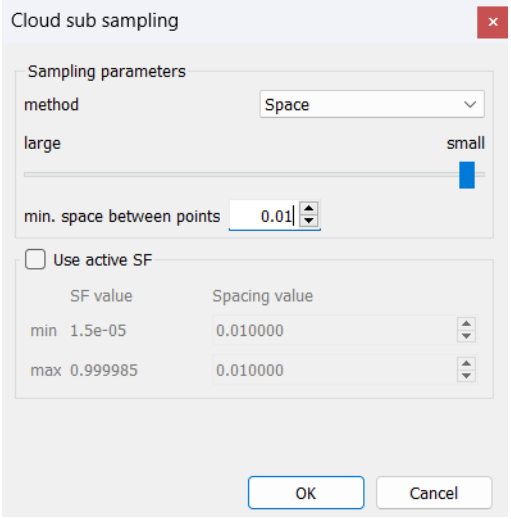
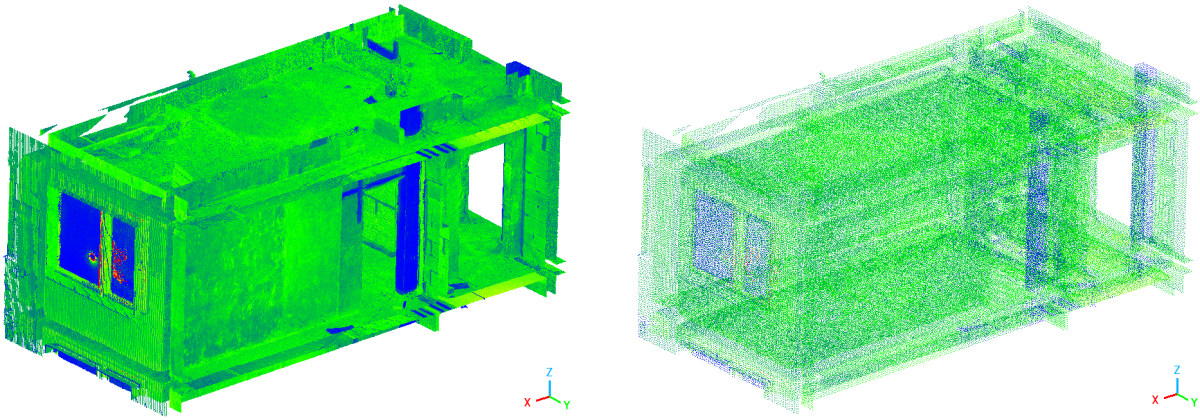


Figure 4.1: CloudCompare settings used for thinning the point cloud.



(a) Original (b) Thinned

Figure 4.2: Comparison of point cloud data before and after density thinning with CloudCompare.

4.2 Methods

Numerous algorithms have been developed to determine the precise geometry of common geometric shapes from a point cloud. When developing or selecting the most suitable algorithm, it is essential to consider how it handles data. Iterative algorithms may not be suitable, particularly when working with slower programming languages and extensive point cloud dataset. When segmenting the point cloud, the algorithm was consistently configured to leverage the characteristic features of the element, such as the orientation of the space. In the subsequent chapter, individual elements and their corresponding algorithms will be described sequentially.

4.2.1 Slab

Based on: Zbirovský, S. and Nežerka, V. 'Transforming Point Cloud Data into 3D BIM Model: A Case Study on Slabs (Preprint)', Acta Polytechnica CTU Proceedings.

The process of generating and situating slabs within a building involves several key steps. In the initial phase, the point cloud is divided into strips, which correspond to planes aligned with the x-y axes. Strips that exhibit a point density surpassing a predefined threshold (in our study, this threshold was established at 50%) in relation to the strip with the highest point count are singled out and extracted. In Figure 4.3, a section of the point cloud is shown along with a histogram displaying individual horizontal surfaces that are identifiable as peaks in the histogram. For these selected strips the median z-coordinate is determined based on the point index and they are subsequently designated as the lower or upper surface. Subsequently, we consolidate adjacent surfaces formed by points to create both the top and bottom surfaces of the slab.

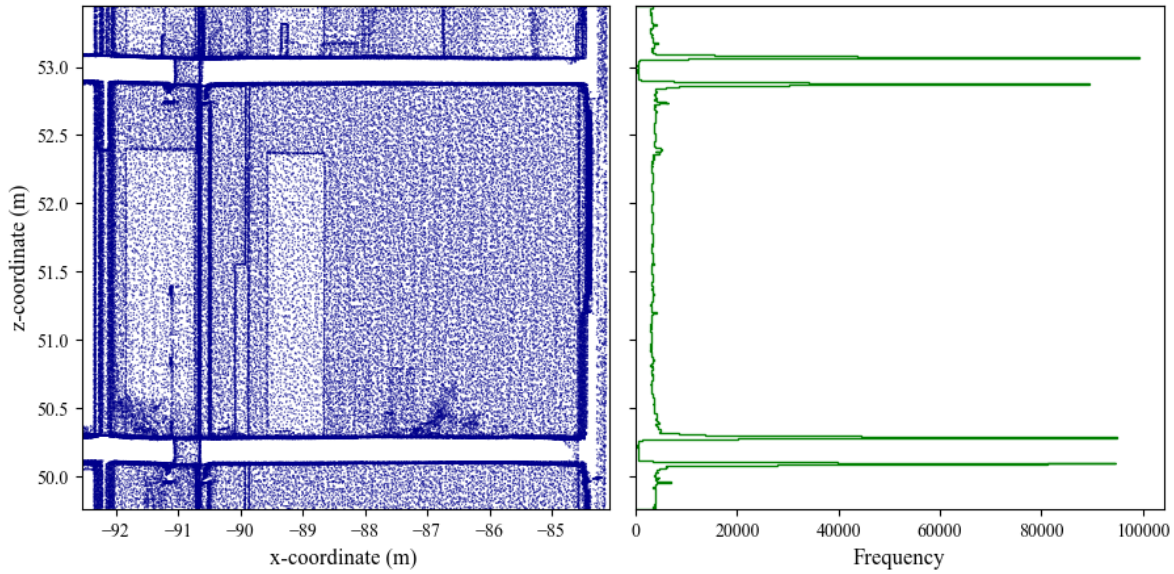


Figure 4.3: Point cloud cross-section (on the left). A z-coordinate histogram of point cloud data to detect slab surface candidates (on the right).

The final stage involves calling the *create_hull_alphashape* function. This function, as detailed in Section 3.3, is utilized to generate the hull and takes a collection of 2D points as its input. Starts the Delaunay triangulation process, then eliminates edges exceeding the specified alpha threshold, resulting in a non-convex polygon. The remaining edges define the boundaries of the hull polygon, and the function provides a set of vertices and edges that bound the polygon.

4.2.2 Wall

After dividing the building into horizontal sections, which are made up of floors, the next step is segmentation and obtaining the exact geometry of the walls. The prerequisite for creating walls was orthogonality to the x-y plane, constant wall thickness along its entire length. The algorithm was designed to detect only straight walls not curved walls. The first step was the preparation of data for wall detection, for this reason the *split_pointcloud_to_storeys* function was used on point cloud, and the ceiling layer was filtered from this data and a 2D histogram was applied to its x,y,z points. The essential function in this section is the subsequent binarization (thresholding) of this data into the mask previously mentioned in Section 3.4. In this binary mask, white pixels indicate regions of interest, these pixels had to exceed the set value (0.01 in this case), black pixels signify areas that are not part of the mask. Subsequently, the morphological operation "closing" previously mentioned in Section 3.4 is applied, which is used to adjust the binary image. The *cv2.findContours* function is applied to this binary image, which identifies contours in the binary image where the white area represents objects or regions of interest. The result of these operations is shown in Figure 4.4, where individual segmented wall surfaces are displayed in a binarized image.

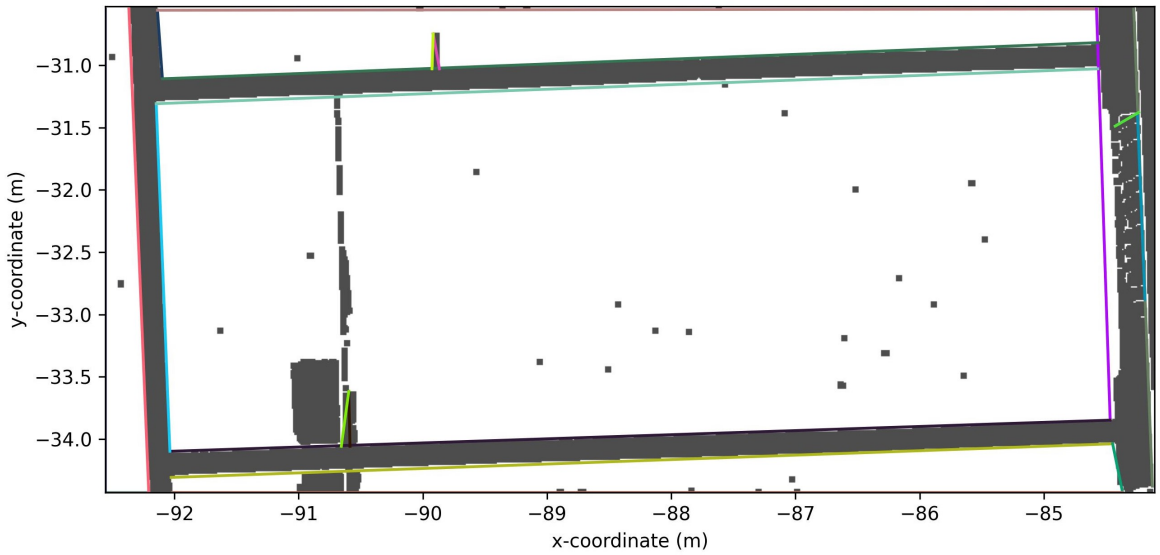


Figure 4.4: Binarized image from 2D histogram (white) and found surfaces (colored).

The following steps involve the application of various rules and functions. First rule, segments that do not have sufficient length are filtered based on the variable *min_wall_length* (in this case st on value = 0.4 m). In the second step, the segments are joined into longer sections. Subsequently, the segments that meet the condition of parallelism within the specified angle deviation (set on 1 degree) of the individual surfaces and the minimum and maximum thickness criteria are connected. In next step determination of the optimal axis for a group of parallel line segments is performed for each group of parallel line segments. It is done by finding the longer and shorter segments, computing the axis direction, and choosing the axis position that minimizes the total distance from the line segments, thus providing an accurate representation of the group's alignment. In Figure 4.5 below, the final parallel segments with axes are pre-

sented. Each wall is depicted in its own distinct color. In the background of this figure, you can observe a green-colored point cloud and a white mask. The function returns the start and end points of wall axes, wall thicknesses and wall materials as its output.

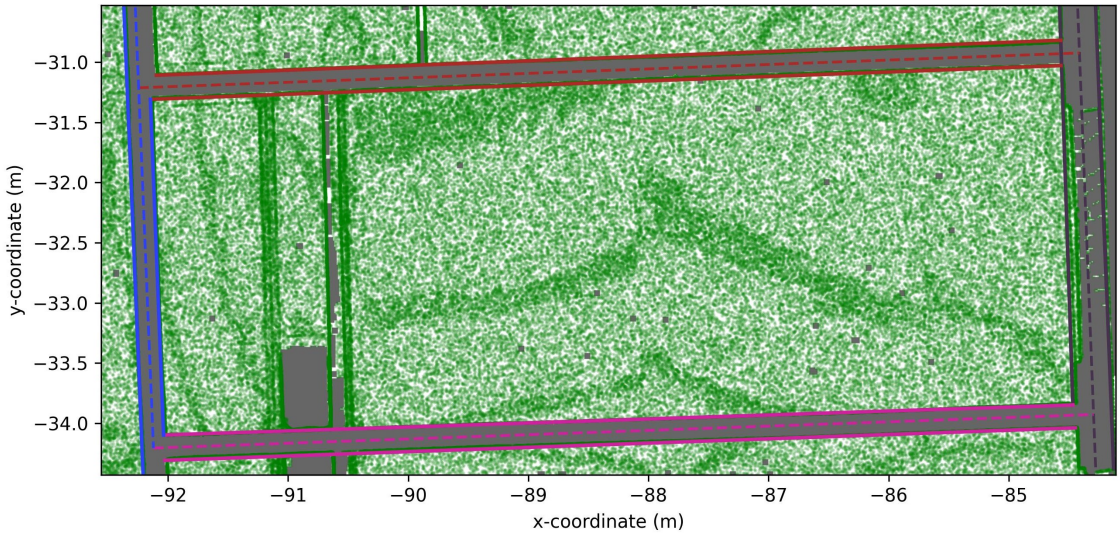


Figure 4.5: Final walls (colored) segmented from point cloud (green). On the background mask.

4.2.3 Openings

Opening detection is based on the pre-established positions of walls within the point cloud data. The analysis occurs on specific sections of the point cloud where the segmented wall is located. The focus area for assessment is the inner surface of the wall. Detection involves analyzing the density of the point cloud initially along the wall's longitudinal axis (x-coordinate), followed by its height (z-coordinate). For all detected orthogonal openings, the minimum and maximum dimensions, as well as the height-to-width ratio, are verified. Based on the position of the sill, an opening is classified as either a window or a door. Consequently, the output of this process includes the basic dimensions (rectangular or square) of an opening, along with its location relative to the wall's origin. Figure 4.6 illustrates a side view of a wall represented by a point cloud (in green) with a detected window opening (in blue). Histograms that represent the density of the point cloud and the set threshold values (indicated by dashed lines) are also displayed to validate the accuracy of the operations.

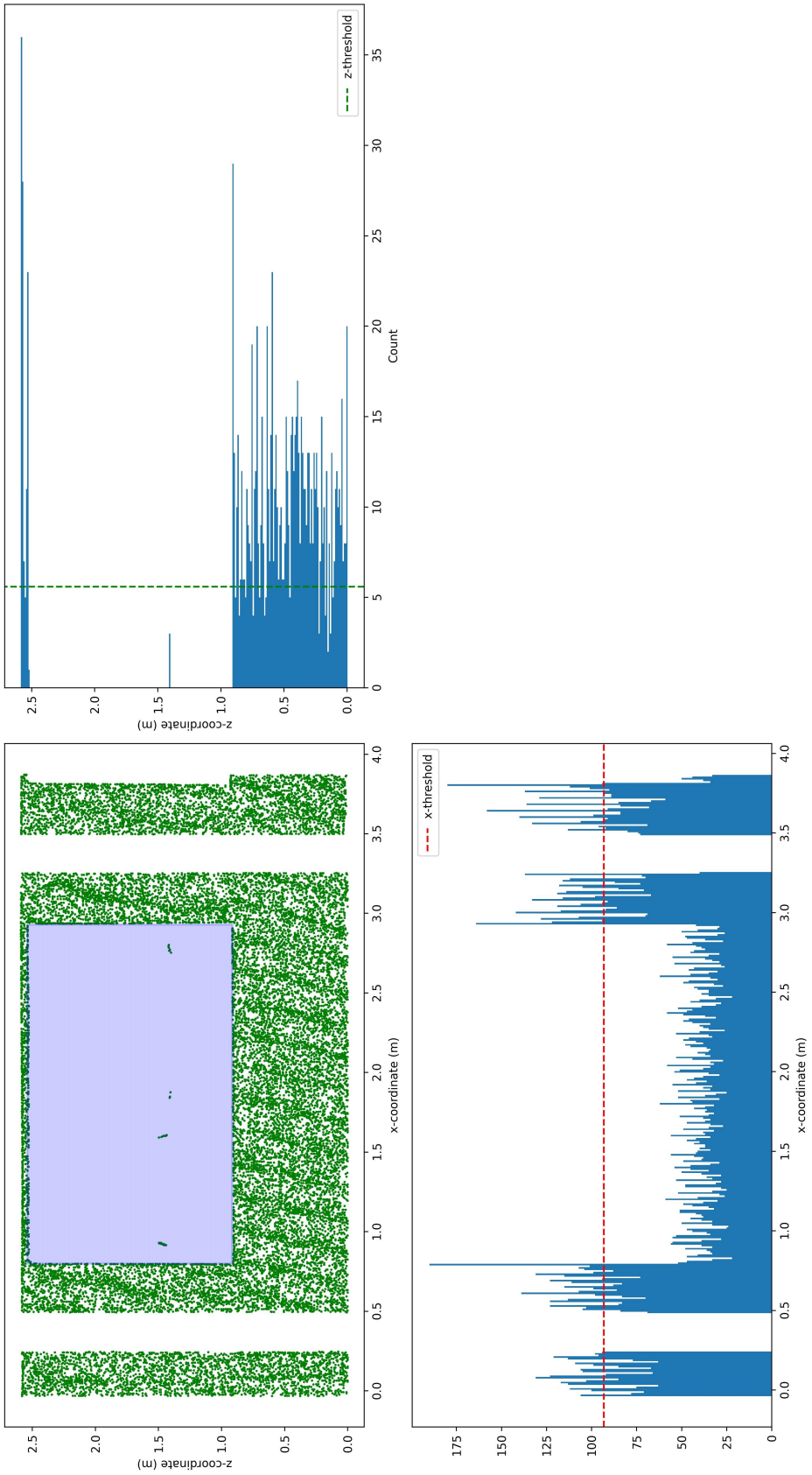


Figure 4.6: Wall opening detection with points density analysis. Z-coordinate point cloud density histogram (top left), x-coordinate point cloud density histogram (bottom right) and side view on evaluated wall point cloud (bottom left) with detected window opening (blue surface).

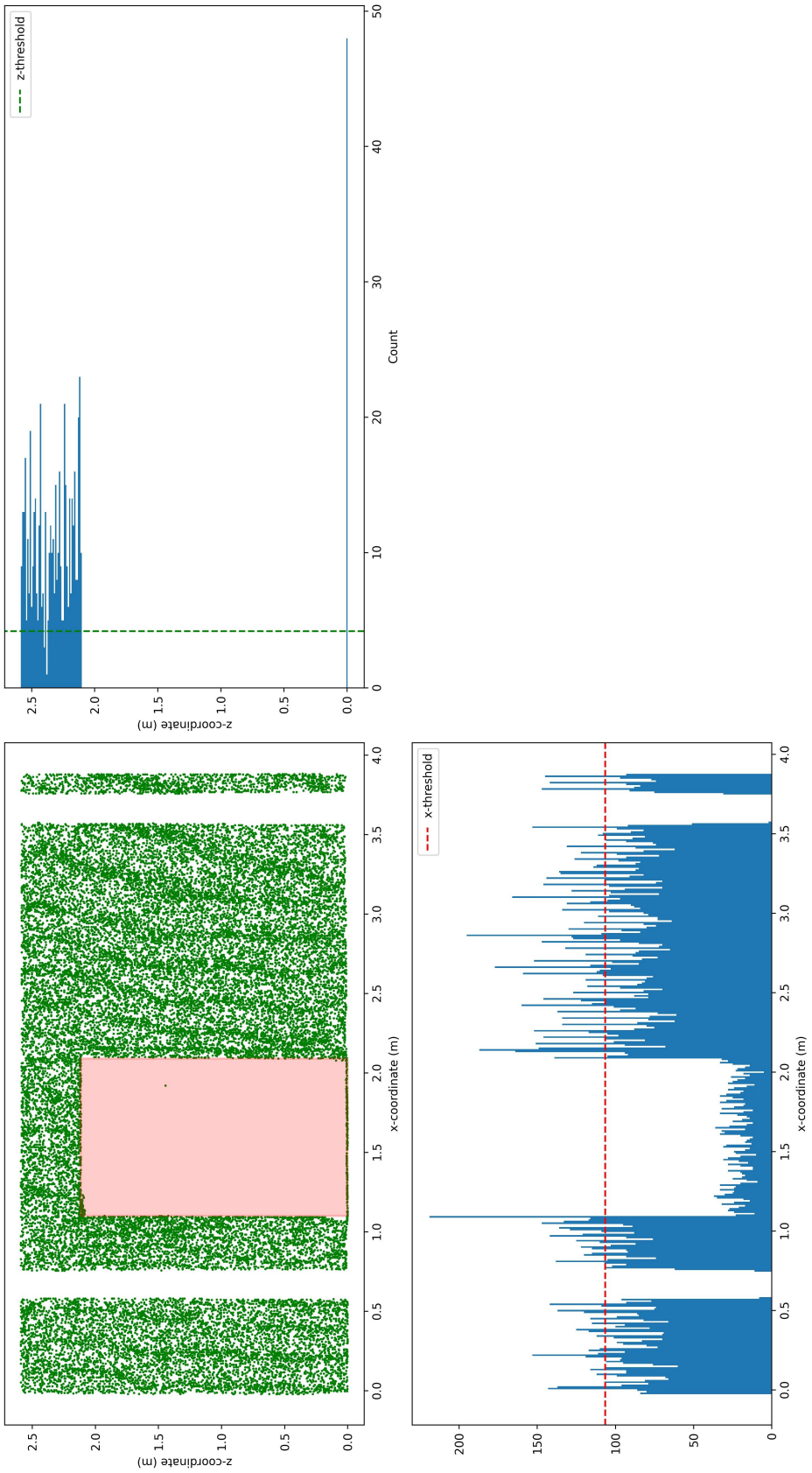


Figure 4.7: Wall opening detection with points density analysis. Z-coordinate point cloud density histogram (top left), x-coordinate point cloud density histogram (bottom right) and side view on evaluated wall point cloud (bottom left) with detected door opening (red surface).

4.3 Algorithms

This section describes the *"Cloud2Entities"* software solution, developed for converting point cloud data into BIM models. Written in Python 3.10, this software is designed to streamline the process of creating 3D building models from point cloud data. The main software is supported by two additional code components, as illustrated in Fig. 5. The first component, *"aux_functions"*, comprises essential functions for segmenting building elements from point cloud data. The second component, *"generate_ifc,"* focuses on generating IFC files.

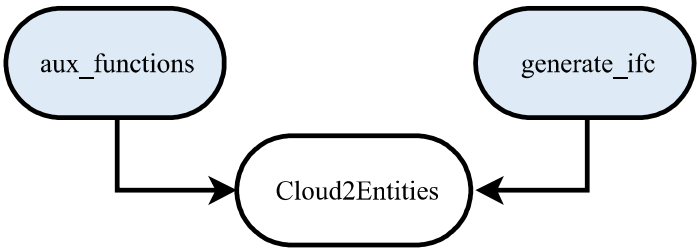


Figure 4.8: Main code supplemented with additional parts aux_function and generate_ifc.

4.3.1 Data Reading

Multiple .e57 or .xyz files can be used as program input. The code first checks for the existence of the *e57_input* variable to determine if it should proceed with loading data from E57 files. If the variable exists, an empty list called *imported_e57_data* is created to hold the data. Function then iterates through a list of E57 file names. Logging messages for each file's reading, and calls a function to read and store data from these files in the *imported_e57_data* list. Once all E57 files are processed, the code logs a message confirming the conversion of the data to the XYZ format and saves it to a file. If there is no E57 input. In this case, the code proceeds to iterate through a list of XYZ file names. For each file, it logs messages regarding data extraction, loads the xyz coordinates and RGB data from these files, rounds the xyz coordinates to three decimal places, and finally logs a message confirming the successful import of all point cloud data. The following code listing 4.1 shows the part where the data are loaded.

```
from aux_functions import *

e57_file_names = ["input_e57/06th.e57", "input_e57/07th.e57"]
if e57_input:
    for e57_file_name in e57_file_names:
        last_time = log('Reading %s.' % e57_file_name, last_time,
            log_filename)
        imported_e57_data.append(read_e57(e57_file_name))
    last_time = log('Saving the data to %s...' % xyz_filenames[0],
        last_time, log_filename)
    e57_data_to_xyz(imported_e57_data, xyz_filenames[0], chunk_size=1000)
    last_time = log('e57 file(s) converted to ASCII format, saved as %s.' %
        xyz_filenames[0], last_time, log_filename)
```

Code Listing 4.1: Data reading

4.3.2 Slabs Identification

The Python function, *identify_slabs_from_point_cloud*, takes a point cloud as input, which includes 3D coordinates (X, Y, Z) and RGB color information. It processes this point cloud data to identify horizontal surfaces and retrieve information about them. Additionally, it offers the option to visualize these segmented surfaces. The output can be saved in .jpg format or visualized using the Open3D library's *open3d.visualization* function [66]. The input parameter *z_step* determines the step size for evaluating the point density in the histogram. A more detailed description of this algorithm can be found in Section 4.2.1. In Figure 4.9 you'll find a process flowchart illustrating the section of the code responsible for extracting slabs. Code Listing 4.2 is a detailed breakdown of the essential steps performed within the function. These steps include:

- Calculation of the number of steps, denoted as *n_steps*.
- Iterating through horizontal surface candidates using a for loop:

```
for i in tqdm(range(n_steps)).
```

- Determining the number of points in each layer:

```
n_points_array.append(len(idx_selected_xyz)).
```

- Identifying the layer with the most points:

```
max_n_points_array = max(n_points_array).
```

- Selecting strips that reach at least 50 % of the maximum points using a for loop:

```
for i in range(len(n_points_array)).
```

- Merging the lower and upper surfaces of each *horiz_surface* using a for loop:

```
for i in range(len(horiz_surface_candidates)).
```

- Calculating the thickness of the slab from *slab_bottom_z_coord* and *slab_top_z_coord*.
- Determining the board polygon using the function *create_hull_alphashape* (Code listing 4.3). The input to this function is the alpha parameter discussed in Section 3.3, namely *concavity_level*.
- The final step involves storing the obtained values in the dictionary *slabs*. The stored values include: *polygon_id*, *polygon_x_coords*, *polygon_y_coords*, *slab_bottom_z_coord* and *slab_thickness*.

```
def identify_slabs_from_point_cloud(points_xyz, points_rgb, z_step,
    plot_segmented_plane=False):
    ...
    n_steps = int((z_max - z_min) / z_step + 1)
    ...
    for i in tqdm(range(n_steps), desc="Progress searching for
        horiz_surface candidate z-coordinates"):
        z = z_min + i * z_step
        idx_selected_xyz = np.where((z < points_xyz[:, 2]) & (points_xyz[:,
            2] < (z + z_step)))[0]
        z_array.append(z)
        n_points_array.append(len(idx_selected_xyz))
    ...
    max_n_points_array = max(n_points_array)
    ...
    for i in range(len(n_points_array)):
        if n_points_array[i] > 0.5 * max_n_points_array:
            horiz_surface_candidates.append([z_array[i], z_array[i] +
                z_step])
    ...
    for i in range(len(horiz_surface_candidates)):
        if (i % 2) == 1:
            ...
            slab_bottom_z_coord = np.median(horiz_surface_planes[i - 1][:,
                2])
            slab_top_z_coord = np.median(horiz_surface_planes[i][:, 2])
            slab_thickness = slab_top_z_coord - slab_bottom_z_coord
            slab_points = np.concatenate((horiz_surface_planes[i - 1],
                horiz_surface_planes[i]), axis=0)
            x_coords, y_coords, polygon = create_hull_alphashape(
                slab_points, concavity_level=0.0)  # 0.0 -> convex
            slabs.append({'polygon': polygon, 'polygon_x_coords': x_coords,
                'polygon_y_coords': y_coords,
                          'slab_bottom_z_coord': slab_bottom_z_coord, '
                              thickness': slab_thickness})
            ...
    return slabs, horiz_surface_planes
```

Code Listing 4.2: Slabs identification

```
def create_hull_alphashape(points_3d, concavity_level=1.0):
    points_2d = [[x, y] for x, y, _ in points_3d]
    alpha_shape = alphashape.alphashape(points_2d, concavity_level)
    hull = alpha_shape.convex_hull
    ...
    return x_coords, y_coords, polygon
```

Code Listing 4.3: Hull creation function using the Alphashape algorithm [52].

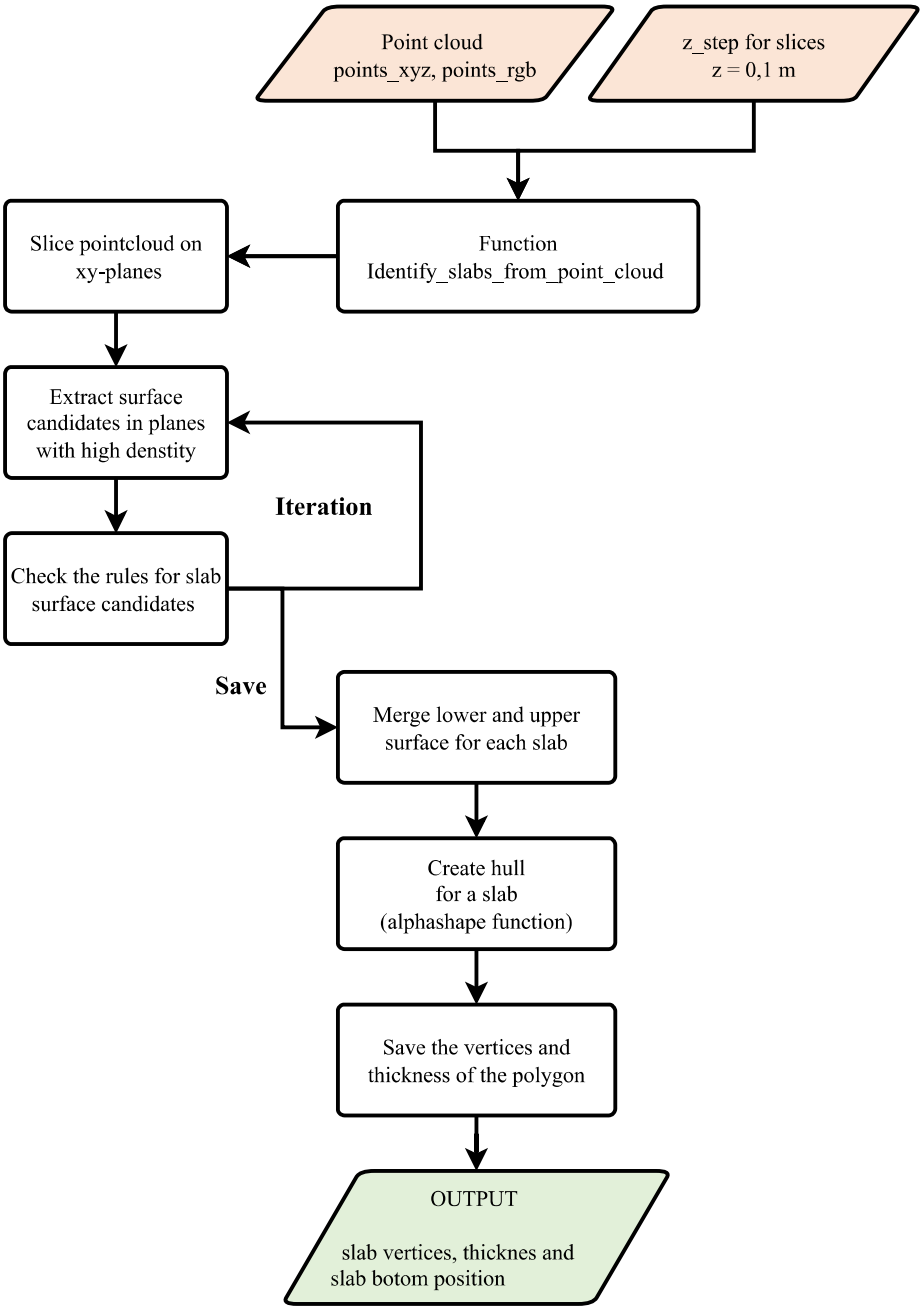


Figure 4.9: A flowchart describing the code for the algorithm for detecting slabs.

4.3.3 Splitting Point Cloud into Storey

Python function, *split_pointcloud_to_storeys*, takes as input a 3D point cloud represented as a NumPy array of (x, y, z) coordinates (*points_xyz*) and a list of slabs. The function's purpose is to split the input point cloud into separate 3D point clouds for each storey in a building, based on the provided slab definitions. The preparation of height-segmented data will help with subsequent wall segmentation. Here's a breakdown of what the function (Code listing 4.4) does:

- The function iterates through the slabs list and calculates two heights for each storey, *bottom_z_of_upper_slab* and *top_z_of_bottom_slab* (Figure 4.10).
- Using NumPy indexing, it extracts points from the *points_xyz* array that fall within the height range defined by *bottom_z_of_upper_slab* and *top_z_of_bottom_slab*.
- Finally, it returns the *segmented_pointclouds_3d* list, which contains separate 3D point clouds for each storey in the building.

```
def split_pointcloud_to_storeys(points_xyz, slabs):
    for i in range(len(slabs) - 1):
        bottom_z_of_upper_slab = slabs[i + 1]['slab_bottom_z_coord']
        top_z_of_bottom_slab = slabs[i]['slab_bottom_z_coord']
        + slabs[i]['thickness']

        segmented_pointcloud_idx = np.where((top_z_of_bottom_slab <
            points_xyz[:, 2]) & (points_xyz[:, 2] <
             bottom_z_of_upper_slab))[0]

        if len(segmented_pointcloud_idx) > 0:
            segmented_pointcloud_points_in_storey = points_xyz[
                segmented_pointcloud_idx]
            segmented_pointclouds_3d.append(
                segmented_pointcloud_points_in_storey)

    return segmented_pointclouds_3d
```

Code Listing 4.4: Point cloud segmentation algorithm for storey points extraction.

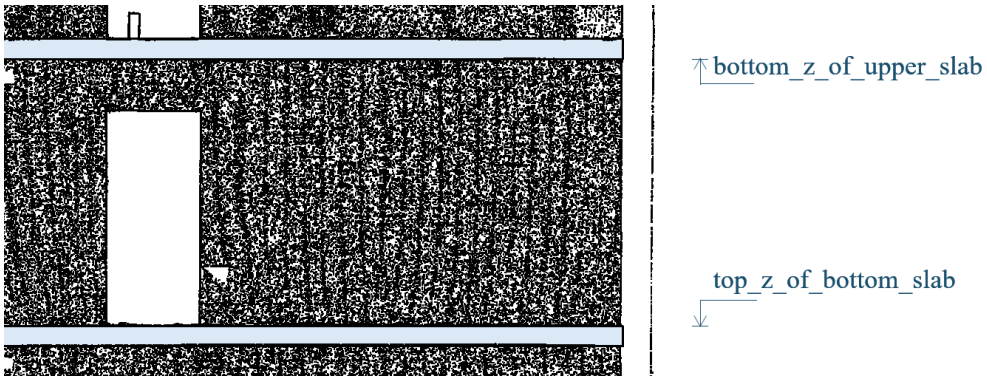


Figure 4.10: Section through the point cloud with marked boundaries for the division into storeys.

4.3.4 Wall Identification

The *identify_walls* function has various user inputs, including 3D point cloud data, the resolution of the point cloud data, the minimum length of a wall segment, the minimum and maximum wall thickness, and the assigned storey or level within the building. Additionally, the function contains inputs that were defined based on the distribution of points in space and are not user-accessible, these are *z_section_boundaries* and *grid_coefficient*. To obtain the precise geometry of walls, this function performs several key steps:

- The *zip(*pointcloud)* expression is used to unzip the 3D point cloud into separate arrays for x, y, and z coordinates.
- *z_section_boundaries* is defined by two height percentages that represent the lower and upper boundaries of a section within a floor. In testing, setting the section to start at 90 % floor height and go up to 100 % worked well.
- *grid_coefficient* is used to control the size of the computational grid. It multiplies the *point_cloud_resolution*, indicating how fine the grid should be.
- *points_2d* is created by selecting the x and y coordinates of the filtered points using the *filtered_indices* list that iterates through the z-coordinates and keeps the indices of points that fall within the calculated z-coordinate limits.
- The *np.histogram2d* function is then called to create a 2D histogram of points based on their x and y coordinates from *points_2d*. The bins parameter accepts the *x_values_full* and *y_values_full* arrays as the boundaries of the histogram pixels.
- The result is stored in *grid_full*, which is a 2D array containing counts of points in each histogram pixel.
- *threshold* variable sets the threshold value, which determines whether a pixel in the histogram should be considered part of the binary mask or not.
- The partial result is stored in the variable *binary_image*, is a binary mask where each pixel represents whether the corresponding region in the original 2D histogram had a value greater than the specified threshold (0.01 in this case). If the value was greater, the pixel in the mask is set to 255 (white), indicating that it's part of the mask. Otherwise, it's set to 0 (black), indicating that it's not part of the mask.
- A closing morphological operation (mentioned in Section 3.4) is applied to the binary image. It uses a 5x5 square structuring element:

```
closing(binary_image, square(5)).
```

- The *cv2.findContours* function accepts a *binary_image* where the white area represents objects or regions of interest, and two constants (mentioned in section 3.4) and return set of points representing contours:

```
cv2.findContours(binary_image, RETR_EXTERNAL, CHAIN_APPROX_NONE).
```

- Approximation of line segments is done with Douglas-Peucker algorithm (mentioned in Section 3.4). Function *cv2.approxPolyDP* takes as input *contours* and *epsilon_factor*.
- List of line segments *all_segments* is converted from pixel coordinates to world coordinates by applying the appropriate scaling factor $pixel\_size$.
- Line segments that do not meet the minimum length requirement specified by $minimum\_wall\_length$ are removed.
- Line segments that are approximately co-linear are merged, based on the specified variables $max\_thickness$ and $max\_distance$.
- The axis for a group of parallel line segments is calculated by analyzing their lengths, directions, and distances from points in the segments, function returns the axis start point and end point along with point to axis mean distance.

4.3.5 Wall Faces Identification

The $identify\_wall\_faces$ function detects the main surfaces of a wall from a point cloud dataset. It extracts the vertical positions of points and uses them to create a histogram that reflects the point density at different location. The function sets a threshold to identify the most significant peaks in the histogram, which represent the wall's surfaces. It adjusts these peaks by the resolution of the point cloud to ensure the surface is fully captured. The following Figure 4.11 displays the histogram of the y-coordinates along with the identified peaks and vertical lines marking the positions of the wall surfaces, as described in the following function 4.5.

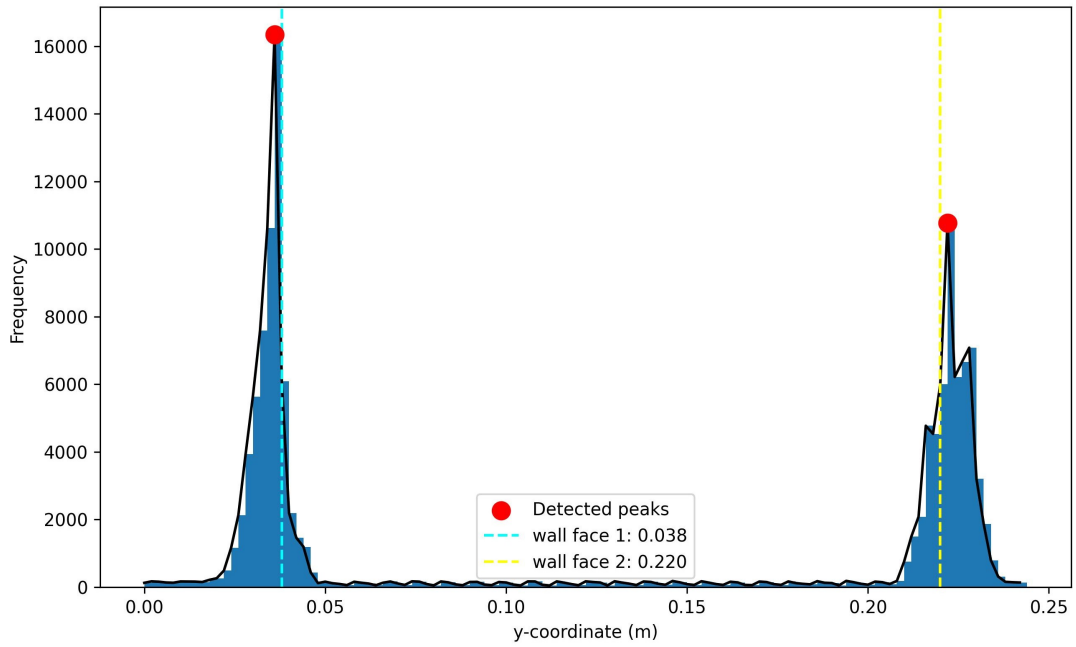


Figure 4.11: The histogram of the density of points on the y-coordinate along with vertices identified and vertical lines indicating the positions of the wall surfaces.

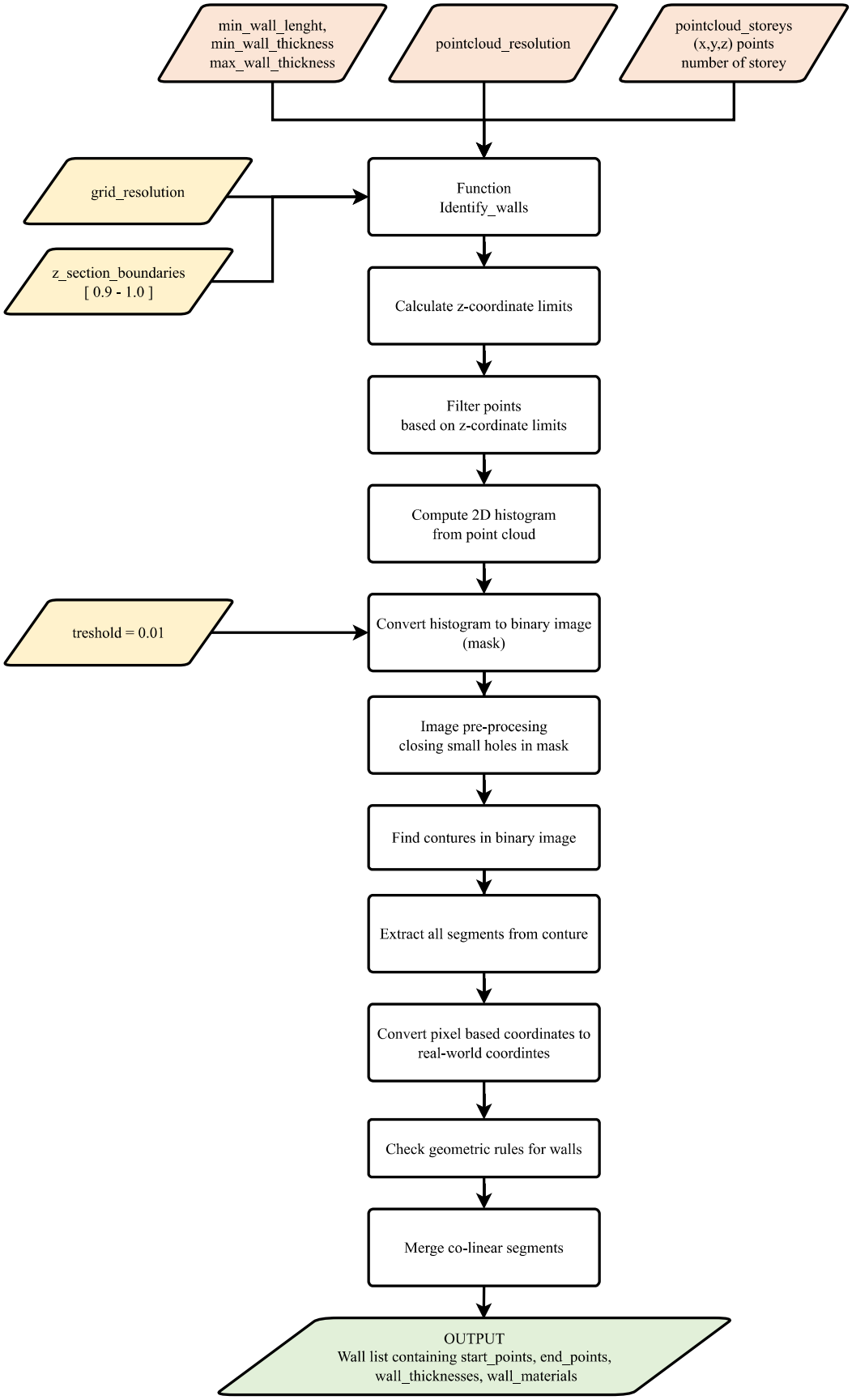


Figure 4.12: A flowchart describing algorithm for walls detection.

4.3.6 Rectangular Opening Detection

The *detect_rectangular_openings* function is designed to identify potential door and window openings in a wall by analyzing point cloud data density. It takes several parameters, including *wall_number* (identifier for the wall), *wall_points* (3D coordinates of points in *.xyz* format), *resolution* and *grid_roughness* (parameters for defining bins in histograms). Another group of inputs include the *histogram_threshold* for deciding if a bin contains an opening, *min_opening_width* and *min_opening_height* to filter by size, *max_opening_aspect_ratio* to control aspect ratios, *door_z_min* to classify openings as doors or windows based on their z-coordinate, and *thickness_for_extraction* to define the region of interest for projecting points onto the x-z plane.

The function begins by determining the front and back surfaces of the wall using the *identify_wall_faces* function (Code listing 4.5), which provides two y-coordinates representing these surfaces. With these coordinates, it establishes a region of interest (ROI) for the wall points by setting an inner and outer threshold around the front wall surface based on the given thickness value in the *thickness_for_extraction* variable. The correct setting of this parameter ensures that there will be no discontinuity of the wall surface, which could be caused by surface unevenness.

```
def identify_wall_faces(wall_number, points, point_cloud_resolution,
    min_distance=3, plot_histograms_for_walls=True):
    y_coords = [point[1] for point in points]
    y_min, y_max = min(y_coords), max(y_coords)
    bin_edges = np.arange(y_min, y_max, point_cloud_resolution)
    hist, _ = np.histogram(y_coords, bins=bin_edges)
    height_threshold = 0.5 * max(hist)
    peaks, properties = find_peaks(hist, distance=min_distance, height=
        height_threshold,
                                   prominence=0.25 * height_threshold)
    if len(peaks) < 2:
        print("Warning: Unable to identify both floor and ceiling surfaces.
            ")
        return None, None

    y1 = bin_edges[peaks[0]] + point_cloud_resolution
    y2 = bin_edges[peaks[-1]] - point_cloud_resolution

    return y1, y2
```

Code Listing 4.5: Key components of the wall surfaces identification function.

Next, the function projects the points within the ROI onto the x-z plane to a 2D representation for easier analysis. The rotation of all relevant points is done based on the unit vector that belongs to the rotated wall. The variable in code is called *direction_vector* and is calculated from the following relation:

$$\mathbf{D} = \left(\frac{x_2 - x_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}}, \frac{y_2 - y_1}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \right), \tag{4.1}$$

where x_2 is the x-coordinate of the second point and x_1 is the x-coordinate of the first point, while y_2 is the y-coordinate of the second point, and y_1 is the y-coordinate of the first point,

$\mathbf{D}$ represents the direction vector. A histogram of projected points is then created, with bins sized according to the resolution and grid roughness parameters. The histogram displayed in Figure 4.6 helps to visualize the distribution of points along the wall's length and can indicate potential openings based on gaps in the points distribution. The threshold for determining floor plan dimension of openings is set by multiplying the 10th highest value in the histogram ($max10$) by the $histogram\_threshold$ parameter.

The "$openings$" list is used to store the identified opening intervals. The function iterates through the elements of the "$hist$" array, which represents the histogram, and checks whether the count of points within a bin falls below a threshold ("$x\_threshold$"). When a bin with a count below the threshold is encountered, it marks the start of an opening, and the function keeps track of this with the "$in\_opening$" variable. The start and end points of an opening are recorded, and if the difference between these points exceeds a minimum opening width ("$min\_opening\_width$"), the opening is considered valid and added to the openings list.

For each valid opening, the function calculates the middle point along the x-axis within the opening, and then it extracts the corresponding z-values (heights) of points falling within a certain tolerance around the middle point. A z-histogram is constructed from these z-values, and the second highest count ("$max2$") in the z-histogram is used to determine a z-threshold, which helps segment the opening's precise height. The function then identifies candidates for the opening's height by analyzing the z-histogram and selects the height range with the maximum vertical span. The width of the opening is calculated based on its x-coordinates, and if the height of the opening exceeds a minimum threshold, and its aspect ratio falls within specified constraints, the opening is classified as either a 'door' or 'window' based on its z-coordinate with respect to a predefined "$door\_z\_min$".

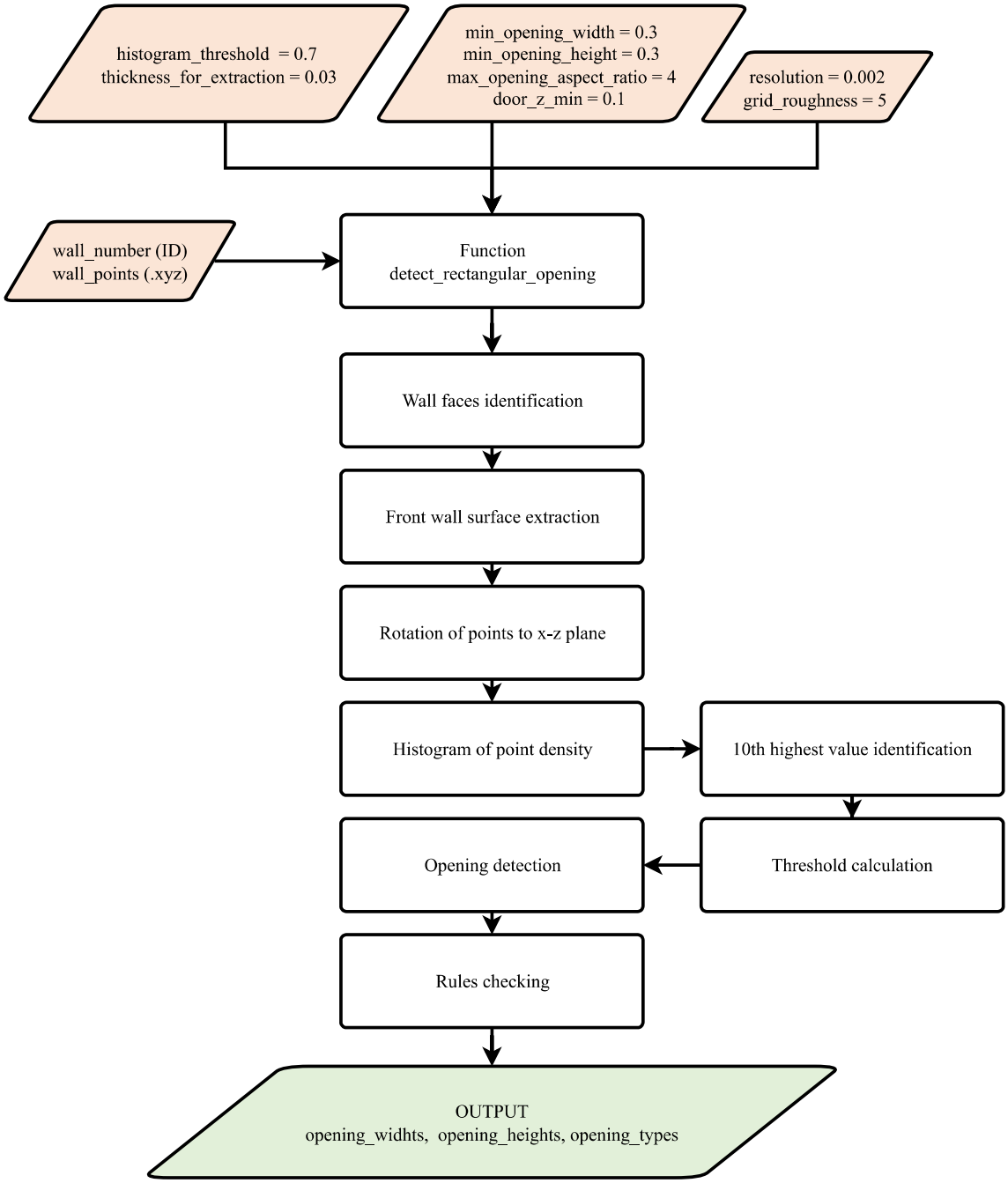


Figure 4.13: A flowchart describing algorithm for openings detection.

4.4 IFC File Generation

To ensure openBIM output after generating the geometry and to guarantee accessibility by various software applications, the IFC (Industry Foundation Classes) format has been selected as the format in which the obtained geometry will be stored. The output generation in IFC 4 ADD2 TC1 format was done using the Ifcopenshell library [67]. DesignTransferView_V1.0 was chosen as the Model View Definition (MVD) for further possible editing and the nature of how the data on individual geometric entities should be generated. After generating the file in .ifc format, validation was performed using the online validation tool from BuildingSMART [68]. This tool tests the file in three areas:

- STEP Physical File Syntax / IFC Schema
- Rules
- bSDD

The following figure shows the fulfillment of all conditions in the generated ifc file from point cloud data.

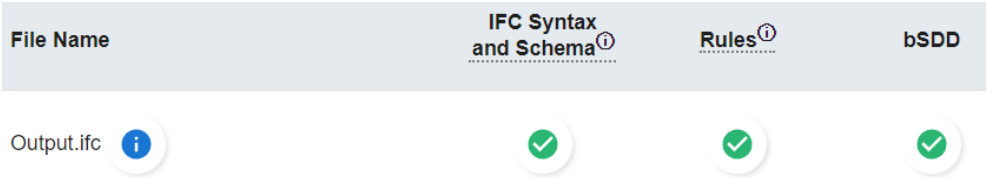


Figure 4.14: Successful output validation with BuildingSMART validation instrument [68].

Algorithm
Python code *"generate_ifc.py"* defines a class named *"IFCmodel"* designed for the creation and management of IFC files, which are utilized in the representation of building and construction industry data. The class constructor initializes various attributes, including project name, output file directory path, and header data for the IFC file. The ifcopenshell library is employed to handle IFC file operations. Upon instantiation of the class, an instance of *"IFCmodel"* is created with specific project and output file details. The overall structure of the code facilitates the organization of IFC model creation logic within a class, offering a clear and modular approach for users to define project and author details before generating an IFC model instance. There are several methods in IFCmodel class:

- The *"define_author_information"* method setting the authorship information in both the class instance and the header of the IFC file being generated.
- The *"create_unit_assignment"* method is responsible for generating a unit assignment for the project. This unit assignment includes units for length, area, volume, plane angle, and solid angle.

- The *"assign_material"* method facilitates the creation of an association between a material and a product in the IFC model. This is achieved by creating an *"IfcRelAssociatesMaterial"* entity with appropriate attributes, linking the material to the specified product.
- The *"define_project_data"* method orchestrates the creation of various entities within the IFC model, including units, geometric representations, project structure components, and their relationships. This method ensures that the IFC model accurately represents the specified project details.
- The *"create_building_storey"* method generate a building storey in the IFC model. It defines a new local coordinate system related to the storey, creates an *"IfcBuildingStorey"* entity, and establishes a relationship between the building and the newly created storey.
- The *"create_slab"* method within the IFCmodel class is responsible for generating a slab in the IFC model. This method involves the creation of various entities such as coordinate systems, profiles, and material associations to represent the slab accurately.
- The *"create_material_layer"* method creates an *"IfcMaterialLayer"* entity representing a single layer of a material. It takes optional parameters *"wall_thickness"* (default is 0.3 m) and *"material_name"* (default is Masonry - brick").
- The *"create_material_layer_set"* method is used to create *"IfcMaterialLayerSet"* entity representing a set of material layers. And is used with multi layered walls and slabs.
- The *"wall_placement"* method establishes the placement of a wall within the IFC model by creating a coordinate system *"IfcAxis2Placement3D"* with an elevation *"z_placement"*. This coordinate system is then used to define the local placement of the wall *"IfcLocalPlacement"*.
- The *"wall_axis_placement"* method creating an *"IfcPolyline"* entity that represents the axis or geometry of a wall. This is achieved by specifying the start and end points of the polyline.
- The *"wall_axis_representation"* method create an *"IfcShapeRepresentation"* entity that represents the geometric representation of the wall axis. This involves specifying the context, representation type, and the actual geometry.
- The *"wall_swept_solid_representation"* method is responsible for creating a swept solid representation of a wall within an IFC model. This involves defining a rectangular profile, extruding it to create a three-dimensional solid, and then representing it as a swept solid in the IFC model.
- The *"create_wall"* method in the IFCmodel class create an *"IfcWall"* entity within the IFC model. This involves specifying various parameters such as global identifier, owner history, name, description, object type, object placement, representation, tag, and predefined type.
- The *"create_wall_type"* method encapsulates the process of creating an *"IfcWallType"* entity within the IFC model. It takes parameters for various properties of the wall type, creates relationships with the wall entity.

- The *"create_wall_opening"* method in the *"IFCmodel"* class is responsible for creating an *"IfcOpeningElement"* entity within the IFC model. This involves specifying various parameters such as global identifier, owner history, name, description, object type, object placement, representation, tag, and predefined type.
- The *"opening_placement"* method defining the placement of an opening element within the context of a wall. It uses *"IfcAxis2Placement3D"* for specifying the location and *"IfcLocalPlacement"* for establishing the relative placement within the wall's context.
- The *"opening_closed_profile_def"* method is responsible for defining a closed two dimensional profile for an opening element, and it uses *"IfcArbitraryClosedProfileDef"* with a polyline as the outer curve to represent this profile.
- The *"opening_extrusion"* method creating an *"IfcExtrudedAreaSolid"*, which represents the extrusion of a 2D profile along a straight path to form a 3D solid.
- The *"create_rel_voids_element"* method creates an *"IfcRelVoidsElement"* relationship between a building element and an opening element in your IFC model. This Boolean operation results in a void space within the building element's geometry, creating an opening.

4.4.1 Spatial Hierarchy

For the simple inclusion of building elements into their respective locations, a hierarchy as shown in Figure 4.15 has been created. This hierarchy represents the common structure of the project. In each IFC file, exactly one instance of *IfcProject* or another *IfcContext* such as *IfcProjectLibrary* must be defined. All other data in the file is then related to this IFC class. This class also defines essential attributes such as a list of unit definitions, project name, project phase, etc. A snippet of the code part 4.6 where the IFC class *IfcProject* is created using the Python programming language and the function provided by the IfcOpenShell package [69].

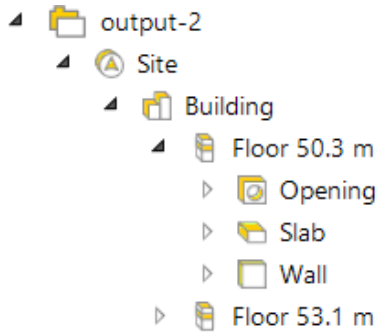


Figure 4.15: IFC spatial hierarchy displayed in BIMcollab ZOOM viewer.

```
import ifcopenshell
class IFCmodel:
    def define_project_data(self, project_description, object_type,
        long_project_name, construction_phase, version, organization,
        person_given_name, person_family_name, latitude, longitude,
        elevation):

        ...

        self.project = self.ifc_file.create_entity(
            "IfcProject",
            GlobalId=ifcopenshell.guid.new(),
            Name=self.project_name,
            LongName=self.long_project_name,
            ObjectType=self.object_type,
            Description=self.project_description,
            Phase=self.construction_phase,
            UnitsInContext=unit_assignment,
        )
```

Code Listing 4.6: IfcProject class atributes declaration

4.4.2 Geometric Representation

Slab

The entity representation of the ceiling slab is provided in IFC 4 using the *IfcSlab* class. The geometric representation is based on a swept solid approach. The *IfcArbitraryClosedProfileDef* defining a closed profile, in this case a polyline, describing the perimeter of the slab. The *IfcExtrudedAreaSolid* entity is then employed to create a solid extrusion from this profile along a specified axis. Parameters such as the extrusion depth and the local placement of the slab within the building are also defined. Individual ifc classes are shown in Figure 4.16.

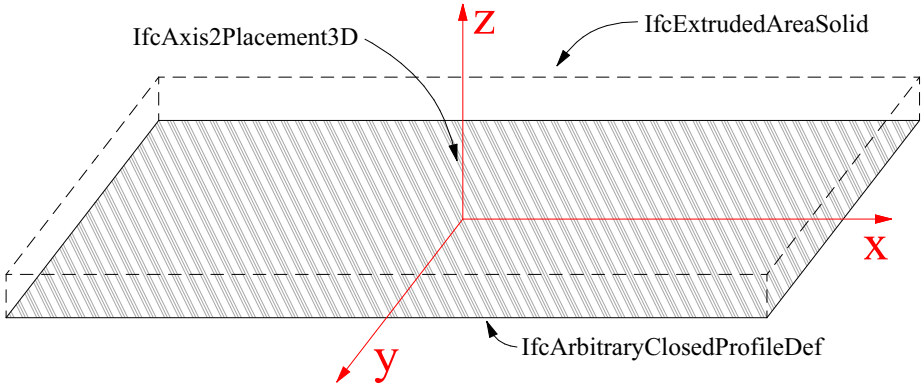


Figure 4.16: Geometric representation of slab created by ifc sweptsolid geometry.

Wall

The wall is represented in IFC by the class *IfcWall* or class *IfcWallType*, which represents individual groups of walls with the same thickness parameters and layer composition. The geometry is defined by a start and end point *IfcCartesianPoint*, which forms the axis of the wall. From wall axis and predefined wall thickness, the *IfcArbitraryClosedProfileDef* extraction polygon is then calculated. The resulting height for the extrusion is considered to be the full height of the floor and is given in the class *IfcExtrudedAreaSolid*. The following Figure 4.17 shows the main classes used to create the geometric representation of the wall.

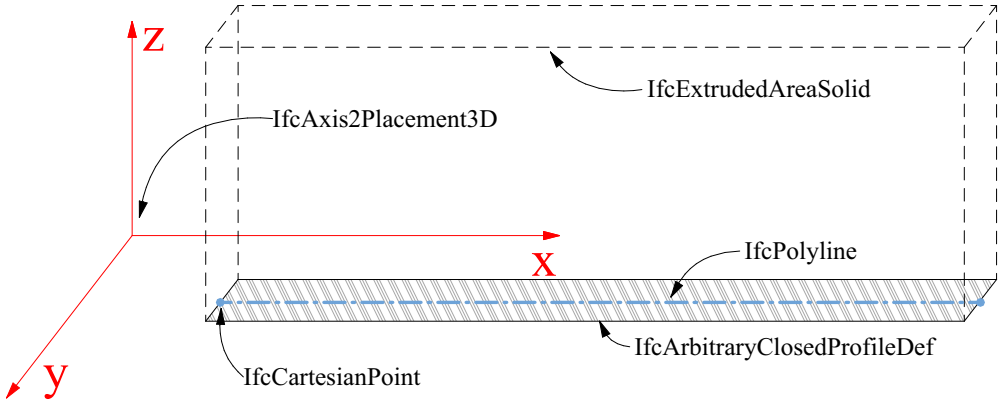


Figure 4.17: Main IFC classes used to create geometric representation of wall.

The following code snippet 4.7 shows the creation of the *IfcExtrudedAreaSolid* class and storing it in the *wall_extruded_area* variable, the direction of the extrusion is given by the unit vector in the z-axis direction, the depth of the extrusion is set by the *wall_height* variable. Code snippet 4.8 shows the mentioned class published to the IFC 4 output in STEP physical file format.

```
wall_extruded_area = self.ifc_file.create_entity(
            "IfcExtrudedAreaSolid",
            SweptArea=rectangle_profile,
            Position=None,
            ExtrudedDirection=self.ifc_file.create_entity("IfcDirection",
                DirectionRatios=(0.0, 0.0, 1.0)),
            Depth=wall_height,
        )
```

Code Listing 4.7: IfcExtrudedAreaSolid class atributes declaration in Python code.

```
#130=IFCRECTANGLEPROFILEDEF(.AREA.,'Wall Perim',#129,7.833,0.203);
#131=IFCDIRECTION((0.,0.,1.));
#132=IFCEXTRUDEDAREASOLID(#130,$,#131,2.591);
```

Code Listing 4.8: IfcExtrudedAreaSolid class in IFC 4 output in STEP physical file format.

Opening

The geometric representation of the opening shown in Figure 4.18 is performed using object definition class *IfcOpening*. This class relates to its parent wall using the relationship *IfcRelVoidsElement*. The location of the opening is provided by the *IfcAxis2Placement3D* classes, where one class is global for the wall and the other is local for the opening in this wall. The opening itself is created using the already mentioned swept solid technique. A rectangle is created with $IfcPolyline$ that represents the area of the window sill or door threshold and is then stretched in the direction of the z axis to the required height of the opening with *IfcExtrudedAreaSolid* class.

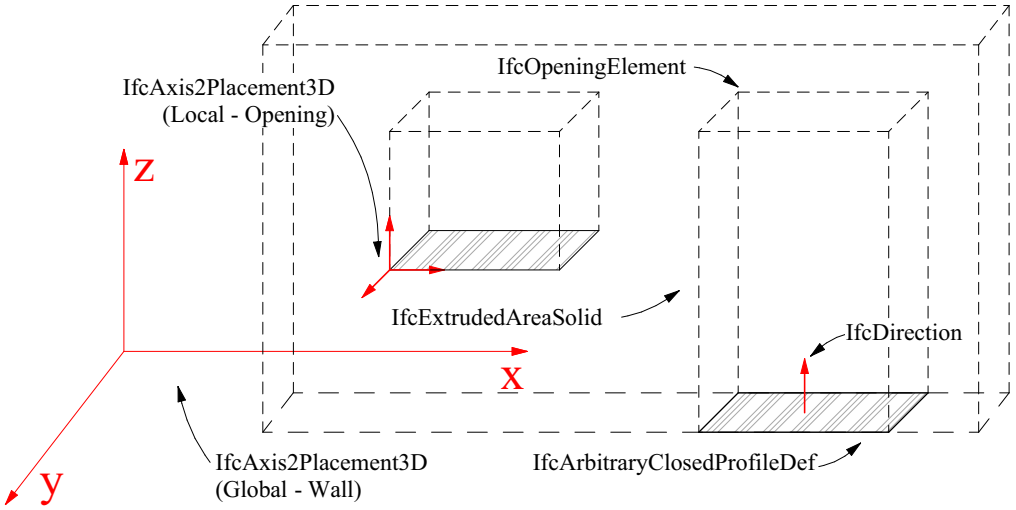


Figure 4.18: Main IFC classes used to create geometric representation of opening.

Chapter 5

Research Outcomes and Discussion

The following section will describe the output of the program and evaluate the pros and cons of the used solution, the approach will also be compared with alternative approaches in the literature. To assess the program's functionality, a point cloud representing a single room within a multi-storey former hotel was prepared. The laser-scanned room contained an external wall with a window and three internal walls, and door openings in them. Horizontally, the point cloud was defined by two ceiling slabs. Figure 4.2 shows the point cloud for testing.

In the first phase, the ceiling slabs were segmented and classified, and the building was divided into individual floors. This procedure is also performed in the manual creation of a 3D BIM model from point cloud data and was therefore a logical first step. The solution relies on the assumption of a higher point density in the height planes of the horizontal surfaces of the ceiling slabs. A similar approach was also followed by Jung et.al. [70]. However, his proposed solution only allows segmentation of a single-story building and therefore uses the floor and ceiling surfaces only as boundaries to obtain the height of the segmented walls. A similarly limited solution for only single-storey buildings was presented by Previtali et.al. [71]. In their paper focuses on modelling the interior space of a building and implements the RANSAC algorithm to obtain floor and ceiling surfaces, which is widely implemented in other works and allows the detection of common shapes such as sphere, cone and cylinder surfaces [20, 72]. To obtain the plan dimension of the ceiling slabs, the algorithm Aplhapshape [52] was applied. For the application of this algorithm, it is important to mention that the calculation of the concave shape is very computationally demanding due to the Delauney triangulation that is applied to all points lying in the ceiling plane. The result of this operation is a ceiling slab that follows the outer envelope of the building by its edge.

The second step in creating the entire building model involved obtaining the floor plan shape and the positions of the walls. During the wall creation process, an assumption was made to consider only straight edges, thereby excluding curved walls. A similar approach was adopted by Thomson [72] and Ochmann [73]. This simplification not only streamlines the modeling process but also proves beneficial in representing the final geometry in IFC format. Another commonly used assumption is the Manhattan-world scene, which posits that walls are strictly orthogonal. However, this work did not strictly adhere to this assumption, and the chosen method allows for the detection of walls that are not perpendicular to each other. The advantage of this assumption in facilitating semantic segmentation is well-documented, as evidenced

in studies such as those by Previtali [71] and Murali et al. [74]. In this work, walls have been modeled as volumetric parametric objects, representing one of the various approaches to wall modeling. This method aligns with similar solutions, such as the one demonstrated by Bassier [75]. Contrasting with this, some methodologies define walls, floors, and ceilings primarily in terms of rooms, characterizing these elements as zones within the modeling software. This perspective emphasizes the spatial aspect as the key element of information, focusing less on the physical structures themselves. However, from the author's standpoint, this latter approach is considered less suitable, particularly in the context of this work. It is believed that the volumetric parametric representation offers a more comprehensive and practical approach for the objectives set forth in this thesis.

When the precise location of the wall is established, analyzing the openings, like doors and windows, from a side elevation perspective seems to be an appropriate method. This approach, used for determining the geometry and positions of openings in the wall, was similarly employed by Cheng et al. [76]. A different approach, presented by Díaz-Vilariño [77], relied on the assumption of a known path followed by a SLAM-based scanner. Although this solution offers high accuracy, it is notably inefficient due to the unavailability of the scanner's trajectory in some cases and the limitations of equipment that can be used for indoor data collection. Another approach, which involves density histogram analysis, was introduced by Chen [78]. Chen utilized a gradient-based maximum to identify the starting and ending locations of openings. For the purpose of simpler segmentation and visualization of openings, the assumption that all doors and windows are square was made. This assumption facilitates the histogram analysis of point density in the wall.

The program outputs a file in the IFC 4 ADD TC2 format, which is commonly implemented in the exchange of data within the field of BIM collaboration. Some solutions found in the literature use output formats or data representations that are not OpenBIM, preventing display and editing across different software platforms for the average user. Relying on a single software, such as Planar5D used by Murali [74], is highly inefficient and restrictive for end-users. The format has multiple versions and is constantly evolving. The previously prevalent IFC2x3 version, as used by Thomson [72], has now been largely supplanted by the more current IFC 4 version, which is employed in this work.

The results of this research are visually represented in the subsequent figures. Figure 5.1 illustrates the generated floor plan of the tested room, as displayed in ArchiCAD 26, highlighting the precision and clarity achieved through the developed methodologies. Following this, Figure 5.2 showcases the program's output for the same room, also visualized in ArchiCAD 26 3D view. These visual representations not only demonstrate the practical applicability of the software but also affirm its effectiveness in producing detailed and accurate BIM models.

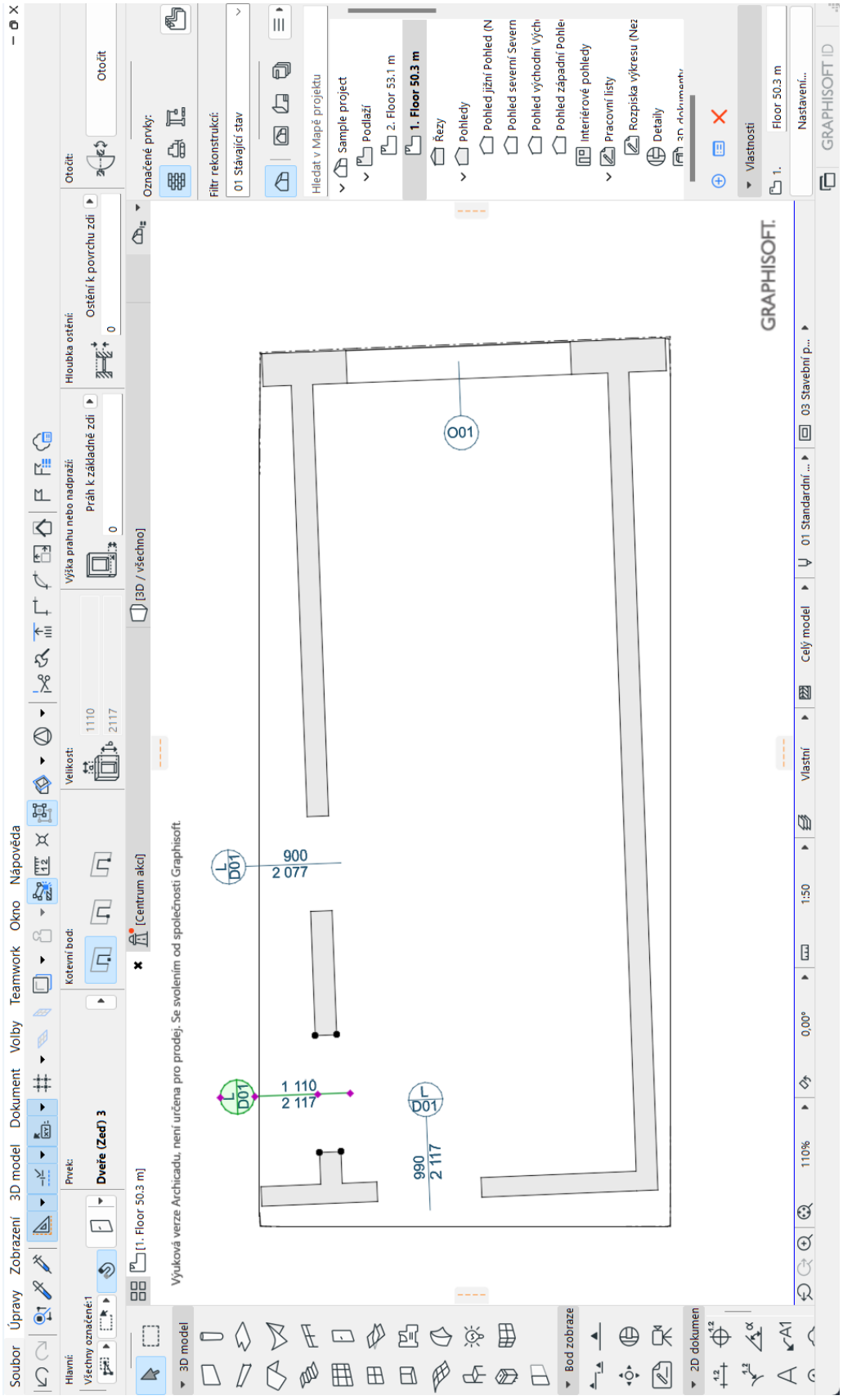


Figure 5.1: Generated floor plan of testing room displayed in ArchiCAD 26.

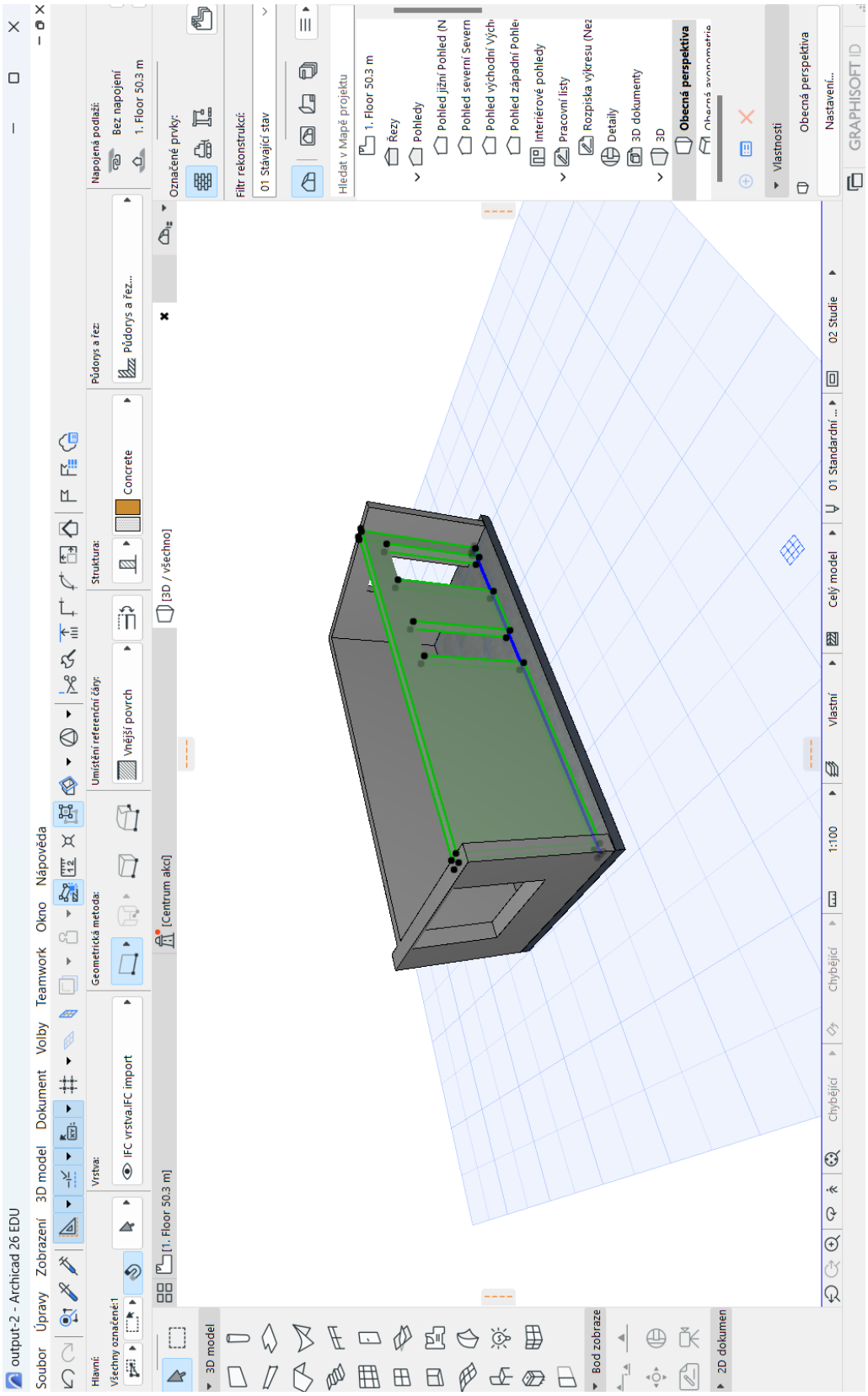


Figure 5.2: Program output of testing room displayed in ArchiCAD 26.

Conclusion

This diploma thesis has been devoted to advancing the methodology for transforming point cloud data into 3D models. Traditionally, this transformation has been a labor-intensive task, requiring manual input of building elements into modeling software such as ArchiCAD, Revit, and Allplan. This manual approach demanded precision in aligning these elements with the point cloud's scanned surfaces. In contrast, the novel approach developed in this thesis automates this conversion process, representing a significant leap forward in efficiency and accuracy.

The thesis was divided into two main parts, and it includes an attachment to a GitHub repository where the program's source codes are stored. The first part described already known principles, techniques, and algorithms usable for the segmentation and classification of point clouds. The second, practical part, discusses data preparation, methods used for segmenting individual structural elements, and the functioning of key algorithms. The program is based on the principle of analyzing point density in space and image processing with morphology operations. The proposed solution can segment and classify building elements such as ceiling slabs, walls, windows, and doors. The program subsequently saves the geometry of these building elements in a created IFC4 file, facilitating the further use of the 3D BIM model.

Future work could extend these capabilities to include the detection of curved walls, roof planes, ceiling ducts, and other building elements. Additionally, the IFC output could be enhanced to include automatically detected materials, possibly leveraging machine learning for texture recognition.

The development of the software not only marks a significant step forward in automating the Scan-to-BIM process but also paves the way for more efficient and sustainable practices in the construction and architectural fields. By facilitating quicker and more accurate conversions of point cloud data into usable 3D models, this work contributes to the reduction of time and resources typically expended in manual processes, underscoring the growing importance of technological integration in building lifecycle management.

References

1. KARMAKAR, Ankan; DELHI, V. S. Kumar. Construction 4.0: what we know and where we are headed? *Journal of Information Technology in Construction*. 2021, vol. 26. Available from DOI: 10.36680/j.itcon.2021.028.
2. *Energy, transport and environment statistics*. Publications Office of the European Union Luxembourg, 2019. ISBN 978-92-76-20736-8. ISSN 2363-2372. Available from DOI: 10.2785/522192.
3. DAKWALE, Vaidehi; RALEGAONKAR, Rahul; MANDAVGANE, Sachin. Improving environmental performance of building through increased energy efficiency: A review. *Sustainable Cities and Society*. 2011, vol. 1, no. 4, pp. 211–218. ISSN 2210-6707. Available from DOI: 10.1016/j.scs.2011.07.007.
4. GE, Xin Janet; LIVESEY, Peter, et al. Deconstruction waste management through 3d reconstruction and bim: a case study. *Visualization in engineering*. 2017, vol. 5, no. 1, pp. 1–15. Available from DOI: 10.1186/s40327-017-0050-5.
5. HONIC, Meliha; KOVACIC, Iva, et al. Material Passports for the end-of-life stage of buildings: Challenges and potentials. *Journal of Cleaner Production*. 2021, vol. 319, p. 128702. ISSN 0959-6526. Available from DOI: 10.1016/j.jclepro.2021.128702.
6. SESANA, Marta; SALVALAI, Graziano. A review on Building Renovation Passport: Potentialities and barriers on current initiatives. *Energy and Buildings*. 2018, vol. 173, pp. 195–205. ISSN 0378-7788. Available from DOI: 10.1016/j.enbuild.2018.05.027.
7. AKBARIEH, Arghavan; JAYASINGHE, Bhagya, et al. BIM-Based End-of-Lifecycle Decision Making and Digital Deconstruction: Literature Review. *Sustainability*. 2020, vol. 12, p. 2670. Available from DOI: 10.3390/su12072670.
8. TAPPONI, Omar et al. Renovation of heritage assets using BIM: A case study of the Durham Cathedral. In: *Proceedings of the 32nd CIB W78 Conference, Eindhoven, The Netherlands*. 2015, pp. 27–29.
9. ROCHA, Gustavo et al. A scan-to-BIM methodology applied to heritage buildings. *Heritage*. 2020, vol. 3, no. 1, pp. 47–67. Available from DOI: 10.3390/heritage3010004.
10. GRAPHISOFT. *Hungarian State Opera House* [online]. 2023. [visited on 2023-10-31]. Available from: www.graphisoft.com/sg/case-studies/hungarian-state-opera-survey.

11. KLINC, Robert; JOTANOVIČ, Uroš; KREGAR, Klemen. Point clouds for use in building information models (BIM). *Geodetski vestnik*. 2021. Available also from: https://api.semanticscholar.org/CorpusID:245681638.

12. MACHER, Hélène; LANDES, Tania; GRUSSENMEYER, Pierre. From point clouds to building information models: 3D semi-automatic reconstruction of indoors of existing buildings. *Applied Sciences*. 2017, vol. 7, no. 10, p. 1030. Available from DOI: 10.3390/app7101030.

13. JIANG, Jianan et al. A Hybrid Approach to Semi-Automatic Conversion of 3D Point Cloud Data to Building Information Model. In: *Construction Research Congress 2020*. American Society of Civil Engineers Reston, VA, 2020, pp. 408–417. Available from DOI: 10.1061/9780784482858.045.

14. SUNGCHUL, Hong; JAEHOON, Jung, et al. Semi-automated approach to indoor mapping for 3D as-built building information modeling. *Computers, Environment and Urban Systems*. 2015, vol. 51, pp. 34–46. ISSN 0198-9715. Available from DOI: 10.1016/j.compenvurbsys.2015.01.005.

15. TAMKE, Martin; EVERS, Henrik, et al. An automated approach to the generation of structured building information models from unstructured 3D point cloud scans. In: *Proceedings of IASS Annual Symposia*. International Association for Shell and Spatial Structures (IASS), 2016, vol. 2016, pp. 1–10. No. 17.

16. HUANG, H. S.; TANG, S. J., et al. From BIM To Pointcloud: Automatic Generation of Labeled Indoor Pointcloud. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2022, vol. XLIII-B5-2022, pp. 73–78. Available from DOI: 10.5194/isprs-archives-XLIII-B5-2022-73-2022.

17. KWADJO, Danielle Tchuinkou et al. From PC2BIM: Automatic Model generation from Indoor Point Cloud. *Proceedings of the 13th International Conference on Distributed Smart Cameras*. 2019. Available also from: https://api.semanticscholar.org/CorpusID:203415358.

18. ROMERO-JARÉN, R.; ARRANZ, J.J. Automatic segmentation and classification of BIM elements from point clouds. *Automation in Construction*. 2021, vol. 124, p. 103576. ISSN 0926-5805. Available from DOI: 10.1016/j.autcon.2021.103576.

19. WANG, Chao; CHO, Yong K.; KIM, Changwan. Automatic BIM component extraction from point clouds of existing buildings for sustainability applications. *Automation in Construction*. 2015, vol. 56, pp. 1–13. ISSN 0926-5805. Available from DOI: 10.1016/j.autcon.2015.04.001.

20. SCHNABEL, Ruwen; WAHL, Roland; KLEIN, Reinhard. Efficient RANSAC for point-cloud shape detection. In: *Computer graphics forum*. Wiley Online Library, 2007, vol. 26, pp. 214–226. No. 2. Available from DOI: 10.1111/j.1467-8659.2007.01016.x.

21. ABDULLAH, Ahsan; BAJWA, Reema, et al. 3D Architectural Modeling: Efficient RANSAC for n-gonal Primitive Fitting. In: *Eurographics (Short Papers)* [online]. 2015, pp. 5–8 [visited on 2023-09-30]. Available from: www.researchgate.net/publication/312057903.

22. BORRMANN, Dorit; ELSEBERG, Jan, et al. The 3D Hough Transform for plane detection in pointouds: A review and a new accumulator design. *3D Research*. 2011, vol. 0202. Available from DOI: 10.1007/3DRes.02(2011)3.

23. RABBANI, Tahir; HEUVEL, Frank. Efficient Hough transform for automatic detection of cylinders in point clouds. *Proc ISPRS Workshop Laser Scan 2005, ISPRS Arch* [online]. 2005, vol. 36 [visited on 2023-09-30]. Available from: www.researchgate.net/publication/228422302.

24. TARSHA-KURDI, Fayez; LANDES, Tania; GRUSSENMEYER, Pierre. Hough-transform and extended ransac algorithms for automatic detection of 3d building roof planes from lidar data. In: *ISPRS Workshop on Laser Scanning 2007 and SilviLaser 2007*. 2007, vol. 36, pp. 407–412.

25. POUX, Florent; MATTES, Christian, et al. Automatic region-growing system for the segmentation of large point clouds. *Automation in Construction*. 2022, vol. 138, p. 104250. ISSN 0926-5805. Available from DOI: 10.1016/j.autcon.2022.104250.

26. EASTMAN, Charles. The Use of Computers Instead of Drawings in Building Design. *AIA Journal*. 1975, vol. 63.

27. CHENG, Jack; MA, Lauren. A BIM-based system for demolition and renovation waste estimation and planning. *Waste management (New York, N.Y.)* 2013, vol. 33. Available from DOI: 10.1016/j.wasman.2013.01.001.

28. SHI, Yi; XU, Jiuping. BIM-based information system for econo-enviro-friendly end-of-life disposal of construction and demolition waste. *Automation in Construction*. 2021, vol. 125, p. 103611. ISSN 0926-5805. Available from DOI: 10.1016/j.autcon.2021.103611.

29. CHAREF, Rabia; ALAKA, Hafiz; EMMITT, Stephen. Beyond the third dimension of BIM: A systematic review of literature and assessment of professional views. *Journal of Building Engineering*. 2018, vol. 19, pp. 242–257. ISSN 2352-7102. Available from DOI: 10.1016/j.jobe.2018.04.028.

30. GRIEVES, Michael. Origins of the Digital Twin Concept. 2016. Available from DOI: 10.13140/RG.2.2.26367.61609.

31. GLAESSGEN, Edward; STARGEL, David. The digital twin paradigm for future NASA and US Air Force vehicles. In: *53rd AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference 20th AIAA/ASME/AHS adaptive structures conference 14th AIAA*. 2012, p. 1818.

32. OETTINGHAUS, Sven. Digital construction with Building Information Modeling-the digital twin. *Wasserwirtschaft*. 2019, vol. 109, no. 5, pp. 86–89.

33. ZHONG, Dong; XIA, Zhelei; ZHU, Yian; DUAN, Junhua. Overview of predictive maintenance based on digital twin technology. *Heliyon*. 2023, vol. 9, no. 4, e14534. ISSN 2405-8440. Available from DOI: 10.1016/j.heliyon.2023.e14534.

34. TANNER, Karl. *What is L.O.D? Unederstanding Level of Development* [online]. 2021. [visited on 2023-10-17]. Available from: www.revitiq.com/level-of-development.

35. ISO CENTRAL SECRETARY. *Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries — Part 1: Data schema* [online]. Geneva, CH, 2018 [visited on 2023-10-02]. Standard, ISO 16739-1:2018. International Organization for Standardization. Available from: www.iso.org/standard/70303.html.

36. TETERVOV, Vitalij. *IFC, MVD, IDS - What's it all about?* [online]. 2022. [visited on 2023-10-08]. Available from: www.linkedin.com/pulse/ifc-mvd-ids-whats-all-vitalij-tetervov.

37. ISO CENTRAL SECRETARY. *Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure* [online]. Geneva, CH, 2016 [visited on 2023-10-06]. Standard, ISO 10303-21:2016. International Organization for Standardization. Available from: www.iso.org/standard/63141.html.

38. ISO CENTRAL SECRETARY. *Industrial automation systems and integration — Product data representation and exchange — Part 28: Implementation methods: XML representations of EXPRESS schemas and data, using XML schemas* [online]. Geneva, CH, 2007 [visited on 2023-10-06]. Standard, ISO 10303-28:2007. International Organization for Standardization. Available from: www.iso.org/standard/40646.html.

39. *IFC Data File Formats* [online]. 2023. [visited on 2023-10-06]. Available from: www.docs.fileformat.com/cad/ifc/.

40. BUILDINGSAMRT. *MVD Database* [online]. 2023. [visited on 2023-10-09]. Available from: www.technical.buildingsmart.org/standards/ifc/mvd/mvd-database.

41. BORRMANN, Andre; BEETZ, Jakob; KOCH, Christian; LIEBICH, T.; MUHIC, Sergej. Industry Foundation Classes: A Standardized Data Model for the Vendor-Neutral Exchange of Digital Building Models. In: 2018, pp. 81–126. ISBN 978-3-319-92861-6. Available from DOI: 10.1007/978-3-319-92862-3_5.

42. EBRAHIM, Mostafa Abdel-Bary. 3D laser scanners' techniques overview. *International Journal of Science and Research* [online]. 2015, vol. 4, no. 10, pp. 323–331 [visited on 2023-10-06]. ISSN 319-7064. Available from: www.researchgate.net/publication/282753883.

43. MIKHAIL, Edward; BETHEL, James S; MCGLONE, J Chris. *Introduction to modern photogrammetry*. John Wiley & Sons, 2001.

44. MOON, Daeyoon; CHUNG, Suwan, et al. Comparison and utilization of point cloud generated from photogrammetry and laser scanning: 3D world model for smart heavy equipment planning. *Automation in Construction*. 2019, vol. 98, pp. 322–331. Available from DOI: 10.1016/j.autcon.2018.07.020.

45. DELAUNAY, Boris et al. Sur la sphere vide. *Bulletin de l'Academie des Sciences de l'URSS*. 1934, vol. 7, no. 793-800, pp. 1–2.

46. MIKY, Yehia; KAMEL, Abdullah; ALSHOUNY, Ahmed. A combined contour lines iteration algorithm and Delaunay triangulation for terrain modeling enhancement. *Geospatial Information Science*. 2022, pp. 1–19. Available from DOI: 10.1080/10095020.2022.2070553.

47. WANG, Chengfeng; GAVRILOVA, Marina L. Delaunay Triangulation Algorithm for Fingerprint Matching. 2006, pp. 208–216. Available from DOI: 10.1109/ISVD.2006.19.

48. WANG, Xiaoling; LI, Songmin, et al. Application of visual simulation technology to a soil erosion protection project. *Landscape and Urban Planning*. 2008, vol. 84, no. 1, pp. 52–61. ISSN 0169-2046. Available from DOI: 10.1016/j.landurbplan.2007.06.008.

49. LUO, Yiming; MI, Zhenxing; TAO, Wenbing. DeepDT: Learning geometry from Delaunay triangulation for surface reconstruction. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2021, vol. 35, no. 3, pp. 2277–2285. Available from DOI: 10.1609/aaai.v35i3.16327.

50. VIRTANEN, Pauli; GOMMERS, Ralf, et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*. 2020, vol. 17, pp. 261–272. Available from DOI: 10.1038/s41592-019-0686-2.

51. EDELSBRUNNER, Herbert; KIRKPATRICK, David; SEIDEL, Raimund. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*. 1983, vol. 29, no. 4, pp. 551–559. Available from DOI: 10.1109/TIT.1983.1056714.

52. BELLOCK, KENNETH. *Toolbox for generating alpha shapes.* Python Software Foundation, 2021. Available also from: https://pypi.org/project/alphashape.

53. SANTOS, Renato César dos; GALO, Mauricio; CARRILHO, André Caceres. Extraction of Building Roof Boundaries From LiDAR Data Using an Adaptive Alpha-Shape Algorithm. *IEEE Geoscience and Remote Sensing Letters*. 2019, vol. 16, no. 8, pp. 1289–1293. Available from DOI: 10.1109/LGRS.2019.2894098.

54. SEENOUVONG, Nilakorn; WATCHAREERUETAI, Ukrit, et al. A computer vision based vehicle detection and counting system. In: *2016 8th International Conference on Knowledge and Smart Technology (KST)*. 2016, pp. 224–227. Available from DOI: 10.1109/KST.2016.7440510.

55. KURTULMUŞ, Ferhat; KAVDIR, İsmail. Detecting corn tassels using computer vision and support vector machines. *Expert Systems with Applications*. 2014, vol. 41, no. 16, pp. 7390–7397. ISSN 0957-4174. Available from DOI: 10.1016/j.eswa.2014.06.013.

56. YEHLIU, Kuen; VANDER WAL, Randy L.; BOEHMAN, André L. Development of an HRTEM image analysis method to quantify carbon nanostructure. *Combustion and Flame*. 2011, vol. 158, no. 9, pp. 1837–1851. ISSN 0010-2180. Available from DOI: 10.1016/j.combustflame.2011.01.009.

57. CHUNXIANG WANG Tao Jin, Ming Yang; WANG, Bing. Robust and Real-Time Traffic Lights Recognition in Complex Urban Environments. *International Journal of Computational Intelligence Systems*. 2011, vol. 4, no. 6, pp. 1383–1390. Available from DOI: 10.1080/18756891.2011.9727889.

58. *Thresholding (image processing) — Wikipedia, The Free Encyclopedia* [online]. 2023. [visited on 2023-10-31]. Available from: https://en.wikipedia.org/w/index.phptitle=Thresholding_(image_processing)&oldid=1140317997.

59. HARRIS, Charles R.; MILLMAN, K. Jarrod, et al. Array programming with NumPy. *Nature*. 2020, vol. 585, no. 7825, pp. 357–362. Available from DOI: 10.1038/s41586-020-2649-2.

60. BRADSKI, Gary. The openCV library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*. 2000, vol. 25, no. 11, pp. 120–123.

61. TEH, C-H; CHIN, Roland T. On the detection of dominant points on digital curves. *IEEE Transactions on pattern analysis and machine intelligence*. 1989, vol. 11, no. 8, pp. 859–872. Available from DOI: 10.1109/34.31447.

62. WU, S.-T.; MARQUEZ, M.R.G. A non-self-intersection Douglas-Peucker algorithm. In: *16th Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI 2003)*. 2003, pp. 60–66. Available from DOI: 10.1109/SIBGRA.2003.1240992.

63. DAL SANTO, Mariane Alves; WOSNY, GC; OLIVERIA, FH de. Algorithms for automated line generalization in GIS. In: *Proceedings of the Twenty-Eighth Annual ESRI User Conference*. 2008.

64. DOUGLAS, David H; PEUCKER, Thomas K. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization*. 1973, vol. 10, no. 2, pp. 112–122.

65. CLOUDCOMPARE. *Edit - Subsample* [online]. 2023. [visited on 2023-10-09]. Available from: www.cloudcompare.org/doc/wiki/index.php/Edit%5CSubsample.

66. ZHOU, Qian-Yi; PARK, Jaesik; KOLTUN, Vladlen. Open3D: A Modern Library for 3D Data Processing. *arXiv:1801.09847*. 2018.

67. *IfcOpenShell 0.7.0 Documentation* [Online]. 2023. [visited on 2023-07-06]. Available from eprint: www.pypi.org/project/ifcopenshell.

68. BUILDINGSMART. *Validation a service by buildingSMART international BETA Version 0.5.5* [online]. 2023. [visited on 2023-10-01]. Available from: www.validate.buildingsmart.org.

69. *IFC projects and contexts* [online]. 2020. [visited on 2023-10-05]. Available from: www.wiki.osarch.org/index.php?title=IFC_projects_and_contexts.

70. JUNG, Jaehoon; STACHNISS, Cyrill; JU, Sungha; HEO, Joon. Automated 3D volumetric reconstruction of multiple-room building interiors for as-built BIM. *Advanced Engineering Informatics*. 2018, vol. 38, pp. 811–825. ISSN 1474-0346. Available from DOI: 10.1016/j.aei.2018.10.007.

71. PREVITALI, Mattia; DÍAZ-VILARIÑO, Lucía; SCAIONI, Marco. Indoor building reconstruction from occluded point clouds using graph-cut and ray-tracing. *Applied Sciences*. 2018, vol. 8, no. 9, p. 1529.

72. THOMSON, Charles; BOEHM, Jan. Automatic geometry generation from point clouds for BIM. *Remote Sensing*. 2015, vol. 7, no. 9, pp. 11753–11775.

73. OCHMANN, Sebastian; VOCK, Richard; KLEIN, Reinhard. Automatic reconstruction of fully volumetric 3D building models from oriented point clouds. *ISPRS journal of photogrammetry and remote sensing*. 2019, vol. 151, pp. 251–262.

74. MURALI, Srivathsan; SPECIALE, Pablo; OSWALD, Martin R.; POLLEFEYS, Marc. Indoor Scan2BIM: Building information models of house interiors. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 6126–6133. Available from DOI: 10.1109/IROS.2017.8206513.

75. BASSIER, Maarten; VERGAUWEN, Maarten. Unsupervised reconstruction of Building Information Modeling wall objects from point cloud data. *Automation in Construction*. 2020, vol. 120, p. 103338. ISSN 0926-5805. Available from DOI: 10.1016/j.autcon.2020.103338.

76. CHENG, Baoquan et al. Windows and Doors Extraction from Point Cloud Data Combining Semantic Features and Material Characteristics. *Buildings*. 2023, vol. 13, no. 2, p. 507.

77. DÍAZ-VILARIÑO, L et al. Indoor modelling from SLAM-based laser scanner: Door detection to envelope reconstruction. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. 2017, vol. 42, pp. 345–352.

78. CHEN, Ziyuan. From Point Cloud to as-built BIM: automated Detection of Openings in existing Buildings. In: *Proceedings of 33. Forum Bauinformatik*. 2022.

List of Figures

List of Tables

Code List

List of Abbreviations and Symbols

BIM	Building Information Modeling
LiDAR	Light Detection and Ranging
RGB	Red-Green-Blue Color Model
DT	Digital Twin
CAD	Computer Aided Design
IFC	Industry Foundation Classes
ASCII	American Standard Code for Information Interchange
PLY	Polygon File Format
RANSAC	Random Sample Consensus
CCD	Charge-coupled Device
CMOS	Complementary Metal–Oxide–Semiconductor
MVD	Model View Definition
IAI	International Alliance for Interoperability
AEC	Architectural Engineering and Construction
FM	Facility Management
gbXML	Green Building XML
TIN	Triangulated Irregular Network
ROI	Region of Interest