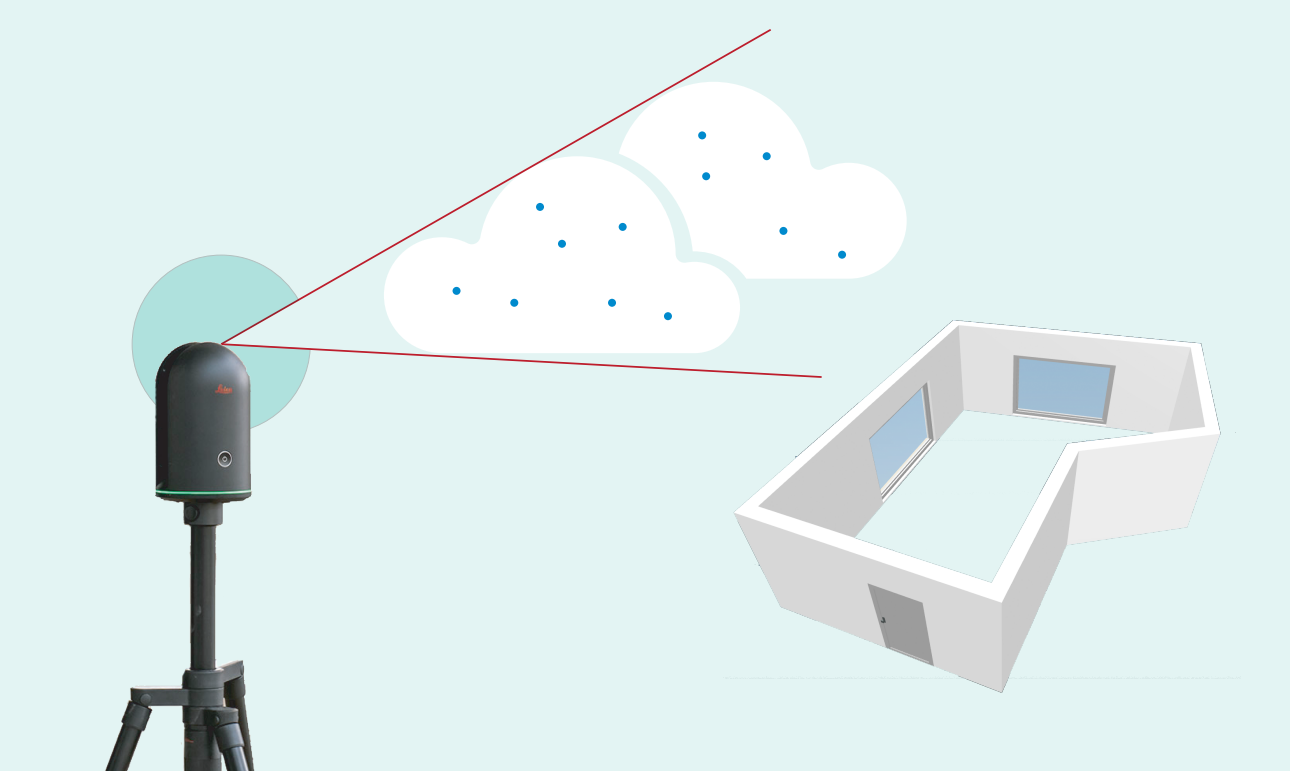
Architectural Floorplan Reconstruction



Bachelor Thesis
1st February 2022 - 1st August 2022

Adilla Böhmer-Mzee, 07-163-819

Supervisors:
Prof. Dr. Renato Pajarola
Lizeth J. Fuentes Perez

Visualization and MultiMedia Lab
Department of Informatics
University of Zürich

University of Zurich[UZH]

VISUALIZATIONANDMULTIMEDIALAB

Cover Image:

From laser scanner data to BIM model - PCD2CAD

Design by Adilla Böhmer

Zusammenfassung

Diese Bachelorarbeit befasst sich mit der automatischen Rekonstruktion von Architekturplänen aus Punktewolken. Oft stimmen gezeichnete Pläne nach einiger Zeit infolge von nicht dokumentierten Umbauarbeiten nicht mehr mit dem Ist-Zustand überein. Es ist auch möglich, dass alte Pläne noch nicht in digitaler Form vorhanden sind oder, dass Pläne im Laufe der Zeit verloren gegangen sind. In diesen Fällen ist es hilfreich, wenn neue, aktuelle Pläne schnell und präzise erstellt werden können. Eine rasche Aufnahmemöglichkeit der Räume bietet der Laserscanner BLK360 von Leica. Um die Punktewolke möglichst zeit- und kostensparend nachbilden zu können, wurde in dieser Arbeit ein autonomes Werkzeug entwickelt, dass die bis anhin manuelle, zeitintensive und fehleranfällige Rekonstruktion automatisiert. Das Tool bietet eine optimale Schnittstelle zwischen der Datenaufnahme und der Verwendung in Building Information Modeling (BIM) Anwendungen. Die vorgestellte Pipeline sieht vor, die wichtigsten baulichen Elemente zu erkennen, zu extrahieren und für den Export zu modellieren. Der Ablauf zeichnet sich dadurch aus, dass kein Umweg über die Erstellung von 2D Schnittebenen gemacht wird, sondern eine direkte 3D Modellierung angestrebt wird. Die Punktewolken werden mit einem Laserscanner erfasst und mit Hilfe der Point Cloud Library (PCL) in C++ verarbeitet. Um die Nähe und die Anbindung an gängige Computer Aided Design (CAD) Programme zu garantieren, wird ein Industry Foundation Classes (IFC) File aufgebaut und bereitgestellt. Im vorliegenden schriftlichen Teil dieser Arbeit werden zunächst verschiedene Abhandlungen zum Thema verglichen und analysiert. Dabei werden diverse Methoden und Abläufe gegenübergestellt, bevor im Hauptteil der Arbeit die Implementation der vorgeschlagenen Lösung präsentiert wird. Zum Abschluss findet der Leser eine Auswertung der Software sowie eine Schlussfolgerung und einen Ausblick auf zukünftige Erweiterungsmöglichkeiten des präsentierten Algorithmus.

Abstract

”What I do is the opposite of building walls. I build bridges. A bridge is something that connects instead of separating.” This statement by the world-famous architect Santiago Calatrava is to be turned on its head in this thesis. Even if the quotation is to be understood in a figurative sense, it is to be shown that walls not only separate but also create spaces and connect zones. Walls are to be reproduced in this project, creating a bridge between two technologies (laser scanner and Computer Aided Design (CAD) software). The captured point clouds are to be separated so that the spaces can be connected and provided for further processing.

This thesis focuses on the vision of an automatic reconstruction of the built environment based on a point cloud captured by a laser scanner. Architectural plans that document the as-is state of a built environment are in great demand. Often they have to be re-recorded in the view of the fact that, for example, only old hand drawings are available in which later modifications were not documented or for the reason that blueprints have been lost. It is likely that digital plans have not been kept up to date or that old CAD data are no longer compatible with new versions. In such cases, it should be possible to reproduce new or updated plans as swiftly and precisely as possible. For this purpose, an autonomous tool was developed that offers an optimal interface between data acquisition and the usage in Building Information Modeling (BIM) applications. The presented pipeline aims to recognise and extract the most important structural elements and to model them for export. The process is characterised by the fact that no diversion is made by the creation of 2D planes, however, a direct 3D modelling is aimed at. The point clouds are acquired with the Leica BLK360 laser scanner and processed in C++ using the Point Cloud Library (PCL). In order to guarantee the integration and the connection to common CAD programs, an Industry Foundation Classes (IFC) file is created and provided.

Contents

List of Figures

Listings

List of Abbreviations

BIM	Building Information Modeling
CAD	Computer Aided Design
CGAL	Computational Geometry Algorithms Library
GIS	Geographical Information System
GUI	General User Interface
IFC	Industry Foundation Classes
OBB	Oriented Bounding Box
PCD	Point Cloud Data
PCL	Point Cloud Library
RANSAC	Random Sample Consensus
SIA	Schweizerischer Ingenieur- und Architektenverein
SVG	Scalable Vector Graphics
TOF	Time of Flight

1 Introduction

A traditional profession such as architecture is always subject to small and large changes over the course of time. There is a constant change in demands, requirements and ideas, just as well, ever-changing and new technologies influence the daily business. For example, twenty years ago many plans were still drawn by hand and measurements were taken with a ruler, whereas today Computer Aided Design (CAD) programs and laser scanners are replacing the practice of the past. Taking older buildings into account, there is often only the availability of hand drawings, plans may have been lost over the years or structural changes were simply not documented. Furthermore, new building materials, new construction techniques or new tools influence building activities. Architecture is an extremely complex craft with different phases and processes, it is therefore, very crucial that building plans documenting the actual state of the building are available as fast and accurately as possible, with limited time and budget expenditure, and in a usable form. Such plans are needed for each of the different phases of planning, execution and maintenance of the buildings. In each phase, there are various professions and craftsmen involved and one of the main objectives of architects is to optimally plan, coordinate and, in the end, control all the sections and the work involved. To achieve this goal, there is a steady increase in the number of planners and contractors turning to Building Information Modeling (BIM) technology. According to Wikipedia [Wika], BIM, the acronym stands for Building Information Modelling, describes the networked planning, construction and management of buildings and structures with the help of software. The building data is digitally gathered, modelled and combined. In the past 10-15 years, this possibility of networking has brought about a strong change in thinking and the use of new technical means. In today's contemporary world, information is bundled and passed on to a wide range of stakeholders. It is precisely this complexity, that gives an idea of how far-reaching the consequences of a change in this process can be and how much time and good planning must be invested.

1.1 Motivation

The intensive planning and coordination in the construction industry are time-consuming and error-prone processes. The automation of the pipeline to generate a finished CAD file from a point cloud captured by a laser scanner is the focus of this paper. The motivation lies in the simplification and improvement of the processes in order to optimise data generation and data exchange. In particular, the aim of this work is to provide a tool that facilitates the step of plan generation from a point cloud. Through the recording with the help of a laser scanner, all points in the room are combined in a cloud and they are available to the planners at any given time, even if they are no longer on site. All measurement points can be retrieved whenever they are needed and the automated processing enables the availability of an Industry Foundation Classes (IFC) file within a very short time, which

can be optimally integrated into the design process. As a consequence, the tool contributes to minimising errors and helps to save time. Many of the possibilities offered by laser scanners are still not being fully exploited. On the one hand, this may be due to its high cost, however, on the other hand, new technologies will become more popular as soon as the time to be saved is significant. It can be assumed that the popularity of new tools will grow steadily, the more they are utilised and the more they can be applied in all layers of planning.

1.2 State of the art

Although the technical possibilities have advanced considerably in the meantime, a great deal of architectural offices in this country still work in a rather traditional way. In recent decades, the switch from hand drawings to CAD plans has been successful almost across the board, nonetheless, not all the possibilities offered by the current state of the art have been fully exploited. It would appear that that there are two areas where improvements could be made: to begin with, the possibilities of the existing software in the offices are not being sufficiently utilised, while in second place, there is limited investment in innovative technologies due to the fact that there is lack of awareness and high-cost. Only a few architectural offices use laser scanners or 3D printers in their daily work. Many offices still rely on conventional manual work for model construction or the creation of building recordings.

Most CAD programs are able to read in point clouds and work with them. However, the recording devices are still relatively expensive, it is not worthwhile for small companies to purchase them, hence the possibility of quick measurement recordings is simply left out. Instead, rooms and entire houses are measured and traced at great expense in terms of time and money. Therefore, if an architectural office does not have a laser measuring device at its disposal, built structures are recorded with a laser distance meter, or in some cases a tape measure, or double meter is still used for this purpose. Even if in some cases an external order is placed to measure a building using a laser scanner, the post-processing on the computer is regardless done manually. The drawn plans immediately give rise to the next problem: the compatibility of the CAD data and the exchange of data between the various planners and contractors. Often the interfaces of the individual programs are not compatible with each other and the respective user groups lack the knowledge of the technical background to make the data exchange as fast and easy as possible. For this reason, plans that are exchanged via different planner groups are often redrawn, which in turn involves an enormous amount of time and money.

1.3 Outline

In this thesis, various related works are first considered and compared. An overview of the possibilities offered by point cloud processing is given. Related concepts are discussed and the advantages and disadvantages of related workflows are described and listed. In the main part of the thesis, Chapter 3, the whole pipeline is presented and the implementation is shown in detail. We start with the acquisition of the point cloud, followed by the reading and pre-processing of the data. The extraction process and the reconstruction of the structural elements are explained step by step and illustrated with examples. Finally, the export of the generated IFC data

is explained and the integration into a CAD program is presented.

The purpose of this study is to generate an automated process for the creation of architectural floorplans from point clouds. The focus is on optimal usability in the daily life of architects and building planners. In order to generate the greatest possible benefit for planners, the creation of 2D floorplans is explicitly omitted, as these can be generated from a 3D model in a CAD program with just a few clicks. The aim is to achieve a level of detail that generates added value for the daily work of an architect, i.e., the generated plans should neither get lost in too small details, nor should the level of abstraction of the plans be too large. Furthermore, it is important that the plans can be generated in a reasonable time frame in order to achieve an optimal cost/ benefit balance. Since not all visions can be equally fulfilled within the scope of this thesis, we concentrate on the automatic extraction of floors, ceilings, walls, doors and windows. An evaluation of the results can be found in Chapter 4, where important conclusions are drawn. It would be exciting to extract more scale-related objects in an advanced research. Finally, in Chapter 5, these visions for the future are shown and discussed.

This paper is characterised by the fact that it is not dependent on specific laser scanners or specific CAD programs. The implemented program is able to read different input file types and by exporting in the IFC format, the output can be used not only by certain CAD programs, but as well by the whole range of drawing and design programs. The program can be used without prior knowledge and without the use of auxiliary programs. In addition, a video was created showing the process, the features and the handling of the software.

2 Related Work

In the field of point cloud processing, there are numerous research papers that deal with transforming unstructured data into a geometrically and semantically adequate and enriched form. Two main groups of related study can be distinguished. On the one hand there are studies, like the one by Mura/ Mattausch/ Villanueva/ Gobbetti and Pajarola [Mur+14], which solve sub-problems in a very robust and optimised way, however, they pursue higher-level goals and are not limited to the extraction of floor plans or to the later use in CAD programs. On the other hand, there are research papers, such as the one by Gankhuyan and Han [GH20], which, like the present study, are focused on the generation of architectural floorplans. The former differ from this paper in their input data, their methods, or their output. In particular, this study differs from others in that all steps are automated and, as resulting output, a file is generated for direct integration into a CAD program. The export is done in the IFC format. This is an open standard in the construction industry for the digital description of building models [ISO]. An additional highlight, is the fact that the input data was generated with a Leica BLK360. The Leica BLK360 is the smallest and lightest laser scanner of its kind and is straightforward and easy to use. It allows full field-of-view-scanning in standard resolution in less than 3 minutes (360,000 laser points/ second) [Lei].

2.1 Related concepts

Several methods and algorithms are used to model the pipeline proposed in this thesis. These are mostly well-known concepts from point cloud processing that focus their attention on different aspects. The algorithms differ, depending on the goal in terms of cost, performance, accuracy and other criteria.

Normal estimation

As one of the first concepts in the processing of point clouds, a normal estimation is used. Both Boulch and Marlet [BM12] and Mura/ Wyss and Pajarola [MWP18] present a robust solution for these tasks. Boulch et al. deploy a deep learning approach on robust normal estimation [BM16] and Ben-Shabat/ Lindenbaum and Fischer [BLF19] rely on neural networks. With reliable and usable progress in these current research fields, another milestone could be achieved for this study as well.

For time and effort management reasons, this implementation uses the method already integrated in the Point Cloud Library to estimate the normals [Rusb]. Since it was discovered through a brief comparison with the method used in the Computational Geometry Algorithms Library (CGAL) [All+] that the Point Cloud Library (PCL) method is faster for our point cloud, the latter was chosen. The exact reasons behind the difference

in performance were not further explored, however, in the test example used, the performance of the KD-tree [OLea] in the PCL was better than the one in the CGAL.

RANSAC and Hough transform

In a further step, the proposed pipeline uses a region growing algorithm [Ushb]. The resulting clusters are filtered, based on their normals and some other criteria before the remaining point sets undergo a plane model segmentation [Rusc]. Here, a plane is placed through each of the separated sets. The computation is done using the **Ran**dom **Sa**mple **C**onsensus, abbreviated RANSAC, and it is based on the 1981 publication of Fischler and Bolles' algorithm [OLeb]. The algorithm is generally used to recognise basic shapes in unorganised point clouds. Schnabel/ Degner and Klein as well present an automated algorithm for the detection of shapes in unorganised point clouds [SWK07]. They show that the RANSAC paradigms of Fischler and Bolles [FB81] and the Hough transformation of Hough [Hou62] are the two best known methods for shape extraction. Both have been shown to successfully detect shapes in 2D and 3D, and they are reliable even with a high percentage of outliers. Fischler et al. emphasise that lack of efficiency or high memory consumption is the biggest drawback of these methods. According to Schnabel et al., a considerable number of acceleration techniques have been proposed for both methods. None of them, alone or in combination with others, has been shown to provide an algorithm as efficient as the one proposed by Schnabel et al. for the task of extracting 3D primitive shapes [SWK07].

Min-cut, graph-cut and inside/ outside labelling

The algorithms presented below are used for binary segmentation of point clouds. In the min-cut algorithm from PCL, the cloud is divided into two groups based on the object centre and its radius: foreground and background points (points that belong to the object and those that do not) [Usha]. Oesau/ Lafarge and Alliez present a method for reconstruction using graph-cut. They process interior models by automatically extracting permanent structures, such as walls, floors and ceilings from raw point clouds. First, a room partitioning is performed where the point cloud is divided into horizontal slices. Vertical wall structures are detected by feature-preserving multi-scale line fitting and a clustering in a Hough transform space is performed. Graph-cut refinement is then used to extract the surface and a watertight surface mesh is obtained. Fidelity to the measured data is thus traded off against geometric complexity [OLA14]. The contribution of Lafarge and Alliez is a further development of the work of Oesau et al., where robustness, flexibility and efficiency are promoted in the extended approach. Their main innovation is in a structure-preserving approach to surface reconstruction. The graph-cut partitioning is formulated on the 3D Delaunay triangulation of the structured point set, with the tetrahedral labeled as lying inside or outside the cells [LA13].

The graph-cut method used in this paper is mainly based on the work of Michailidis and Pajarola [MP17], and as well the paper by Mura/ Mattausch and Pajarola [MMP16] is groundbreaking in this field and deals with the piece-wise planar reconstruction of interiors.

Triangulation

A substantial number of the pipelines compared in Section 2.2 differ from this work in the generation of grids that enclose the points, so-called meshes. Delaunay triangulation is a common method for creating a triangular mesh from a set of points [Wikb]. Turner and Zakhor present an approach that generates watertight triangulated surfaces from input point clouds. The input is converted to a voxelised representation, using a memory-efficient data structure. The triangulation is created by analysing spatial regions [TZ13]. Another important contribution is made by Schnabel/ Degner and Klein, who propose a method for completing and extending primitive shapes [SDK09].

Plane sweeping

A further concept, based on a segmentation process, is the plane sweeping method used by Budroni and Böhm. It builds on a hypothesis-and-testing strategy [BB10]. The paper by Budroni et al. follows the goal of CAD integration, nonetheless, the work is limited by the extraction of walls, floors, and ceilings. A further distinction with respect to the proposed pipeline is the fact that it operates with meshes. The method proposed in this paper does not require voxel transformation, or the generation of grids, or similar procedures.

Polygonal surface reconstruction

In the pipeline description of the next chapter, other implemented procedures are discussed, such as the use of the Oriented Bounding Box (OBB), the concave/ convex hull function, or the α-shapes algorithm. Similar to the polygonal surface reconstruction (PolyFit) function from CGAL, which provides a method for reconstructing piece-wise planar objects from point clouds [Nan], the previously mentioned methods provide tools that can be used for object detection in the point clouds. The use of this CGAL method was omitted with respect to the output of a watertight surface mesh. In contrast to these methods, the presented approach is based on an object-related reconstruction of the structural elements. Thus, the goal is not to place meshes over the points that are as detailed as possible, but to extract walls, ceilings, floors, openings, etc. directly as objects or, in other words, as structural elements. Nan and Wonka propose a novel framework for reconstructing lightweight polygonal surfaces. The reconstruction strategy based on hypothesis generation and selection is designed to reconstruct simple polygonal surfaces. Since the method has only been tested in smaller, more concise examples, it is unclear whether this method could be used in complex interior settings [NW17]. Stojanovic/ Trapp/ Richter and Döllner provide an alternative algorithm to the α-shapes approach. They focus on the detection and evaluation of primary (PB) and secondary (SB) boundary elements [Sto+19].

2.2 Related pipelines

The pipeline described in detail in Chapter 3 includes three core parts. First, there is the reading of the point cloud and the preparation of the data. As it shall be seen, this first step is very important to provide a good base for the subsequent algorithms. In the next step, the main building structures are detected and extracted,

Table 2.1: Related research overview with associated features. (Note: In the column *Processing* we distinguish between automatic processing and the use of learning algorithms. *VOX* stands for voxelised point cloud)

	Input		Output		Extracting		Processing	
	Point Cloud	*Spec.*	*IFC*	*2D*	*Walls*	*Openings*	*Autom.*	*Learn.*
Gankhuyag [GH20]	✓	–	✓	✓	✓	–	✓	–
Jung et al. [Jun+18]	✓	–	BIM	(✓)	✓	✓	✓	–
Macher et al. [MLG17]	✓	–	✓	(✓)	✓	✓	✓	–
Ochmann [Och19]	✓	–	3D	(✓)	✓	✓	✓	–
Okron et al. [Oko+10]	✓	–	–	✓	✓	–	✓	–
Pexman [Pex21]	✓	–	BIM	✓	✓	✓	✓	–
Thomson et al. [TB15]	✓	E57	✓	(✓)	✓	–	✓	–
Xiong et al. [Xio+13]	✓	VOX	(✓)	–	✓	✓	✓	✓

and to finish the process, the central architectural elements (walls, doors, windows, floors and ceilings) are reconstructed and stored in the IFC format. Addressing the connection to related process flows, the three core building blocks to analyse similar work shall be used. An overview of related work and associated features is shown in Table 2.1.

2.2.1 Pre-processing of point cloud data

As mentioned in the beginning, one goal of this work is to process point clouds acquired with a Leica BLK360. In contrast to this, another field of research has opened up in the last decades, which deals with data generated with the help of an RGB-D camera. An overview of current research in 3D reconstruction with such cameras is provided by Zollhöfe et al. [Zol+18]. Ambrus/ Claici and Wendt use the rather simple and inexpensive technology for automated room segmentation in unstructured 3D data of interiors [ACW17]. Furthermore, Murali/ Speciale/ Oswald and Pollefeys propose a system that generates automatic 3D layouts from building interiors created with mobile 3D deep sensors. They work with 3D mesh and do not consider windows [Mur+17].

One of the most important steps in the process pipeline involves reading and pre-processing the acquired data. This process is of absolute importance for the reason that the further algorithms are based on it and only with good starting data, the optimal conditions for a good result are given. For this purpose, the most robust methods possible should be applied, which guarantee a positive consensus between performance and accuracy. Ideally, we get a result that contains all structural elements and is free of clutter, noise, and unwanted objects such as furniture, etc.

To achieve a good starting point for the extraction of the architectural elements, it is important to free the spaces from as much furniture and other clutter as possible. For this, Monzeglio's PointCloudAnnotator [Mon20] offers a solid approach. With the toolkit presented, sets of points can be grouped by selecting individual points, or entire regions of points can be selected together. There is in addition, a possibility of boolean union of existing groups. Since the selection of points is usually done by mouse clicks, however, in this case, an automated solution is aimed at, this tool was omitted.

A similar solution oriented approach, which deals with the division of the point cloud into subgroups such as furniture, windows and other elements, is that of Armeni et al. Here, the point cloud is first divided into disjoint spaces before a classification by semantic elements is performed [Arm+16]. As this shall be addressed in the introduction of the pipeline in the next chapter, the freeing of spaces from furniture and clutter is done in a similar way.

As already shown in Section 2.1 *Related concepts*, the estimation of normals is the first important step in dealing with point clouds. Building on this, Mura et al. present a workflow for automatic room detection and reconstruction in cluttered indoor environments with complex room layouts [Mur+14]. In a slightly different form, this work forms the core of the pre-processing pipeline of point cloud data presented in Chapter 3.

2.2.2 Extracting planar primitives

A large number of related studies follow the approach of projecting the point clouds into a 2D ground plane to subsequently extract walls from the generated voxel space using appropriate methods [Oko+10; GH20; MLG17; Jun+18; Pex21; Xio+13]. As the example of Okron/ Xiong/ Akinci and Huber shows, this involves first removing the floor and ceiling points and then discretising the remaining points into a voxel space. By projecting them onto the x-y plane, this produces a 2D density histogram. Okorn et al. show in their paper that it can be useful to refine the cutting planes for discretising the voxel space. Since there is more clutter near the floor due to furniture and other objects, and there is likely more clutter in the ceiling area due to lamps, they suggest computing the ground plane histogram using selected height segments [Oko+10]. The detection and extraction of the walls is done, for example, as in Gankhuyag et al. using a RANSAC algorithm [GH20] or a Hough transform algorithm as we encounter in Okron et al. [Oko+10]. Since it is robust to noise, Oesau et al. in addition use, among others, the least square method for this purpose, nevertheless, it is not very robust to outliers [OLA14]. In their article, Macher/ Landes and Grussenmeyer present a processing chain for point clouds designed for 3D reconstruction of interiors of existing buildings, which is completed with integration into BIM software [MLG17]. This pipeline has some parallels with the proposed idea, however, the work differs in certain respects. For instance, in contrast to this work, Macher et al. present a semi-automatic solution and they use a binary image to segment the point cloud [MLG17]. The two pipelines have their commitment to ideal BIM integration in common. However, they differ in their methods.

The paper by Gankhuyag et al. follows the same input/ output strategy as our pipeline. The paper is characterised by its closeness to CAD integration. The presented solution additionally includes a Scalable Vector Graphics (SVG) file, making it possible to use the data extracted from the point cloud without a professional architecture program. The pipeline differs from this work in its use of a grid structure, or depth image, which is used to extract the building elements [GH20]. Unfortunately, this pipeline so far neglects any openings such as windows or doors.

Furthermore, a closely related solution is presented by Jung et al. After the height estimation phase the pipeline provides a room segmentation followed by a floor-wall boundary extraction. After the processing of this extraction, a three-step modelling phase follows in which the 3D model is built. The pipeline is able to extract and model doors and windows, which is in line with the idea of this work. The drawback in the paper by Jung et al. is that the proposed solution is not fully automated. Some dimensions have to be measured manually beforehand. Unfortunately, the export of the data is not in the IFC format [Jun+18].

The master thesis of Pexman provides a comprehensive insight into the possibilities of automated creation of floor plans and architectural models from point clouds. The concepts are described in detail. The algorithms have been analysed and partially optimised and the results have been systematically reviewed. The pipeline is close to ours, although the specific input and output data differ from each other [Pex21].

In particular, the work of Ochmann [Och19] should be highlighted. His PhD thesis is very comprehensive and we will look at his approach to door detection in more detail in Section 3.1.2.

2.2.3 Output generation

In principle, all pipelines considered pursue the intention of generating added value from the point clouds. Since this study aims at a strong BIM integration, which should be fulfilled by an IFC file, our attention is mainly focused on the studies that pursue a similar ambition. Thomson and Boehm demonstrate a very analogous idea, omitting the extraction of openings. Like Macher et al. [MLG17] and Gankhuyan et al. [GH20], the work of Thomson et al. generates and exports an IFC file. Furthermore, Thomson et al. present quality indicators for the reconstructed geometric elements, as well as a framework to evaluate the quality of the reconstructed geometry compared to a reference [TB15].

Xiong/ Adan/ Akinci and Huber use a deep learning approach for their pipeline. Their method starts by extracting planar pieces from a voxelised version of the input point cloud. This involves extracting walls, floors, ceilings, windows, and doors, as in our algorithm. In contrast, however, in Xiong et al., this is done by an algorithm that learns to recognise the unique features of different surface types and the contextual relationships between them, and it uses this knowledge to automatically label patches as walls, ceilings or floors. Xiong et al. recognised the importance of BIM integration early on and are working to convert their surface-based method into a volumetric model and export an IFC file [Xio+13].

3 Implementation

The introduction states that the goal of this project is to provide a program for automated creation of architectural plans. What is special about this is that, for the first time, the entire process from reading the point cloud to exporting an IFC file that can be used in a CAD program is automated to a valuable level of detail. Up to this point, either only parts of this pipeline were implemented or the process was dependent on manual input. The large scale of the workflow being realised in this project creates many opportunities for intervention in the overall algorithm. The speed at which the IFC model is computed is good and the point cloud replication is as expected. It is conceivable that a revision of the extraction methods will allow an even better mapping.

The pipeline was implemented on an Apple M1 pro in the C++ programming language, for which the JetBrains IDE CLion was used. The Point Cloud Library (PCL) was mainly used for processing the point cloud. Due to all demands and desires could already be covered by the PCL, the use of CGAL was omitted. Some short tests between the two libraries were performed, which showed that the PCL is currently better suited for our needs. In order not to have to transfer the point cloud unnecessarily from one format to the other, a combination of the two libraries was avoided. For the General User Interface (GUI) we worked with QT [QT], a library for cross-platform development of graphical user interfaces. IFC++ [Ger], an open source IFC implementation for C++, was used to create the IFC file.

3.1 Pipeline

The entire implemented pipeline is described in detail below. To get an overview, consider Figure 3.1, where the main components of the pipeline are shown. We start by reading in the data followed by converting the point cloud to the Point Cloud Data (PCD) format that can be used in the PCL. This is followed by all the pre-processing of the point cloud in order to extract all the required structures and finally provide an IFC file. The latter is done by IFC modelling of the extracted objects and elements. An overview of the entire pipeline can be seen in Figure 3.2.

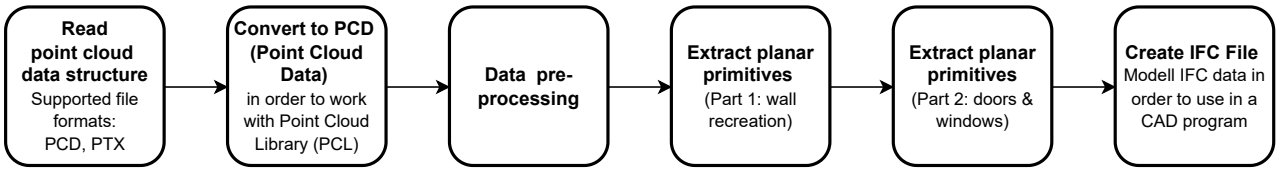


Figure 3.1: An overview of the main components of the project pipeline

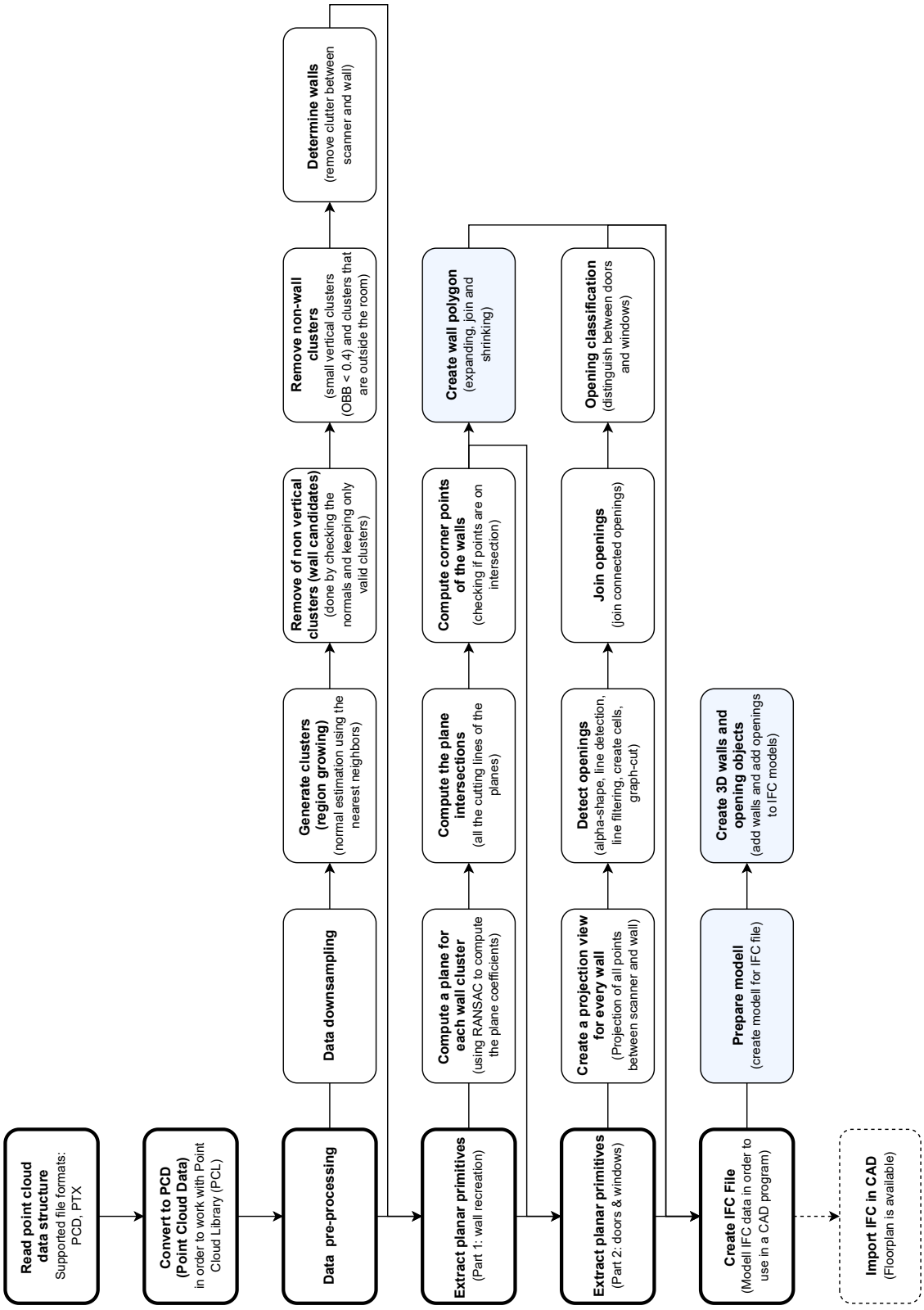


Figure 3.2: Detailed representation of the process pipeline

3.1.1 Point cloud conversion

The Leica BLK360 laser scanner is capable of capturing a complete full dome scan in less than 3 minutes. This corresponds to a measurement rate of around 360,000 points/ second. The high-speed Time of Flight (TOF) distance measuring system achieves an accuracy of 6 mm at a distance of 10 meters and a precision of 8 mm at a distance of 20 meters. Points can be recorded at a distance of 0.6 meters - up to 60 meters from the scanner and, as the name suggests, they can be recorded at a horizontal angle of 360°. Vertically, an angle of 300° is possible, however, the area directly under the scanner is not reached. The scanner can be controlled either directly via the device or with the help of an Apple iPad Pro and the corresponding iPad app [Lei].

For our test data sets, the Leica BLK360 with the iPad Pro app was used and eleven scans were taken on the first floor of the IFI building at the University of Zurich. This results in an E57 format when exporting. In Register360, Leica's point cloud registration tool, it is then possible to create a PTX file. The Leica PTX format has a header followed by the x, y and z coordinates of each point. One line corresponds to one point where the first three values correspond to the XYZ values, followed by the α-value and the colour of the point as RGB (another three values). A PCD file further consists of a header and the recorded data points. Here, the points are represented by the XYZ values and one RGB value.

For processing the point cloud with the help of PCL, the PTX file is first converted into a PCD file when it is opened. Both formats are shown below. The header of the PTX file can be found in Listing 3.1 and an excerpt from a real world PTX file can be found in Listing 3.2. Similarly, the header of a PCD file can be found in Listing 3.3 and the real world example in Listing 3.4. The difference between the two formats lies mainly in the header, nonetheless, the data points are represented somewhat differently [Bou; Poi]. The already mentioned E57 format is not needed in this work, however, for additional information please refer to the paper *The ASTM E57 File Format for 3D Imaging Data Exchange* by Huber [Hub11].

Listing 3.1: Header of a Leica PTX file

```
1 number of columns
2 number of rows
3 st1 st2 st3     // scanner registered position
4 sx1 sx2 sx3     // scanner registered axis 'X'
5 sy1 sy2 sy3     // scanner registered axis 'Y'
6 sz1 sz2 sz3     // scanner registered axis 'Z'
7 r11 r12 r13 0   // transformation matrix
8 r21 r22 r23 0   // this is a simple rotation and translation 4x4 matrix
9 r31 r32 r33 0   // just apply to each point to get the transformed coordinate
10 tr1 tr2 tr3 1   // use double-precision variables
```

Listing 3.2: Sample snippet of a PXT file

```
2578
1068
0.00000000 0.00000000 462.07976183
-0.11866899 0.99289775 -0.00846890
-0.99292812 -0.11863497 0.00441436
0.00337830 0.00893285 0.99995439
-0.11866899 0.99289775 -0.00846890 0.00000000
-0.99292812 -0.11863497 0.00441436 0.00000000
0.00337830 0.00893285 0.99995439 0.00000000
0.00000000 0.00000000 0 1.00000000
0.75981 -0.00091 -1.32350 0.37909 34 43 63
0.76487 -0.00091 -1.32475 0.34392 30 39 58
0.76867 -0.00092 -1.32382 0.34001 30 38 57
0.77596 -0.00093 -1.32883 0.34783 30 38 59
0.78378 -0.00094 -1.33468 0.35955 31 38 61
0.00000 -0.00000 -0.00000 0.34001 30 38 57
0.78405 -0.00095 -1.32024 0.32829 29 35 55
0.78842 -0.00096 -1.32020 0.32438 29 35 54
0.79526 -0.00097 -1.32425 0.32047 28 36 54
0.80152 -0.00098 -1.32726 0.33219 28 36 57
0.80794 -0.00099 -1.33050 0.33610 29 38 57
...
```

Listing 3.3: Header of a PCD file

```
# .PCD v0.7 - Point Cloud Data file format  // The official entry point for the PCD
VERSION 0.7                 // PCD file version
FIELDS x y z rgb            // name of each dimension/field that a point can have
SIZE 4 4 4 4                // size of each dimension in bytes
TYPE F F F U                // type of each dimension as a char
COUNT 1 1 1 1               // how many elements does each dimension have
WIDTH 230                   // width of the point cloud data set (no. of points) *
HEIGHT 1068                 // height of the point cloud data set (no. of points) **
VIEWPOINT 0 0 0 1 0 0 0     // acquisition viewpoint for the points in the dates
POINTS 2753304              // total number of points in the cloud
DATA ascii                  // data type that the point cloud data is stored in

// * WIDTH has two meanings: it can specify the total number of points in the cloud
     for unorganised data sets; it can specify the width (total number of points in a
     row) of an organised point cloud data set
// ** HEIGHT has two meanings: it can specify the height (total number of rows) of an
     organised point cloud data set; it is set to 1 for unorganised data sets (thus,
     used to check whether a data set is organised or not).
```

Listing 3.4: Sample snippet of a PCD file

```
# .PCD v0.7 - Point Cloud Data file format
VERSION 0.7
FIELDS x y z rgb
SIZE 4 4 4 4
TYPE F F F U
COUNT 1 1 1 1
WIDTH 2578
HEIGHT 1068
VIEWPOINT 0.0000 0.0000 0.0000 1 0 0 0
POINTS 2753304
DATA ascii
-0.0937335 0.742699 -1.32988 2239295
-0.0943382 0.747712 -1.33117 1976122
-0.0947761 0.751494 -1.33027 1975865
-0.0956481 0.758689 -1.33535 1975867
-0.096586 0.766402 -1.34126 2041405
-0.0965593 0.766801 -1.32682 1909559
-0.0970678 0.771141 -1.32682 1909558
-0.0978833 0.777898 -1.33093 1844278
-0.0986264 0.784087 -1.33399 1844281
-0.0993893 0.790434 -1.33729 1910329
...
```

3.1.2 Pre-processing of point cloud data

Taking a look at the pre-processing step of our pipeline, it is divided into five sub-steps, which are described in the following. An overview of it is depicted in Figure 3.3. This first main step is very important for the later modelling of the architectural objects since all calculations, or the reproduction of the structural elements, are based on this data cleansing. The idea and the approach in this section is based on the work of Mura et al. [Mur+14], who in Chapter 4 *Occlusion-aware selection of candidate walls* of their work, show a robust solution for the selection of candidate walls.

All five steps of this sub-process are repeated. This means that each scanner image is read individually and each room or image goes through this process. The main reason for this procedure is that the Leica Register360 tool stores the individual scans in PTX format one by one. In Register360, all scans can be aligned and merged, while the scanner position remains stored. Unlike the PTX format, the E57 format stores all data in a bundle. This would as well be possible with PTX to a certain extent, nevertheless, would hardly offer an advantage, since on the contrary, a larger amount of data would have to be handled at once.

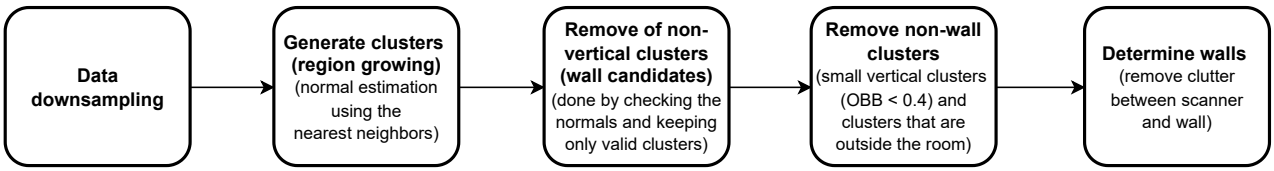


Figure 3.3: Representation of the pre-processing pipeline

Data down-sampling

The PCL method *Voxel Grid* is used for data reduction of the point cloud. One can imagine a division of the point cloud into small boxes in which the points are summarised and returned in reduced form. A voxel size (cube size) of 1.5 cm was chosen for the reduction. Thus, a three-dimensional grid is placed over the point cloud. In contrast to other similar methods, the chosen approach does not calculate the centre of a voxel, therefor, the centroid of the voxel in relation to the points within it. This approach is somewhat slower, thus, for this more accurate. The grid divisions, i.e. the voxels, are only used for the calculation, the result of the function is a reduced point cloud in the original format [Rusa].

Generate clusters

This step was carried out with the help of the PCL. A region growing segmentation was used, whereby the chosen algorithm first requires a robust estimation of the normals, which we have already seen in Chapter 2.1. This estimation was solved by the PCL *Estimating Surface Normals in a PointCloud* algorithm [Rusb]. Especially in areas like room corners, the estimation of the normals is not trivial and the parameters have to be adjusted accordingly to generate an acceptable result. In our case, the threshold for the k-search was set to fifty points. Alternatively, a radius of a few centimetres could have been chosen.

The region growing segmentation groups the points that are close enough to each other in terms of smoothness conditions [Ushb]. The aim of grouping is to be able to decide later which groups are relevant and which can be deleted. For this reason it is important that, for example, all points that belong to the same wall or the same furniture side respectively the same surface are grouped in the same cluster. There are as well various parameters to be set for this purpose. An overview of the chosen values can be found in the code snippet of Listing 3.5. Figure 3.4 shows the state of the point cloud after it has been reduced and divided into clusters (region growing).

Listing 3.5: Code snippet: Region growing segmentation

```
pcl::RegionGrowing<point_t, pcl::Normal> regGrow;

    regGrow.setMinClusterSize(1000);
    regGrow.setSearchMethod(tree);
    regGrow.setNumberOfNeighbours(30);
    regGrow.setInputCloud(inPointCloud);
    regGrow.setInputNormals(normals);
    regGrow.setSmoothnessThreshold(3.0/180.0*M_PI); // radians to degrees (3 degrees)
    regGrow.setCurvatureThreshold(1.0);
    regGrow.extract(clusters);
```

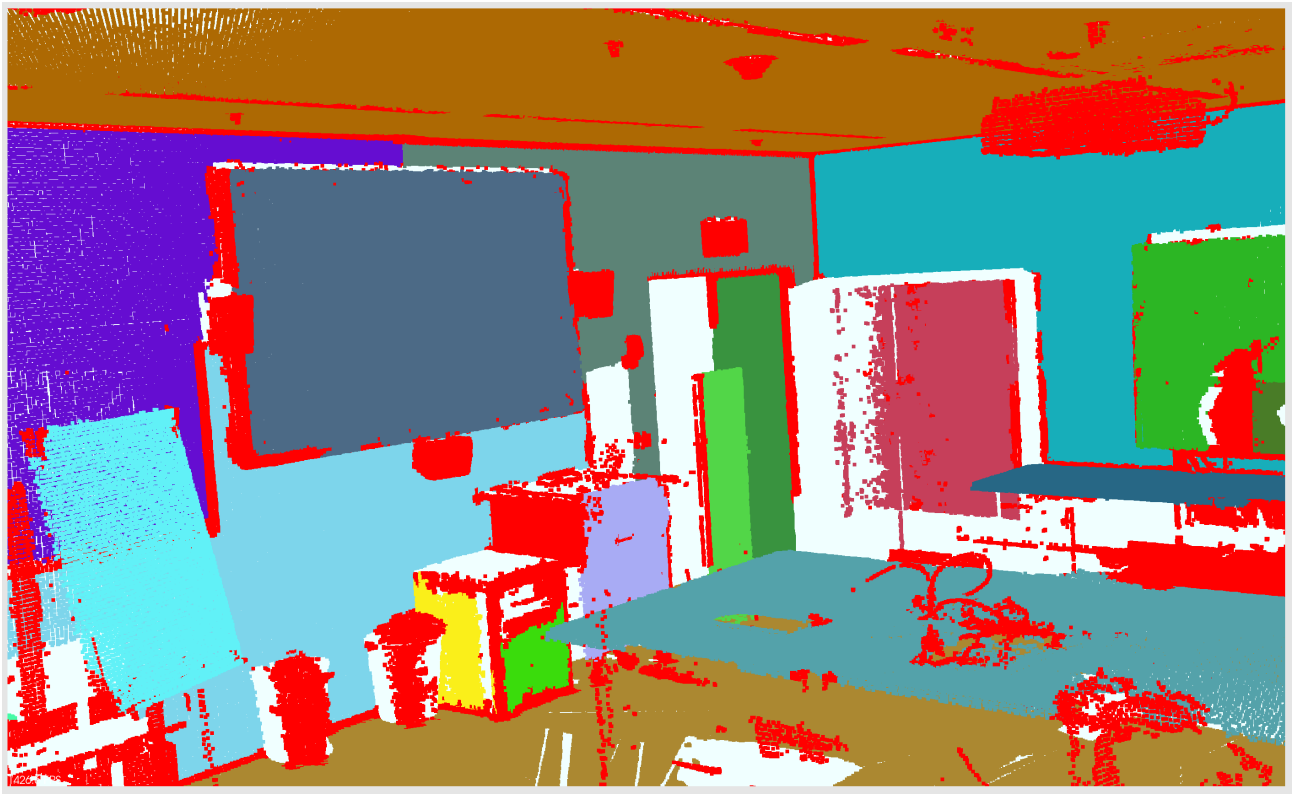


Figure 3.4: Result of the generated clusters after down-sampling and region growing

Remove non-vertical clusters

All clusters that are not vertical are eliminated in this step. This is a very strong simplification of the point cloud, which does not only bring advantages. Although all surfaces such as table planes or the seat of a sofa are removed, the disadvantage is that, for example, roof slopes or ramps are removed as well, restricting further processes. The idea of this solution is that walls are vertical, which is often, however, not always true. Specifically, all vertical cluster indices, i.e. indices of clusters for which the equation $|\mathbf{n} * \mathbf{v}_z| < \varepsilon$ is

fulfilled, are stored. ε is chosen as close to 0 as possible. The normals of all points were estimated before the clusters were created and used to calculate the region growing segmentation. Therefore, we already have the information on how the clusters are aligned in space. Consequently, **n** is already the averaged normal vector of the corresponding cluster. Figure 3.5 illustrates the idea of the calculation.

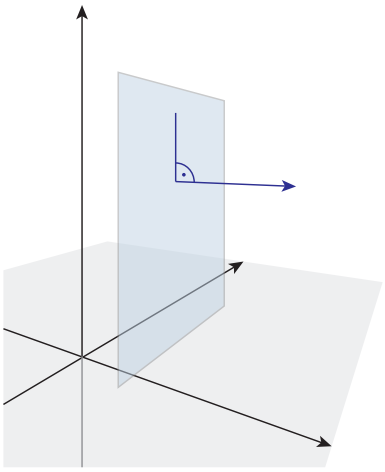


Figure 3.5: Vertical cluster with corresponding normal vector

Remove non-wall clusters

Besides the walls, there are most likely other vertical fragments left. To decide which of the remaining clusters actually represent walls and which still need to be removed, a method was developed that involves several decision steps. First, an OBB is calculated for each cluster. Then, all clusters with an extension of less than 0.4 m are eliminated. For this, both the length and the width of the OBB are taken into account. At the same time, those clusters are in addition deleted whose OBB have a depth dimension that is greater than a certain threshold value (in our case more than 0.1 m). This is the case, for example, with roundings. At the moment, the proposed procedure cannot process a rounding that represents a pillar, for example, in a meaningful way. This is because, the wall detection algorithm, which is explained in detail in Section 3.1.3, so far only works for planes. After all, a column consists of many narrow planes, which is why the pipeline can be supplemented to this effect at a later stage. In the IFC model, pillars can be modelled without any problems, which is certainly an advantage for a future extension. Somewhat more complex to model are round walls, which have to be composed of many planes or very short wall segments.

Next, all the clusters that are outside the room are removed. They can be recognised by the fact that their minimum or maximum height is greater than the floor or ceiling height. An example of this can be found in Figure 3.6. This phenomenon occurs when recording points through a window. In the view of the fact that the pipeline was designed so that no room segmentation is necessary (i.e., it is not necessary to calculate which recorded points belong to which room), points that were recorded through a door and are in addition still present as a vertical cluster cannot yet be filtered. Therefore, at this point in time, it is still important that the

recorded point clouds do not contain such outlier points if possible. This can be ensured when exporting the point cloud from Register360. In works like Ochmann's [Och19] this problem is solved by splitting the point cloud into disjoint spaces. The advantage of splitting is in addition that later doors can be detected by checking for overlapping zones when viewed from two scanner points. This can further be seen in Ochmann [Och19]. Since in the algorithm proposed in this paper the space division is not absolutely necessary, it was omitted in favour of the performance and the detection of the doors was solved somewhat differently, as we will see in Section 3.1.4.

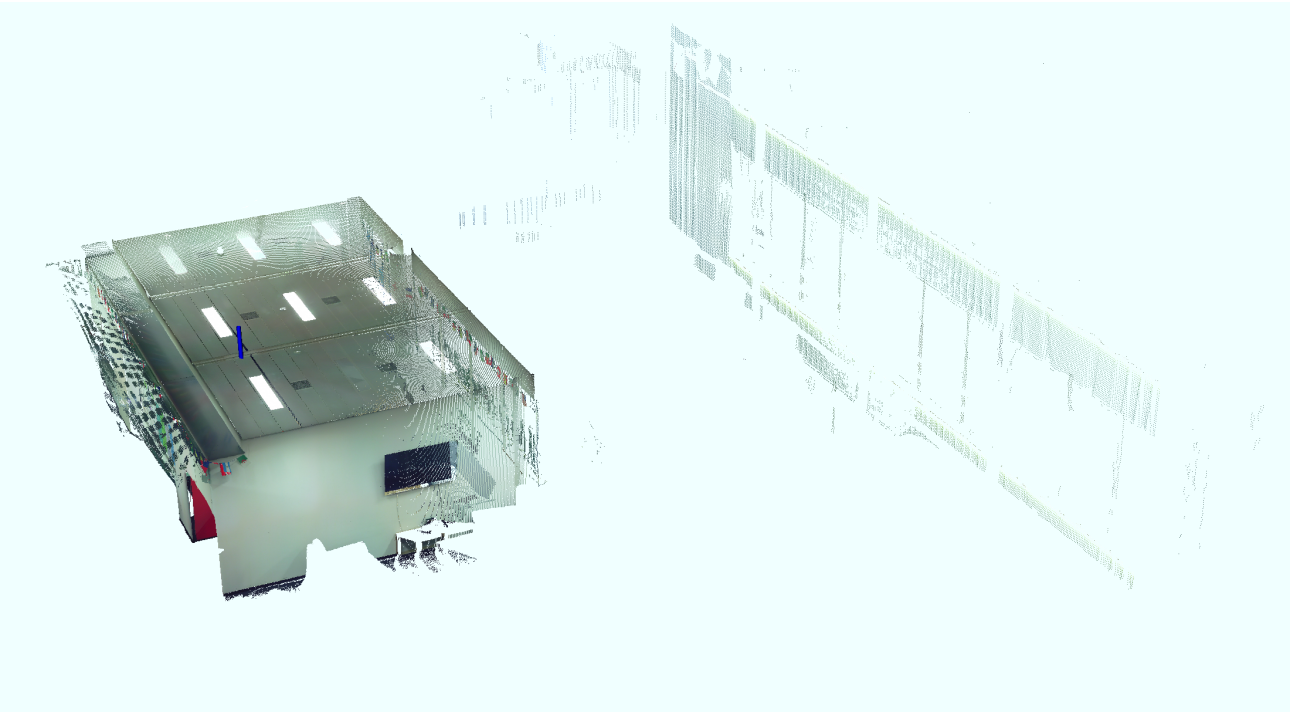


Figure 3.6: Point cloud with outside-room clutter

Determine walls

In the last step of the pre-processing pipeline, all clusters that are located between the scanner and the wall are removed. For this purpose, a vector with a fixed scalar is placed from the scanner through the four corners of a z-axis OBB of each cluster. The field resembling a truncated pyramid that is created by this projection (behind the cluster) is used to detect any collisions with other clusters in the field of view. Specifically, it is checked whether there is another object behind a cluster. If this is the case, it can be assumed that the cluster under consideration is not a wall. In principle, it is calculated whether there is a z-axis OBB in the truncated pyramid-like field. For this, a pragmatic solution was implemented in which the z-axis OBBs are filled with points and it is checked whether these are located in the corresponding volume. The described procedure can be found in Figure 3.7 and the intermediate status after the pre-processing phase can be found in Figure 3.8.

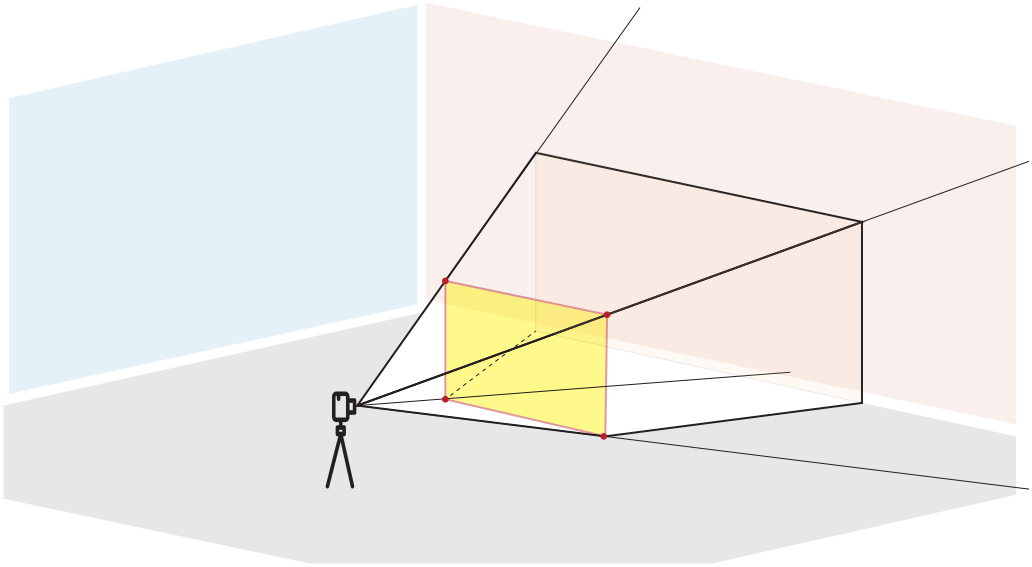


Figure 3.7: Sketch of the calculation basis with the line construction through the corner points of the z-axes OBB

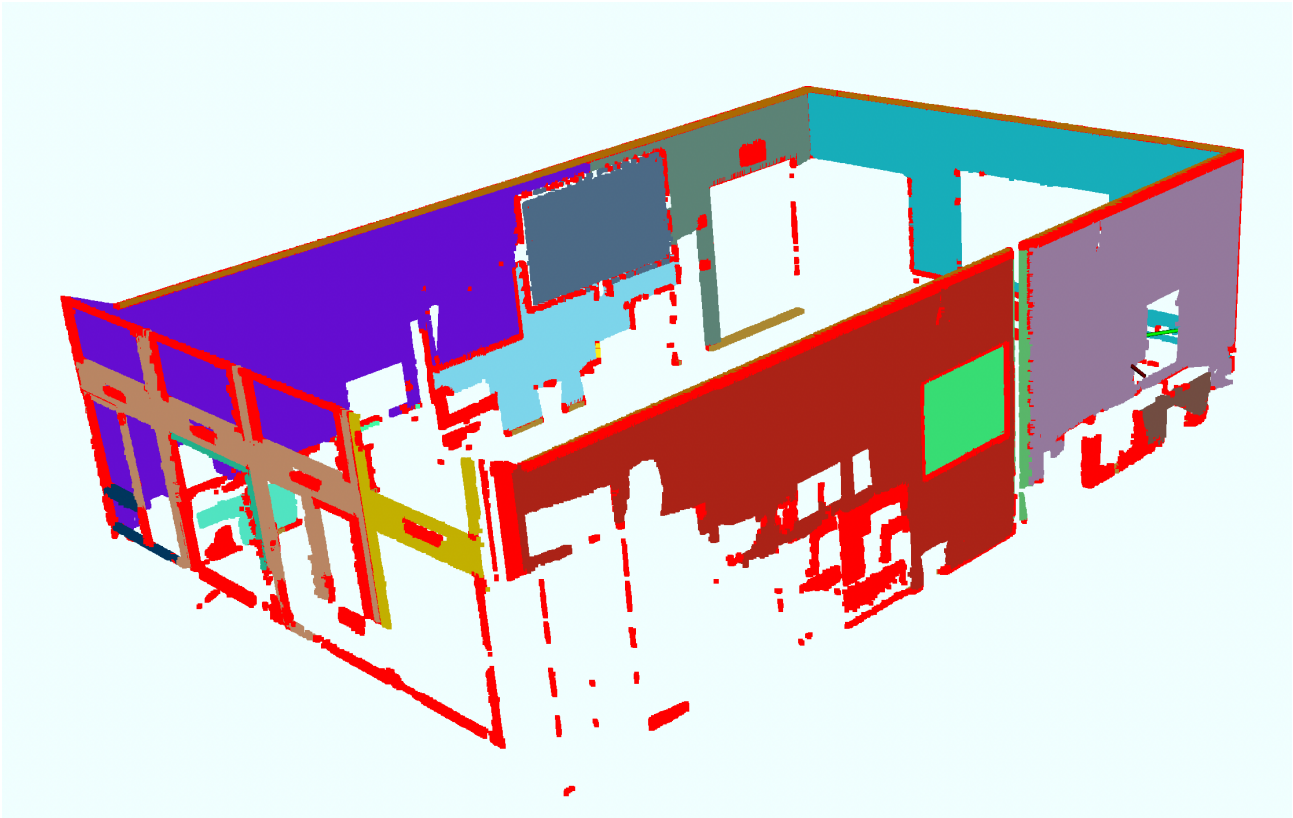


Figure 3.8: Intermediate status of the point cloud after the pre-processing phase

3.1.3 Extracting planar primitives (Part 1: walls)

In the first part of the extraction phase, the walls are reproduced. An overview of the corresponding steps can be seen in Figure 3.9. As it can be seen here, this process is divided into four steps, whereby the first three steps are repeated (i.e. for each PTX file read) and the last step only has to be carried out once.

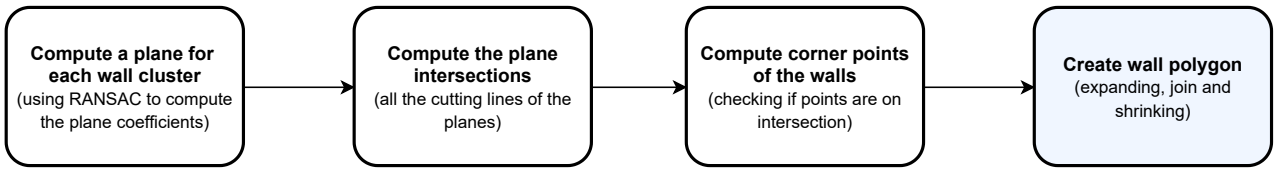


Figure 3.9: Pipeline of the planar primitives extraction (Part 1: wall recreation)

Compute a plane for each wall cluster

This step is practically straightforward. With the help of the RANSAC algorithm, a plane is placed through each cluster. For this purpose, the PCL algorithm *Plane model segmentation* [Rusc] is used. This algorithm takes a point cloud cluster as input, fits the plane, and returns the four coefficients of the plane equation. The plane then has the form $ax_1 + bx_2 + cx_3 + d = 0$, where a, b, c and d are the coefficients calculated by the algorithm and all x_1, x_2 and x_3 that satisfy this equation are points that lie on this plane. For illustration, consider the parameter form that explains this situation geometrically, as we see it in Figure 3.10. The plane is determined by the location vector $\boldsymbol{r}$ and the two direction vectors $\boldsymbol{v}$ and $\boldsymbol{w}$ (compare Equation (3.1)). All points that fulfil the plane equation and whose parameters r and s lie between 0 and 1 belong to the cluster, which was the basis for the plane equation. In addition, each plane is aligned vertically, as the clusters can be slightly skewed due to the parameters that were used for the calculation.

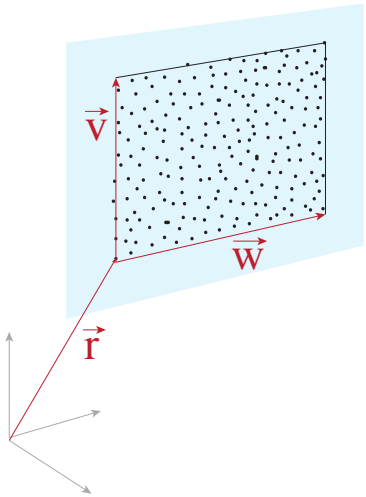


Figure 3.10: Sketch of the plane model segmentation calculation

$$E : \vec{x} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix} + s \cdot \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} + t \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} \text{ where } r, s \in \mathbb{R} \tag{3.1}$$

Compute the plane intersections

The procedure in this step is likewise very intuitive. There are three ways in the three-dimensional space in which two planes can lie in relation to each other. They are parallel, lie on top of each other, or intersect. For the calculation of the intersection line, the latter are therefore of importance. In general, we calculate the intersection line g of two planes by placing the parametric form of one plane into the coordinate notation of the other plane (Figure 3.11). Now, we can rearrange the equation according to one of the parameters s or t and insert the parameter into the parameter form. Since only one parameter is left, this form corresponds to the linear equation of g and all that remains is to check how many solutions the set has. If the solution is always true, the planes are identical. In contrast, if the equation is false for every case, no solution exists, and the two planes must be parallel. In the third case, the planes intersect and there is a solution that corresponds exactly to the equation of a line. Since we assume in our case that the z-axis is orthogonal to the x-y plane, the procedure could be optimised. However, in order to be able to extend the program later for other cases, this was not done.

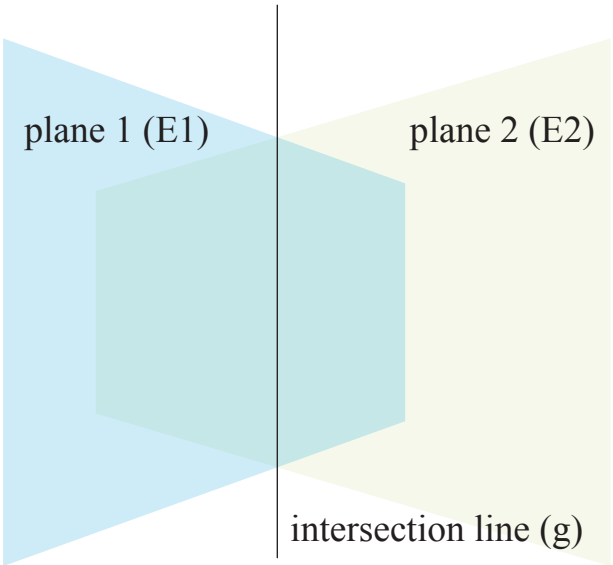


Figure 3.11: The plane intersection of two planes and the resulting intersection line

Compute the corner points of the wall

This calculation step has been simplified in comparison to an earlier solution, in which the planes of all walls of one storey were calculated simultaneously, in that only the planes of one room are processed at a time. The

calculation of the walls remains the same in principle (it is only split up), however, it is no longer necessary to check which walls form a room together. In the previous step, we calculated the intersecting lines of the planes. In doing so, each plane was intersected with every other plane and we obtained all the intersection lines. Now, we look at each plane in turn with its corresponding point cloud cluster. The intersection lines that lie on this plane are sorted and the floor and ceiling heights are taken as horizontal boundaries. We then check whether points from the cluster under consideration are located in the cells (sectors) created in this way. This process is used to determine the effective length of the wall or to filter which sectors are parts of the wall and which sections are created by walls further away. At the end of this process, we know all the corner points of each wall and, thus, all the corner points of the room. Optionally, a wall can be improved here in that all points belonging to the wall surface under consideration are projected into this surface. This option can be selected in the settings of the program. The entire process is illustrated in Figure 3.12.

Create wall polygon

In order to create the basis for the wall modelling in the IFC file, a polygon must be placed around all exterior walls of the rooms in the last step of the wall extraction process. This step is not necessary if the exterior walls have been recorded as well, or if the location of the exterior walls can be determined, for example by an exterior wall extraction from a Geographical Information System (GIS) file (see Figure 4.6). For both of these options, it makes sense to include all interior rooms, otherwise zones where rooms are missing will be *filled in* by thick wall structures. In contrast, a facade that has not been recorded and is estimated by the polygon created in this step may be deformed by missing room recordings. A possible solution for all three cases would be to check the walls for a maximum wall thickness. This task could be taken up in a later step, if necessary.

The polygon itself is easy to create. Each room polygon is expanded by a certain factor (usually corresponding to the outer wall thickness), which leads to an overlap of the rooms. With a simple join, the outer wall polygon can now be created. The wall thickness can be chosen arbitrarily. Later, it is very easy to adjust the outer wall thickness in the CAD program. The polygon of the inner walls is obtained by shrinking the outer wall polygon by the wall thickness and subtracting the room areas from the resulting area. These steps are shown in Figure 3.13.

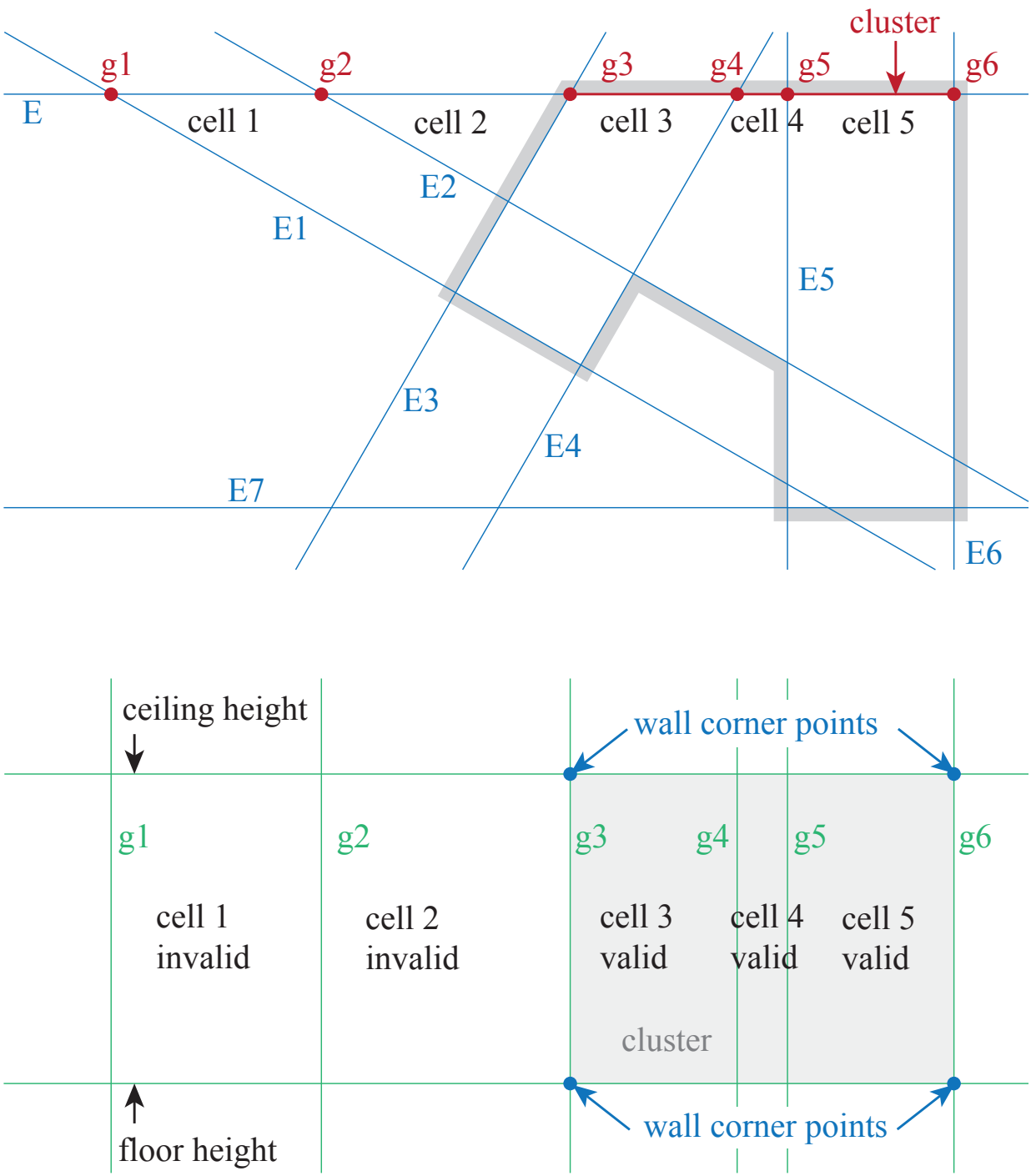


Figure 3.12: Above: floorplan with the planes (blue lines) and intersection lines (red dots); below: cell complex (green) with cluster (gray) and wall corner points (blue)

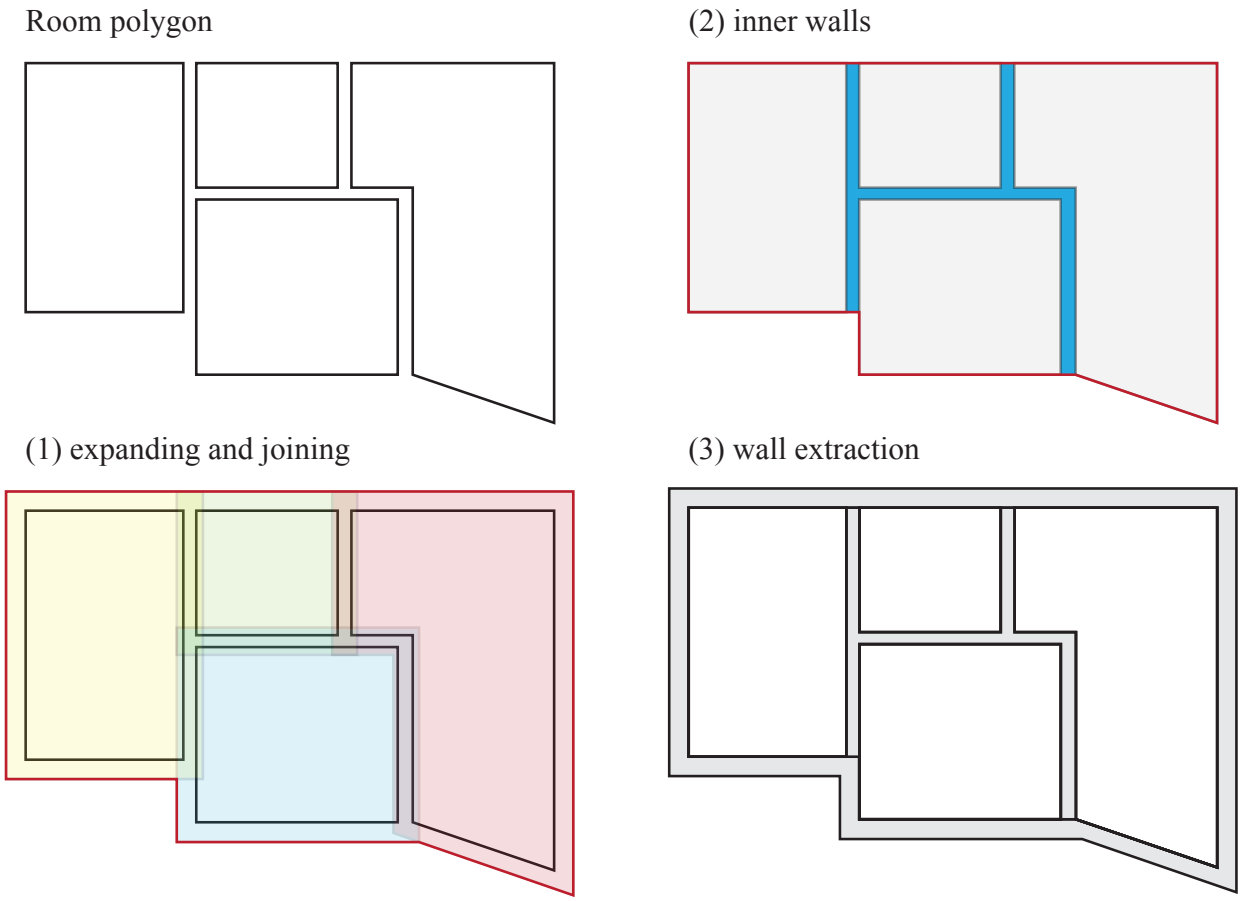


Figure 3.13: Steps of creating the wall polygon for the IFC model: top left: room polygons; bottom left: process of expanding and joining the room polygons; top right: result of the outer polygon and the inner walls; bottom right: result of the wall extraction

3.1.4 Extracting planar primitives (Part 2: openings)

In the meantime, the walls are ready to be modelled in the IFC file. In this subsection we will see how we can model the doors and windows. The process consists of four main steps, the first of them *Projection view* could in principle be postponed. It comes first as this gives an advantage in the implementation, however, logically speaking, the result of this step is significant at a later stage (for the graph-cut). The process pipeline can be examined in Figure 3.14.

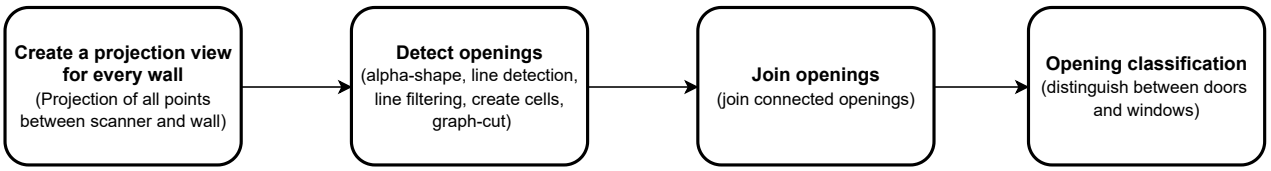


Figure 3.14: Pipeline of the planar primitives extraction (Part 2: doors and windows)

Projection view

In this step, the aim is to examine whether holes in a wall cluster actually represent an opening or whether they are merely the result of furniture or other clutter between the scanner and the wall. For this purpose, all points between the scanner and the wall are projected onto the wall. Problems in finding openings occur when furniture obscures the view of the window or when an object, for example a plant, stands directly in front of the window. It is best to make sure that these cases are avoided when recording the points. Curtains or blinds that do not allow a clear view through the window are unfavourable as well.

In a first attempt to detect openings, a point cloud was used from which the furniture in the interior had been removed. However, this procedure has the disadvantage that the projection view explained here cannot be created and it cannot be traced where the pointless areas in a wall come from. The procedure does not guarantee that the windows will be found correctly, nevertheless, it massively improves the result. The projection view is shown in Figure 3.15.

Detect openings

The concept in this section is based on the work of Michailidis et al. [MP17]. An α-shape is placed around each wall cluster that we extracted in Section 3.1.3. Then a RANSAC line model fitting algorithm is used to find all the lines in the α-shape. Next, all lines that are not approximately horizontal or vertical are deleted. Lines that deviate by a small angle are straightened to avoid slightly skewed shapes later. After this process, refinement is necessary since in real-world examples there are usually several lines that are very close to each other and can be merged without loss. Now, a cell complex is created, which is used to check whether there are a certain number of points per cell. The procedure is similar to the procedure in Section 3.1.3, where cells are used to check whether there are points of the wall cluster in them or not. The cell cluster (graph-cut) is created and the examination is then basically run twice. Once with the wall cluster without the *projection view*, i.e. with the

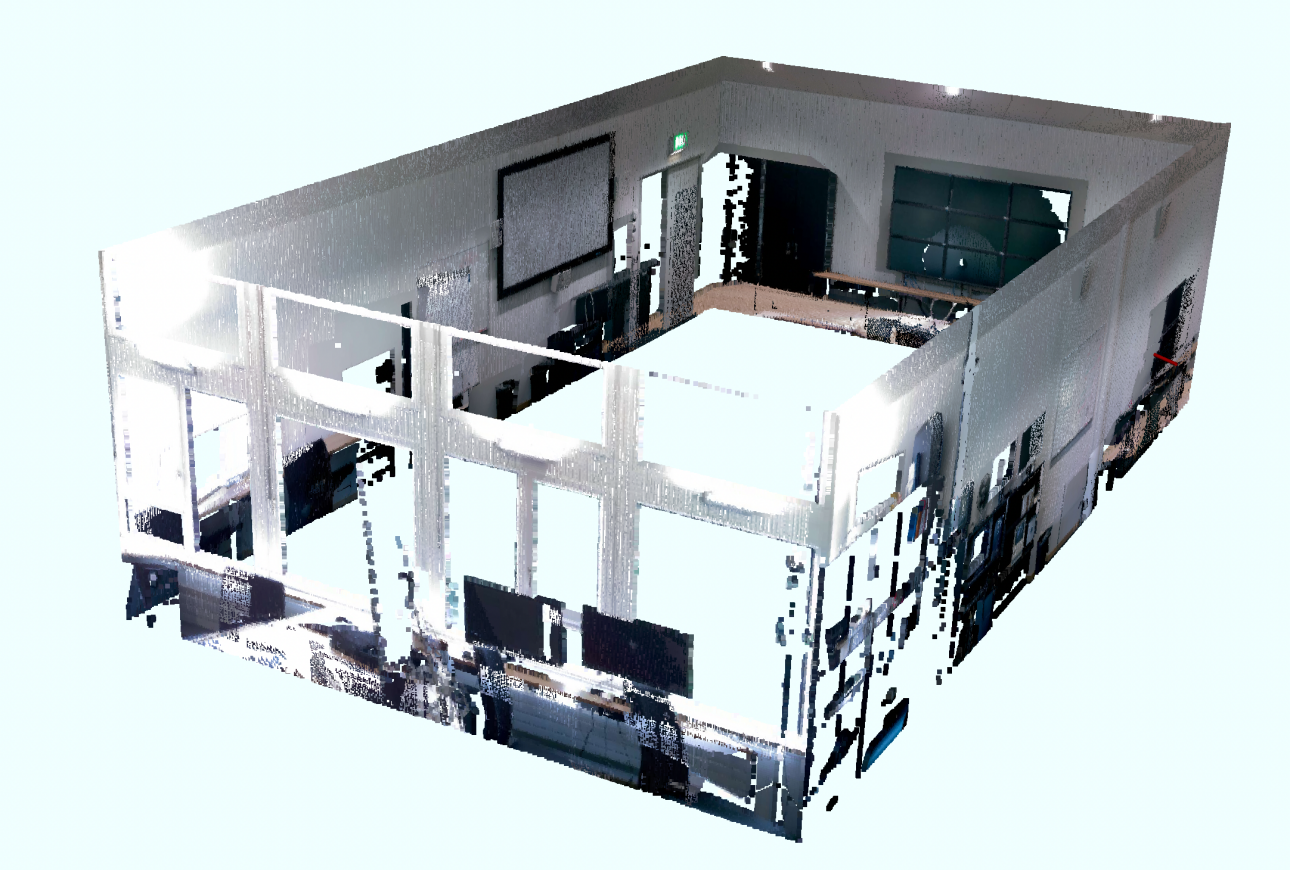


Figure 3.15: Projection view of the Point cloud: points between the scanner and the cluster are projected to the wall

view in which there are more pointless places (the same view that was used to calculate the α-shape). This first run is done to estimate whether a cell that has some points is more likely to be a window or not. As a rule, it can be assumed that if points are found in a cell during this run, then the cell is more likely not a window. The second run is done with the *projection view* (analogue Figure 3.1.4). Here it is checked again whether there are points in a cell. In contrast to the first run, as many points as possible must be present in a cell in order not to assign it to a wall. The entire assessment works for the reason that the cell complex was calculated based on the wall data (not on the *projection view* data).

The diagram by Michailidis et al. clearly illustrates the process in Figure 3.16, and Figure 3.17 shows the steps using the example point cloud.

Join openings

After the openings have been found, they most likely consist of several *fragmented* parts that have to be joined together. This circumstance arises in the case of very narrow areas, no or too few points may have been found in the associated segment. Furthermore, it is in fact possible that the lines originate from neighbouring openings. Hence, areas that are close enough to each other and have a high probability of belonging to the same opening

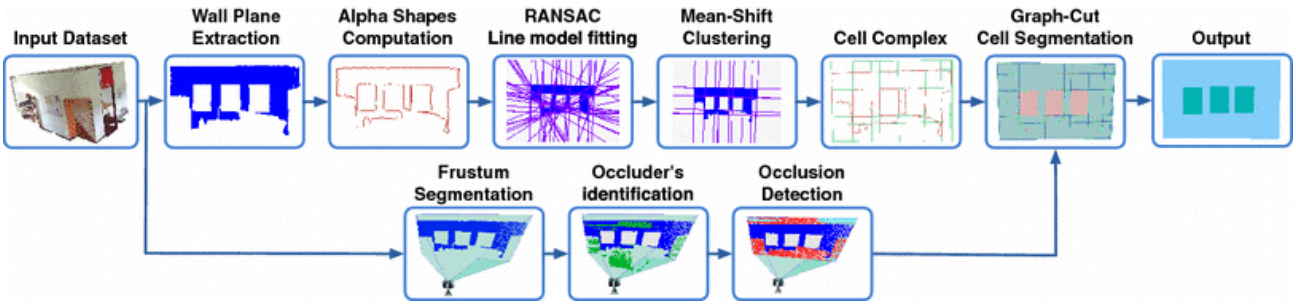


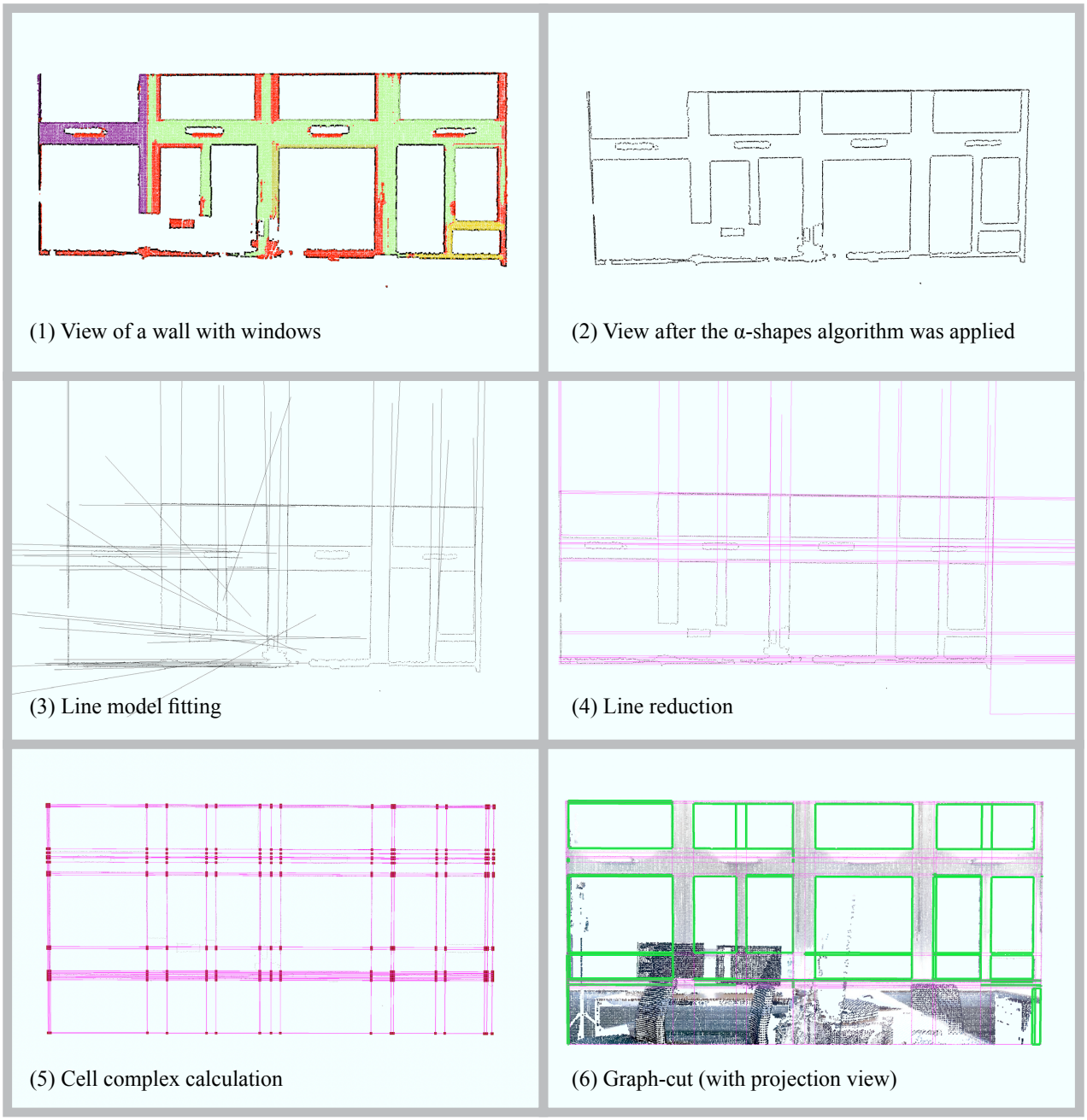
Figure 3.16: Block diagram of Michailidis et al.

are joined. The exact threshold value can be set manually and the set value can be changed respectively. This is useful, for example, if the object is a chalet whose windows have glazing bars. In this case, a value as small as possible must be selected. This is in contrast to a modern office building with large window fronts where a larger threshold value must be chosen.

For the calculation, the cells are classified. Each cell receives an evaluation whether it is closed, i.e. it represents a wall piece, whether it is open, i.e. it is an opening, and whether the cell is narrow or wide. The classification of the cells is stored in a matrix, where the different numerical values represent one of the previously mentioned states. The matrix is then first scanned column by column and in the second pass row by row to identify which cells need to be connected to form a single opening. Each entry used, is in addition, assigned a numerical code. A sample matrix with the corresponding cell illustration is shown in Figure 3.18, and Figure 3.19 shows the joined cells.

Opening classification

Meanwhile, we have found all the openings and this last step is to categorise them, i.e. to classify them into doors and windows. Here, we simply check whether the opening reaches the floor and from there reaches a certain height (in our case a default minimum height of 1.89 m was chosen). The width is also checked and both the minimum permitted height and the width that a door may have can be adjusted. For the width, the default value was set to a range of 0.7 - 0.9 m. If an opening is not recognised as a door, it is automatically assigned to the window category.



(1) View of a wall with windows

(2) View after the α-shapes algorithm was applied

(3) Line model fitting

(4) Line reduction

(5) Cell complex calculation

(6) Graph-cut (with projection view)

Figure 3.17: Steps of the opening detection algorithm

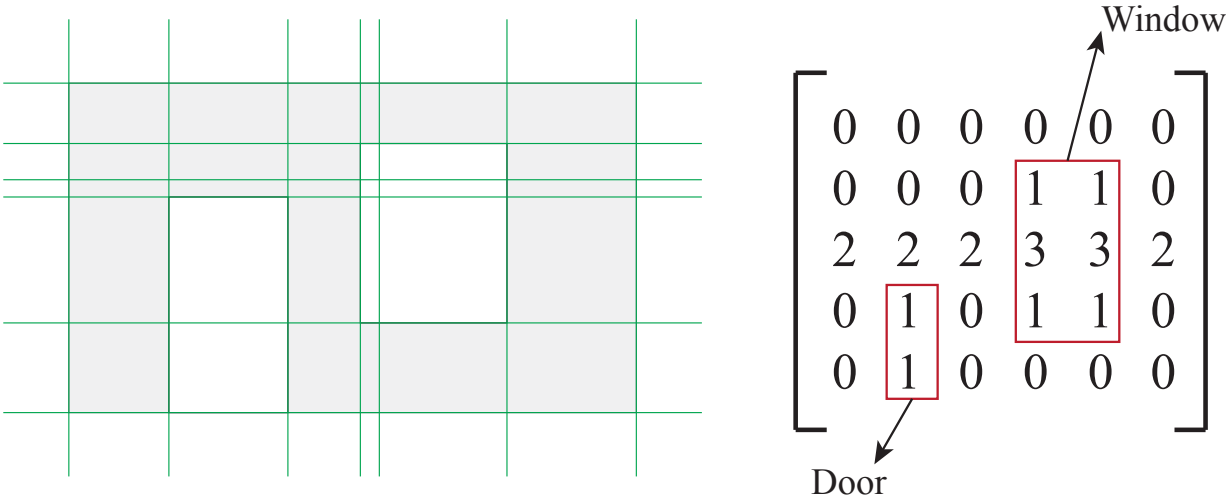


Figure 3.18: Cell complex and the corresponding matrix for the openings detection

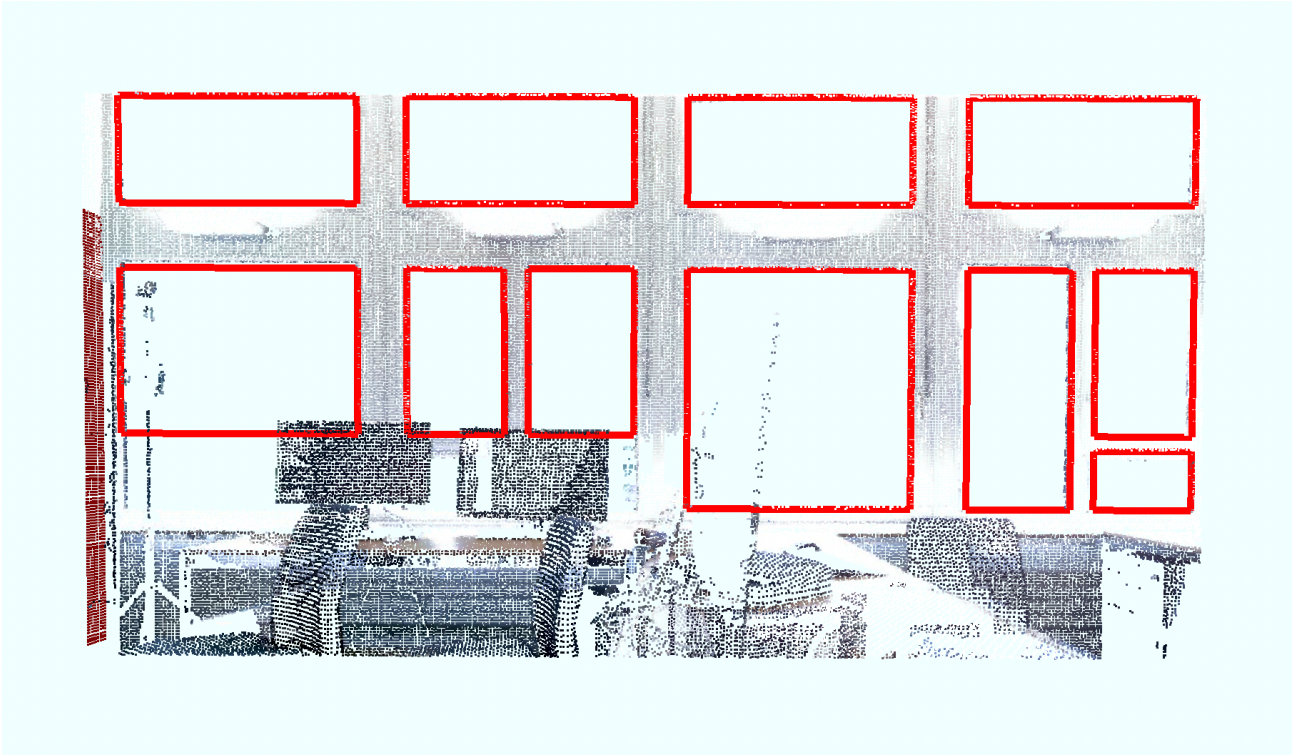


Figure 3.19: Result of the *join openings* algorithm

3.1.5 Output generation

The whole pipeline works towards recreating, modelling and exporting all architecture-related elements into a BIM format at the end. The IFC file that is formed in this process is composed of all extracted elements. Step by step, the objects are added to the file. The implemented code was taken directly from the IFC++ open source library and only slightly simplified. The implementation was adapted in order that the structures not used for our pipeline were omitted and the passing values were adjusted to our implementation. More information about the IFC++ library can be found directly on the IFC++ project page [Ger] and the structure of the IFC file can be seen in Listing 3.6 and the folder structure is presented in Figure 3.20.

Listing 3.6: Simple IFC file

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION(('ViewDefinition [CoordinationView]'),'2;1');
FILE_NAME('../../../Output/SimpleWall.ifc','2022-07-19T21:54:28',(''),(''),(''),'
    IfcPlusPlus','');
FILE_SCHEMA(('IFC4'));
ENDSEC;
DATA;
#1= IFCPROJECT('3EBsybjiv05Bc1N3p0vnFW',#26,$,$,$,$,$,(#29),#28);
#2= IFCSITE('0RebVSJjP8PPAZpysVlLk6',#26,$,$,$,$,$,$,$,$,$,$,$,$);
#3= IFCBUILDING('2Ij3PQhqX6vRWjf037K0bE',#26,$,$,$,#17,$,$,$,$,$,$);
#4= IFCBUILDINGSTOREY('001C0vOPnEuu_Bw1MuI1Bx',#26,'Storey 1',$,$,$,$,$,$,$);
#5= IFCRELAGGREGATES('3hbGGiyVv9w9dIgq8e54QE',#26,$,$,#1,(#2));
#6= IFCRELAGGREGATES('23h7q7lsf5S8dixw1eUIAf',#26,$,$,#2,(#3));
#7= IFCRELAGGREGATES('22CVhWvpL5U93b461XtT9T',#26,$,$,#3,(#4));
#8= IFCAXIS2PLACEMENT3D(#9,#10,#11);
#9= IFCCARTESIANPOINT((0.,0.,0.));
#10= IFCDIRECTION((0.,0.,1.));
#11= IFCDIRECTION((1.,0.,0.));
#12= IFCLOCALPLACEMENT($,#8);
#13= IFCAXIS2PLACEMENT3D(#14,#15,#16);
#14= IFCCARTESIANPOINT((0.,0.,0.));
#15= IFCDIRECTION((0.,0.,1.));
#16= IFCDIRECTION((1.,0.,0.));
#17= IFCLOCALPLACEMENT(#12,#13);
#18= IFCAXIS2PLACEMENT3D(#19,#20,#21);
#19= IFCCARTESIANPOINT((0.,0.,0.));
#20= IFCDIRECTION((0.,0.,1.));
#21= IFCDIRECTION((1.,0.,0.));
#22= IFCLOCALPLACEMENT(#17,#18);
#23= IFCPERSON('MyID','MyFamilyName','MyGivenName',$,$,$,$,$);
#24= IFCORGANIZATION('MyOrganization','My organizations name','My organizations
    description',$,$);
#25= IFCPERSONANDORGANIZATION(#23,#24,$);
#26= IFCOWNERHISTORY(#25,$,$,$,$,$,$,$);
```

```
#27= IFCSIUNIT(*,.LENGTHUNIT.,$,.METRE.);
#28= IFCUNITASSIGNMENT((#27));
#29= IFCGEOMETRICREPRESENTATIONCONTEXT($,'Model',3,0.00001,#8,$);
#30= IFCPROPERTYSET('0TmL_2rKH1tuPmbnpvRurd',#26,'Pset_Walls','Pset to defineWalls'
    ,(#31,#32));
#31= IFCPROPERTYSINGLEVALUE('Identifier_Walls',$,IFCIDENTIFIER('Id_Walls'),$);
#32= IFCPROPERTYSINGLEVALUE('Footprint area Walls',$,IFCREAL(0.8),$);
#33= IFCRELDEFINESBYPROPERTIES('3i3042bfv0Vg7UjscD7rcc',#26,$,$,(#62),#30);
#34= IFCSHAPEREPRESENTATION(#29,'Body','SweptSolid',(#36));
#35= IFCPRODUCTDEFINITIONSHAPE($,$,(#34));
#36= IFCEXTRUDEDAREASOLID(#42,#37,#41,2.499997995793819);
#37= IFCAXIS2PLACEMENT3D(#40,#38,#39);
#38= IFCDIRECTION((0.,0.,1.));
#39= IFCDIRECTION((1.,0.,0.));
#40= IFCCARTESIANPOINT((0.,0.,0.100000001490116));
#41= IFCDIRECTION((0.,0.,1.));
#42= IFCARBITRARYPROFILEDEFWITHVOIDS($,$,#43,(#57));
        // missing lines: some IFCPOLYLINE(...) followed by IFCCARTESIANPOINT(...);
#57= IFCPOLYLINE((#58,#59,#60,#61));
#58= IFCCARTESIANPOINT((3.899999380111694,3.749999523162842,0.100000001490116));
#59= IFCCARTESIANPOINT((3.899999380111694,6.,0.100000001490116));
#60= IFCCARTESIANPOINT((6.749999046325684,5.999999523162842,0.100000001490116));
#61= IFCCARTESIANPOINT((6.749999046325684,3.749999523162842,0.100000001490116));
#62= IFCWALL('3sOjNz1r9BUudlMJvTb4tP',#26,'Walls','Walls',$,$,#35,$,$);
#63= IFCRELCONTAINEDINSPATIALSTRUCTURE('2u7Heas1D9kQyTXOQ9FDRk',#26,$,$,(#62),#4);
#64= IFCOPENINGELEMENT('1Tc33Sj0LFNB3Rng0Ul6DS',#26,'Door-000',$,$,$,#66,$,$);
#65= IFCSHAPEREPRESENTATION(#29,'Opening','BRep',(#67));
#66= IFCPRODUCTDEFINITIONSHAPE($,$,(#65));
#67= IFCFACETEDBREP(#68);
        // missing lines: some more IFCPOLYLINE(...); IFCCARTESIANPOINT(...);
#77= IFCFACE((#78));
#78= IFCFACEOUTERBOUND(#79,.T.);
#79= IFCPOLYLOOP((#69,#72,#71,#70));
        // missing lines: some IFCFACE(...); IFCFACEOUTERBOUND(...); IFCPOLYLOOP(...);
#95= IFCRELVOIDSELEMENT('0NHpxWw7X3XA4ZVWPgxTFc',#26,$,$,#62,#64);
ENDSEC;
END-ISO-10303-21;
```

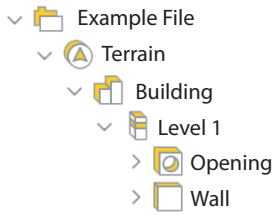


Figure 3.20: Folder structure of an IFC file with openings and walls

3.2 Software structure

In the last section we saw the entire pipeline and in this passage, we will have a look at the software structure to see how the complete process has been implemented. Finally, the chapter concludes with a look at the user interface in Section 3.3. We will look at the most important classes and functions and for illustration purposes, in Figure 3.21, the corresponding class diagram is shown and in the Figures 3.22 - 3.24 the different sequence diagrams are presented.

As mentioned earlier the program was implemented using CLion from JetBrain on an Apple MacBook Pro. For the Graphical User Interface (GUI) the QtCreator from The QT Company Ltd. was utilised. In addition to the C++ standard library, the following libraries were used:

PCL	Point cloud handling
IFC++	IFC file generation
QT5	General user interface
VTK	General user interface
Boost	Room polygon
Eigen	Polygon and vector handling

3.2.1 Class diagram

In the following class diagram the implemented classes and their relationship are depicted (Figure 3.21). Subsequently, the responsibilities and functions of the classes are explained in detail. The files *main.cpp* and *global.h* are not shown in the diagram, nevertheless, they should be mentioned briefly: The file *main.cpp* includes the main function that starts the programme and the file *global.h* contains the various typedefs and structs that are used in several classes.

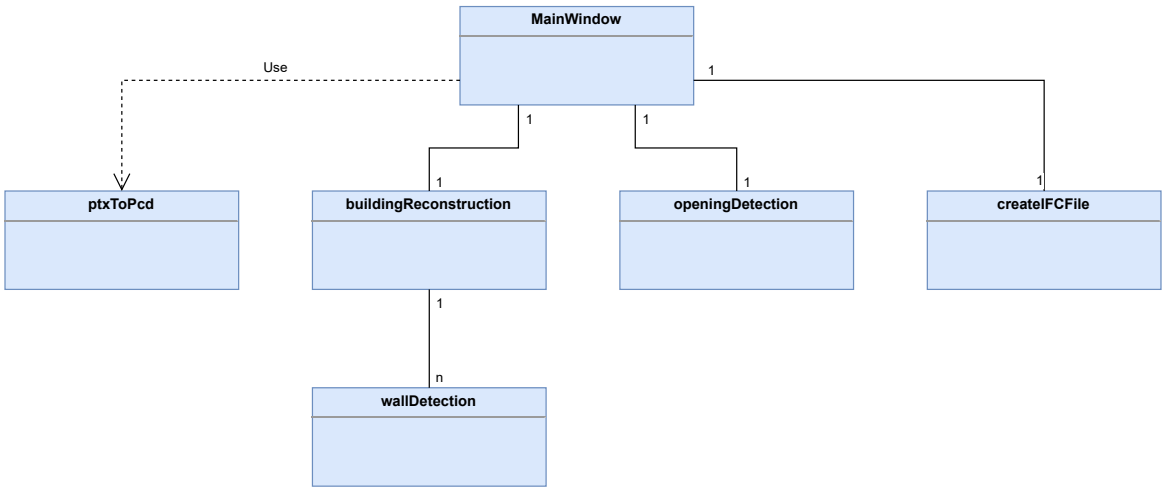


Figure 3.21: Class diagram of the implemented software (PointCloudToCAD)

mainWindow.cpp/.h/.ui This class contains all functions that are triggered in the GUI, as well as the functions for visualising the point clouds and the results of the pipeline.

buildingReconstruction.cpp/.h Here, the different rooms that are loaded and evaluated are controlled. The class contains the functions that are needed to calculate the wall polygons.

wallDetection.cpp/.h Objects of this class represent a single room. All the necessary calculation steps for the reconstruction of the walls are carried out and saved within it. Most of the intermediate steps of the *Pre-processing* and *Wall-processing* pipeline are executed in this class.

openingDetection.cpp/.h The responsibility of this class is to identify all openings. All data that will be needed to build the IFC file are stored here. The class corresponds to the *Opening processing* of the pipeline.

createIFCFile.cpp/.h The IFC file is composed in this class. The data gathered from the previously described classes *buildingReconstruction* and *openingDetection* are combined, merged, evaluated and used to model the architectural elements.

ptxToPcd.cpp/.h This class transforms a PTX point cloud data (Leica format) to a *colored* PCD file that can be used with the PCL library. The generated file is saved as .pcd on the disk.

3.2.2 Sequence diagram

In order to review the implemented processes in more detail, simplified sequence diagrams are shown below. Not all sequences need to be discussed, however, the three main buttons *Add Room*, *Process Room* and *Create IFC*, which represent the entire process pipeline and can later be seen in Section 3.2, are worth mentioning.

Add room

The first sequence diagram (Figure 3.22) shows the events after pushing the *Add Room* button. After the process is triggered by pressing this button, a QT file dialogue opens and the desired file can be loaded. If a PTX file is selected, it is converted to a PCD file and saved as a new file with the same name and the new suffix. The program automatically continues working with the new file. If a PCD file is selected, this first process step is not necessary, in any case, the point cloud is loaded with *addCloud* and made available for the further steps. In the process pipeline, this corresponds to the first two main steps (read and convert).

Process room

The execution of the second sequence diagram is triggered by activating the *Process Room* button. This initiates the pipeline steps *pre-processing* and *extract planar primitives 1 and 2*. With *addNewRoom* a new room is created and the point cloud is reduced in size and stored. As the name suggests, all the calculation steps needed to reconstruct the walls are carried out in *processWallReconstruction*. Afterwards, the currently processed room is added to the list of all rooms that have been appended so far (*addRoomToFloor*). In order to provide the

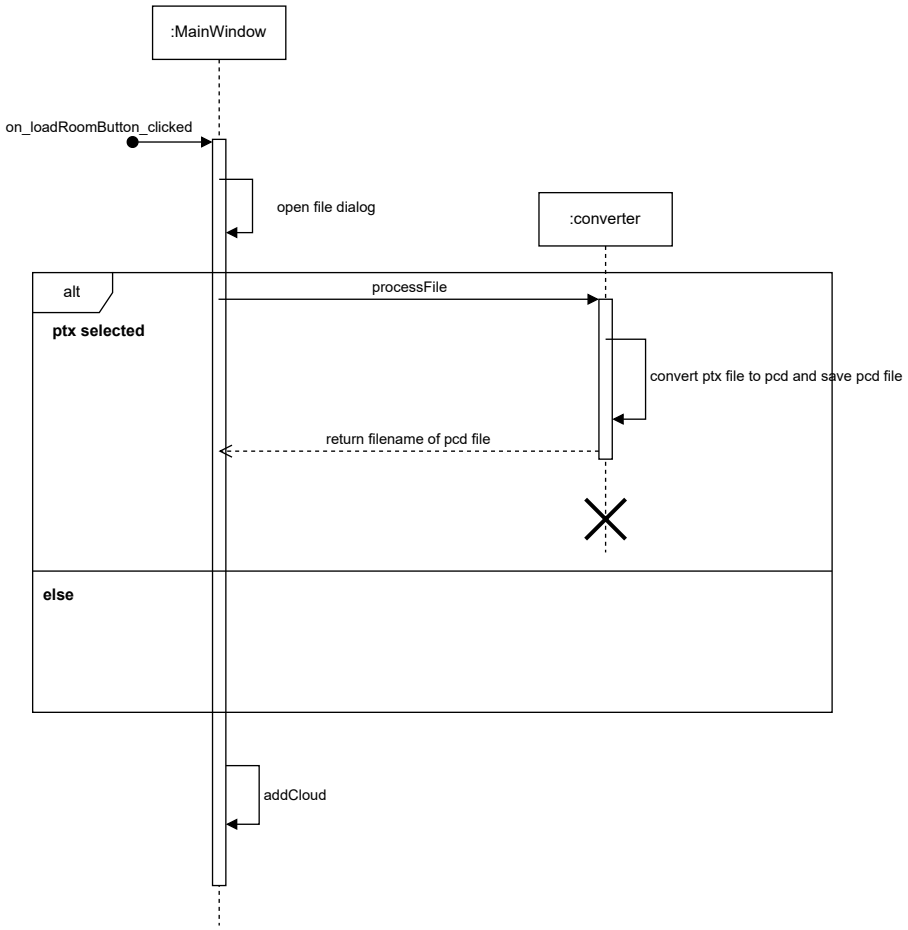


Figure 3.22: Sequence diagram: On click *Add Room* button

necessary information for the IFC file, a room polygon is created in this step. This would not need to be done in every iteration, as already discussed in the pipeline, but for the sake of simplicity this step was implemented this way. For the later detection of the openings, all found walls of the current room under consideration, are loaded and provided with *getWall*. To determine the wall openings, the required data is now loaded. This step is performed individually for each wall. Having almost reached the end of the sequence diagram, all steps for determining the windows and doors are now carried out (*processOpeningReconstruction*). Again, the process is executed separately for each wall. Finally, all views are updated and can be viewed via the various tabs.

Create IFC file

The final sequence diagram shows the process *Create IFC*. After clicking on the corresponding button, the process that begins with *createModel* is set in motion. In this first act the IFC model is set up and the initial values are generated. The process continues with *addWallExtruded*, which adds the walls to the model. The same is done with all openings, i.e. with all windows and doors by *addAllOpenings*. Finally the model is written to a IFC file and with *writeIFCFile* the file is saved.

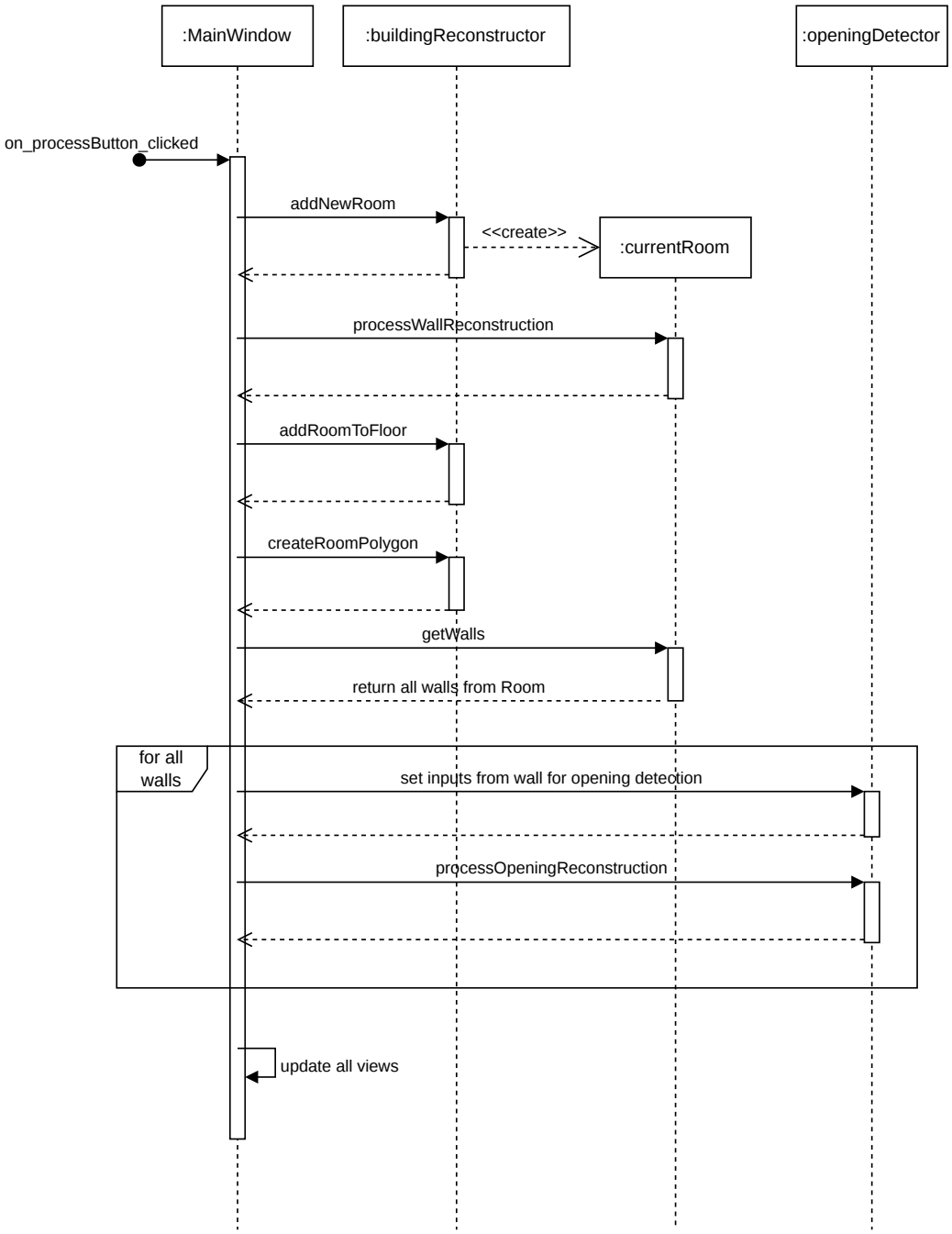


Figure 3.23: Sequence diagram: On click *Process Room* button

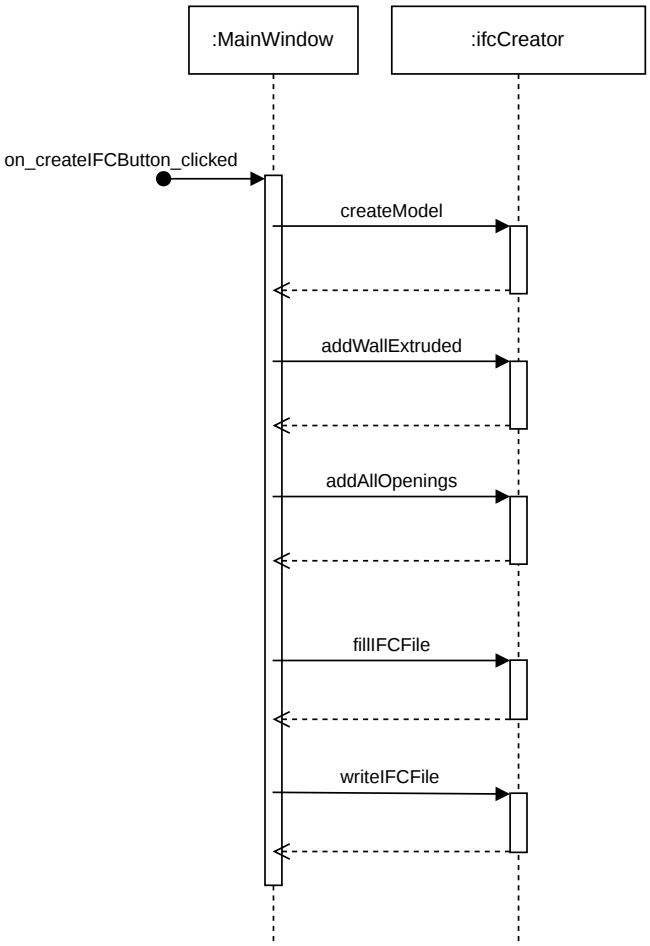


Figure 3.24: Sequence diagram: On click *Create IFC* button

3.3 Program user interface

The user interface of the PointcloudToCADFloorplan converter is composed of a main menu bar, a sidebar and a main window for displaying the different views. As you can see in Figure 3.25, the main menu is divided into the following tabs and subcategories:

File Open (PTX; PCD)

Pre-Processing

- Down Sampling
- Generate Clusters
- Remove non-Verticals
- Remove non-Walls
- Determine Walls

Wall Processing

- Create Planes
- Wall Corner Points
- Wall Polygon

Opening Processing

- Wall Projected Cloud
- Detect Openings (Alpha-Shapes; Line Model Fitting; Line Reduction; Cell Complex; Graph-Cut)
- Join Openings
- Opening Classification

IFC Processing

- Create Model
- Add Walls
- Add Openings
- Create IFC File

Show Results

- Down Sampling
- Generate Clusters
- Remove non-Verticals
- Remove non-Walls
- Determine Walls
- **********
- Create Planes
- Wall Corner Points
- **********
- Wall Projected Cloud
- Detect Openings (Alpha-Shapes; Line Model Fitting; Line Reduction; Cell Complex; Graph-Cut)
- Join Openings

The point clouds can be read under *File*. The first three *Processing* tabs (*Pre-, Wall, Opening*) are used to calculate all intermediate steps that have a changed output. They can be displayed later under *Show Results*. The *IFC Processing* tab is special in that the individual steps cannot be executed independently of each other. This is possible with the other menu items. All *Processing* menu items are for debugging or experiment purposes rather than for basic use of the program.

The sidebar is used to read in and process the point cloud scans. Four buttons are available for this purpose, which control the entire process and, in contrast to the points in the main menu, calculate all the steps at once. With the *Add Room* button the scans are added, with the *Additional Room Data* button a second point cloud of the same room can be loaded and with the *Process Room* button the calculations are carried out. Finally, the export file can be created by clicking on the *Create IFC* button and the created file can be read in a CAD program. As shown in Figure 3.26, there are some information fields that indicate, for example, which room number and which wall is currently being visualised. In case a project is complete and a new one is to be started, the reset button is available.

In the main window, it is possible to switch back and forth between five tabs. The *Current Room Point Cloud* view shows the active point cloud. The *Process* view shows the current state of the calculation. The *Building Point Cloud* window shows all the rooms that have already been loaded and the *Floorplan Preview* window shows the basis that will later be used to model the IFC file. In the last field, the *Settings* area, all modifiable parameters can be found. These can be changed as desired and the *Update Parameters* button can be used to apply the selected dimensions and thresholds to the current project. It is important that a file is initially loaded for changing the parameters. Then the parameters can be set and the processing can be started. In Figure 3.27, the changeable values are shown.

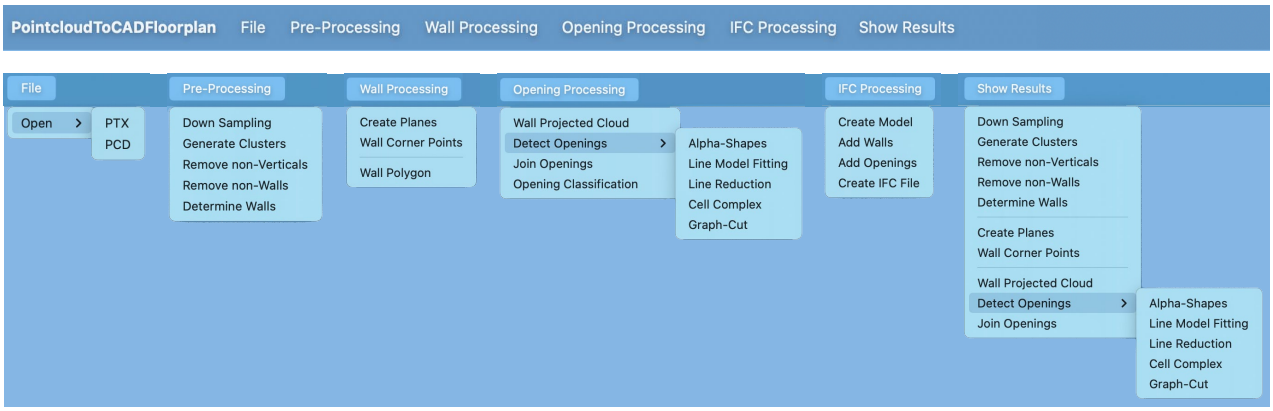


Figure 3.25: Overview of the user interface menu

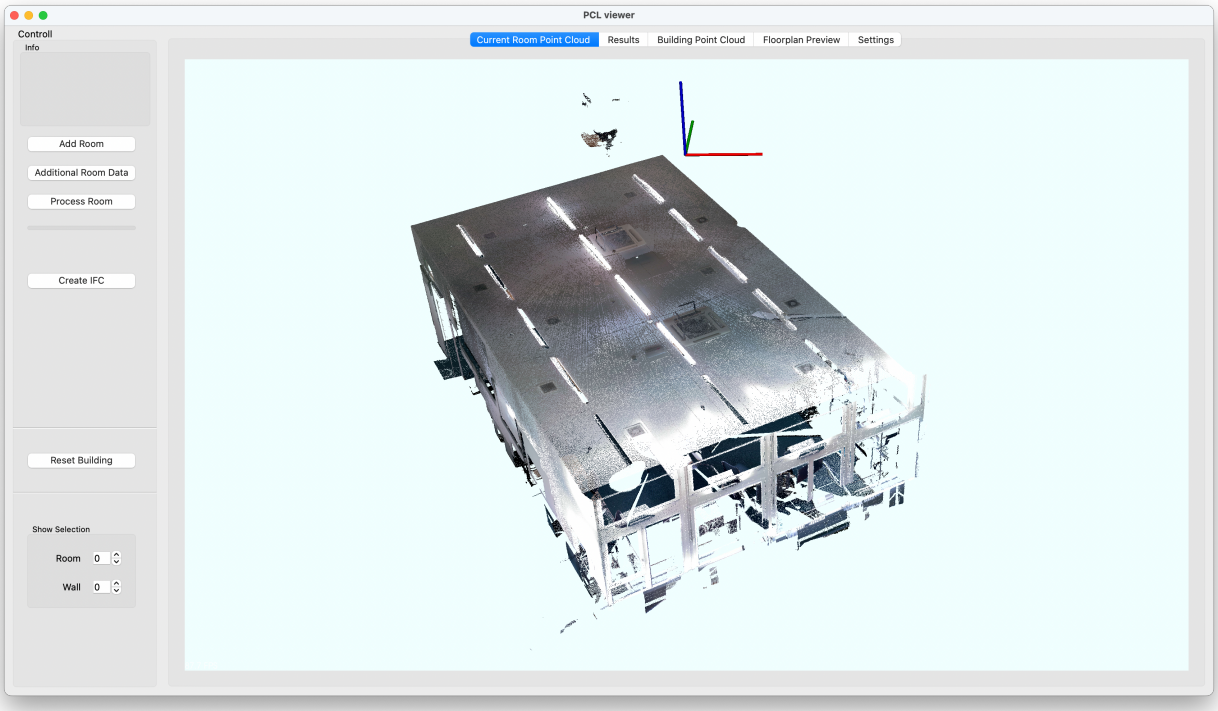


Figure 3.26: The general user interface

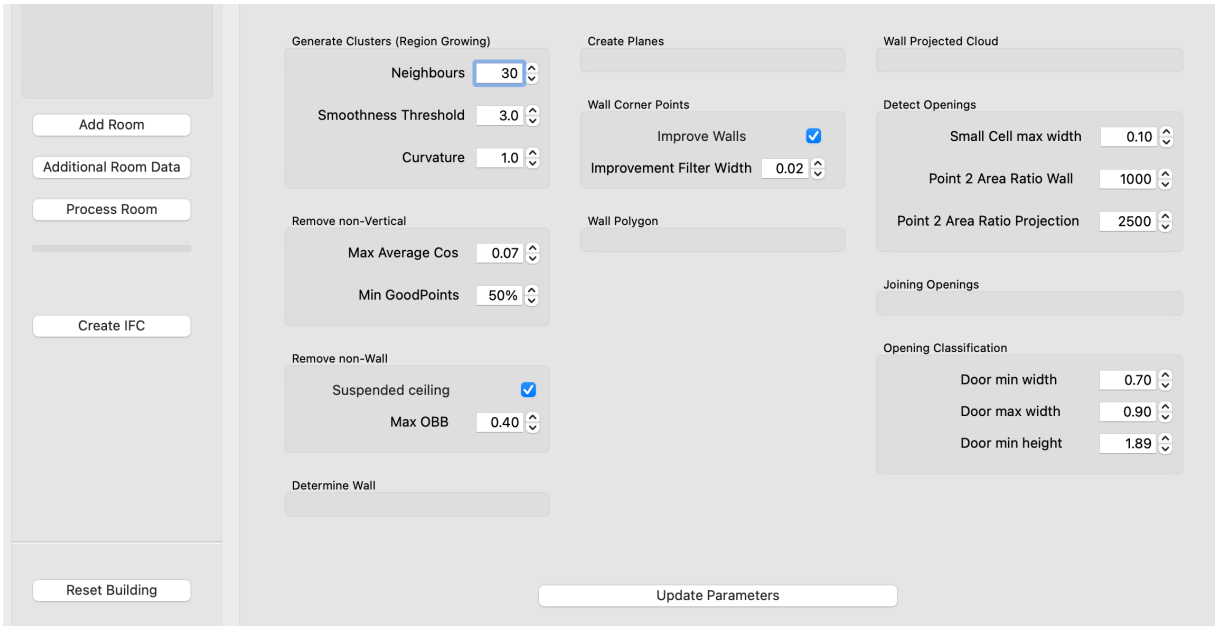


Figure 3.27: The parameters that can be modified

4 Evaluation

In the paper by Pexman [Pex21] an evaluation schema is presented that shows how well the automatically generated architectural models match the registered point cloud. With suitable metrics, the quality of the reconstructions can be well determined and evaluated. For example, it can be measured how far away wall points from the point cloud are compared to the modelled wall or how fast the chosen algorithms process the data. Unfortunately, such evaluation methods are beyond the scope of this paper, which is why a purely visual comparison was made between the recorded point cloud and the final IFC file. The IFC data was superimposed on the point cloud in ArchiCAD [IDC] for direct comparison. The result of this overlap can be seen in Figure 4.1.

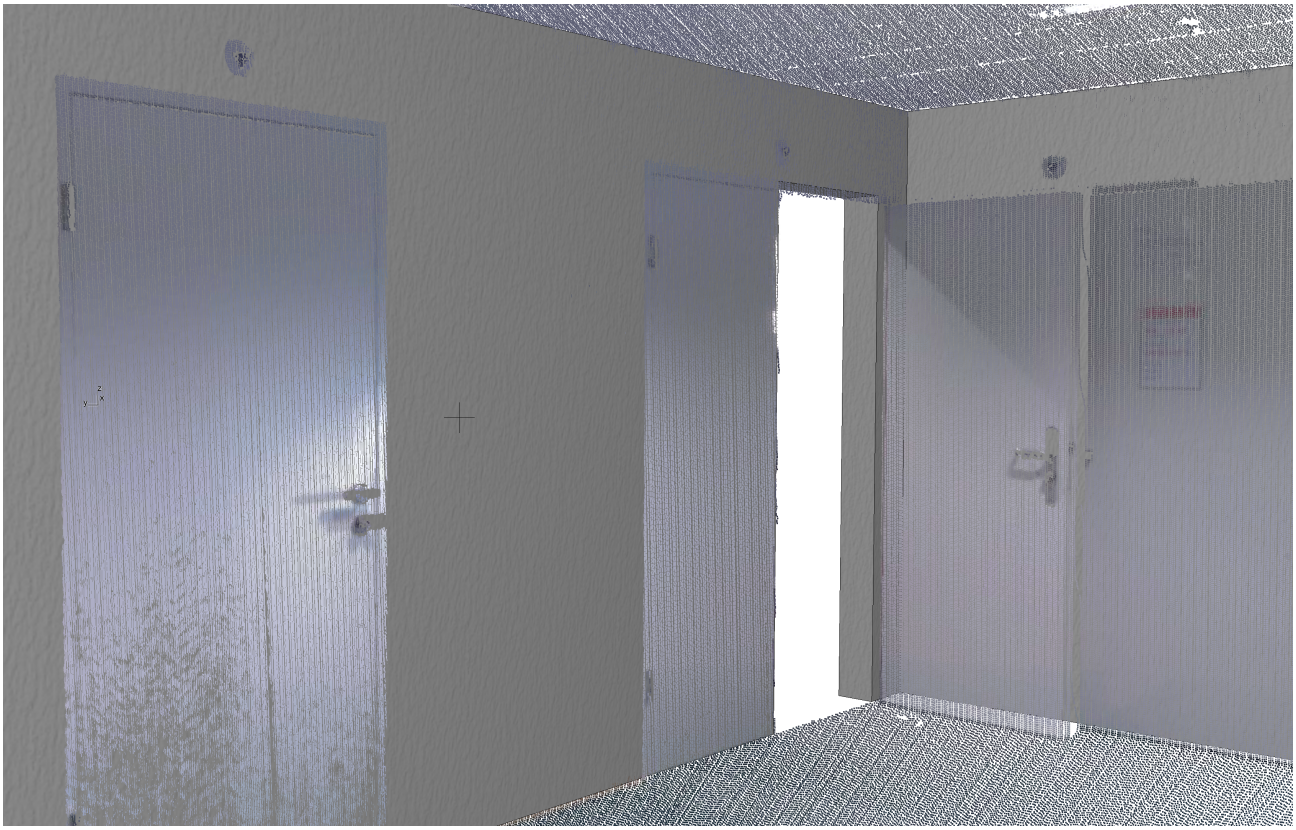


Figure 4.1: Overlap image of the point cloud data and the final IFC model

4.1 Results

The Swiss Society of Architects and Engineers, known as Schweizerischer Ingenieur- und Architektenverein (SIA), regulates a wide range of construction issues. The SIA standard 414/1 deals with dimensional tolerances in the building industry, whereby not every component or every trade has to meet the same requirements [SIA]. This has an influence on the generated results insofar as there is not necessarily a claim to maximum accuracy. For example, it can be completely sufficient if walls are extracted to +/- 2 cm accuracy. For planning, a certain degree of abstraction is advantageous anyway, for the reason that it allows any (slight) curvatures of a wall or similar structure to be smoothed out. In the development of buildings, such small deviations are not important. On the other hand, it may be desirable to know the exact dimensions of windows so that they can be reordered for renovation purposes, for example. A clear boundary for the definition of metrics is therefore not entirely trivial.

As far as the comparison of the point clouds used in this work with the generated IFC data is concerned, a positive conclusion can be drawn. All walls could be successfully extracted and modelled. Even more challenging elements such as windows and doors were successfully recognised and modelled. Viewed in a larger context, it can be emphasised that it is certainly not a matter to be taken for granted that the right objects are recognised in the right place. For instance, we find in Jung et al. [Jun+18] that, in contrast to our results, doors were found where there were none or windows were not recognised where there should be some. However, it should be mentioned that only two data sets were used for the evaluation (a synthetic one and the one captured by the laser scanner), which is not very representative. In addition, as precision is concerned, some details could certainly be refined through precisely defined quality requirements and the resulting improvement approaches. In general, however, the quality gain compared to other approaches should be highlighted.

To get an idea of the results, the three figures below illustrate the process with the generated real-world data. Figure 4.2 shows the imported point cloud, figure 4.3 shows the polygon lines needed to construct the IFC file and figure 4.4 shows the modelled IFC File.

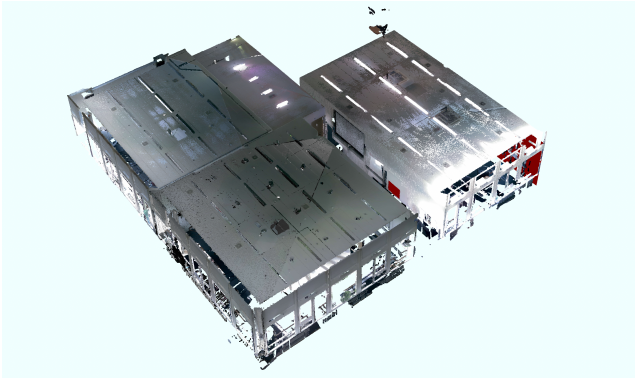


Figure 4.2: The point cloud after adding and processing the rooms of the IFI building at the University of Zurich

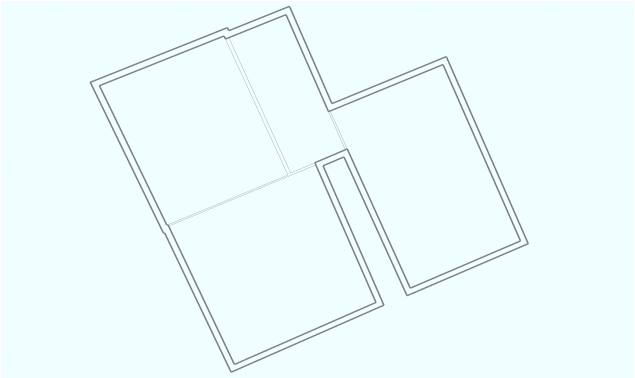


Figure 4.3: Polygon lines for constructing the IFC file

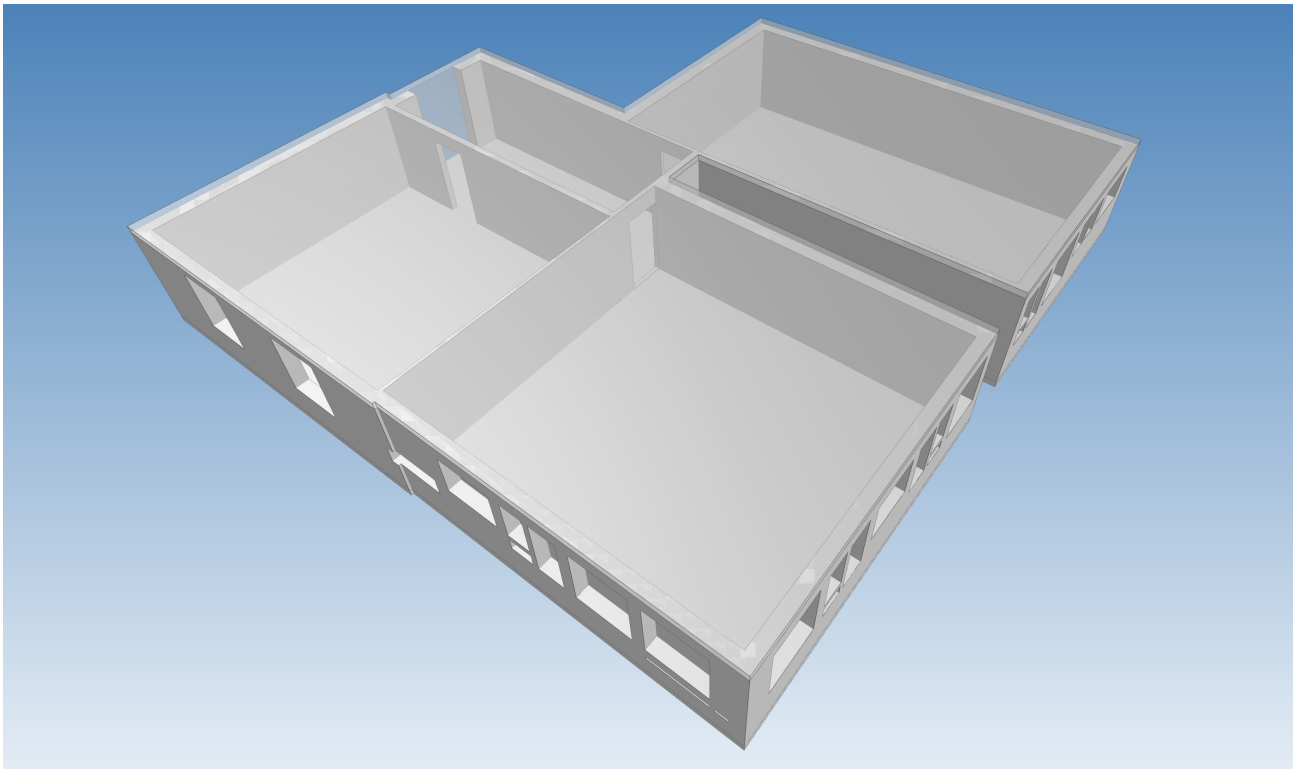


Figure 4.4: IFC model of the scanned rooms at the IFI building

4.2 Limitations

Finally, a number of potential limitations and shortfalls need to be considered. First, the proposed pipeline is limited mainly by the pre-processing of the point cloud. As we have seen in Chapter 3, all non-vertical point groupings are removed. This automatically has the consequence that, for example, roof slopes or ramps cannot be reproduced with this solution approach. Pillars and round wall segments are as well still a limit to the possibilities at this point.

Second, the extraction of stairs are in addition, a shortcoming and a disadvantage of this study. They extend over several floors, which is why they have not yet been recorded or included in the data generation. In principle, extraction would be possible with the solution presented, as the stairs always have a horizontal and a vertical component. In digital architectural plans, stairs correspond to an object group, which is why they should certainly be considered in later studies.

A further shortfall is the differentiation between wall segments and built-in cupboards or kitchen units. Here, it is currently not possible to distinguish between certain built-in objects and wall segments and we are not able to determine whether they are (nearly) room-high built-in objects or corners made of solid bricks. The differences in representation can be found in Figure 4.5. Such differentiation are essential for planners, which

is why it would be important to develop a procedure for differentiation.

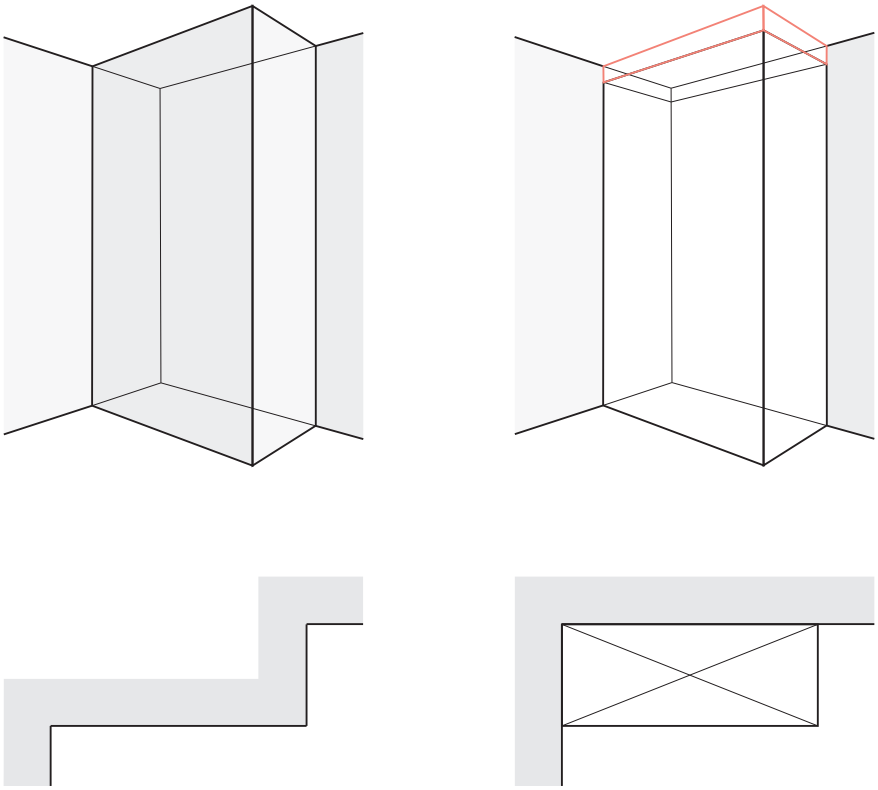


Figure 4.5: A solid wall (left) and the corner case of a cupboard (right)

The fact that the external views are not yet fully taken into account is another limitation. Thus, balconies are not taken into account; nor do we have a binding wall thickness of the exterior walls. These are still estimated or measured manually at the moment. In principle, however, it is no problem to in addition take records of the facade and to adapt the pipeline so that these walls are calculated automatically as well and the disadvantage can be eliminated. By taking an outdoor image, the calculation method for the windows could be adjusted somewhat, since in this case most of the disturbances found in indoor environments could be circumvented. If no exterior recordings exist, the simultaneous absence of interior areas can lead to deformed exterior walls. If an outside scan exists while there is a lack of an inside capture, this leads to thick wall formations. In the Paragraph *Create wall polygon* in Section 3.1.3 an approach to solve this problem is sketched.

In the GIS, building exterior lines can be looked up [GIS]. These plans can be quickly and easily integrated into a CAD program and by overlaying the IFC data, the wall thickness can be adjusted. An example of such a GIS file can be seen in Figure 4.6. Since the GIS plans show the bird's eye perspective and therefore mainly the roof lines are shown, this solution is not always applicable, especially when the floors vary in shape.

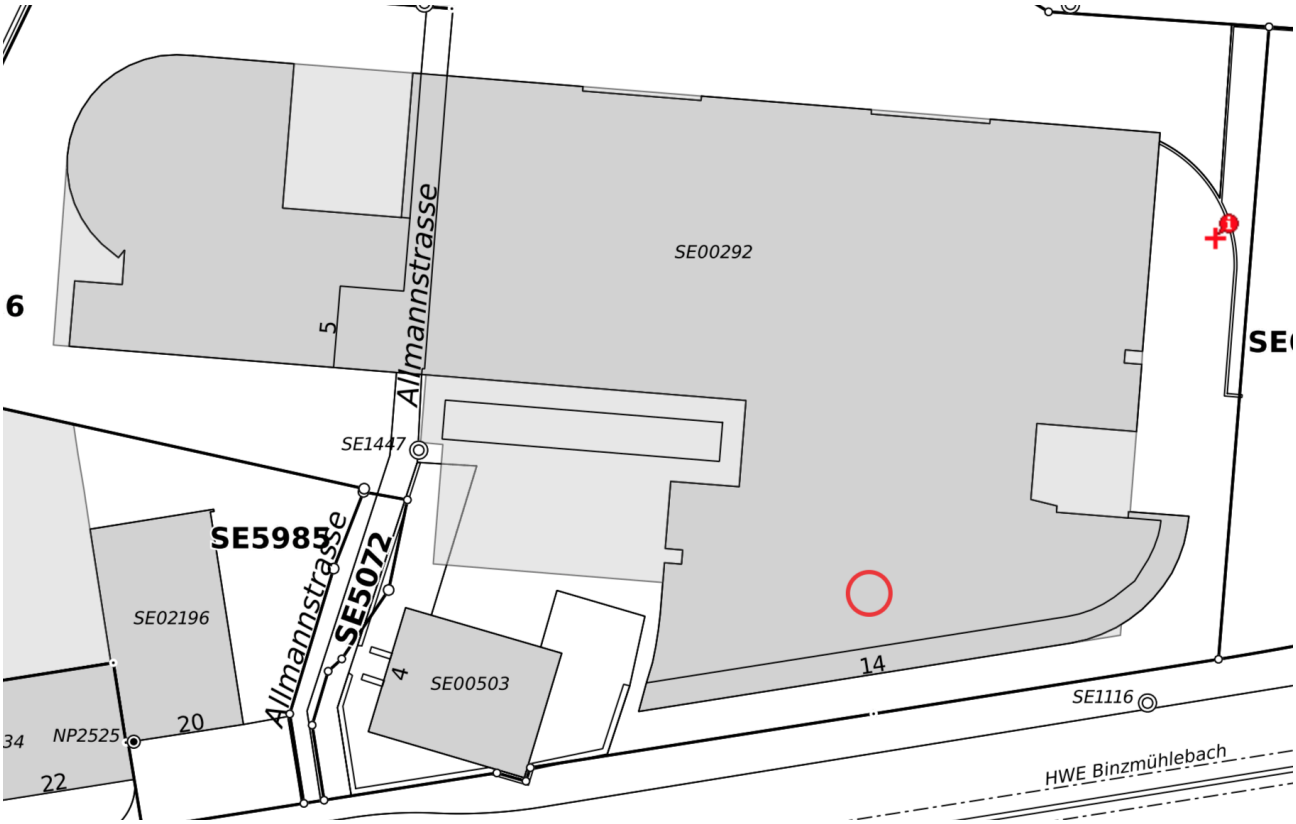


Figure 4.6: GIS file of the IFI building at the University of Zurich

5 Conclusion and Discussion

In conclusion, a tool that generates finished CAD plans from point clouds is very valuable. In this paper, the focus is clearly on the usability for architects, however, this should not be a restriction for other possible areas of use. In contrast to other applications, a certain degree of abstraction is important in construction planning so that the building plans are comprehensible and can be used purposefully by the corresponding user group. A bricklayer, for example, needs different information than a carpenter, and for both, reconstructions such as those needed for virtual reality applications would be of little help. Ludwig Mies van der Rohe, one of the most important architects of modernism, was a master of the principle ”less is more”. This principle is in no way intended to limit the possibility of filtering a multitude of elements and objects. Rather, it is intended to show why so much emphasis was placed on exporting in the IFC format in this study and why mesh reconstruction or similarly detailed methods were not used. The decision to prioritise usability for architects over the possibility of photo realistic extraction was made intentionally.

5.1 Future work

In the view of the fact that the generated tool offers an enormous number of possibilities, it is very simple to list a few features that could be pursued next, in order to expand the work that has been done. In this context, one thinks, for example, of staircases, roof slopes, kitchen units or built-in cupboards that could be reconstructed in an extension. The reconstruction could be even more substantial by extracting and modelling power connections, built-in lights, vents, radiators, water installations, and toilets or sinks, etc. Furthermore, balconies, or more generally, building measurements from the outside have not been discussed or implemented so far. For working in a BIM environment, it would in addition be exciting to record information about the type or the material of walls, windows, doors, sun-blinds, or the structure of a facade. These are all points that are important for the architect’s planning and the more of these points there are that can be automatically generated, the greater the benefit of this process.

My vision for further work in addition lies in a second area: in the even more sensitive extraction of elements and objects from the point cloud. Imagine that the rooms no longer have to be *cleared out* but that all elements could be recognised individually and the points therefore grouped together. Objects such as furniture could thus be grouped on a furniture layer and faded in or out depending on the depth of representation. This is already done in the planning, which is why the proposed procedure would represent a desirable extension of the pipeline. However, care should be taken that here, too, no unnecessarily detailed meshes are formed, but rather

simplified vector bodies.

Furthermore, methods might be considered which make use of artificial intelligence. We have already encountered a few ideas on this in Chapter 2. Such systems require multiple resources, however, they could contribute to the fast and automated recognition of objects in the recorded environments. It is relatively easy for humans to distinguish and group objects and elements based on point clouds and reconstruct them. The benefit of a technology that is highly oriented towards this human ability would, in my opinion, be enormous.

Last but not least, it should be emphasised that almost all steps of the pipeline can be improved. This study has focused on automating the entire process. Many concepts have been kept simple, thus, these could certainly be optimised to achieve better performance or more accurate reconstructions, for example. The study is characterised by the overall process and the usability in a BIM environment.

6 Acknowledgements

This research was made possible by the Visualization and MultiMedia Lab at the Department of Informatics at the University of Zurich. Support was given by the Institute of Prof. Dr. Renato Pajarola, who funded the work in all its initial stages. I gratefully acknowledge the help provided by Prof. Dr. Renato Pajarola. I would like to express my sincere thanks to him for giving me the opportunity to write this thesis. His work already done in this research area was very inspiring and due to my educational path, which brought me from architecture to mathematics and finally to computer science, the topic was perfectly suited to me. I am very grateful that these very bachelor thesis was available, because, since I started studying informatics, it has been my wish to be enabled to realise this project. My thanks as well go to Lizeth J. Fuentes Perez, who has been excellent in guiding me through my work. Thank you for the numerous inputs and the valuable suggestions and discussions. I am very grateful for that.

Furthermore, I gratefully acknowledge my mother, who has always stood by my side. Not only during the process of writing this thesis, thus, throughout my entire educational and life journey so far. This gratefully thanks as well goes to my husband, since without the always inspiring conversations and the guidance through my study time, I would not be at this point today. Last but not least, I would like to thank my children Malia and Lennox. You are the light of my life that drives me to do my best every day.

Bibliography

[ACW17] Rareş Ambruş, Sebastian Claici, and Axel Wendt. "Automatic Room Segmentation from Unstructured 3-D Data of Indoor Environments". In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 749–756.

[All+] Pierre Alliez, Simon Giraudot, Clément Jamin, Florent Lafarge, Quentin Mérigot, Jocelyn Meyron, Laurent Saboret, Nader Salman, Wu Shihao, and Necip Fazil Yildiran. *Point Set Processing Normal Estimation*. URL: https://doc.cgal.org/latest/Point_set_processing_3/index.html#Point_set_processing_3NormalEstimation. (Accessed on 22/06/2022).

[Arm+16] Iro Armeni, Ozan Sener, Amir R Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. "3D Semantic Parsing of Large-Scale Indoor Spaces". In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 1534–1543.

[BB10] Angela Budroni and Jan Böhm. "Automated 3D Reconstruction of Interiors from Point Clouds". In: *International Journal of Architectural Computing* 8.1 (2010), pp. 55–73.

[BLF19] Yizhak Ben-Shabat, Michael Lindenbaum, and Anath Fischer. "Nesti-Net: Normal Estimation for Unstructured 3D Point Clouds using Convolutional Neural Networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10112–10120.

[BM12] Alexandre Boulch and Renaud Marlet. "Fast and Robust Normal Estimation for Point Clouds with Sharp Features". In: *Computer Graphics Forum* 31.5 (2012), pp. 1765–1774.

[BM16] Alexandre Boulch and Renaud Marlet. "Deep Learning for Robust Normal Estimation in Unstructured Point Clouds". In: *Processings Eurographics Symposium on Geometry Processing*. Vol. 35. 5. 2016, pp. 281–290.

[Bou] Paul Bourke. *File format PTX*. URL: http://paulbourke.net/dataformats/ptx/. (Accessed on 09/07/2022).

[FB81] Martin A Fischler and Robert C Bolles. "Graphics and Image Processing Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Communications of the ACM* 24.6 (1981), pp. 381–395.

[Ger] Fabian Gerold. *IFC++-Open Source IFC Implementation*. URL: https://www.ifcquery.com. (Accessed on 22/07/2022).

[GH20] Uuganbayar Gankhuyag and Ji Hyeong Han. "Automatic 2D Floorplan CAD Generation from 3D Point Clouds". In: *Applied Sciences (Switzerland)* 10.8 (Apr. 2020), pp. 1–12.

[GIS] GIS-ZH. *Geographical Information System*. URL: https://www.zh.ch/de/planen-bauen/geoinformation.html. (Accessed on 09/07/2022).

[Hou62] Paul VC Hough. *Method and Means for Recognizing Complex Patterns*. 1962.

[Hub11] Daniel Huber. "The ASTM E57 File Format for 3D Imaging Data Exchange". In: *Three-Dimensional Imaging, Interaction, and Measurement*. Vol. 7864. 2011, pp. 88–96.

[IDC] IDC.ch. *ArchiCAD*. URL: https://www.idc.ch. (Accessed on 09/07/2022).

[ISO] ISO.org. *Industry Foundation Classes*. ISO 16739-1:2018. URL: https://www.iso.org/standard/70303.html. (Accessed on 08/07/2022).

[Jun+18] Jaehoon Jung, Cyrill Stachniss, Sungha Ju, and Joon Heo. "Automated 3D Volumetric Reconstruction of Multiple-Room Building Interiors for as-built BIM". In: *Advanced Engineering Informatics* 38 (2018), pp. 811–825.

[LA13] Florent Lafarge and Pierre Alliez. "Surface Reconstruction through Point Set Structuring". In: *Computer Graphics Forum* 32.2 (2013), pp. 1–10.

[Lei] Leica. *Estimating Surface Normals in a PointCloud*. Point Cloud Library. URL: https://leica-geosystems.com/en-in/products/laser-scanners/scanners/blk360. (Accessed on 30/06/2022).

[MLG17] Hélène Macher, Tania Landes, and Pierre Grussenmeyer. "From Point Clouds to Building Information Models: 3D Semi-Automatic Reconstruction of Indoors of Existing Buildings". In: *Applied Sciences (Switzerland)* 7.10 (2017), p. 30.

[MMP16] Claudio Mura, Oliver Mattausch, and Renato Pajarola. "Piecewise-planar reconstruction of multi-room interiors with arbitrary wall arrangements". In: *Computer graphics forum*. Vol. 35. 7. Wiley Online Library. 2016, pp. 179–188.

[Mon20] Francesca Monzeglio. "A Toolkit for the Geometric and Semantic Annotation of Point Cloud Data". MA thesis. University of Zurich, 2020, pp. 1–108.

[MP17] Georgios-Tsampikos Michailidis and Renato Pajarola. "Bayesian Graph-cut Optimization for Wall Surfaces Reconstruction in Indoor Environments". In: *The Visual Computer* 33 (Oct. 2017), pp. 1347–1355.

[Mur+14] Claudio Mura, Oliver Mattausch, Alberto Jaspe Villanueva, Enrico Gobbetti, and Renato Pajarola. "Automatic Room Detection and Reconstruction in Cluttered Indoor Environments with Complex Room Layouts". In: *Computers and Graphics (Pergamon)* 44.1 (2014), pp. 20–32.

[Mur+17] Srivathsan Murali, Pablo Speciale, Martin R. Oswald, and Marc Pollefeys. "Indoor Scan2BIM: Building Information Models of House Interiors". In: *International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 6126–6133.

[MWP18] Claudio Mura, Gregory Wyss, and Renato Pajarola. "Robust Normal Estimation in Unstructured 3D Point Clouds by Selective Normal Space Exploration". In: *The Visual Computer* 34.6 (2018), pp. 961–971.

[Nan] Liangliang Nan. *Polygonal Surface Reconstruction Reference*. URL: `https://doc.cgal.org/latest/Polygonal_surface_reconstruction`. (Accessed on 22/06/2022).

[NW17] Liangliang Nan and Peter Wonka. "PolyFit: Polygonal Surface Reconstruction from Point Clouds". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2017, pp. 2372–2380.

[Och19] Sebastian Klaus Ochmann. "Automatic Reconstruction of Parametric, Volumetric Building Models from 3D Point Clouds". PhD thesis. Universitäts- und Landesbibliothek Bonn, 2019.

[Oko+10] Brian Okorn, Xuehan Xiong, Burcu Akinci, and Daniel Huber. "Toward Automated Modeling of Floor Plans". In: *Proceedings Symposium on 3D Data Processing, Visualization, and Transmission* (2010).

[OLA14] Sven Oesau, Florent Lafarge, and Pierre Alliez. "Indoor Scene Reconstruction using Feature Sensitive Primitive Extraction and Graph-cut". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 90 (2014), pp. 68–82.

[OLea] Gabe O'Leary. *How to use a KdTree to search*. Point Cloud Library. URL: `https://pcl.readthedocs.io/en/latest/kdtree_search.html`. (Accessed on 22/06/2022).

[OLeb] Gabe O'Leary. *How to use Random Sample Consensus model*. Point Cloud Library. URL: `https://pcl.readthedocs.io/en/latest/random_sample_consensus.html`. (Accessed on 22/06/2022).

[Pex21] Katherine Pexman. "Automated Floor Plan and Building Model Creation for Cultural Heritage Buildings from Laser Scanner Data". PhD thesis. University of Calgary, 2021.

[Poi] Pointclouds.org. *File format PCD*. URL: `https://pointclouds.org/documentation/tutorials/pcd_file_format.html`. (Accessed on 09/07/2022).

[QT] QT. *QT cross-platform software for creating graphical user interfaces*. URL: `https://www.qt.io`. (Accessed on 09/07/2022).

[Rusa] Radu B. Rusu. *Downsampling a PointCloud using a VoxelGrid filter*. Point Cloud Library. URL: `https://pcl.readthedocs.io/projects/tutorials/en/latest/voxel_grid.html`. (Accessed on 22/06/2022).

[Rusb] Radu B. Rusu. *Estimating Surface Normals in a PointCloud*. Point Cloud Library. URL: `https://pcl.readthedocs.io/en/latest/normal_estimation.html`. (Accessed on 22/06/2022).

[Rusc] Radu B. Rusu. *Plane Model Segmentation*. Point Cloud Library. URL: `https://pcl.readthedocs.io/en/latest/planar_segmentation.html`. (Accessed on 22/06/2022).

[SDK09] Ruwen Schnabel, Patrick Degener, and Reinhard Klein. "Completion and Reconstruction with Primitive Shapes". In: *Proceedings of the symposium on 3D data processing, visualization and transmission*. Vol. 28. 2. 2009, pp. 503–512.

[SIA] SIA. *Normenwerk des Schweizerischen Ingenieure und Architektenverein*. URL: http://shop.sia.ch/normenwerk/architekt/414-1_2016_d/D/Product. (Accessed on 09/07/2022).

[Sto+19] Vladeta Stojanovic, Matthias Trapp, Rico Richter, and Jürgen Döllner. "Generation of Approximate 2D and 3D Floor Plans from 3D Point Clouds". In: *VISIGRAPP*. Vol. 1. 2019, pp. 177–184.

[SWK07] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. "Efficient RANSAC for Point-Cloud Shape Detection". In: *Computer Graphics Forum* 26.2 (2007), pp. 1–12.

[TB15] Charles Thomson and Jan Böhm. "Automatic Geometry Generation from Point Clouds for BIM". In: *Remote Sensing* 7.9 (2015), pp. 11753–11775.

[TZ13] Eric Turner and Avideh Zakhor. "Watertight Planar Surface Meshing of Indoor Point-clouds with Voxel Carving". In: *Proceedings 3DV*. 2013, pp. 41–48.

[Usha] Sergey Ushakov. *Min-Cut Based Segmentation*. Point Cloud Library. URL: https://pcl.readthedocs.io/projects/tutorials/en/latest/min_cut_segmentation.html. (Accessed on 22/06/2022).

[Ushb] Sergey Ushakov. *Region Growing Segmentation*. Point Cloud Library. URL: https://pcl.readthedocs.io/en/latest/region_growing_segmentation.html. (Accessed on 22/06/2022).

[Wika] Wikipedia. *Building Information Modeling*. URL: https://en.wikipedia.org/wiki/Building_information_modeling. (Accessed on 30/06/2022).

[Wikb] Wikipedia. *Delaunay-Triangulierung*. URL: https://de.wikipedia.org/wiki/Delaunay-Triangulierung. (Accessed on 22/06/2022).

[Xio+13] Xuehan Xiong, Antonio Adan, Burcu Akinci, and Daniel Huber. "Automatic Creation of Semantically Rich 3D Building Models from Laser Scanner Data". In: *Automation in Construction* 31 (2013), pp. 325–337.

[Zol+18] Michael Zollhöfer, Patrick Stotko, Andreas Görlitz, Christian Theobalt, Matthias Nießner, Reinhard Klein, and Andreas Kolb. "State of the Art on 3D Reconstruction with RGB-D Cameras". In: *Computer Graphics Forum*. Vol. 37. 2. 2018, pp. 625–652.