MSc thesis in Geomatics

Automatic building feature detection and reconstruction in IFC models

Jasper van der Vaart

2022

TU Delft

MSc thesis in Geomatics

Automatic building feature detection and reconstruction in IFC models

Jasper van der Vaart

June 2022

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

The work in this thesis was carried out in the:

3D geoinformation group
Delft University of Technology

Supervisors:	Dr. Ken Arroyo Ohori
	ir. Nadine Hobeika
Ext. supervisor	ir. Francesca Noardo
Co-reader:	Prof.dr. J.E. Stoter

Abstract

The flexibility of the Industry Foundation Classes (IFC) format makes it challenging to blindly rely on the data it stores. This is because, to allow its flexibility, the stored data in an IFC file can be missing, incorrect, redundant, or stored in vague ways while the file can still be considered format compliant. This missing reliability is an issue for a format that is so central to the Architecture, Engineering and Construction (AEC) industry. This also severely hampers the chances for automated processes to be applied in the AEC industry. Chances that could reduce the time and cost spend on a construction while enlarging its quality. Prior done research on this subject is sparse. The small subset of this research that covers the "repair" or reliable extraction of IFC data primarily focuses on idealized situations that do not often occur in practice. Due to the combination of these factors, this master thesis attempts to discover if it is possible to extract reliable data from the unreliable IFC files with an automated method.

Because of the large flexibility and broadness of the IFC format, the scope of the thesis had to be limited. This was done by focusing on the extraction/reconstruction of storeys (and their elevations), rooms, and apartments. For these three subjects not only extraction/reconstruction methods have been created, but they have also been implemented in an open source software tool[1].

To develop reliable processes, the processes have to be based upon reliable data. From the data available in the IFC format, the tangible geometry is considered to be the most reliable. However, not all the other data can be discarded. Only the data that is implicitly stored in the tangible geometry can be ignored without losing information. In practice this means that most, if not all, topologic data and a subset of the semantic data can be ignored.

It was found that for the extraction/reconstruction of storeys (and their elevations), a z-value extraction based on created *IfcSlab*/*IfcRoof* groups was fairly effective. These groups were made based on their size, height, and location. This process does show a lowered reliability in buildings with complex storey structures.

It was found that for the extraction/reconstruction of rooms, a voxelization approach functioned very well. The boolean refinement that followed, which took the shape from a rough voxelization to the precise room shape, showed some reliability issues.

Finally, it was found that for the extraction/reconstruction of apartments that rules applied to a created space syntax graph showed promise. The rules that were used were simple, resulting in lower reliability in structures that had more complex room relationships.

In conclusion, this research has been unable to give a definite answer to the issues at hand. However, all three the subjects that have been covered showed promising results and can be considered steps towards a solution.

[1]The tool is available from: `https://github.com/jaspervdv/IFC_ApartmentDetection`

Acknowledgements

In this section I want to thank everyone that was in some degree involved with this project. Everybody giving me any pointers have helped in shaping this project, regardless if this was via spoken conversations, email exchanges or via forum messages, thank you. However, I want to thank a selection of people in particular without whom this project could not have come to fruition.

Firstly, I want to thank my supervisors Ken Arroyo Ohori, Nadine Hobeika and Francesca Noardo. Thank you for giving me the freedom to develop and experiment in my own ways. When I got stuck feedback and tips came swift, which allowed this project to move quickly. I felt that, when needed, a lot of information was supplied without the attempt to steer the project in a certain direction. This allowed me to make mistakes, and learn from them myself. In the end, I believe this resulted in a better final product. I also want to thank Jantien Stoter for co-reading the thesis, giving feedback, and asking critical questions. The current flexibility and user interaction of tool has been added due to this.

Secondly, I want to thank the developer of the IfcOpenShell library, Thomas Krijnen, and the developer of the Syntactic plugin, Pirouz Nourian. Without Thomas' explanations and answers (to a plethora of questions) the tool would not have been close to the quality it is in currently. Pirouz helped me understand the inner workings of the Syntactic tool better. This enabled me to make the apartment detection process a lot more refined than prior was anticipated.

Finally, I want to thank my mother for having listened to me rambling about programming and 3D models, for a full year.

Contents

List of Figures

List of Tables

Acronyms

1. Introduction

1.1. Background & Motivation

The Architecture, Engineering and Construction (AEC) industry is complex. A large amount of parties have to communicate and collaborate to successfully realize a building or construction [Bouchlaghem et al., 2005]. These parties create masses of data that has to be exchanged in an easy and reliable manner. The Industry Foundation Classes (IFC) format was introduced to ease this exchange of data [Ozturk, 2020]. However, to support the diverse needs of the different disciplines in the AEC sector the IFC format is very flexible [Venugopal et al., 2012]. This flexibility can result in data that is missing, incorrect, redundant, or stored in vague ways while the IFC file is still being considered format compliant [Venugopal et al., 2012; Noardo et al., 2021; Ying and Lee, 2021a]. So, although the IFC schema should improve interoperability and clarity, in practice the format is unclear and error-prone [Venugopal et al., 2015; Noardo et al., 2021]. This makes it challenging for a party to rely blindly on the received data that is stored in IFC files. These parties, and the AEC industry as a whole, would benefit from a novel approach that could extract accurate and reliable data based on these possibly unreliable IFC files in an automated manner.

The variable quality of the data stored in IFC files makes it challenging for automated processes to function in an effective and reliable manner. Unreliable IFC files often create unreliable automated output. This forces involved parties to evaluate the data present in an IFC file and manually bring it up to a standard that ensures reliable output of the automated process. An example of this can be seen in the parties that cover the Building Energy Modelling (BEM) computations [Lilis et al., 2015]. BEM computations require special geometry which could be automatically extracted from IFC files with established software packages. However, errors in the IFC files can result in an output with an insufficient quality. This forces the parties to either avoid automated processes and manually extract and correct the required data or manually validate the automated output. The BEM sector, but also other parties in the AEC industry, would be significantly helped by an automated process that could function in a reliable manner on faulty IFC data. An example of an issue that could arise by blindly trusting an IFC file can be seen in Figure 1.1.

A reduced need of the semi-manual validation of the IFC files before automated processing would allow automated processes to be applied more easily and take a more central role in the AEC industry. This would allow the parties to put more focus, time, and effort in the manual creation of design solutions while allowing the computer to resolve the detection and collection of data [Türkyılmaz, 2016]. This will not only save resources at parties individually, but also to the collaborative group as a whole. Involved parties will be able to share products and more accurate feedback in a shorter time span. The improved exchange can possibly result in an overall reduction of cost in both the development and subsequent life of a building [Venugopal et al., 2015] while allowing for more innovative design solutions to be created [Hu et al., 2016].

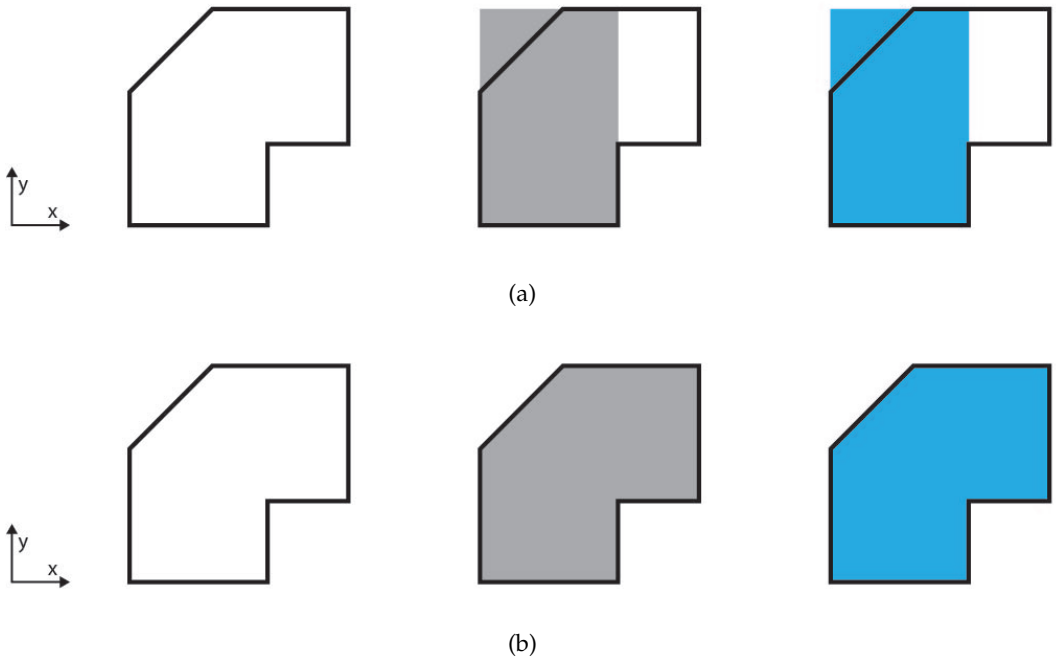


Figure 1.1.: An example of an issue that could be created when relying on IFC data that may not be correct. (a) shows the outlines of a space bound by walls (left). An incorrect space object can be seen in the middle, on which a valid relationship space boundary is based (right), however this is not correct due to the space object being incorrect. (b) shows the desired situation where the space object complies with its surrounding walls.

One could reason that the reliability and accuracy of the data stored in IFC files has to be ensured by the original creator of the IFC file. Thus, the original Building Information Modeling (BIM) model has to be repaired and/or the model creators should be better educated. However, this is a partially flawed idea due to the significant amount of errors that are either hidden from the user and/or are created by the software packages when exporting the models [Katsigarakis et al., 2019; Arroyo Ohori et al., 2018]. These are often errors that the user has no direct control over and can be completely unaware of. Aside from this, having automated software that functions on sub-optimally made IFC files allows firms to more easily share models even though their BIM expertise is limited.

Although the chances that the automated extraction of reliable data from unreliable IFC files opens up, it seems that it has not been extensively covered in prior research. The small subset of research that does cover a form of automated repair or reconstruction often presume an idealised but unreal situation. This research often covers singular errors or only take format and non-standard use into account. However, in practice, a variety of different errors occur, often having effect on each other. These errors range from minor intrusive ones, e.g. presence of small wall clashes and unreferenced features[1], to heavily interoperability limiting ones,

[1]Note that although the presence of a small number of unreferenced features could pass by without notice. Larger numbers of unreferenced features does bloat the already large file size of the IFC format resulting in, among other things, reduced interoperability.

e.g. absence of topologic data, incorrect geometry[2], or incorrectly labelled data.

1.2. Research Goal

The potential that an automated approach of reliable data extraction from unreliable IFC data could provide in combination with its trivial coverage in literature opens up an interesting space for this research. The goal of this research is to not only develop a method that can automatically extract reliable data from faulty IFC files but also correct and enrich the file. This will be attempted to be done by ignoring the majority of the data that is considered error-prone and recreating this data based on data that is considered to be reliable. The output of this method should be a reliable base for already established automation methods to function upon. To successfully create this method this research will address the following question:

- How can building features be automatically detected from an IFC file in a reliable manner?

The term features is used in this thesis as an encompassing term for objects and values. Objects are sets of, but not limited to, walls, floors, and rooms. Values are sets of, but not limited to, dimensions (e.g. storey elevation heights) and classifications (e.g. complex or simple rooms). The research question can only be answered if the following question is answered as well:

- What data in an IFC file can be considered accurate and/or reliable and is the minimal needed data to base the reconstruction upon?

1.3. Research Scope

The presented research questions will be the main focus of the thesis and will not only be attempted to be answered but will also be attempted to be implemented (and tested) in an open source software tool. Due to the flexibility and broadness of the IFC format there are too many building features and objects to evaluate in a reasonable time-span. This makes it challenging to answer these questions for the complete IFC format. And thus, a subset is taken. This research will focus on the detection, reconstruction and storing of storeys, rooms, and apartments. These three elements were selected due to all three being of importance for the building permit sector and the building energy modeling sector. The apartments have an added complexity in the sense that they are often not stored in IFC files or, when they are stored, it is done in non-standard ways.

Effectively, this limited scope creates three main research questions out of the overarching question:

- How can storeys and their elevations be automatically detected from an IFC file in a reliable manner?
- How can rooms be automatically detected from an IFC file in a reliable manner?
- How can apartments be automatically detected from an IFC file in a reliable manner?

[2] e.g. a room shape that does not comply with the walls that should bound it.

1.4. Thesis Structure

- Chapter 2 covers related works. This will be split into two distinct parts. The first part will cover the related works that have been done in the IFC/BIM domain. The second part will cover feature detection works outside of the IFC domain that show interesting methods which could be applied in the IFC/BIM domain.
- Chapter 3 covers the methodology. This is split into four parts. The first part will cover the minimal reliable subset. The following parts will cover the detections, reconstruction, and storage of each of the three selected IFC elements.
- Chapter 4 will go into some of the implementation details of the tool and supportive software. It also covers the datasets which were used to test the tool upon.
- Chapter 5 covers the results of the described methodologies. This chapter is split into three parts where every part covers the result of the detection, reconstruction, and storage of each of the three selected IFC elements. Every part will also cover relevant discussions.
- Chapter 6 will give a brief conclusion, a list of input requirements for the tool, and recommendations for future works.

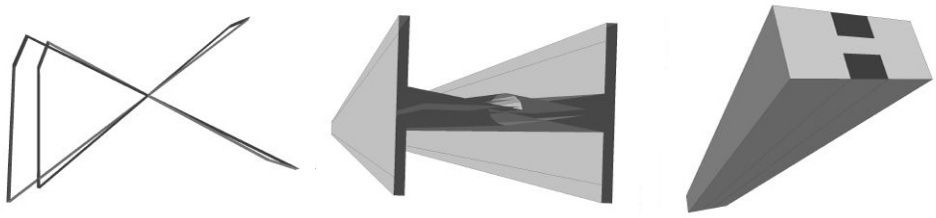
2. Related work

In this chapter, relevant knowledge and related methods are presented. Section 2.1 will primarily focus on the IFC format and its problems. Section 2.2 will cover the already prior created methods/tools that help detect, reconstruct, and/or repair features in the IFC format. Due to the limited research done on the detection, reconstruction, and/or repair of the three selected features in the IFC format, Section 2.3 will cover potential applicable methods of detection and reconstruction in other domains. The apartment detection is a classification process that is usually covered in disconnect from any format. The literature that describes approaches that could be applied to the IFC format can be found in Section 2.4.

2.1. The reliability of the IFC format

As has been mentioned in Chapter 1, the IFC format has been introduced to improve the interoperability of BIM data [Ozturk, 2020]. BIM data should be reliable and easily exchangeable to function in its central role in the AEC industry. If the IFC format is considered as a collection of geometry, semantic data, and topologic data, it can be noted that the reliability in all three types of data is limited in practice.

Arroyo Ohori et al. [2018] showed that the geometric objects in an IFC file can be filled with errors, see Figure 2.1. The most common errors that were found by Arroyo Ohori et al. [2018] were objects that are self-intersecting. Aside from this, planes were not planar while they were supposed to be planar and disconnected objects modelled as one singular object were also noted. An important comment was that these errors were generated by the software with which the original BIM model was created. The users often did not notice these errors and these errors were able to slip through an entire design and construction process. Krijnen et al. [2020] corroborate the presence of incorrect geometric objects in IFC files. This research evaluated space boundaries in three different IFC models. A space boundary is a relationship between a space and its bounding elements. Occasionally a space boundary was incorrect or completely missing. These errors were of a different nature than the ones noted by Arroyo Ohori et al. [2018]. In the described cases by Arroyo Ohori et al. [2018] the object itself was placed correctly and largely intact (all vertices and edges were present) although parts (often faces) were incorrectly shaped. Krijnen et al. [2020] presented objects that could not be classified as being largely intact. These objects had incorrect faces, missing faces, or the objects were completely missing. Krijnen et al. [2020] expanded their research to inaccuracies found in semantic and topologic data as well. The semantic data that was tested was the assignment of objects to certain storeys, this was found to be occasionally incorrect. The tested topologic data that was found to be incorrect was the wall connectivity data. In two of the three models, this data was partially incorrect. These erroneous topologic connections were e.g. missing relations between walls that were connected and walls that were not connected but were marked as connected. Arroyo Ohori et al. [2018] noted that of the errors they



(a) (b) (c)

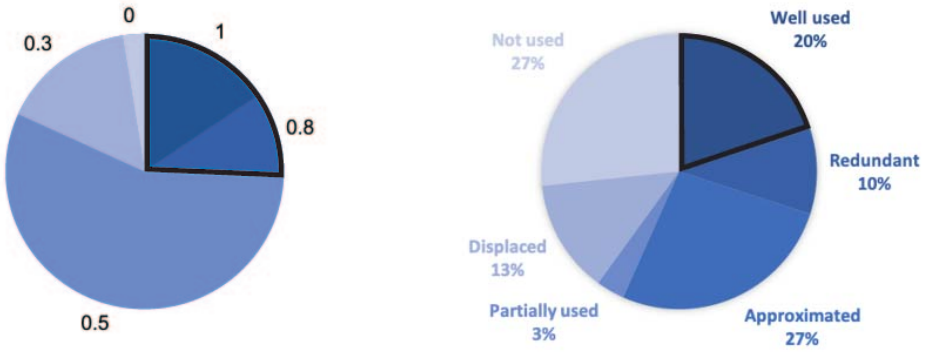
Figure 2.1.: A set of the errors in IFC files found by Arroyo Ohori et al. [2018]. a) A self intersection, b) A self intersections in planes that should be planar c) A topological manifold that contains non-obvious geometric intersections.

found at least a subset can be resolved to support further reliable processing. Creating all the processes that resolve the issues took however too long to implement completely.

As mentioned before these errors were to an extent created by processes that were out of direct control of the creator of the BIM files. This does however not mean that the errors are only introduced by the software packages that are used to create the IFC data. Noardo et al. [2021] studied the use of IFC data in practice. They highlight the user errors that can be present parallel to issues that are potentially created by the software. It is noted that these issues can be actual mistakes but that the IFC format also allows multiple interpretations. This suggests that a subset of these errors have gotten into the file accidentally due to the creator being unaware of that his approach was faulty or non-standard.

Noardo et al. [2021] also demonstrated issues within semantic data. Most of the semantic data of the models they evaluated scored fairly well on their created scale, except for the use of *IfcBuildingElementProxy* objects. *IfcBuildingElementProxy* objects allow users to store data as a generic type. Although the IFC format is very flexible and often enables the user to store data as a variety of predefined types, there can still be situations where a correct type to represent an object is not available. The use of a generic object, the *IfcBuildingElementProxy*, allows users to still store this data in the file. Noardo et al. [2021] mentioned that these generic elements are often used to store data that could be stored in a predefined type. The use of *IfcBuildingElementProxy* for objects that can be correctly stored in a predefined type creates a loss of semantic data that has to be reconstructed or deducted in another way. Even if this incorrect use of the *IfcBuildingElementProxy* is applied to a small set of objects, it can have noticeable effects on the quality of the results created by the automated processes. Around 75% of the models were considered to be unsuitable for automated processes based on the *IfcBuildingElementProxy* use alone. A visual representation of this number can be found in Figure 2.2a.

The research of Noardo et al. [2021] was not limited only to semantic data but extended over a multitude of other types. The most interesting one was the covering of spaces, or *IfcSpace* objects. Only 20% of the considered IFC models had well used *IfcSpace* objects. Another 27% had approximated volumes, these are cases where the rooms did not comply with angled roofs but were just boxes. The other 53% had either displaced *IfcSpace* objects, missing *IfcSpace* objects, or overlapping/misused/redundant *IfcSpace* objects. A visual representation of these numbers can be found in Figure 2.2b.



(a) (b)

Figure 2.2.: Two edited charts with data presented by Noardo et al. [2021]. (a) displays a score given to 39 models related to the presence of *IfcBuildingElementProxy*. The minimum score requirement for reliable automatic processing is 0.6 (the highlighted area). (b) displays the quality of the *IfcSpace* use in these models, only 20% has no apparent mistakes (the highlighted area).

2.2. Automatic detection of features in IFC Files

Noardo et al. [2020] developed an open source tool[1] that reads IFC files and semi-automatically extracts data that can be used to ease the building permit process. The tool is able to check the maximum height, reciprocal overlap and overhang of building parts with respect to the base of the building. The main focus of this research is not on the creation of reliable data based on IFC models with severe errors and does not cover the implementation in depth. However, it does show a way of processing IFC data while ignoring some errors that could be present in an IFC file.

In a series of papers [Ying and Lee, 2020, 2021b,a], the detection, repair, and generation of second order Space Boundaries (SBs) is covered. A second order SB is an IFC object that represents the geometry used for BEM. This series presents a fast and robust approach that creates and corrects more complex SB shapes than established software packages allow [Ying and Lee, 2021a]. The SBs are detected in a building by evaluating curved objects, *IfcSpace* objects, and *IfcSite* objects. These objects are used to extract the Common Boundaries (Cds) between the SBs and their surrounding objects. The use of the curved objects instead of relying solely on the *IfcSpace* and *IfcSite* objects is due to their limited IFC support for curved objects. Reconstructing the curved Cds from the surrounding objects enables more accurate reconstruction. The extraction of the shape of curved objects and the merging of it to the relating *IfcSpace* object is not covered in detail.

To check for errors in SB, 38 rules are constructed that test both semantic and syntactic errors [Ying and Lee, 2021b]. Geometric errors are also tested with the help of a Monte Carlo ray tracing approach. This way it can detect gaps in SBs but also overlapping or overhanging surfaces.

Lilis et al. [2015] cover the detection and (semi-) automatic repair of geometric errors in IFC files, with a partial focus on spaces (rooms). Three different errors are detected and repaired:

[1]The tool can be found at: `https://github.com/tudelft3d/GEOBIM_Tool`

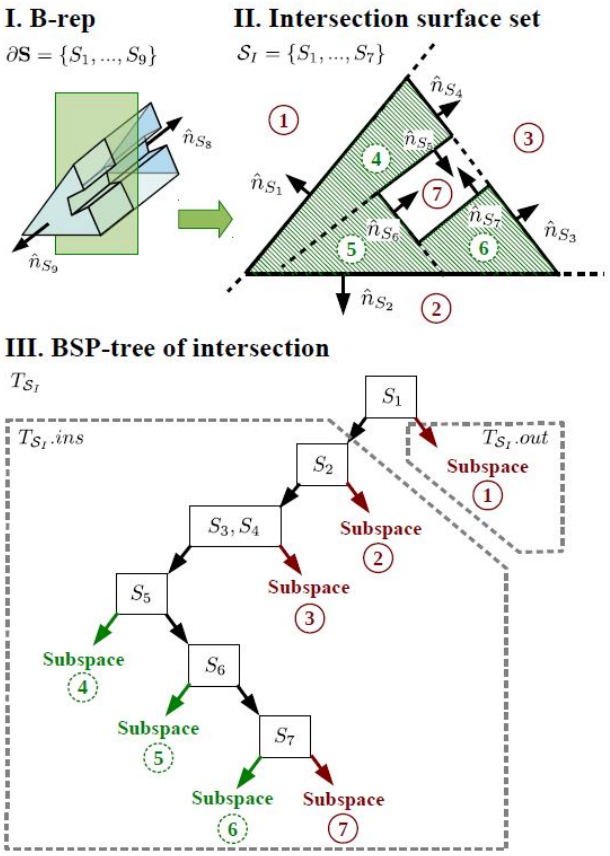


Figure 2.3.: Example of the BSP tree that is being used in the work of Lilis et al. [2015].

clash errors, space definition errors, and incomplete shell errors. The errors are detected, and resolved, with the help of a BSP tree. Every IFC object is used as (or transformed to, details to this are not mentioned) a B-rep. This B-rep is split over the BSP tree where every node of the tree exists out of one, or more, coplanar surface polygons with the same normal direction. The two branches of these nodes represent what lies outside the normal direction and what lies inside of the normal direction, see Figure 2.3. With this BSP tree, it is fairly easy to determine and resolve geometric intersections and contacts with other B-reps. It is however important that all the B-rep normals are correctly orientated before the BSP tree is generated. This way of working is (partially) extended to not only the detection and repair but also the generation of SBs by Lilis et al. [2017]. In this expansion the construction objects are presented. These are objects that bound second order SBs. These objects are: *IfcWall*, *IfcSlab*, *IfcCovering*, *IfcRoof*, *IfcCurtainWall*, *IfcColumn*, *IfcBeam* and *IfcBuildingElementProxy* objects. This research has similar issues as the one of Ying and Lee [2021a] where the success of the detection of the second order SB is dependent on the correct placement and validity of *IfcSpace* objects.

2.3. Automatic room detection in other domains

Research regarding room detection in IFC files without relying on *IfcSpace* objects in any significant way has not been found in literature. However it has been covered in two closely related fields. Interior space extraction directly from BIM models [Xiong et al., 2017] and room detection from paper floor plans [Ahmed et al., 2012, 2011].

Xiong et al. [2017] cover a method for the extraction of interior spaces from complex CityGML LoD4 models. Although these spaces are created from the CityGML format, the presented method could be implemented in the IFC format. The extraction of interior spaces is based on a voxelization approach that can be summarized in three steps. The first step is the voxelization of the original model where the created voxels inherit the semantic relationship of the rooms they represent in the model. Step two is a course extraction of spaces which helps eliminate invalid spaces. The final step uses these rough spaces and the stored semantic data to refine the space shapes. This approach is fairly fast and accurate, however, the created room shapes do not completely comply with the surrounding geometry.

In a series of papers [Ahmed et al., 2012, 2011], syntactic and semantic data is extracted from paper floor plans. A major part of this paper covers extracting texts, icons and wall objects from the flat plans. The central elements for creating the room shape, walls, are however readily available in the IFC model and can be selected with ease. This means that the detection process of walls from this paper can be disregarded. However, the step from wall objects to rooms is of particular interest. This is done by following connected building components until a closed loop is formed [Ahmed et al., 2011]. This loop represents the outer boundary of a room. This outer boundary is combined with earlier extracted semantic data to get a rich room object.

Methods and research covering the transformation of point clouds to 3D volumetric geometry (either b-reps or meshes) have been evaluated but they have been found to generally have unneeded complexity. This complexity is often introduced in point cloud processing due to an uncertainty if geometry that has been detected is actual real geometry that is present in the building or is just an artifact in the point cloud [Mura et al., 2014]. In IFC models, it is however clear which geometry is actual geometry, so this uncertainty is not present.

2.4. Space syntax in indoor environment

Space syntax is a facet of graph theory that represents a spatial configuration as a labelled graph, which enables a designer (or other reviewer) to directly evaluate the configuration and its performance in terms of connectivity [Hillier and Tzortzi, 2006], see Figure 2.4. This tool can be applied to the architectural space and buildings by showing the relations between rooms, hallways, and spaces.

An apartment can be considered a group of rooms. Possibly apartments can be detected from certain connectivity traits or patterns that are recognizable in the graph of a building. Although assessment methods have been developed with graph theory [Jeong and Ban, 2011; Kim et al., 2008], it seems none attempted to find patterns in the spatial configuration. This does not mean that the done research is not usable however. Jeong and Ban [2011] do show a way of displaying the graph data in a matrix shape enabling easy pattern detection. Both

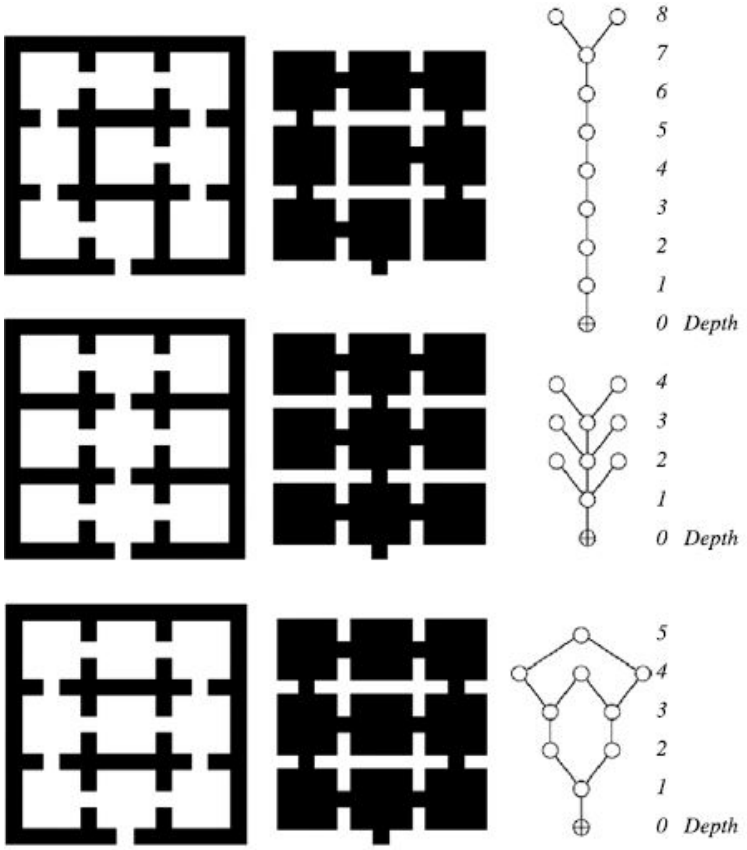


Figure 2.4.: Examples of the graph representations of floor plans. Figure from Jeong and Ban [2011]

Jeong and Ban [2011] and Kim et al. [2008] cover graph extraction based on IFC models. However, the presented extraction of room data from the IFC model is a complex process that relies on only reading the IFC data, hopefully by adjusting the IFC data the need for such a complex technique can be avoided.

3. Methodology

This chapter presents the developed methodology. This methodology is based on the described objectives in Chapter 1 and the covered literature in Chapter 2. The methodology can be split in four parts that are covered in separate sections. Section 3.1 clarifies which subset of data is being used for the detection and reconstruction methods that follow it. Section 3.2 will cover the detection of storey elevations and the correct labelling of objects to those storeys. Section 3.3 will cover the detection and/or creation of the rooms/*IfcSpace* objects' geometry, topological, and semantic data. Section 3.4 will be closely related to the described processes in Section 3.3 but will deviate into the grouping of *IfcSpace* objects into apartments. A schematic overview of the methodology can be found in Figure 3.1.

3.1. Minimal reliable subset

As mentioned in Chapter 1 and 2, the data stored in the IFC format is not always correct or reliable enough to allow downstream processes to function well. To enlarge the reliability of the output of these processes the input data has to be filtered. Not all the available data present in IFC files is required for processing and faulty data could be avoided by ignoring the unneeded data. However, when reducing the total amount of data on which a process is based the negative effect of incorrect data could be amplified. This means that the subset of data that is used to execute further processing should be reliable and chosen thoughtfully in an attempt to avoid the most, if not all, errors.

This subset of reliable data is considered to be the tangible geometry in the file. Tangible geometry are objects that will be tangible when a building is realized, e.g. walls, floors, columns and doors. Intangible geometry are objects that are not actually realized in a building, e.g. values, and objects like room shapes and annotations. The tangible geometry is considered to be plainly visible for users and its correctness is crucial for both automated and non-automated processes. This means that aside from errors in this data being fairly easy to see there is also a large group of people checking this data when passing over it. When the creator/user misses errors in this data incorrect tangible geometry will also be noted by other collaborators, reducing the chances of errors being overlooked. Tangible geometry is often also important for a large group of collaborators, enlarging the chances of errors being spotted and rectified. Tangible geometry has also shown to be a possible source for the reconstruction of other types of data, as for example SBs Lilis et al. [2015]. It is important to note that tangible geometry is not completely free of any errors [Arroyo Ohori et al., 2018]. However, it has also been stated that these errors can often be resolved with already developed tools. The process of resolving these issues is outside of the scope of this research which is why for this research the tangible geometry is presumed to be completely correct.

The only data that can be reconstructed with the help of tangible geometry is data that is implicitly stored in that tangible geometry. For example, the explicitly defined walls bound a room, thus the walls are storing the room boundaries implicitly, see Figure 3.2. Semantic

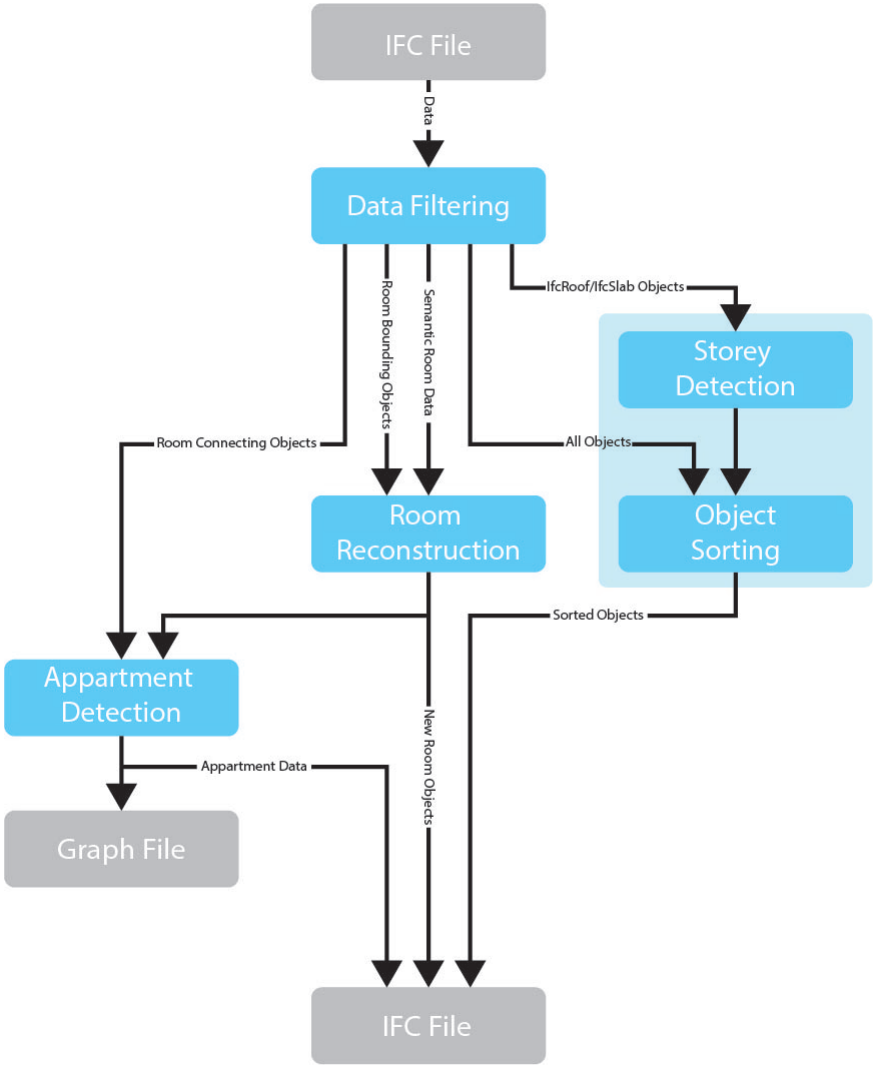
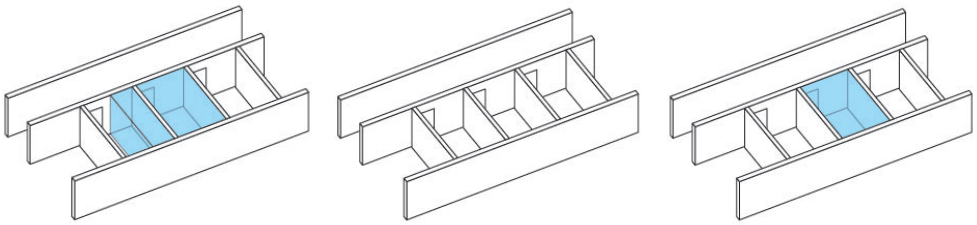


Figure 3.1.: A simplified schematic overview of the methodology. Every blue block will be covered in a section of this chapter. The light blue block encapsulating the storey detection and object sorting will be covered in a single section.



(a) (b) (c)

Figure 3.2.: Example of the possibly derivation of data from other data. (a) shows the original situation where a room (highlighted in blue) does not comply with the walls that should bound it. (b) shows the data that is considered reliable. (c) shows a newly created room (highlighted in blue) that is created based on this data. Note that the floors have been removed from this view to allow for easy display.

data is often not implicitly stored in tangible, or any other, geometry however. This means that the developed solution can not completely rely on tangible geometry only, if it would do so it would result in large data losses. The reliance on other data than tangible geometry will be forced to a minimum. Topological data is regarded as data that is completely recoverable and is thus completely disregarded.

3.2. Detection of storey elevations and the labelling of objects

This section is split in two parts. The first part will cover the way that the storey elevations are approximated. The second part will cover how the objects in the IFC files are sorted into storeys.

3.2.1. Approximation of storey elevation

The literature covered the detection of storey elevations very minimally, this means that a novel method had to be created. This novel method approximates the storey elevations by grouping floor representing objects that are considered to be on the same storey. If the grouping is executed correctly the z-value of every group will resemble the storey elevations of the evaluated building or construction. *IfcSlab* objects are regularly used in the IFC format to represent objects that have a primarily horizontal nature, including, but not limited to, both roofing and flooring objects. The *IfcSlab* objects are not the only objects that should be considered for roofing. *IfcRoof* objects, when they are not used as aggregates, can represent the geometry of a roofing structure. Directly extracting the z-values of all the *IfcSlab*/*IfcRoof* objects in a model will rarely create a reliable storey elevation list. The correct values are presumed to be in that list but depending on the nature and complexity of the evaluated model the list will be filled with noise. This noise can manifest itself in e.g. very small elevation differences, duplicate values or incorrect z-values. That is why the data needs to be preprocessed and filtered. A visual summary of this preprocessing and filtering can be found in Figure 3.3.

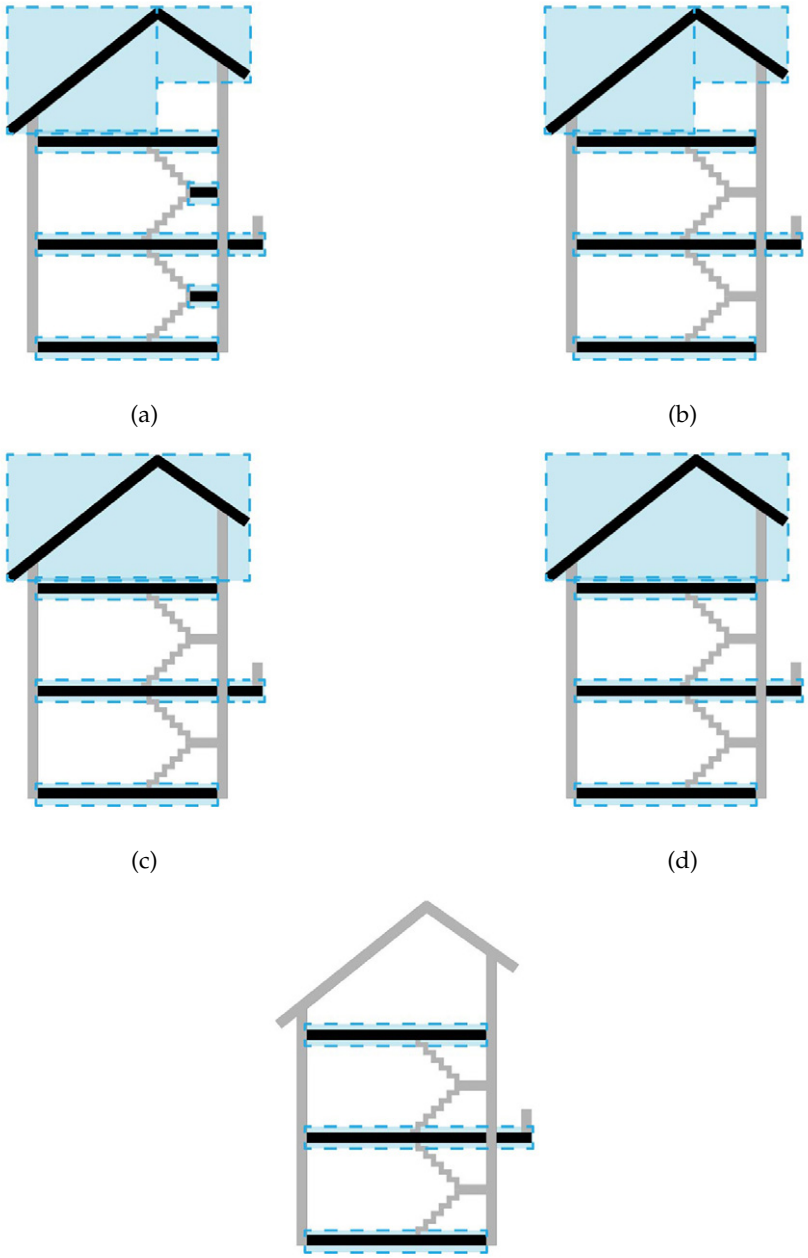


Figure 3.3.: Visual summary of the storey elevation extraction process. (a) is the original situation. (b) *IfcSlab* objects with small top face areas are ignored. (c) neighboring slabs are merged. (d) unique elevation values are extracted. (e) the lowest elevation values from groups of small elevation differences are picked.

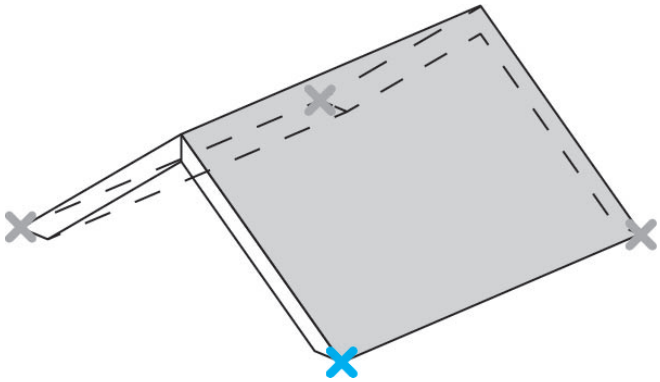


Figure 3.4.: An example of the vertex that is used to extract the z-value of a single *IfcSlab/IfcRoof* grouping, marked in blue. The highlighted in blue vertex is the lowest possible vertex of all the top face vertices of all the objects in one group. In this particular case there are three other options that have the same z-value, marked in gray. If multiple vertices are present with the same z-value the first vertex in the data structure with the desirable z-value is taken.

All the *IfcSlab* and *IfcRoof* objects are initially collected from an IFC file. After the collection this pool of objects is filtered on their top face area, see Figure 3.3b. Small slab objects are ignored ($\leq$10% of the median of all *IfcSlab* and *IfcRoof* top face areas). Doing this results in three benefits:

- It reduces the chance of elevations being evaluated of objects that do not represent flooring or roofing.

  e.g. incorrectly made furniture models or poles and beams that are incorrectly classified as *IfcSlab* objects.

- It reduces the chance of elevations being evaluated of objects that do represent flooring, but are floors that should not be considered on their own storey.

  e.g. balconies and the intermediate landings between stairs.

- Using the median instead of the average to filter the objects reduces the chances of incorrect influence when a small number of extremely large top faced *IfcSlab/IfcRoof* objects are present in the model.

The preprocessing starts with the merging of *IfcSlab/IfcRoof* objects that are neighboring into singular shapes, see Figure 3.3c. This results in horizontally planar *IfcSlab/IfcRoof* objects being seen as one group. This is advantageous for the efficiency of the following computations. Although this is an effect that the process has, it is not the major goal of this step. The major goal is merging the slanted *IfcSlab/IfcRoof* objects that rest against each other into one single group. This enables us to evaluate complex roofing structures as a one shape instead of a collection of shapes. The pairing of neighboring *IfcSlab/IfcRoof* objects is done by evaluating the length of an edge of an *IfcSlab/IfcRoof* object and the length of the begin point of that edge, to the vertex of a potential neighbor, to the end point of that edge. If this distance is sufficiently close to the length of the edge, the objects are considered to be neighbors.

For each of the resulting groups, the z-value is extracted. This z-value is taken from the single lowest possible vertex of all the top face vertices of all the objects in one group, see Figure

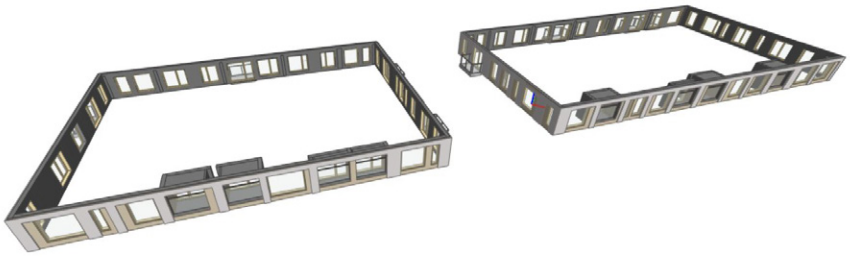


Figure 3.5.: The isolated 8th floor of the facade model of the Boompjes. It can be clearly seen that this model has no data that can enable the software to detect the storey elevation of this floor.

3.4. After ordering this list, it will represent a rough storey elevation list. This list is however still filled with noise that has to be removed. Firstly, all the duplicates are removed, only unique values are allowed to be present in the storey elevation list, see Figure 3.3d. After this step, the elevations with a small difference are grouped. From every group of closely lying elevations the lowest value is taken. The resulting elevation list is considered to be a close approximation of the storey elevation list, see Figure 3.3e.

When multiple files are used to represent a building, the storey approximation process will only be required to evaluate the construction model. This is due to the expectation that a construction model encompasses the entire building and that it includes all the constructional *IfcSlab/IfcRoof* objects representing floors and roofs. The storey elevation list resulting from the construction model is afterwards forced upon the other files, completely discarding the original elevations and other features present in these models. This creates a situation where there is no deviation from the number of storeys and their elevations between files. Aside from this consistency across files, it also enables processing of files that do not have the required data present to enable an accurate approximation of the storey elevations. This is often the case in interior or facade models, see Figure 3.5.

There should always be an extra option that bypasses the approximation of the storey elevations. If the method does not correctly process the data present in the IFC file, it should not influence the following processes.

3.2.2. The sorting and labeling of objects

After the storey elevations are approximated and updated within the IFC files, it is possible to sort the objects present in each file into the new storeys. This is done by taking the z-value of a base point of each object and measuring the z-distance from that object to every storey elevation. The storey where the z-distance from elevation to object base is the smallest while still positive is considered to be the storey at which the object is placed. The object will be labelled with this storey number. For a visual explanation see Figure 3.6. For most objects this base point is the lowest point of the object offset by +20 centimeters, If the object is small (less than 1 meter in height) the lowest point of the object is taken without this offset. Taking

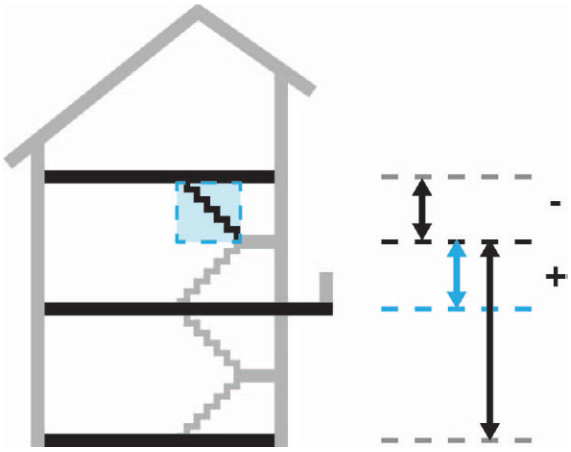


Figure 3.6.: Visual summary of the storey sorting process of an object. The object (the top staircase in this example, highlighted in blue) is to be assigned to a storey. The distance from every storey to the base point of the object is measured, the storey with the smallest positive distance is considered to be the correct storey at which this object is located (highlighted in blue). The distance is positive when the storey elevation is lower than the base point while the distance is negative when it is higher.

this 20 centimeter translated location as a base point resolves the issues that can occur when objects are slightly misplaced due to rotations. An example of this are columns that are placed diagonally, which can create a situation where the lowest point is located below the floor it actually rests upon.

There are some objects processed in a slightly different way. Firstly, the *IfcSlab* and *IfcRoof* objects do not use the z-value of the base point to compute the z-distance to the storey but the lowest point of the top face of the object. This is the same point that is used to get the z-value of a *IfcSlab*/*IfcRoof* object, see Figure 3.4. Taking this z-value results in the *IfcSlab* object being matched to the storey with the same (correct) level instead of one storey too low. The *IfcWall* and *IfcStair* objects use points to compute the z-value that are a $1/3$ of their total height translated up from the base point. This enables reoccurring exceptions to be matched to the correct storey. For example: walls comprising the facade often start lower than the height of the storey they belong to or a *IfcStair* object that occasionally starts below the elevation of the storey they belong to.

3.3. Detection and reconstruction of rooms

This section is split in three parts. The first part will cover the correction of incorrectly classified *IfcSpace* objects present in the inputted IFC files. This has to be done to find the *IfcSpace* objects that have to be replaced/corrected. The second and central part of this section will cover the creation of geometry that represents a room. The third part will cover the recovering and connecting of semantic data to this newly created room geometry.

3.3.1. Semantic data extraction

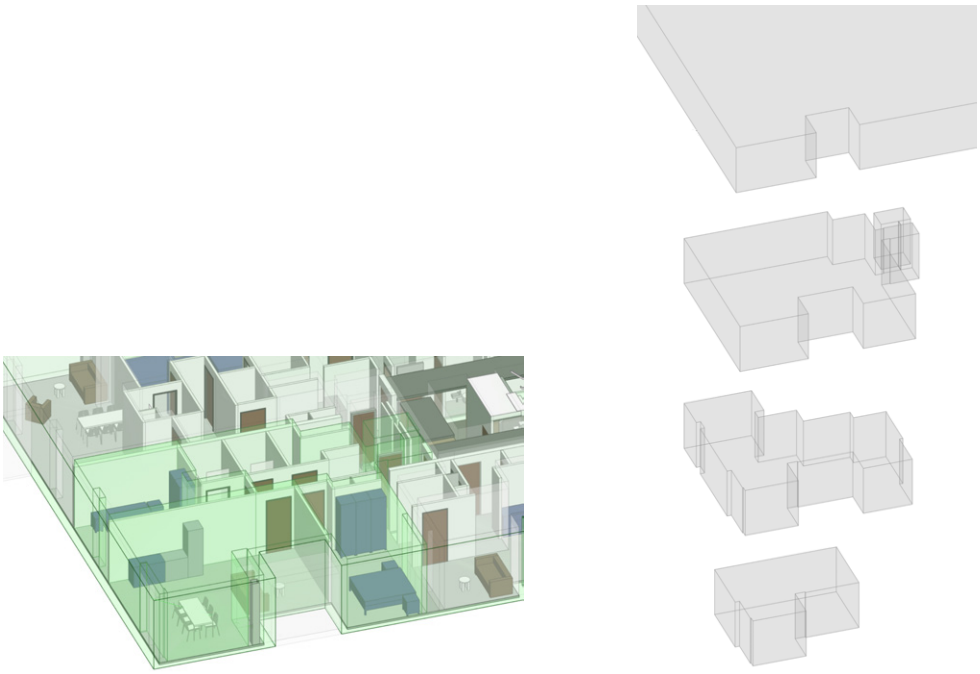
As mentioned in Section 3.1, it is challenging to reconstruct semantic data from other data sources that can be found in an IFC file. When creating new rooms, or *IfcSpace* objects, the semantic data of the original *IfcSpace* object it replaces has to be copied. This means that some of the semantic data has to be acquired from the original input IFC file. The semantic *IfcSpace* data from the input files can rarely be used directly due to the *IfcSpace* objects often being incorrectly classified. *IfcSpace* objects can be classified "Partial", "Element" or "Complex". When classified "Partial" the *IfcSpace* object is presumed to be a part of a space, when classified "Element" it is presumed to be a complete but singular space and when classified "Complex" it is presumed to encompass a group of spaces. *IfcSpace* objects representing rooms should thus be classified as "Element" while *IfcSpace* objects representing areas encompassing multiple other *IfcSpace* objects should be classified "Complex". In practice, most, if not all, *IfcSpace* objects in an IFC file are classified "Element", see Figure 3.7. This creates a challenge in finding the actual *IfcSpace* objects representing a room. To find the *IfcSpace* objects representing the rooms, a check will be done to see if an *IfcSpace* object encompasses another *IfcSpace* object. If this is the case it is presumed that the *IfcSpace* object is not representing a room and its classification can be changed to "Complex". If there is no other *IfcSpace* object found inside, it can be classified as "Element" and its semantic data can be stored for later use. This process can be sped up with the use of a spatial index filled with *IfcSpace* objects.

3.3.2. Reconstruction of room geometry

Although the detection and reconstruction of room representing geometry in the IFC format has been covered in literature, it was chosen to adapt a method that was developed outside of the IFC format. The reasoning for this is that the covered methods created in the IFC space required a substantial set of correct tangible and intangible data to function reliably. This is, as stated before, something that can not always be expected from IFC files. The method developed and presented here is based on the method developed by Xiong et al. [2017] where voxelization is used to reconstruct the room representing geometry. This method has to be extended to allow for valid *IfcSpace* geometry. *IfcSpace* geometry has to be watertight[1]. This is not the case with the rough room shapes created by voxelization. Simple smoothing operations are not able to ensure watertight fitting with surrounding geometry and thus simply smoothing the voxelized shapes will not suffice. That is why an alternative voxelization process has been developed. A visual overview of the process can be found in Figure 3.8 and 3.12.

As the first step of the reconstruction process, the room bounding objects are selected from the IFC file. [Lilis et al., 2017] presented a list of bounding SB geometry. These are the *IfcWall*, *IfcSlab*, *IfcCovering*, *IfcRoof*, *IfcCurtainWall*, *IfcColumn*, *IfcBeam*, and *IfcBuildingElementProxy* objects. The *IfcBuildingElementProxy* objects are however excluded from our room bounding objects. *IfcBuildingElementProxy* objects can be room bounding objects but they can be used for a range of different subjects as well, of which many are not room bounding. Primarily, geometry representing the site is incorrectly classified and created with *IfcBuildingElementProxy* objects, see Figure 3.9. Including these objects creates reliability issues and speed performance issues. After analysing IFC models, the list of objects presented by [Lilis et al., 2017] is considered to be fairly limited. To successfully encompass the room bounding objects, a

[1] The *IfcSpace* geometry has to be directly enclosed by its bounding geometry, no space between it and this geometry is allowed.



(a) (b)

Figure 3.7.: Incorrectly classified *IfcSpace* objects in a model used in practice. (a) shows the overlapping *IfcSpace* objects located in the model (highlighted in green). (b) shows an exploded view of the in (a) highlighted *IfcSpace* objects. of these exploded objects only the lowest should be classified as ”Element” while the rest should be classified ”Complex”. However in this model all the *IfcSpace* objects are incorrectly classified ”Element”.

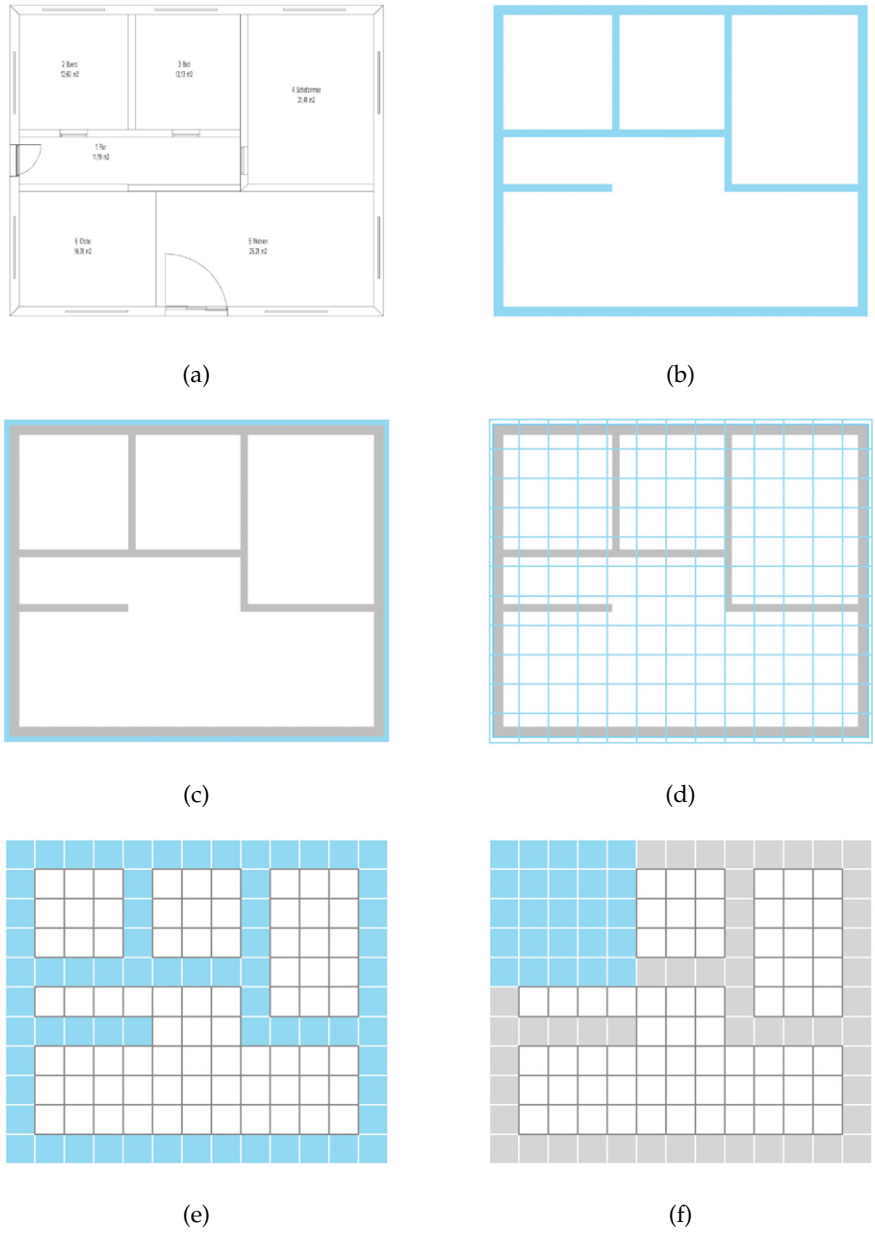
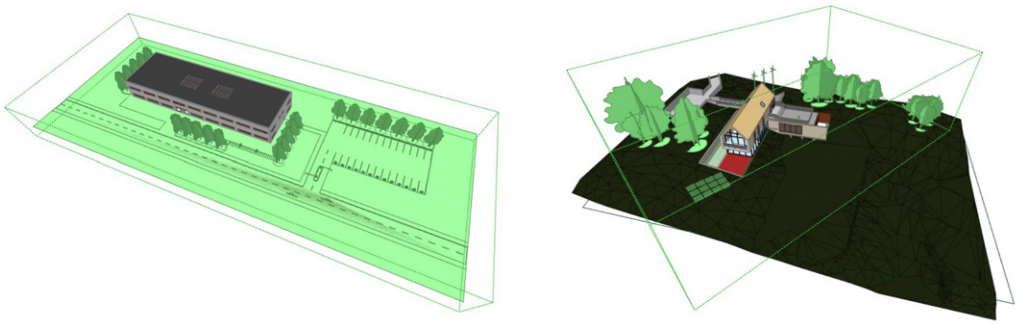


Figure 3.8.: The steps taken when creating a rough room shape. (a) the floor plan of the input IFC model. (b) the room bounding geometry (highlighted in blue) is filtered from the input file. (c) the approximated smallest bounding box (highlighted in blue) is created around the room bounding geometry. (d) the smallest bounding box is rescaled and populated with voxels (highlighted in blue), note that for clarity a large voxel size is chosen. (e) the voxels that intersect with the room bounding geometry are marked as intersecting voxels (highlighted in blue). (f) The first room is grown (highlighted in blue) until no potential voxels are left for this room.



(a) (b)

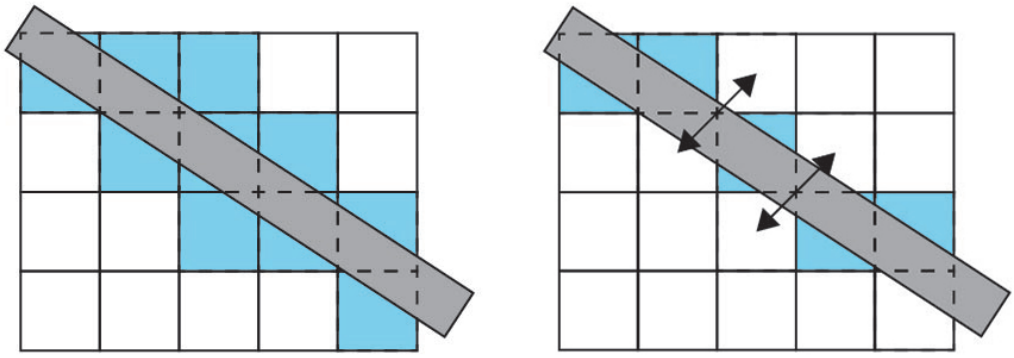
Figure 3.9.: Two examples of incorrectly assigned *IfcBuildingElementProxy* objects (highlighted in green) that are used to represent the site but are assigned as objects representing the building.

new selection is added to this list. The IfcPlate, *IfcCovering*, *IfcMember*, *IfcRoof*, *IfcDoor*, and *IfcWindow* objects are added. All these objects are spatially indexed. This can be done with simple bounding boxes surrounding each of the objects. Although the bounding boxes of IFC objects are occasionally available from the input IFC files, these should not be used. Bounding boxes are not tangible geometry and can be easily created based on the tangible geometry. So, the bounding boxes stored in file have to be discarded. Creating new ones ensures the accuracy of the boxes without the need to check the validity of every box for every object.

When the indexing of the room bounding objects is finished, the approximated smallest oriented bounding box around the construction file is created, see Figure 3.8c for a single file visual explanation. Bounding boxes around the other files are created that have the same rotation angle as the smallest oriented construction bounding box. The construction bounding box is grown to encapsulate the other generated bounding boxes. This creates a rotated bounding box that encapsulates all the objects in a model.

The domain and the rotation angle constructed by the final grown box are used to base the voxel grid upon, see Figure 3.8d. A desired voxel size can be submitted by the user. This voxel size is first used to slightly re-scale the domain. The re-scaling of the domain allows the voxels to completely fill the domain without the need to change voxel's dimensions. This means that the submitted voxel size will be the actual size which is used for the processing. After this, the domain is populated with voxels. The geometry of every voxel is a 3D area that can be used to query the earlier made spatial index of the room bounding objects. This returns a set of objects that could potentially intersect with this voxel. It is tested if there is an actual intersection between the voxel and the object with the signed volume of a tetrahedron approach. An object is considered intersecting with a voxel if it intersects with any of the edges of that voxel. If an intersection is found, the voxel is marked as an intersecting voxel and no further intersections are analysed for this particular voxel. If a voxel does not have any objects that it intersects with, it is not marked as an intersecting voxel. This process results in a voxelization of the room bounding objects, see Figure 3.8e.

Once every voxel has been evaluated for a room bounding object intersection, the room geometry can be grown, see Figure 3.8f. This is done by selecting a voxel that has not been assigned already (voxels can be unassigned or assigned as part of a room, as being outside of



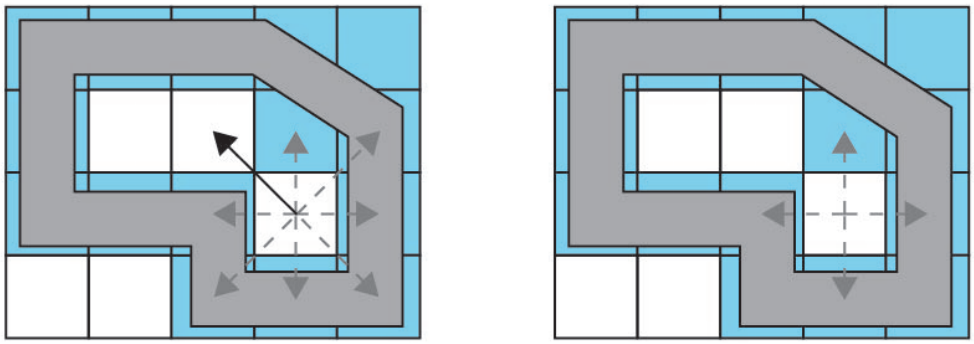
(a) (b)

Figure 3.10.: 2D explanation of the voxel intersection and 26-connectivity where gray object is room bounding geometry and blue are intersecting voxels. (a) the chosen approach where due to the choice to consider a voxel to be intersecting with an object when it intersects with any of its edges disallows intersecting voxels to be connected diagonally. (b) if this rule is changed to e.g. intersection of the midpoint of the voxel intersecting voxels could be connected diagonally. This creates diagonal openings where spaces could grow through when 26-connectivity is used for growing.

the building, or as an intersecting voxel). From this voxel a rough room shape can be grown. The growth is executed via a 26-connectivity growing process. The 26-connectivity voxel approach is chosen because the objects that bound rooms are always volumetric. This means, in combination with the earlier mentioned intersecting voxel classification[2], that there is no possibility of rooms growing through walls, see Figure 3.10. The use of 26-connectivity has as an advantage that it will allow the rooms to grow in narrow diagonal spaces, enlarging the chances of accurate room approximation while using fairly big voxel sizes, see Figure 3.11.

The room growing process grows a room into all the non intersecting voxels until it encounters an intersecting voxel. An intersecting voxel means the boundary of a rough room shape. But, instead of keeping the room bounded by the intersecting voxels, the room is allowed to include the intersecting voxel as room object. This means that intersecting voxels can (and will) be part of multiple rough room shapes while non-intersecting voxels will always be included in a single rough room. This intersecting voxel will however not be used to find any neighboring voxels, locally ending the growing process at that particular voxel. If the growing process encounters a voxel that does not have 26 neighbors, this voxel is deemed as being on the boundary of the domain and thus as being outside of the building. A room is presumed to be always completely encapsulated by space bounding objects. Thus, if the space that is being grown encounters the end of the domain, it means that it is not inside of a building. If the voxel is found to be outside of the building, it will cause all the prior found and upcoming found voxels for the currently being grown room shape to be marked as exterior voxels. Voxels, and in extension rough room shapes, that are marked exterior will not continue into the next step of the room reconstruction process. Instead they are discarded and not used further.

[2]voxels are being considered intersecting voxels if any of their edges intersect with a room bounding object

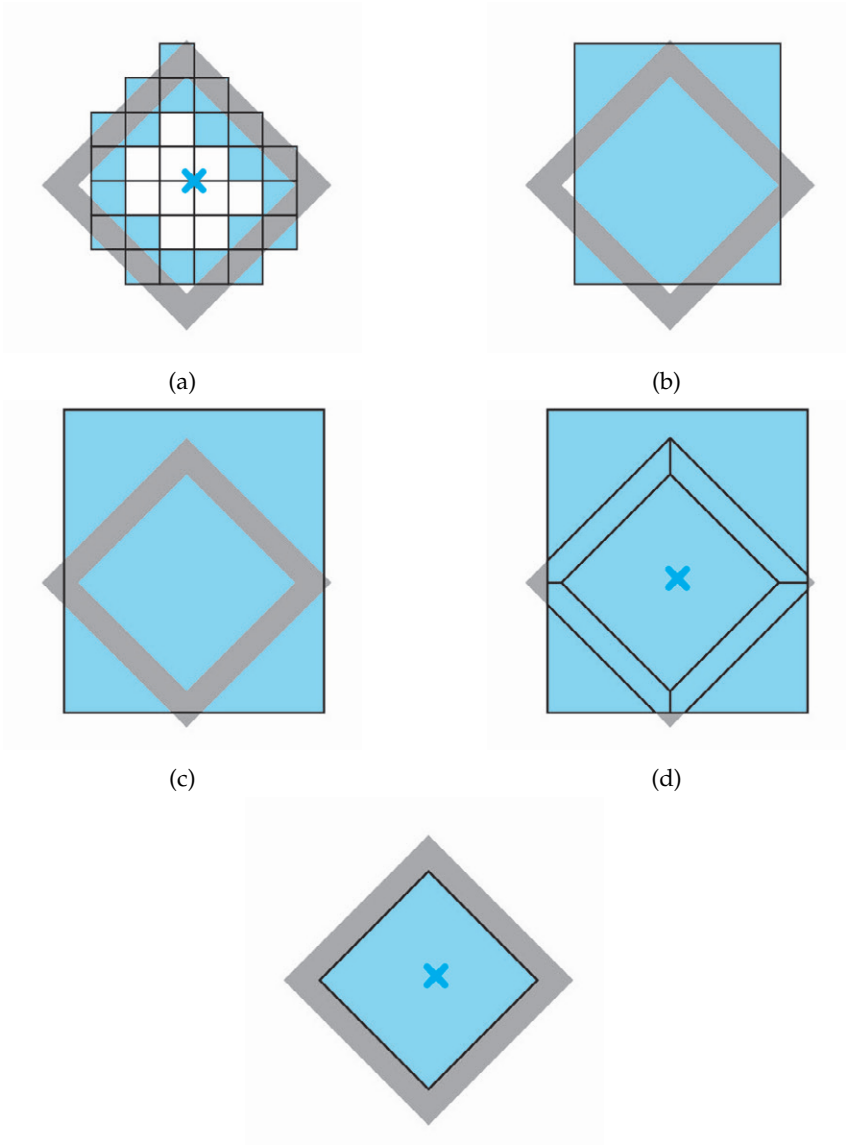


(a) (b)

Figure 3.11.: 2D explanation of the benefit of 26-connectivity when voxel sizes are (too) large. The gray geometry is room bounding geometry and the blue geometry are intersecting voxels. (a) is the chosen approach where due to the 26-connectivity narrow rooms can still be grown. If the lower right voxel is the beginning of the room growing process the other two voxels can be found as part of the rough room shape (the first step is the black arrow). (b) if for example 6-connectivity was picked this would have been impossible, the growing process would only find intersecting voxels from which no growing steps can be taken, resulting in an inaccurate room shape.

The growing of a rough room shape is finished once no unassigned neighboring voxels can be found anymore. If this moment is reached for an interior space, the shape refinement can begin. The first step is to create a bounding box around the rough room shape, see Figure 3.12b. This bounding box is the beginning of the *IfcSpace* representation object. This box is scaled up slightly around its center point (Figure 3.12c). This scaling process is done to ensure that the box encompasses the entire set of room bounding objects that enclose the actual room shape. This can resolve issues that can occasionally occur when a building or construction has slanted roofs or tight wall corners, see Figure 3.12a and 3.12b. The created box is used to query the spatial index with room bounding objects. The returned objects are used to split the box in a subset of pieces (Figure 3.12d). Within these pieces is a solid that correctly bounds the room and can be used as the *IfcSpace* representation object. This solid is found by firstly removing all the objects that have a z-height that is smaller than one meter, no living space is lower than one meters and can thus be discarded. After these objects have been discarded, a known point inside of the room is selected. This point is found by searching the voxels constructing the rough room for a non-intersecting voxel. If this non-intersecting voxel is found, one of its corner points is taken (Figure 3.12a). We know for certain that this point is inside of the room shape. This is due to the non-intersecting voxels being completely free from any intersection. so, it can not lie partially outside of the room shape and thus none of its corner points could lie outside of the room shape. For every residual solid, it is tested if this point lies inside of it. If it does, this solid is considered to be the room representing geometry (Figure 3.12e).

The splitting of solids is a costly process that has a limited robustness. That is why it is necessary to keep the solids used in the split process as simple as possible while still creating



(a) (b) (c) (d) (e)

Figure 3.12.: The steps taken when going from a rough room shape to a refined room shape from a top down plan view, the gray outlines are wall objects. (a) the rough room shape with the intersecting voxels marked in blue and the point that is known to be in the room indicated with a blue x. It can be seen that the room grows into the intersecting voxels. (b) the resulting bounding box of this rough room shape in blue, it can be seen that it does not encapsulate the entire actual room shape. (c) the bounding box is scaled up to reduce the chance of partial encapsulation. (d) the result of the splitting process, where every blue area is a solid that could represent the room shape. (d) the final room shape that is found by testing if earlier found point (the blue x in (a)) fell inside of the solid.

the desired precision for a room shape. *IfcWindow* and *IfcDoor* objects often have a lot more detailed geometry than is required for the creation of a room shape. There is no need for the room shape to accurately follow the outline of the window frame for example. The complexity of the objects also brings a bigger chance of geometric inaccuracies and errors being present in these objects which can severely hamper the performance of the splitting process. This possible presence of inaccuracies was a known issue which was chosen to be ignored by this research. However, when these complex objects are present in the room bounding object pool, they can not be completely discarded. This complex shape issue that is potentially present in the *IfcWindow* and *IfcDoor* objects is addressed in two ways.

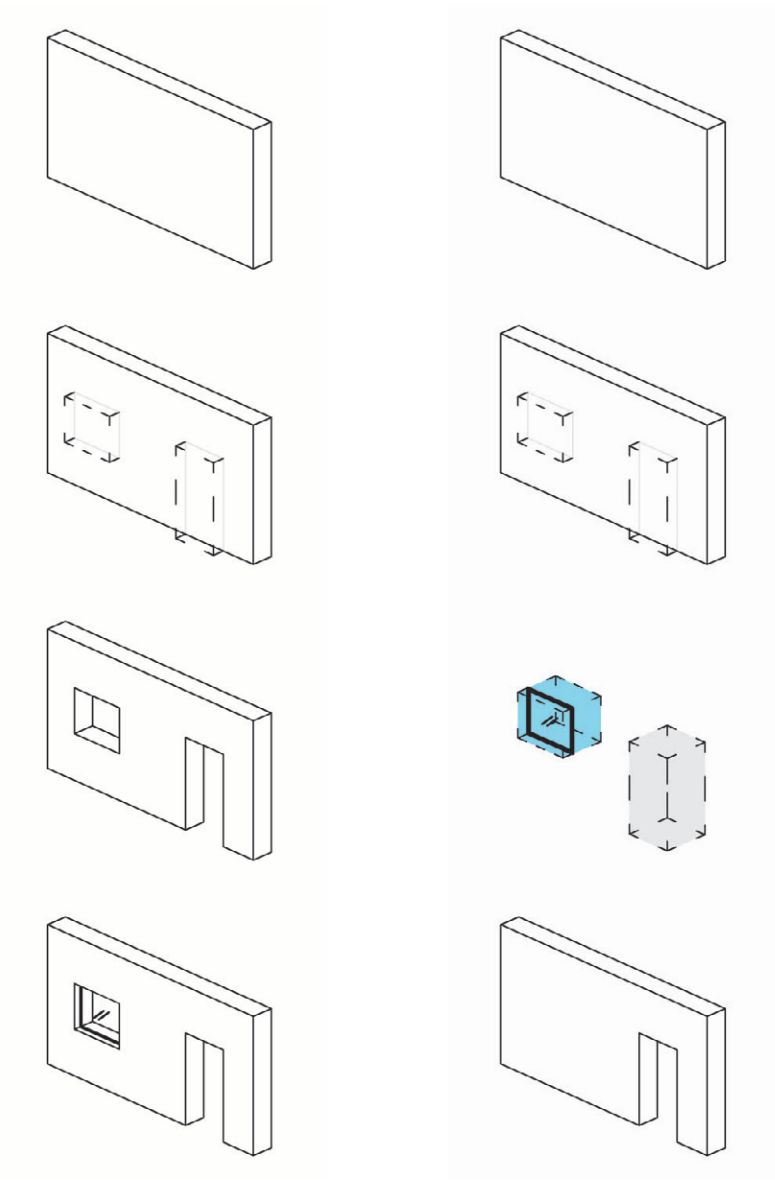
The first way this issue is addressed, is by creating an adjusted application of the *IfcOpening-Element* objects for *IfcWall* objects, see Figure 3.13. *IfcOpeningElement* objects are the spatial voids that create openings in other IfcObjects. Of the *IfcOpeningElement* objects present for a certain *IfcWall* object, only a handful are usually not occupied by *IfcWindow* or *IfcDoor* objects. For the creation of room shapes, these occupied openings do not have to be applied to the *IfcWall* objects; a wall with holes filled with windows can be considered as the same as a wall that has no openings. The only real difference is the added complexity the former introduces. Thus, *IfcOpeningElement* objects that have space bounding objects placed inside can be ignored. The creation of these simplified walls is done by reconstructing the wall geometry by initially ignoring all the *IfcOpeningElement* objects. This means that no opening elements are initially present in the geometry of an *IfcWall* object. For this wall, each of its related *IfcOpeningElement* objects is checked if it encompasses another (space bounding) object. Only if the *IfcOpeningElement* encompasses no space bounding object, the *IfcOpeningElement* is applied to the wall, creating an opening at that location. This will result in wall geometry that only has openings present at locations where no objects occupy the actual opening.

However, not every complex room bounding shape is encompassed by an *IfcOpeningElement*. The second processing way is used when *IfcWindow* and *IfcDoor* objects are not encompassed by *IfcOpeningElement* objects. By creating an oriented approximated smallest bounding box around these objects, they can be very crudely simplified, see Figure 3.14. *IfcWindow* and *IfcDoor* objects are often small and linear in nature, this means that the smallest bounding box is a resource friendly way of not just surrounding the object by a solid but also creating a simplified shape of it. These bounding boxes are used in the further processing of the room geometry.

There are still some objects that have not been addressed. The primary focus so far has been on *IfcWindow* and *IfcDoor* objects. However, bigger and non-linear shapes can occasionally also show geometry inaccuracies. Especially non-watertight solids[3] are a major issue. When these inaccuracies are spotted, the bounding box simplification is not accurate enough. For bigger room bounding objects that are considered complex, the geometry will be handled per face instead of as one complex shape. These bigger objects, although still complex, often do not have a similar level of complexity that is present in the *IfcDoor* or *IfcWindow* objects. This means that using the faces of the objects can still yield decent results while not having a drastic effect on performance.

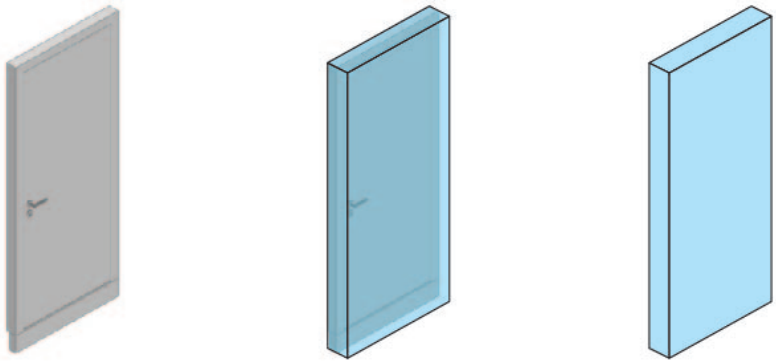
Topological data related to these newly reconstructed *IfcSpace* geometry objects has to be reconstructed as well. This is done by selecting all the objects used to split the room geometry. From each of these objects, a ray is cast from the two points where the object and the refined room shape are closest to each other. If the ray between the object and the room geometry is longer than 0.5 meters, the object is not considered to be topologically connected to the

[3] solids that have faces that are not watertight; have gabs between them.



(a) (b)

Figure 3.13.: An example of simplification by selective *IfcOpeningElement* application. (a) shows the normal application of *IfcOpeningElement* objects. A simple wall is reconstructed, the *IfcOpeningElement* objects are found and they are all applied to the wall object. This results in a fairly complex wall shape where the openings are filled with (often) complex objects. (b) shows the method applied for the room refinement process where, initially, a simple wall is constructed and its *IfcOpeningElement* objects are found. Only now it is tested if these *IfcOpeningElement* objects encompass another (room bounding) object. If they do not they are applied to the simple wall shape. This results in a more simple wall shape and avoids the need to use more complex shapes (for example the shapes of *IfcWindow* objects) to reconstruct the room geometry.



(a) (b) (c)

Figure 3.14.: An example of simplification by bounding box creation. (a) shows a complex *IfcDoor* object. (b) shows this complex object overlaid with its oriented smallest bounding box. (c) shows the resulting bounding box that is used to replace the *IfcDoor* geometry for the room refinement process. The reduction of detail can be clearly seen, every side of the door is constructed out of a single surface and the door handle has been internalized as well.

room. If the ray length is 0 (¡ 0.0001) meters, the split object is considered to be topologically connected to the room. If the ray is between 0 and 0.5 meter in length, it is tested if this ray is intersected by any of the other objects that were used to split the rough room shape. If it is not, this object is also considered to be topologically connected to the room.

3.3.3. Matching semantic data to an IfcSpace object

After the new room geometry is made, the semantic *IfcSpace* data has to be found and copied. As described in 3.3.1, this data has to be copied from the original file. The stored *IfcSpace* objects are classified into ”Element” and ”Complex” objects. The semantic data of the ”Element” *IfcSpace* objects is used to copy. If no semantic data has been found in the file, either due to the file having no *IfcSpace* objects included or due to the process having failed to collect the needed data, the rooms will be given generic names. If the semantic data has been found, this data is related to a point based on each old room’s shape. For every original ”Element” room shape a central lying point is picked, this can not be the center of mass due to L, U, or O-shaped rooms potentially having a center of mass point that lies outside of the room geometry. These central points are then tested to see if they fall within the newly created room geometry. If a point does, the *IfcSpace* object where this geometry belongs to will get the semantic data that is bound to this point (and its original *IfcSpace* object). If multiple points fall inside of the newly created room geometry, the semantic data they supply is appended into one so no information is lost.

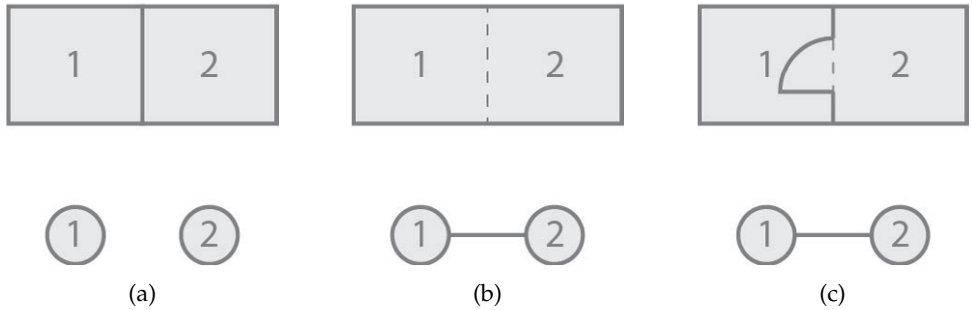


Figure 3.15.: Examples of room connecting objects with the resulting graph connections displayed underneath. (a) shows no connections between two rooms. (b) shows connection via room separators that are discarded from this research. (c) shows connection via a door.

3.4. Detection and reconstruction of apartments

This section is split in two parts. The first part will cover the detection and reconstruction of room connectivity data. The second part will cover chosen rules that are used to pair spaces together into apartments.

3.4.1. Reconstruction of connectivity data

The literature covered room connectivity detection very minimally. When it is covered, it is often used for the navigation within buildings. The goal of these reports was not just creating a graph between rooms but also, to a certain degree, creating valid walking paths. This nuance adds a lot of complexity that is, for the goal of this research, unneeded. That is why a novel method is created that incorporates the connectivity between rooms with as the main goal to "just" create a space syntax graph from a building based on the input IFC files. This method is closely related with the presented method in Section 3.3 and partially uses some of its processes.

As the first step of this process, *IfcStair* objects are spatially indexed. *IfcStair* objects are part of the Room connecting object group. Room connecting objects are the objects that can connect one room to another. These are *IfcDoor* and *IfcStair* objects. In theory, room splits are also room connecting objects; however, the current implementation of the room detection process is unable to split rooms. This means that there are no occurrences of room splits in the input data of this process. A visual representation of some of the connecting objects can be found in Figure 3.15.

The next step in this process locks in with the process that is described in Section 3.3.2. The topologic data of the room that was created includes the *IfcDoor* objects that are connected to that room. The *IfcDoor* objects can be isolated from the complete topologic data and grouped. For these *IfcDoor* objects, it has already been established that they are related to the room. This is the reason why the *IfcDoor* objects are not included in the spatial index of the connecting objects. To query *IfcStair* objects from the spatial index, the upscaled bounding box (see Figure 3.12c) that was created around the rough room shape is used. This bounding box is however often considerably larger than the actual room. Consequently, the result of the query could include *IfcStair* objects that do not have a connection to the room that is being evaluated. To

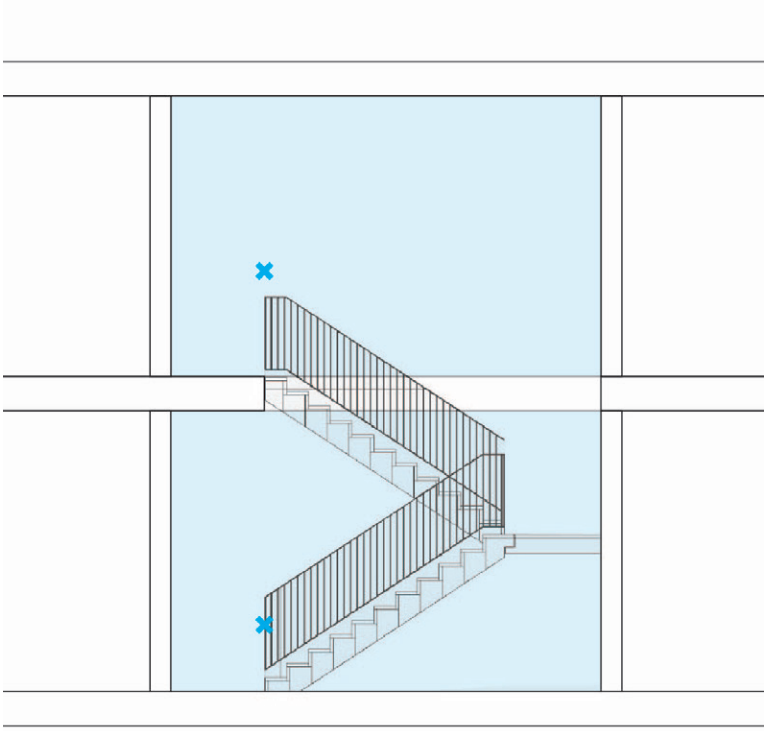


Figure 3.16.: Visual explanation of the staircase to room (highlighed in blue) binding. Two points are extracted from the *IfcStair* object. One is representing the connection at the base and one at the top. If one of these points falls inside of a *IfcSpace* objects the staircase is considered to be connecting to that room.

check if *IfcStair* objects actually connect with a room object, two base points are extracted, see Figure 3.16. One point represents the top of the staircase and one represents the bottom. Both of these points have been translated in a positive z-direction to float slightly higher than the actual bottom and top of the staircase. These points are averaged in the xy-direction reducing the chance of the point resting against a wall (or room) face. If one of these points falls within the room, it is connecting to that room.

If a connection object is found to be connecting to a room, this room is stored as related to the connecting object. If the connecting object already has a related room, it can be concluded that this stored related room connects to the currently evaluated one. A connection between these two rooms can be established. This connectivity data is collected for further processing.

If the entire room connectivity process (and the room detection process described in Section 3.3) is finished, there will be a small subset of connectivity objects that have only one single connection. This, presumably, means that the connectivity object connects an *IfcSpace* object to the outside. This connection to the outside can however be varied in nature. It can be a front door to the garden but it can also be a door on the third floor to a balcony. The door to the balcony does not actually connect the room to the outside. To resolve this issue a rule was introduced: if the connecting object is located below 2 and above -0.5 meter height, it can be marked as a connection to the outside. If not this connection is discarded.

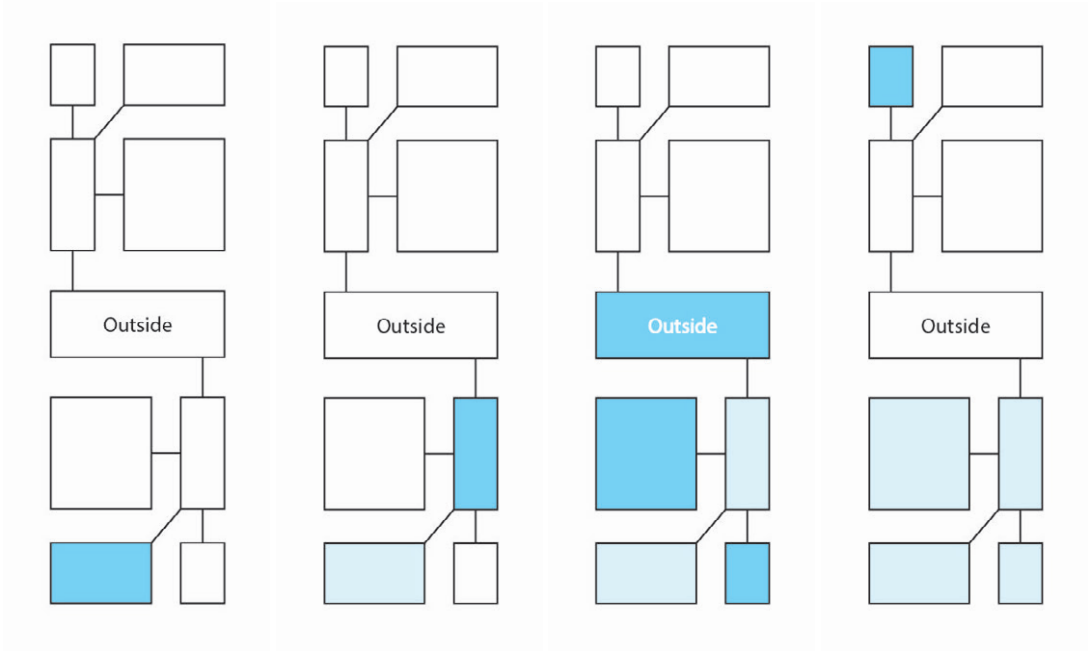
3.4.2. Detection of patterns and creation of rules

The detection of apartments will be done with a simple rule set. With these rules, the spaces and their connectivities can be split into apartments at so called splitting points. These rules are created based on the evaluation of multiple apartment buildings and models but are to a certain extent a simplification of reality. The created rules are:

- Connections to the outside are splitting points.
- An apartment/section has to minimally consist out of two rooms.
- An apartment/section has to have at least one room with a singular connection.
- An apartment/section has to minimally have a floor area of 22 square meters[4].
- A hallway/connection space is a splitting point when it has has five or more door connections it.

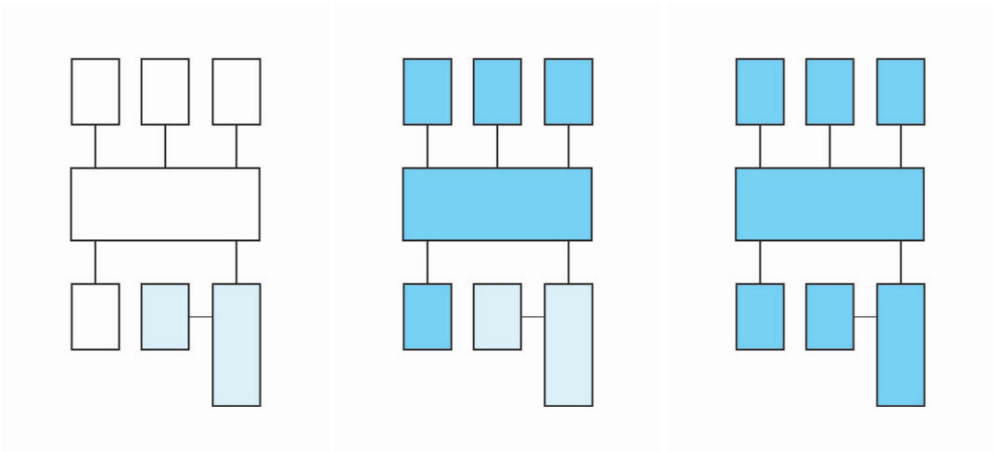
After all the *IfcSpace* objects have been created (so once the processes described in 3.3 and 3.4.1 have been completely finished), apartments can be grown with the help of the created rules. A visual summary of the growing process can be seen in Figure 3.17. The process is started from a room that has only one connection to another room, and, thus, is the outer leaf of a graph. From this leaf room every other connecting room can be added to the apartment. With every step (or series of steps), the apartment and the potential connecting room are tested based on the created rules. If the apartment complies with all the rules when hitting a potential splitting space, that space is marked as such and the apartment can not continue to grow into that space. These splitting points are not definite however. If another apartment grows into this splitting point and does not consider it a splitting point, the splitting point is removed, see Figure 3.18. The apartments that now rest against each other are merged into one singular apartment.

[4]This is the area of the smallest studio in Rotterdam that was available for rent on publicly accessible websites at the time of writing (03/05/2022)



(a) (b) (c) (d)

Figure 3.17.: Visual representation of the apartment growing process. (a) one arbitrary leaf node that has not yet been grouped to an apartment is selected. (b) the grown apartment group checks if it is valid for being an apartment and if the next connecting room is an apartment splitting object, in this case, it is not and it can grow into the next room. (c) this process is repeated, one of the connections is to the outside and thus the apartment is unable to grow in this direction. If the grown apartment group satisfies the set rules and can not continue to grow further it is classified as an apartment. (d) a next arbitrary leaf node that has not yet been grouped to an apartment is selected and grown from.



(a) (b) (c)

Figure 3.18.: Visual representation of the apartment merging process. (a) one apartment (in light blue) has stopped growing due to the next connecting room being considered a splitting point from this perspective. (b) for the second apartment (dark blue) this splitting point is not considered a splitting point and thus the apartment can internalize this room. (c) The two apartments are directly connected to each other without separation of a splitting object and thus they are merged into one.

4. Implementation details

This chapter covers some of the details relating to the development and testing of the open source software tool that were not covered in the previous section. A subset of the implementation details that require more elaboration are covered in this chapter. Section 4.1 elaborates on the IFC files that have been used to test the reliability and effectiveness of the software tool. This is followed by Section 4.2 where the language and libraries used for the software tool are described. This section also covers the way files are processed and how user interactions are dealt with. The final section covers the ways the data is visualized.

4.1. Used datasets

The IFC models used to test and develop the methods range in nature from small single file models to multi file models with thousands of objects. The models can be categorized in simple, moderate, and complex models. The simple models have a fairly low object count, their storey structure is simple and they have a low number of faults present. The moderate models are a lot more complex. These models are not just structurally more complex but they also include a relatively large occurrence of incorrect data and incorrect object use. This is something that is very limited in the simple models. The complex models are very large models or models where clear mistakes and/or structural issues are present. A summary of the used models can be found in Table 4.2. A visual comparison on the difference between the different categories can be seen in Figure 4.1. A complete visual overview of the used models can be found in Appendix B.

This test model collection is a subset of all the available IFC files. The available collection has been vetted for a number of reasons. Firstly, a selection had to be made to be able to keep the time spent on processing and evaluating the models within reason. The set models has been selected to encompass a wide variety of different style of buildings in a range of different sizes and complexities. The models that have a large *IfcBuildingElementProxy* object count or use *IfcBuildingElementProxy* objects as room bounding objects have also been limited. Due to the way the methods have been developed *IfcBuildingElementProxy* are ignored. Thus, it is known that these will perform badly. Table 4.1 shows how many *IfcBuildingElementProxy* objects are present in each of the models.

During the development of the tool, box-like models were used to develop the initial steps of a functionality. These were extremely simple models that were created to test certain aspects of the method/tool. More complex models representing buildings were used to refine the functionality.

Model name (Simplified)	Total # objects	Total # IfcBuilding-ElementProxy objects	Total # room bounding IfcBuilding-ElementProxy objects	% of IfcBuilding-ElementProxy objects	Score (0-1)
FZK-Haus	199	0	0	0%	1
Institute-Var-2	1183	0	0	0%	1
Smiley-West-10	1252	0	0	0%	1
RAC-sample-project	481	69	0	14%	0.3
DigitalHub	5041	0	0	0%	1
ON4 The Building	1414	153	0	11%	0.5
CUVO Building	1750	2	2	0.1%	0.5
Witte_de_Withstraat	1510	104	69	7%	0.3
Projekt Golden Nugget	6138	145	15	2%	0.5
Boompjes	51307	1302	28	3%	0.5
Rabarberstraat	482	120	77	25%	0.5
SubZero	1080	1066	-	99%	0

Table 4.1.: Summary of the *IfcBuildingElementProxy* object count in the tested models. The score column is based on the by Noardo et al. [2021] proposed scores. A score above 0.6 is considered good enough for automated processing. The two separated last entries in the table are examples of models that have not been chosen to be evaluated.

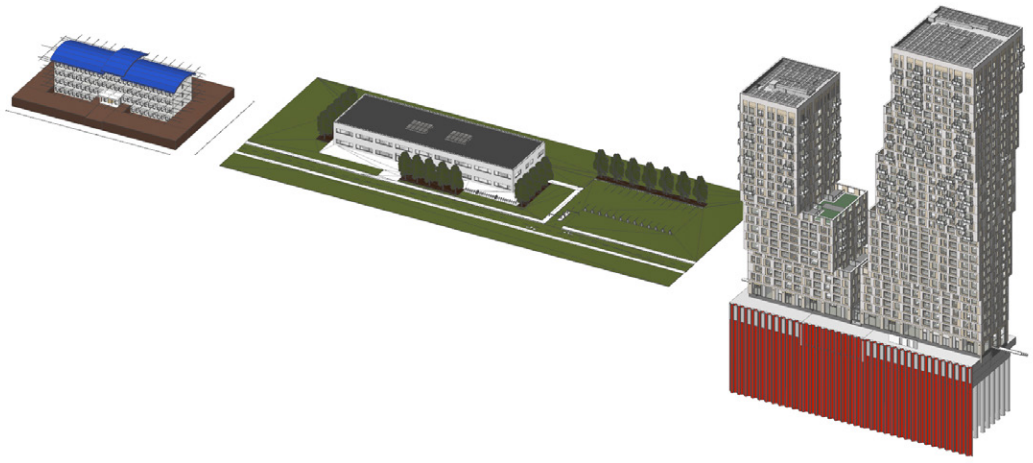


Figure 4.1.: An in scale isometric visual comparison of three test models from the three different complexity categories. From left to right: the `Institute-Var-2` model (simple), the `DigitalHub` model (moderate), and the `Boompjes model` (complex). See Appendix B, for a visual of every model that has been used.

Model name (Simplified)	Source	Overall complexity	# files	# objects	Structural complexity	Faulty data occurance
FZK-Haus	IBPSA repository	simple	1	119	simple	low
Institute-Var-2	IBPSA repository	simple	1	1183	simple	low
Smiley-West-10	IBPSA repository	simple	1	1252	simple	low
RAC-sample-project	AutoDesk Revit	moderate	1	481	moderate	moderate
DigitalHub	Aachen University repository	moderate	4	5041	moderate	moderate
ON4 The Building	self made model	moderate	1	1414	complex	high
CUVO Building	supplied by supervisor	complex	1	1750	moderate	high
Witte_de_Withstraat	supplied by supervisor	complex	1	1510	complex	moderate
Projekt Golden Nugget	AutoDesk Revit	complex	1	6138	complex	high
Boompjes	Manucipality of Rotterdam	complex	3	51307	very complex	moderate

Table 4.2.: Summary of the used models to develop and test the software tool. The final column covering the faulty data covers the occurrence of both faulty data and non-standard use. The simple box-like models have been excluded from this list.

4.2. Software tool

The methodology that was described in Chapter 3, is implemented in an open source software tool[1] that is written in C++. IFC files are opened, parsed, and edited with the help of the open source IfcOpenShell library[2]. This library resolves most of the challenges present when processing an IFC file. This library uses the Open CASCADE library[3] to transform the implicit geometry stored in the IFC files to explicit geometry on which computations can be done more easily. The Open CASCADE library is also used outside of the IfcOpenShell context to ease other geometric operations. The functionality of both libraries is supported and extended by the boost library. Like Open CASCADE, this library is used to ease geometric operations. Primarily, the spatial indexing is done with the help of the boost library.

The goal of the developed methods, and the software tool in which they are implemented, is to function on IFC models that are used in practice. These IFC models are, as described earlier, often comprised out of multiple IFC files representing different disciplines, e.g. construction, facade and interior models. This means that the tool will have to be able to process one or multiple IFC files as if it is one model. The files can not be read and blindly be loaded into memory. By doing this it will lose its connection to the original file. It is considered important to preserve this connection because it is desirable to be able to store the updated data in the original file structuring instead of one singular merged output file.

This is why the helper class is introduced. For every new IFC file that is loaded a new helper is created. The helper stores the data from a single file. Added to this, the helper is a collection of extra information and functions that eases the processing of the IFC data. These helpers are collected into a cluster, which in turn is also a collection of extra information and functions that helps processing. At every process all the helpers in the cluster are looped through linearly. This keeps the data separated while still being able to evaluate the data as one singular model. This enables us to create new objects and data based on all the objects while still knowing which file had which object included. The storing of the data in the correct file will now be an easy and quick process. The data in the model will also be spatially indexed. To guarantee separation between the files, every helper will have its own unique spatial indexes. Every unique file has four different spatial indexes that are used for specific tasks.

The tool is not a blind executable but has a simplistic console interface. Through the interface, the user is given information and is able to select and skip certain processes. The processes related to the detection of storeys, described in Section 3.2, can be ran in isolation. The user is, able to monitor the results of the storey detection and compare it to the found storeys in the IFC file. This allows the user to make an educated choice between the two. The sorting of the objects can be skipped completely if only the room and/or apartment related processes are required. Each of these subsequent processes, aside from the *IfcSpace* creation process, is made to function on both the newly created *IfcSpace* objects and the existing *IfcSpace* objects. This allows the tool to output and improve the data even when the *IfcSpace* creation fails or is unneeded. Most of these processes can be run in isolation as well. For example when the original IFC files only have to have topologic relationships added to the existing *IfcSpace* objects, that is possible.

[1] The tool is available from: https://github.com/jaspervdv/IFC_ApartmentDetection
[2] Available from http://ifcopenshell.org/
[3] Available from https://dev.opencascade.org/

Although the methods clearly described the room bounding objects, it was considered of importance to give the users extra flexibility. To facilitate this flexibility the tool gives the user the ability to use different objects for this process. This is split up in four options:

- The default room bounding objects
- The default room bounding objects with *IfcBuildingElementProxy* objects added
- The default room bounding objects with a custom selection of other objects added
- A complete custom selection of objects

It is important to be aware that only the default room bounding objects have been implemented extensively. The new types of objects that an alternative option can introduce do not go through any special processing or handling. This means that the alternative objects could result in unexpected results or errors.

Similarly the user is also given more control over the apartment creation rules. Although the methods were clearly described the created rules the user is given an option to change those. The rules that can be changed are:

- Minimal apartment room count
- Minimal apartment area size
- Minimal connections needed to be considered splitting point

These alternative options have not been addressed in the rest of the report. The options are present but not part of the core research.

4.3. Visualisation of the data

To help developing the tool and test the methods, Rhino3D and grasshopper were utilized. The ease of visual programming augmented with c# code that is available in this package made it very easy to visualize rough and intermediate output from the tool (point coordinates or simple geometry writing). This visual check made it easy to directly monitor the output and correct issues in the code or the developed method before later stages were reached where the tool could create formatted IFC output. The most important intermediate checking mechanism was the visualization of the rough room shapes. This was done by outputting the corner points of each non-intersecting voxel and their room number. With a Rhino/-Grasshopper script, this was transformed into visuals, see Figure 4.2 and 4.3.

The apartment connectivity data is written to a file that enables us to display it in Rhino 3D with the help of the Syntactic grasshopper plugin[4]. Syntactic is meant to be a set of tools that bring space syntax theory into computational design workflows [Nourian, 2016] but it can also be used to visualize the graph data. This is done by slightly altering the expected input to bend the Syntactic output to a suitable visual. Syntactic usually needs Rhino3D points (representing rooms), edges (representing connections), a list of strings (representing the room names), and a list of doubles (to represent the room areas) to construct a valid graph. The output of the C++ tool is, aside from an updated IFC file, a .txt file that has the graph data included. This graph file has the four categories of data supplied. For every room

[4] Available from https://www.grasshopper3d.com/group/space-syntax

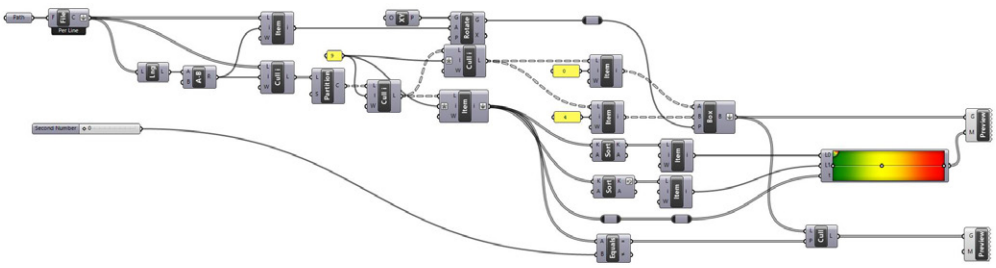
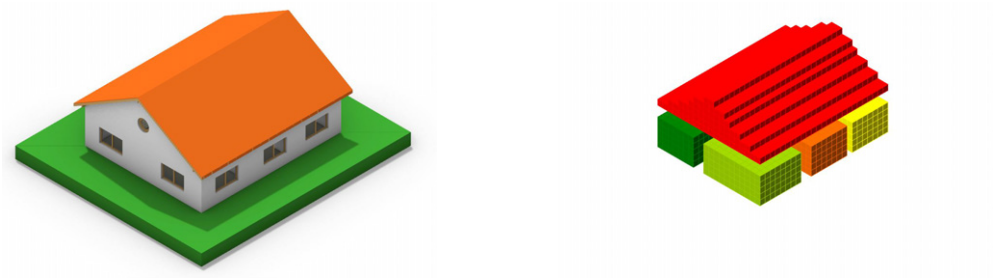


Figure 4.2.: The grasshopper algorithm that is used to display the rough room data.



(a) (b)

Figure 4.3.: The rough room output of the grasshopper script. (a) is a visual of the FZK-Haus model. (b) are the created rough room shapes that the script visualized.

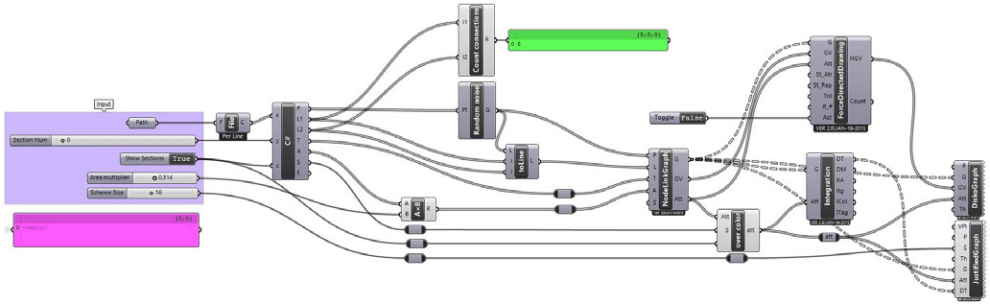


Figure 4.4.: The grasshopper algorithm that is used to display the graph data.

an unique generic point in 2D space is made. Every connection between rooms is translated as edges between these points. The room names and areas are easily copied from their real *IfcSpace* counterpart. With a simple C++ script in grasshopper, this data can be parsed into the building blocks required to let Syntactic function, see Figure 4.4.

A fifth data stream is added to the graph .txt file that signifies the section or apartment the room is part of. The custom created part of the grasshopper script can filter out the data of the other categories based on this section or apartment data. This allows display of the entire building graph with the apartments colored differently and also display all the sub-sections or apartments isolated.

To display the IFC models, two different viewers were used: BIMvision[5] and BIMcollab zoom[6]. BIMvision has an elaborate set of tools allowing to measure and count objects. In addition, BIMvision also notifies the user clearly when objects were present in the file but not drawable by the tool. The price for this is that it struggles to display large files and is unable to have more than two files opened in one session. This prevented the bigger and more complex files to be viewed as one model. BIMcollab zoom was able to process and display these files more easily and was thus used to display the complex models.

[5] Available from https://bimvision.eu/download/
[6] available from https://cutt.ly/vGtRqfy

5. Results & discussion

This chapter is split in five sections. Each of these sections will cover the results of a process that can be found in the described methods[1]. Section 5.1 and 5.2 cover the results of the storey elevation detection and the sorting of the objects, see Section 3.2 for the related methods. This is followed by Section 5.3 where the detection and reconstruction of room data is covered, the method of this process is described in Section 3.3. The final section of this chapter, Section 5.4, covers the detection and reconstruction of apartments, the method of this process is described in Section 3.4.

Every section in this chapter is split in three parts. The first part of every section gives an overview of the results. This part covers the actual results and the way these results are collected. This is followed by a small discussion, which covers an interpretation and explanation of the prior shown results. The closing part of every section will cover the limitations of the method and/or the software tool. Here the causes of, and potential solutions for, unexpected and/or undesirable results are covered.

Every result is based on a uniform voxelgrid of 44 x 44 cm. The tables will have three special values. Values inside of brackets have a clarification in the text. Question marks "?" mean the approximation is too vague to be done accurately, this often also has a clarification in the text. The hyphen "-" means that the code failed to execute the process completely resulting in no output. Some columns have been colored. These columns are displaying the performance of the tool per model. See table 5.1 for the meaning of the used colors.

[1] a subset of the processed files are available from: https://github.com/jaspervdv/IFC_ApartmentDetection/tree/main/models/exports

Indication meaning	Indication color
perfect/reliable result (no errors are present)	
decent result (small amount of errors are present)	
undesirable result (medium amount of errors are present)	
undesirable result (large amount of errors are present)	

Table 5.1.: The indication colors used in the results and their meaning. The indication/score is relative, based on the amount of errors in relation of size of the object population. The two shades per indication were used to easily distinguish between rows, their meaning is identical.

5.1. Detection of storey elevations

5.1.1. Overview of the results

The results of the detected storey elevations can be challenging to evaluate in an objective manner. The deemed most accurate and objective method is manually evaluating the height of every major floor and roofing (group) and compare it to the storey elevation found by the software. This is to an extent manually executing the described method in Section 3.2.1. A major floor in this context is a floor that has a top face area of at least $10m^2$ and has at least one room placed on it[2]. Of every major floor, the top face height is taken as the approximated storey height. For the roofing, the elevation at the base of roofing with a xy-area of at least $10m^2$ is taken as a storey elevation. The manual evaluation of the correctness of the floors placed at the same storey (the elevation merging) is done on a sensible case-by-case basis. The result of this evaluation can be found in Tables 5.2 and 5.3.

Table 5.2 shows the storey counts from different sources per model. The number of expected storeys are manually determined while the number of approximated elevations are the ones outputted by the software tool. For the manual determination of the storey elevations in multi-file models, the construction model is evaluated only. This is exactly the same as the software tool does. The final column shows the amount of storeys in the original IFC file. This column only shows if the outcome corresponds with the original storeys and should not be considered as a marker for accuracy or success of the process. It is only an indicator if the results do comply with the original model. The values in the original model could however be incorrect or established with a different reasoning.

Table 5.3 shows a summary of the approximated storey elevations. The first column shows the manually approximated storey count. The second column shows the amount of storey elevations that comply with the elevations that have been manually approximated. The final column show the amount of approximated storey elevations by the tool that comply with the storey elevations stored in the original IFC model.

A complete overview of the original storey elevations, the manually determined storey elevations, and the the software approximated storey elevations can be found in Appendix C. This appendix includes all the storeys of the evaluated models.

The forcing of the storey elevations upon the other files (the non-structural files of a model) that comprise the model is checked by comparing the storey elevations of all files before and after processing. This is only done for the `DigitalHub` and `Boompjes` model because these models are the only models that are constructed out of multiple files. The results of this can be seen in Table 5.4.

5.1.2. Discussion

Table 5.2 shows that the tool often approximates the same amount of storeys as expected. It also shows that the more complex a model is, the bigger the chance that deviations from this expected amount can occur. Especially the models that have, partially, complex flooring structures show these deviations. These deviations are presumed to be due to different ways of elevation merging. The manual, considered sensible, approximation takes the spatial

[2]This required presence of a room is ignored if the input model has no room objects included.

Model name	Expected # storeys	Approximated # storey elevations	Original # storeys
FZK-Haus	2	2	2
Institute-Var-2	5	5	5
Smiley-West-10	5	5	5
RAC-sample-project	6	3	6
DigitalHub	4	4	3
ON4 Building	6	7	7
CUVO Building	3	3	8
Witte_de_Withstraat	4 (?)	3	8
Projekt Golden Nugget	8	8	12
Boompjes	36 (?)	38	40

Table 5.2.: Comparison of the approximated storey count by the tool. The "?" indicates that the storey structure of the model was (partially) too complex to objectively determine the storey elevations.

Model name	Expected # storeys	# correctly approximated storey elevations	# of correspondence w/ original elevations
FZK-Haus	2	2	2
Institute-Var-2	5	5	5
Smiley-West-10	5	4	4
RAC-sample-project	6	3	0
DigitalHub	4	3	1
ON4 Building	6	6	6
CUVO Building	3	3	1
Witte_de_Withstraat	4 (?)	3	0
Projekt Golden Nugget	8	5	0
Boompjes	36 (?)	37	2

Table 5.3.: Summary of the approximated storey elevations by the tool. The "?" indicates that the storey structure of the model was (partially) too complex to objectively determine the storey elevations.

Model name	# of non-structural files	Correct # of forced elevations
DigitalHub	3	3
Boompjes	2	2

Table 5.4.: Summary of the storey elevation forcing upon non-structural models.

relationship between floors into account while the automated approximation only looks at the floors' z-values. These complex models often have a smaller amount of approximated storey elevations than the original IFC file had included. The most extreme case of this is the `Witte_de_Withstraat` model. However there are occasions where new storeys are created. This is always the case in models that do not have a dedicated roof storey. An example of this can be seen in the top storey of `DigitalHub` model, see Figure 5.2.

Table 5.3 shows us that the computed storey elevations often complies more with the expected values in the simple models. Of the simple models, the `Smiley-West-10` model is the only model of the three where an error occurs. This error occurs at the top of the building. This is the storey where only the roofing structure is located in the model. The more complex models show a worse performance. The performance is worse than is the case in Table 5.2. However, it should be noted that, in complex models, it is also more challenging to manually determine expected storey elevations. This creates a more subjective component to the evaluation and comparison.

If the computed storey elevations are compared to the original storey elevations in a more complex building, we can see some phenomena occur that are worth highlighting. The first one is that even when a structure is too challenging to manually approximate the storey elevations in an accurate way, the software is still able to make an educated guess. However, these educated guesses do deviate both in the number of storeys and their values from both the original model and the sparse amount of manually approximated elevations. An example of where the tool struggles can be seen in Figure 5.1 which shows the lower floors of the `Boompjes` construction model. However, the struggles of the tool do not stop at the lower floors of the model. In the higher, and more simple, part of the building, both the expected and the tool-approximated elevations persistently deviate by 0.08m from the storey elevations that are stored in the original file. Due to this area being fairly simple, the expected elevations can be considered fairly objective. This deviation is caused due to the storey elevations stored in the construction file being based on the interior file of the building. This interior file has a screed floor that has a thickness of 8 cm creating this deviation. The complete unsummarized evaluation for the `Boompjes` model can be found in Table C.10 and Table E.1 in the Appendix.

Table 5.4 shows that the forcing of the storey elevations upon the non-structural files has a 100% success rate on the tested models. This forcing of the storey elevations from the structural to the non-structural files also resolves some errors that can occur in the model. The expected result was that all the storey counts and their elevations across the files are the same. This would make the models more unified and some files more easy to process. However, the `Boompjes` model shows the added advantage that when a storey (elevation) is missing from a file, this can be resolved by the storey forcing. The file representing interior of the `Boompjes` model has the 12^{th} storey missing. This caused the 11^{th} storey to incorrectly represent both the 11^{th} and the 12^{th}. By forcing the found-storey elevations upon this file, this error is resolved. This can be seen in Figure 5.3.

5.1.3. Limitations

The performance of the storey elevation detection is good in isolated cases. These cases are often simple models that have a very low amount of overlapping storeys. The moment models become more complex, inaccuracies occur in the results from the process. All the detected

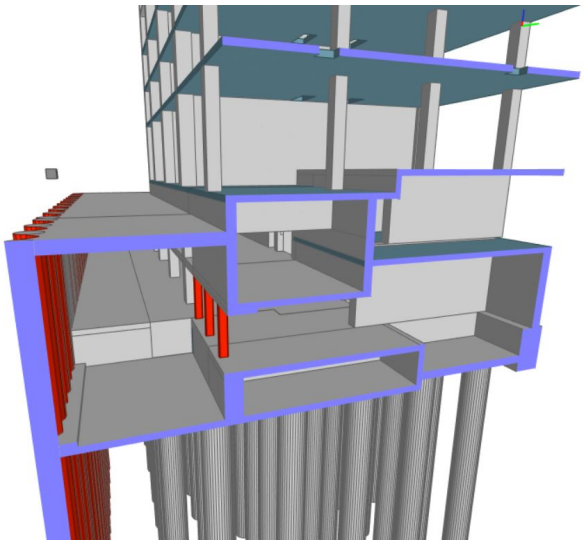


Figure 5.1.: Section of the lower storeys of the `Boompjes` model. The complexity of the flooring structure can be clearly seen.

Figure 5.2.: Example of the roof storey that is added by the tool. The gray storey elevations were already present in the original IFC file of the `DigitalHub` model. It can be seen that the roofing structure of the model has no dedicated storey elevation. This elevation, highlighted in blue, is added by the tool.

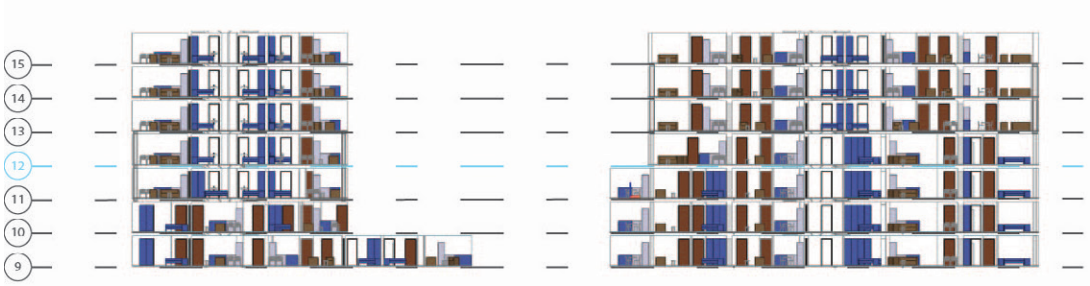


Figure 5.3.: The added storey to the `Boompjes` interior file highlighted in blue. The marked in gray other storeys were already present in the file and correctly numbered, only the 12th storey was missing.

errors (and, when known, their causes) of the storey elevation detection that occurred will be covered and explained in this subsection in order of model complexity.

The `Smiley-West-10` model displayed the first error, the incorrect detection of the roof elevation. In the current state of the tool, it does not process normal *IfcSlab* object or *IfcRoof* and *IfcSlab* objects being part of the roofing structure in different ways. This means that for the storey elevation detection, all the elevations are approximated based on the lowest point of the top face of an *IfcRoof* or *IfcSlab* object. If the object is part of the roofing structure however, its elevation should be taken from the lowest point of the lowest face of the object. This results in the 20 cm difference between the expected and approximated value of storey 3, see Table C.3. This could be resolved by creating a function that checks if the objects are part of the roof or the flooring. The issue can pass by undetected if the roofing structure storey is defined by a closely located floor. This is for the case of the `Institute-Var-2` model, although the elevation of the roofing structure is incorrectly determined this elevation is merged with the one determined for the floor that is located closely, see Figure 5.4. This almost accidentally results in an accurate result.

The tool struggles with complex flooring structures. It was briefly mentioned in the discussion that the manual, considered sensible, approximation takes the spatial relationship between floors into account while the automated process only considers the z-values of the floors. This is due to the core functionality of the process being rather simplistic, and is only able to consider the z-domain. The xy-domain of the building, and its floors, is currently ignored. This results in storey elevations being merged into one, even if the floors that they are based upon are spatially separated from each other. The issue could be resolved by creating an xy-domain for every storey group while determining the grouping of *IfcRoof* and *IfcSlab* objects. This xy-domain could be used as an extra parameter to determine if storeys should be merged into one or not. This xy-domain should also be saved to improve the subsequent sorting process. The introduction of such a domain could potentially reduce the inaccuracies not only in the `RAC-sample-project` model but also in the complex basement area of the `Boompjes` model and the separate buildings of the `Witte_de_Withstraat`.

The `Witte_de_Withstraat` model is however a unique case. The model seems to be comprised out of three different buildings, see Figure 5.5. Each of these have their own storey elevation structure. This is a certain logic that the tool cannot recognize and will be extremely challenging to add. In this particular case, the xy-domain would improve the storey elevation detection and might even create the correct storeys per building without recognizing this logic. This is however presumed to be an outlier due to none of the three buildings having

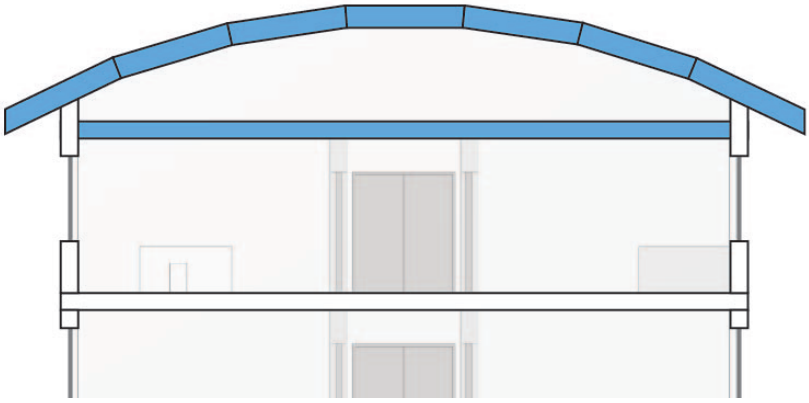


Figure 5.4.: A section of the `Institute-Var-2` model that shows the situation where a closely related floor can hide the incorrect roof detection. The highlighted in blue roofing and flooring structure will have the same storey elevation due to the software merging closely located storey elevations.

storey defining slabs that are located on the same z-height while being located at a close xy-location. If this would have been the case they would have been merged into one. A potential solution to this would be adding a functionality that allows the tool to recognize certain tags or groups as separate building parts. This would enable the user to use certain settings to separate objects, for example the use of *IfcPropertySingleValue* or another label. This is however intangible geometry, thus it would be wise to clearly notify the user that this is an option, but might not be reliable due to the software being unable to check the validity of these labels.

The method and the created tool have been based around the *IfcSlab* and *IfcRoof* objects that are stored in the construction file of a model. The `Boompjes` model showed that in a multi-file model the storey elevations can also be based on another file. In the case of the `Boompjes` model the elevations stored in the IFC file were, partially, based on the on the interior file. The tool and method did not incorporate this. However, exactly the same method can be applied on the interior file instead of the construction file. This is done by selecting the interior file as construction file when prompted by the tool. Doing this yields more compliant results in the upper section of the model with the original storey elevations. The interior file does not cover all the storeys, and thus the storey elevations in the basement levels are incorrectly or not approximated in this case. An added option that would enable the evaluation of all the file instead of a single file would however not directly resolve this. Due to the way that small z-differences between floors are merged to the lowest z-value, it will still malfunction, see Figure 5.6. Therefore, to function properly, not only this option has to be added, but the z-value merging has to be revised as well.

Almost vertical *IfcSlab* or *IfcRoof* objects are processed in the same manner as primarily horizontal *IfcSlab* or *IfcRoof* objects. This is due to the expectations that *IfcSlab* or *IfcRoof* objects only occur primarily horizontal. However, the `Projekt Golden Nugget` model shows that this does not have to be the case, see Figure 5.7. In this example, it creates an extra storey in the upper area of the model that is unexpected and considered to be incorrect. An improvement to the code would be implementing an adjusted process for primarily vertical *IfcSlab*

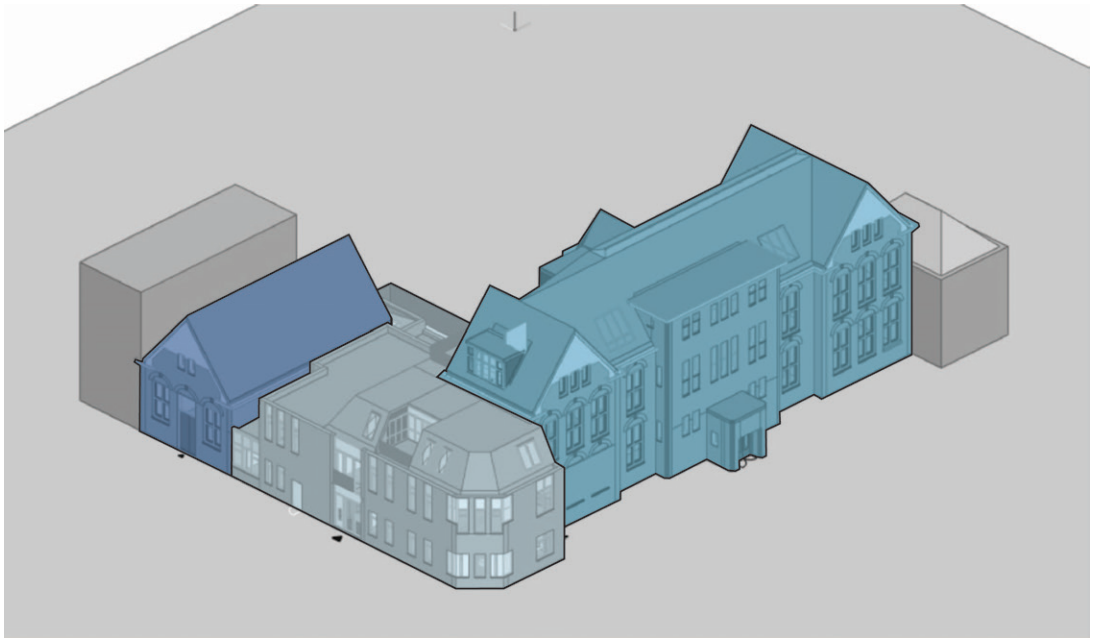


Figure 5.5.: The multi-structure nature of the `Witte_de_Withstraat` model. The different sub-buildings with a different storey structure are highlighted in different shades of blue.

Figure 5.6.: The situation in the `Boompjes` model that creates an undesired z-value extraction. The flooring is constructed out of a structural floor (dark gray) and a screed floor (light gray). Even when both are evaluated by the tool, the tool will merge the z-values found in these floors to the lowest of the tool, resulting in the z-value still being 8 cm lower than expected.

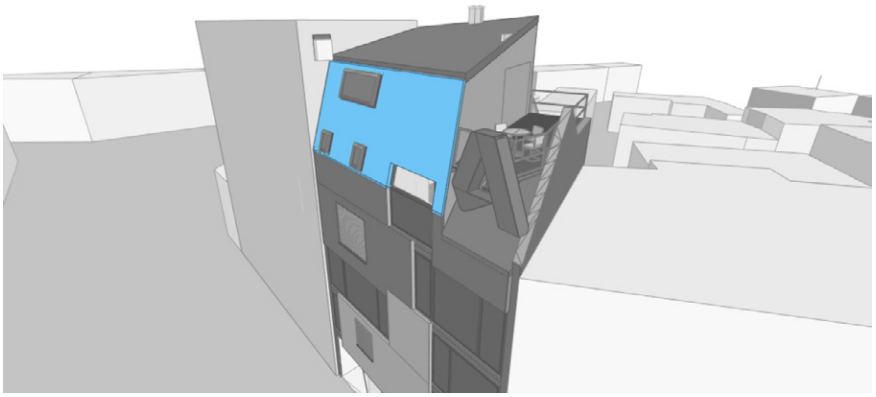


Figure 5.7.: The top area of the `Projekt Golden Nugget` model. The highlighted roof object is an *IfcRoof* object that is primarily vertical. This *IfcRoof* object causes the tool to create an extra storey elevation at the base of it, which is incorrect

and *IfcRoof* objects.

5.2. Object to storey matching

5.2.1. Overview of the results

The accuracy of the object classification can be tested by manually counting the amount of objects that are not classified on the storey at which they are (primarily) located. Doing this results in Table 5.5 and Table 5.6. Table 5.5 covers the accuracy of the sorting process when the storeys that are approximated by the tool are used. Table 5.6 covers the accuracy when the original storeys (and their elevations) that are stored in the IFC file are used. Both tables have the same columns. The total number of objects is the counted number by BIMvision and not manually counted. The original misplaced objects are the manually counted objects that show a clear incorrectly assigned storey before processing. The processed misplaced objects are the manually counted objects that show a clear incorrectly assigned storey after processing. The final column, the processed number unclassified objects, shows the amount of objects that have not been sorted into a storey. These objects can not be drawn in a 3D viewer due to this. This number is acquired by BIMvision. Exploded elevation views based on the labelling of the objects can be found in Appendix D.

5.2.2. discussion

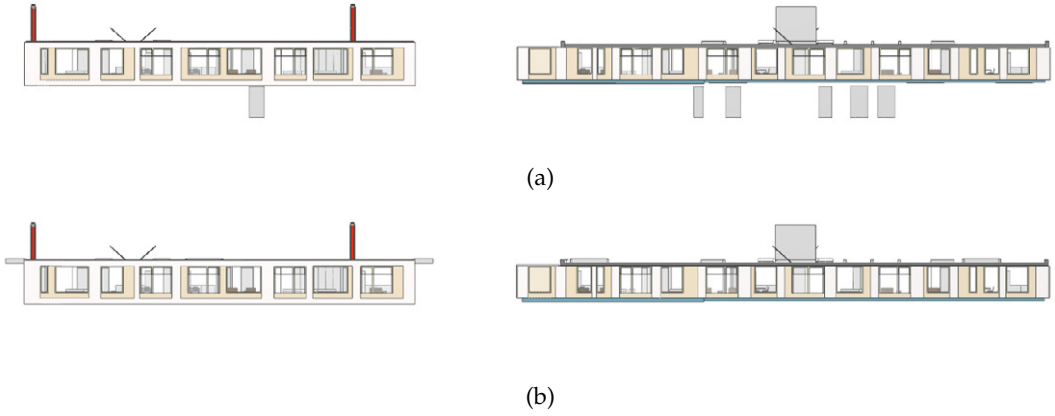
Table 5.5 shows that, when processing the simple models, the software can correctly assign all objects. Even in complex models, the processed files show almost all a significant improvement in the labelling of objects. Some persistent issues are resolved by the tool, see Figure 5.8. However, the more complex models show that errors do still occur. A subset of these errors are created by incorrectly processing certain types of objects. For example, both the

Model name	Total # objects	Original # misplaced objects	Processed # misplaced objects	Processed # unclassified objects
FZK-Haus	199	0	0	0
Institute-Var-2	1183	0	0	0
Smiley-West-10	1252	0	0	0
RAC-sample-project	481	8	5	3
DigitalHub	5041	13	1	6
ON4 Building	1414	3	7	0
CUVO Building	1750	199	0	0
Witte_de_Withstraat	1510	241	7	8
Projekt Golden Nugget	6138	(?)	7	1
Boompjes	51307	923 (120)	379	27

Table 5.5.: The comparison of misplaced objects before and after processing. Note that only extremely misplaced objects have been counted.

Model name	Total # objects	Original # misplaced objects	Processed # misplaced objects	Processed # unclassified objects
FZK-Haus	199	0	0	0
Institute-Var-2	1183	0	0	0
Smiley-West-10	1252	0	0	0
RAC-sample-project	481	8	15	0
DigitalHub	5041	13	41	0
ON4 Building	1414	3	3	0
CUVO Building	1750	199	0	0
Witte_de_Withstraat	1510	241	466	8
Projekt Golden Nugget	6138	(?)	(?)	(?)
Boompjes	51307	923 (120)	929	0

Table 5.6.: The comparison of misplaced objects before and after processing the original storey elevations. Note that only extremely misplaced objects have been counted.



(a)

(b)

Figure 5.8.: Mislabelled objects being corrected in the `Boompjes` model. (a) shows an elevation view of all the objects being labelled as part of the 5th floor. It can be seen that underneath the 5th storey some objects are still labelled as part of the storey while they are actually part of the storey below it. (b) shows the same selection of labelled object but now based on the processed model. It can be clearly seen that the mislabelled objects that were present have been corrected.

`ON4 Building` model and the `Boompjes` model show similar behavior where all the balconies in the model are (partially) incorrectly classified, see Figure 5.9. If the objects representing balconies would be removed from this summary, the processed number of misplaced objects in the `ON4 Building` model would be 3 and in the `Boompjes` model 146.

As has been mentioned in Section 5.1, the lower storey structure of the `Boompjes` model is very complex. Not only the storey detection process, but also this sorting process struggles with these complexities. The results in Table 5.5 are drawing a skewed picture of the resulting output of the `Boompjes` model. We can split the model in two parts, from the -4th floor to the ground floor and from the 1st floor to the top floor. The top section does not have a lot of complexities and only includes a total of 3 incorrectly classified objects in the construction and interior model. The other 143 misplaced objects are all located in the lowest 5 storeys. This is excluding the 233 incorrectly assigned balconies.

The Project Golden Nugget presents a challenge when trying to evaluate the tool's performance. The number of storeys and the way they are populated is so complex and inaccurate that it is challenging to categorise the objects in a reasonably objective way. This is why this model does not have a number of original misplaced objects. However, it can be said that the processed model is a lot more structured than the original one. The degree of improvement is not quantifiable and thus the model is not further considered in this section.

The results in Table 5.5 do have a slight bias. This is because the original misplaced object count is based on the storey elevations stored in the original IFC file while the processed misplaced objects count is based on the storey elevations that have been created by the software tool. Therefore, they are tested against different conditions. The tool is also able to use the original storey elevations to base the sorting process upon. This does still force the elevations of the construction file upon the other files but it does not create new storeys or elevations. The results can be seen in Table 5.6. The table shows that, when using the original elevations,

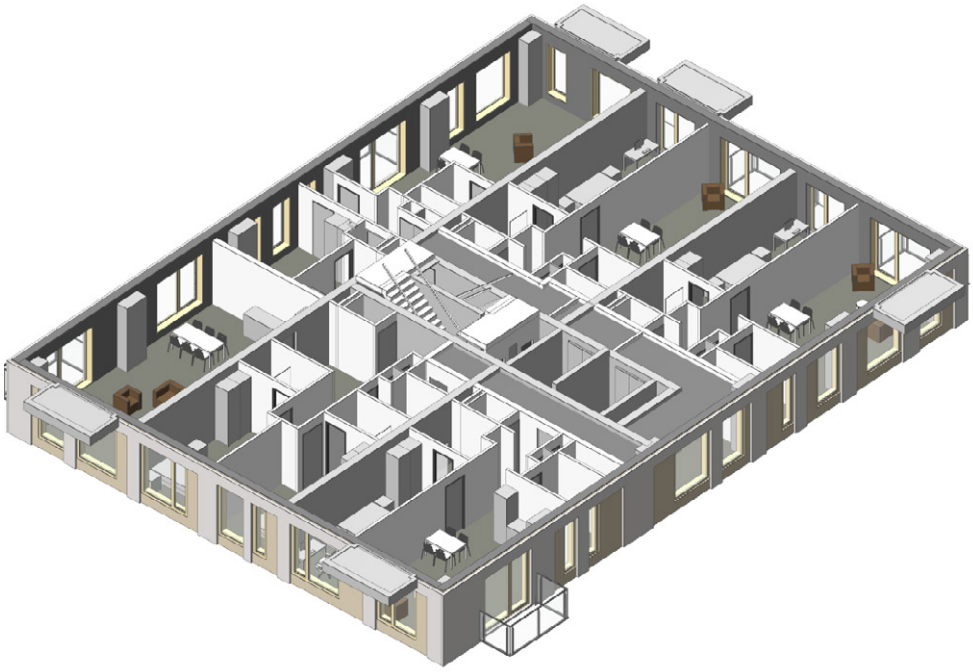


Figure 5.9.: Fragment of the `Boompjes` model showing the incorrect assignment of the balconies. The floors of the balconies are incorrectly assigned to a floor lower than they are actually located.

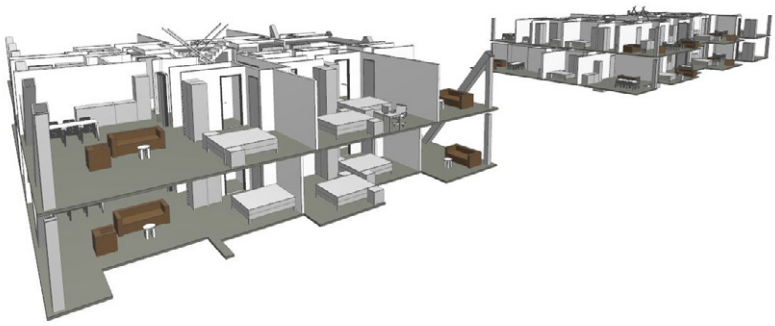


Figure 5.10.: The merged 11th and 12th storey of the `Boompjes` interior model isolated from the rest of the model.

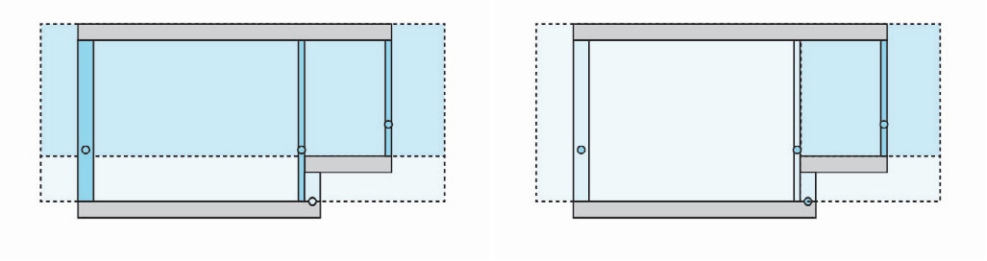
a higher number of misplaced objects are present in the processed file. Across all the models, there is no improvement using the original storey elevations.

The extreme value of misplaced objects in the original model of the Boompjes draws attention. This is due to the 11th floor of the interior model. As described in Section 5.2, the 12th storey is missing in the interior file. This results in the 11th and 12th storey being merged into the 11th storey, see Figure 5.10. The 12th storey is absent from the interior model and the next storey is the 13th. In theory these objects are thus not incorrectly sorted. They fall between the bottom of the 11th and the bottom of the 13th storey. However, in practice, this is presumably undesirable. After processing by the tool, the objects of these storeys are sorted into the correct 11th and 12th storey. This occurs both when the newly computed storey elevations and when the original storey elevations are used. Both inhered the storey elevations from the construction model that has the 12th storey present.

The final columns of Table 5.5 and Table 5.6 show the amount of unclassified objects that are present in the files after processing. Unclassified objects are not displayed when the model is opened in a 3D viewer but are instead (completely) ignored. This is undesirable because it essentially means that objects/data is being lost from the model. Table 5.6 shows that, when the original storeys are used, all objects are successfully classified and no unclassified objects are stored into the processed file. The only exception to this is the `Witte_de_Withstraat`.

5.2.3. Limitations

Similar to the detection of the storey elevations, the sorting process performs well in the simple models. When more complexity is present in the processed model, more errors occur. A major element at play in the errors is the already mentioned absence of an xy-domain in the method. Objects that have their base point located higher off the ground (e.g. walls) or objects that are located higher off the ground themselves (e.g. hanging lights or ceilings) often get incorrectly assigned in models that have storeys that are partially overlapping in the z-direction, see Figure 5.11. This occurs often in the lower section `Boompjes` model. The solution



(a) (b)

Figure 5.11.: Schematic section of the `Boompjes` basement showing the solution that the xy-domain integration could yield. The walls show their basepoints and the colors of the walls reflect to what domain they are assigned (light blue = lower domain, blue = upper domain). (a) shows the current situation where a floor creates a domain that covers the whole xy-range of the building, resulting in walls being incorrectly assigned to the wrong storey when half elevations are present. (b) shows that when the xy-domain is integrated, the floors do not create domains covering the whole xy-range of a building anymore, resulting in more accurate assignment of objects.

would be creating a complete xyz-domain for every storey. This could help determine the potential storey where the object could be situated.

A more complex "score" system could introduce more accuracy as well. This score system would give every storey a score based on how likely it would be for an object to be at the storey. This would enable multiple evaluations to be done when finding the correct storey location. In the current version of the method and tool, the location of the base point of an object is the only evaluation that is done to assign an object to a storey. If an object has a base point that is, for example, slightly lower than the storey where it should be located, it could be incorrectly assigned to a storey lower. This occurs in the `Boompjes` model with the balconies. With a scoring system, more evaluations with each their own weight could be incorporated. This could allow objects to be sorted more precisely. Another example of incorrectly assigned objects are walls that span multiple storeys. Due to their base point being higher off the ground, they are often assigned at other storeys than they are actually placed, see Figure 5.12.

When the newly approximated storey elevations are used, occasionally an object is not classified. These unclassified objects are often objects that are flagged as having no representation while also not aggregating other objects that might have a representation. Objects that have no representation can not be processed by the tool. The flag that signifies the absence of representation does however not mean that the objects have no actual representation. For example, the unclassified objects in the `DigitalHub` objects are all *IfcDoor* objects that are present and visible in a 3D viewer prior to processing. The fact that these viewers can find these objects and display them means that a representation should be available somewhere in the file. Currently the tool has no functionality for dealing with these objects. However, the inability to classify the objects occurs with less than 1% of the objects present in tested models.

Figure 5.12.: The `Witte_de_Withstraat` model with a wall highlighted that is labelled to be on the first floor instead of the ground floor. This is caused by picking the basepoint for the elevation of a wall at a third of its height. This is very high for walls spanning multiple floors.

5.3. Detection and reconstruction of room data

5.3.1. Overview of the results

The classification of the existing *IfcSpace* objects into "element" and "complex" objects is evaluated by manually executing the classification of the existing *IfcSpace* objects and comparing that to the classification that was done by the tool. If an *IfcSpace* object encapsulates another *IfcSpace* object, it should be classified "Complex". From the test models that were evaluated, only the `Witte_de_Withstraat` and the `Boompjes` model have *IfcSpace* objects present that encapsulate or overlap with other *IfcSpace* objects. The `Boompjes` model is, however, a very large model that presents a large amount of different *IfcSpace* objects and object interactions. This allows us to evaluate the accuracy of the process on a large and diverse set of spaces even though only a small set of suitable models is available. To establish the success of the process four floors of the `Boompjes` model are evaluated. The results of the `Boompjes` model can be found in Table 5.8. An overview of the complex space detection on the other models can be seen in Table 5.7.

The evaluation of the created *IfcSpace* representation shapes is again done manually. The evaluation is done at two stages of the *IfcSpace* representation creation process. The first evaluation point establishes if the rough voxelized shape of the space can be found during the voxelization process and if this shape is correct. The second evaluation establishes if a correct shape has been constructed that completely complies with its bounding geometry. The results of these evaluations can be found in Table 5.9. Note that the total amount of spaces present in the model is not equal to the total amount of *IfcSpace* objects present in the model. The amount of spaces present in the model is the manually counted number of, enclosed by room bounding geometry, areas. It does not differ if this area is occupied by zero or multiple *IfcSpace* objects, one single enclosed area will always be counted as one space. Some of the complex room shapes that have been created by the tool can be seen in Figure 5.13. An example of the rough room output can be found in Figure 4.3.

Model name	Total # IfcSpaces	Expected # complex spaces	Approximated # complex spaces
FZK-Haus	5	0	0
Institute-Var-2	82	0	0
Smiley-West-10	140	0 (14)	7
RAC-sample-project	14	0	0
DigitalHub	64	0	0
ON4 Building	168	0	0
CUVO Building	0	0	0
Witte_de_Withstraat	57	3	9
Projekt Golden Nugget	86	0	0

Table 5.7.: The comparison of the expected number of complex spaces and the number of complex spaces approximated by the tool for a subset of the models.

Floor	Total # IfcSpaces	Expected # complex spaces	Approximated # complex spaces
-2	69	24	2
-1	68	24	20
6	238	84	42
20	197	76	49

Table 5.8.: The comparison of the expected number of complex spaces and the number of complex spaces approximated by the tool for the `Boompjes` model.

Model name	Total # spaces	# of correct rough room shapes	# correctly shaped IfcSpaces	# of solids
FZK-Haus	5	5	5	5
Institute-Var-2	82	82	82	82
Smiley-West-10	140 (126)	140	126	126
RAC-sample-project	14	14	3	3
DigitalHub	68	58	58	58
ON4 Building	168	151	0	0
CUVO Building	51 (44)	43	23	23
Witte_de_Withstraat	53	23	15	15
Projekt Golden Nugget	59	44	0	0

Table 5.9.: The summary of the accuracy of the *IfcSpace* shape recovery.

Model name	Total # replaced spaces	# of correctly copied semantic data
FZK-Haus	5	5
Institute-Var-2	82	82
Smiley-West-10	140 (126)	134
RAC-sample-project	0	0
DigitalHub	68	56
ON4 Building	0	0
CUVO Building	0	0
Witte_de_Withstraat	8	5
Projekt Golden Nugget	0	0

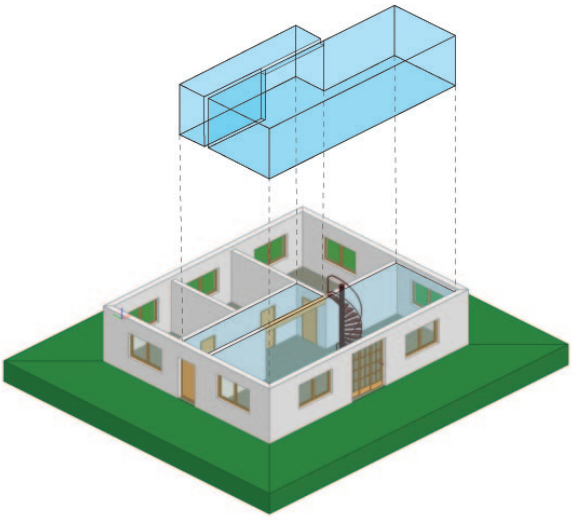
Table 5.10.: Summary of the *IfcSpace* object semantic data matching.

The copying process of the semantic data is evaluated by comparing the *IfcSpace* object's semantic data in the original IFC file to that of the replaced *IfcSpace* object(s) in the processed file. The results of this evaluation can be found in Table 5.10. For this evaluation, only new *IfcSpace* objects with a created representation are used. This means that, when an *IfcSpace* shape that is present in the original IFC file but (for any reason) not replaced by a new *IfcSpace* shape, it will not be included in this semantic evaluation as a failed copy.

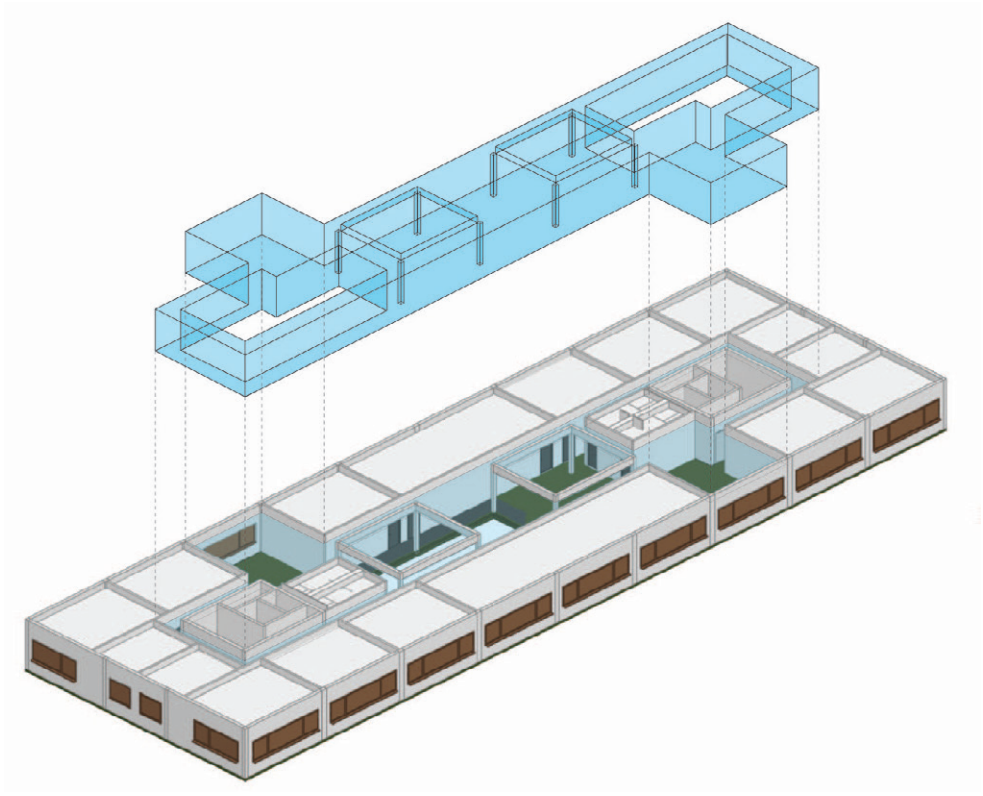
Finally, the accuracy of the topologic relationships of the *IfcSpace* objects is evaluated. BIMvision has a function that can highlight the objects that have a topologic relationship with each other. With the help of this functionality, every newly created *IfcSpace* object can have its topologic relations assessed. The results of this assessment can be found in Table 5.11. The incorrect relationships and missing relationships have been separated in this table. The incorrectly related objects are objects that are marked as related while they should not have a relationship with the *IfcSpace*. The missing related objects are objects that are marked as unrelated but should have a relationship with the *IfcSpace* object. As with the evaluation of the semantic data, the topologic relation is only evaluated on the newly created *IfcSpace* objects. The only exception are the models highlighted with an asterisk. For these models the *IfcSpace* objects from the original model are used as input. A visual example of the created topologic relations can be seen in Figure 5.14.

5.3.2. Discussion

It can be observed in Table 5.8 that the number of complex *IfcSpace* objects approximated by the tool is a lot lower than the expected number. This occurs on every storey and seems to be unrelated to the total amount of expected complex spaces. The presence of more complex *IfcSpace* objects in a file does not signify a smaller percentage of correctly classified spaces. When evaluating the objects in the 3D model, it becomes clear that often the expected to be "Complex" spaces do not completely encapsulate the "Element" spaces, see Figure 5.15. When the shapes do only partially overlap, the tool is unable to determine if it is a complex space or not and will always regard it as not complex. This creates a lower than expected count of complex spaces in the `Boompjes` model.

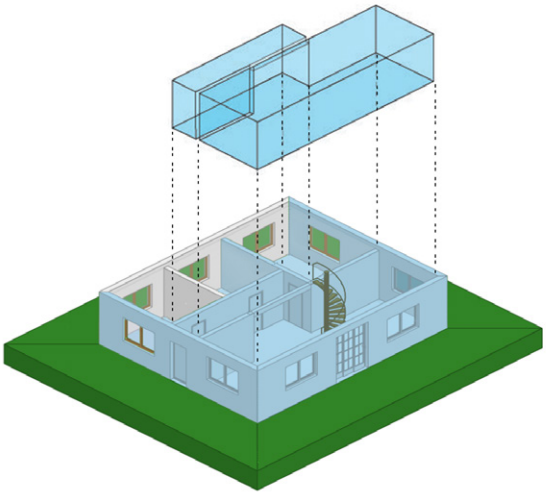


(a)

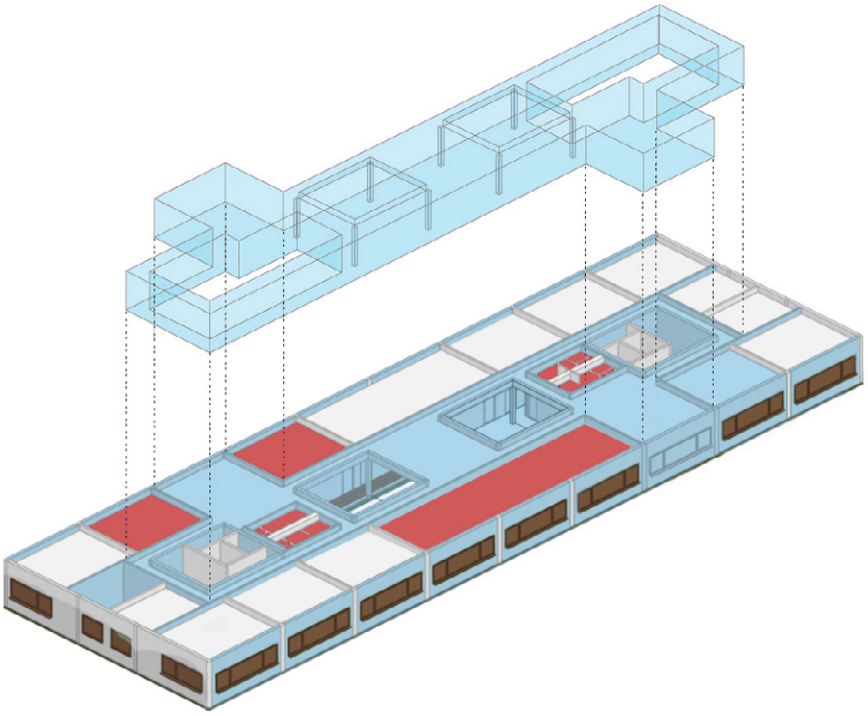


(b)

Figure 5.13.: Visual examples of *IfcSpace* shapes created by the tool. (a) the shape of the ″living″ space of the `FZK-Haus` model that is created by the tool. (b) the shape of the first floor hallway of the `DigitalHub` model that is created by the tool.



(a)

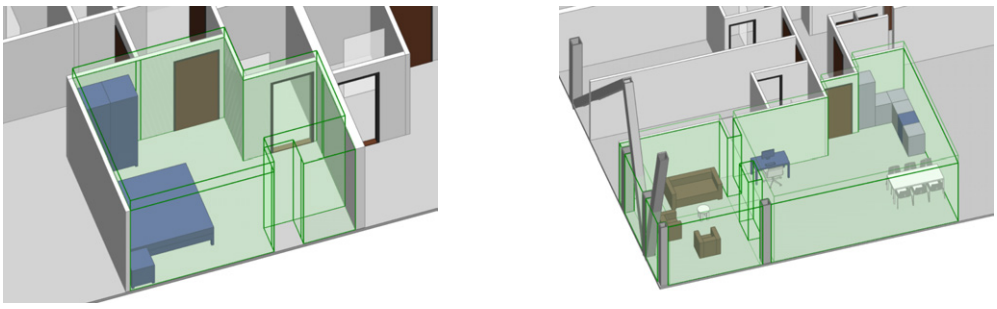


(b)

Figure 5.14.: Visual examples of objects that are found to be topologically related to an *IfcSpace*, according to the tool. The correctly related objects marked in blue and incorrectly related objects marked in red. For clarification, the *IfcSlab* resting directly on top of both the rooms in the examples have been hidden. (a)the `FZK-Haus` model. (b) the `DigitalHub` model.

Model name	Total # IfcSpaces	# IfcSpaces with incorrect related objects	# IfcSpaces with missing related objects
FZK-Haus	5	0	0
Institute-Var-2	82	0	0
Smiley-West-10	140	0	0
RAC-sample-project	3	0	0
DigitalHub	68	6	4
ON4 Building*	168	0	0
CUVO Building	23	0 (9)	0
Witte_de_Withstraat*	15	0	0
Projekt Golden Nugget*	86	83	0

Table 5.11.: Summary of the *IfcSpace* object topologic relation recovery.



(a) (b)

Figure 5.15.: Visual examples of "Complex" *IfcSpace* outliers. (a) a singular space partially encapsulated in the xy-plane but completely in the z-plane. (b) two spaces completely encapsulated in the xy-plane but not in the z-plane

Table 5.7 shows that the only false positives occur in the `Smiley-West-10` model and the `Witte_de_Withstraat` model. The false positives that occurs in the `Smiley-West-10` model are caused by the input data being faulty. The `Smiley-West-10` model has an incorrectly placed wall in 7 of its apartments which resulted in the *IfcSpace* objects' shapes representing the bathroom and the hallway on the first floor to overlap, see Figure 5.16. For every apartment where wall misplacement occurs, two different *IfcSpace* objects are placed in the identical location. This results in the software to incorrectly presume that there are 7 "Complex" *IfcSpace* objects present. The effect of this seemingly minor geometry error can be noticed in all the subsequent processes as well. In Table 5.9, it can be seen that, of the three simple models, only in the Smiley-West-Building the tool is unable to recreate accurate *IfcSpace* shapes for all the spaces. The 14 incorrectly shaped rooms that are present are incorrect due to this wall displacement, see Figure 5.20a. In Table 5.10, again the Smiley-West-Building is the only simple model where the results were not ideal due to the same reason.

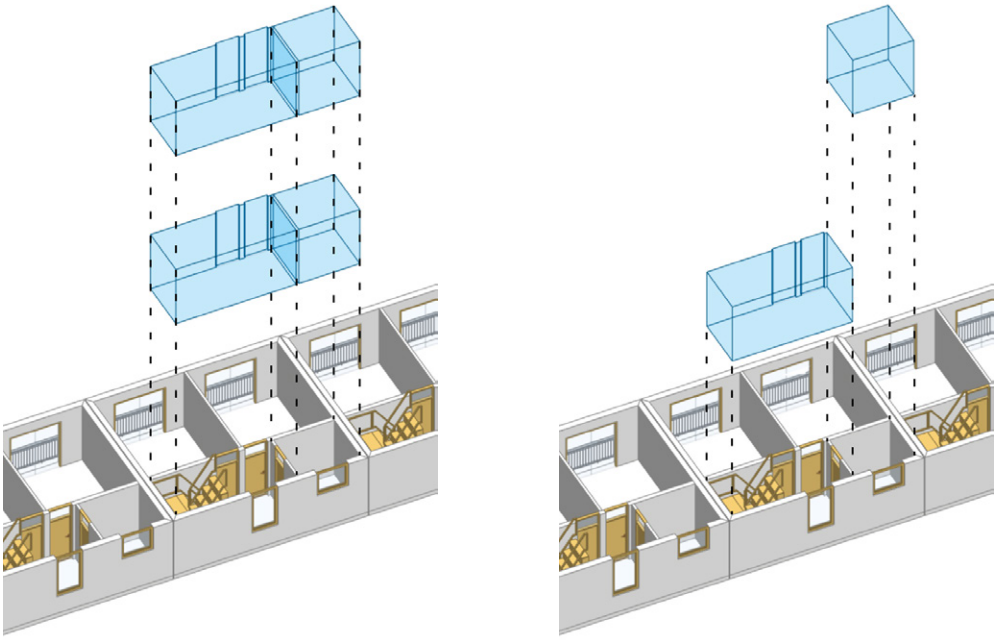
The "Complex"/"Element" classification process functions incorrectly on the `Witte_de_Withstraat` model, but this is not due to faulty data. The file has many redundant *IfcSpace* objects present for different phases. Aside from the phase (and the GUID), these *IfcSpace* objects are almost identical duplicates of each other placed at the same locations. If one of these near identical room representations is slightly bigger than the other, it causes the tool to occasionally incorrectly assign *IfcSpace* objects to "Complex". This is similar to the issue in the `Smiley-West-10` model although the root cause is different.

Table 5.9 shows that the process that creates *IfcSpace* objects performs well on the simple models. However, for complex models the performance is sub-optimal. This quality reduction ranges from a subset of spaces not being created to no spaces being created at all. The table also shows that during the (attempted) creation of the *IfcSpace* representation objects, there are more rough room shapes found than *IfcSpace* objects created. The processes executed during the voxelization are fairly simple and robust. These processes function without the need of solids or watertight closed off faces. The subsequent refinement of the shape[3] is more complex and does require these solids or watertight connected faces. The difference in the requirements of these two processes is clearly reflected in the results in this table. Of the *IfcSpace* representation shapes that have been made, all of them are valid solids.

The `DigitalHub` and `ON4 Building` models show that, although the voxelization process is more robust, it can also struggle in some complex models. This can manifest itself in different manners. The `DigitalHub` model has spaces that have been incorrectly merged into each other, see Figure 5.17. The `ON4 Building` model has missing rough room shapes in seemingly simple surrounding geometry, see Figure 5.18. The more complex a model becomes, the more exceptions to the expected growing and intersection rules can be present.

All these issues with complex models and shapes do however not mean that complex *IfcSpace* shapes cannot be created by the tool. Figure 5.13a and 5.13b show that big and/or intricate room shapes can be extracted in both simple and fairly complex models. The created room shapes successfully follow the room bounding geometry to very small detail. Interestingly, these shapes, in the case of the `DigitalHub` model, seem to be able to be non-manifold (or extremely close to be non-manifold) without the software crashing, creating inaccuracies or having issues with storing the data. Although it is positive to see that non-manifold shapes do not create issues in the tool, it might create issues in subsequent processing by other software.

[3] going from the rough room shape to the correctly shaped *IfcSpace*



(a) (b)

Figure 5.16.: The incorrect original *IfcSpace* representations in the Smiley-West-20 model. (a) is the situation in 7 of the 10 apartments where, due to an incorrectly situated wall, the *IfcSpace* representations are creeping past it. This results in the room representations occupying the same space. (b) shows the desired situation that occurs in the other three apartments. The wall correctly splits the space into two, allowing two separate *IfcSpace* representations to occupy their respective space without overlapping. The software will classify both these *IfcSpace* objects correctly as "Element".

The presented results do not only show issues related to the functioning of the tool but also some related to user errors. All the processes for the complete evaluation were executed with a voxel size of 44 x 44 centimeters. This means that, when a hallway or opening is smaller than 88 centimeters[4], the code could be unable to detect that opening. This creates an error in the toilet spaces of the `DigitalHub` model. This space has areas that are less than 88 centimeters wide and results in the space being detected as three separate spaces instead of one singular space.

Table 5.10 shows that the copying of the semantic data is functioning fairly reliably. The `DigitalHub` model is however showing issues with semantic data copying. When analyzing the processed model, it becomes clear that the majority of the vertical shafts through the building do not get any semantic data assigned to them. Of the multiple spaces that comprise one shaft, none have the correct semantic data copied in a correct manner. All these spaces wear the generic name that the tool gives it when it is unable to find a suitable semantic set.

The final table of this section, Table 5.11, shows that for the simple models the topologic relationships can be accurately recovered. The `DigitalHub` model shows some spaces that have one incorrectly related object, see Figure 5.14b for an example. For a currently unknown reason, all the incorrectly related objects are *IfcCovering* objects, except for one other occurrence. Aside from the `DigitalHub` model, only the `CUVO Building` model shows potentially incorrectly found relations. This model is created in such a way that the basement walls rest against the *IfcSpace* representation shapes of the ground floor with one edge, see Figure 5.19. These walls have a spatial relationship with these *IfcSpace* objects. However, it is questionable if they are desirable due to the relation being 1D instead of 2D.

5.3.3. Limitations

As was the case for the processes described in Section 5.1 and 5.2, the processes related to the creation and recovery of *IfcSpace* data are very accurate on the simple models. Exceptions are present when faulty data is supplied. The `Smiley-West-10` model did include some faulty data resulting in almost all the processes outputting undesirable data. This faulty data was the incorrect placement of a subset of walls. Although the misplacement was extremely minor, the effect rendered the newly created data affected by this useless. The issues were however centralized spatially around the locations of the faulty data and the output that had no relationship with the faulty data remained completely unaffected.

The errors that this particular wall misplacement caused in the space shape approximation process can be completely resolved by introducing fuzzy logic. The fuzzy logic could be used when going from the rough room shape to the accurate room shapes. Currently the boolean split process that creates the accurate shape avoids the use of fuzzy logic. There is an alternative version of the tool where fuzzy logic is used. This version is able to create correctly shaped rooms at the problem location, see Figure 5.20. At other locations in the model that did not show any signs of issues in the original build, it now is unable to create the room representation shapes. The tool would benefit if there was a way to combine the two builds. This could be implemented in such a way that the tool could enable fuzzy logic when issues are encountered in the normal creation of an accurate room shape.

[4] If the model has walls or rooms that are not perpendicular or parallel to the created voxelgrid the smallest allowed hallway or opening is 67.35 centimeters

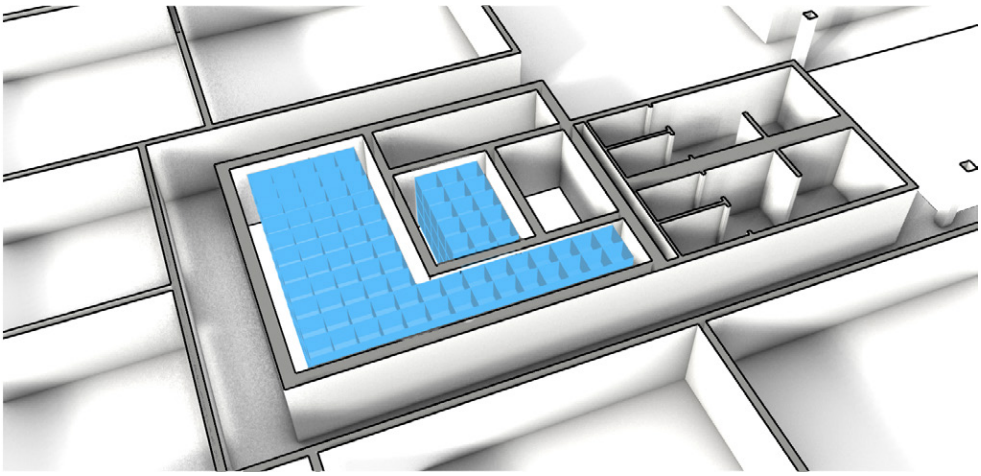


Figure 5.17.: A section of the second floor of the DigitialHub model populated with the non-intersecting voxels representing a rough room shape. It can be seen that this rough room has grown through a door into a second room.

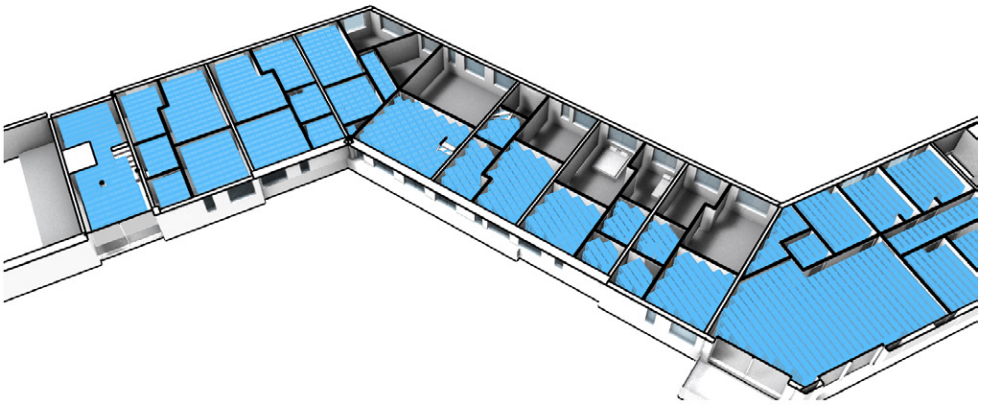


Figure 5.18.: The first floor of the `ON4 Building` model populated with the non-intersecting voxels representing rough room shapes. It can be seen that a subset is not occupied by rough room shapes.

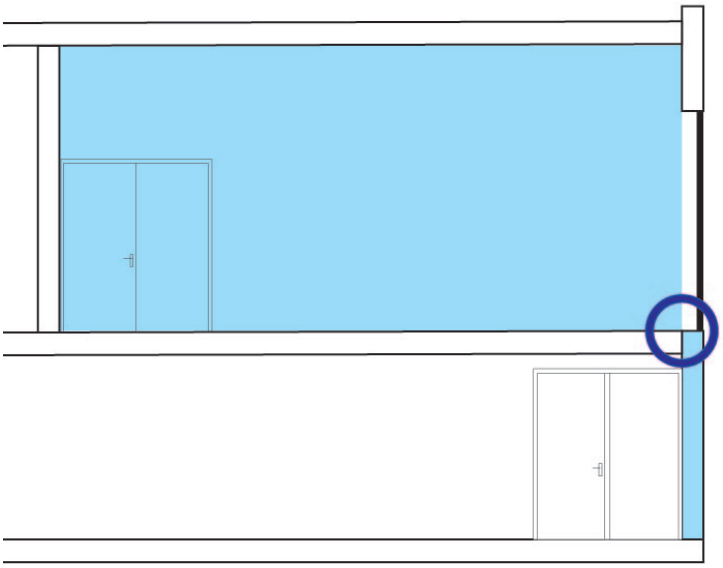
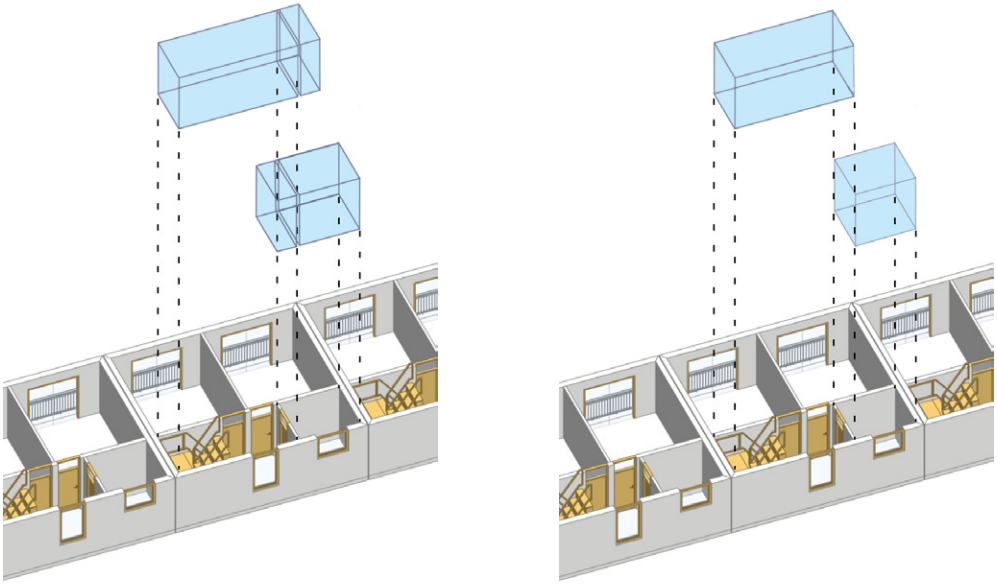


Figure 5.19.: Example of a topologic room wall connection that is 1d, highlighted in dark blue. This occurs in the `CUVO Building` model where the top of the walls of the basement level are flush with the floor of the ground floor.



(a) (b)

Figure 5.20.: The effect of using fuzzy logic in the boolean split function that can create precise spaces. (a) is the result of the process using no fuzzy logic when encountering the misplaced wall in the `Smiley-West-10` model. (b) is the result when using fuzzy logic in this process.

The creation process of the accurately shaped spaces in `RAC-sample-project` and the `ON4 Building` model is performing very badly. This is due to the process encountering walls that are too complex for the Open CASCADE library to parse into solids. The backup approach where the object's faces are used instead of the unified solid also yields no positive effects in this case. This results in the boolean process not using these walls as splitting agents. This means that the process creates invalid and/or incorrect shapes that are often discarded by the tool itself. A crude solution for this would be creating oriented smallest bounding boxes around the walls as a way to simplify them when solid geometry cannot be recovered. This style of simplifying is also done for *IfcDoor* and *IfcWindow* objects and works well if the object they encompass is primarily linear in nature. However, if the walls are not primarily linear in nature, the results will be affected drastically. An alternative approach that could avoid the need of simplification is the earlier mentioned use of fuzzy logic. However, when attempting this, it did not resolve the issues in this case.

Although the `CUVO Building` is a seemingly simple model, it also showed a reduced space creation success. This is mostly caused by the excessive use of *IfcCurtainWall* objects/aggregates like they were *IfcDoor* or *IfcWindow* objects. These *IfcCurtainWall* objects can be, like *IfcDoor* and *IfcWindow*, extremely complex. The tool does however not simplify these *IfcCurtainWall*, which can, and does, result in situations where the curtain wall does not split a room correctly, this can be seen in Figure 5.21. A potential solution is again creating smallest bounding boxes around the objects as a way to simplify them when solid geometry is not recovered. Again, as was noted at the wall simplification, this will only work if the object that is being simplified is primarily linear.

Due to the way the results have been presented, a major issue with the tool has been passed over unnoticed. Slight symptoms might be spotted when looking at the results of Table 5.10 where the shafts in the `DigitalHub` are unable to recover any semantic data. This is due to the tool using simplified *IfcSlab* geometry where no *IfcOpeningElement* objects are applied. This means that no holes are present in the *IfcSlab* geometry that is used to create the *IfcSpace* shapes. This often prevents the created *IfcSpace* shapes to occupy multiple storeys. These simplified *IfcSlab* objects are used because their use resolves many of the issues that could occur when an atrium or open staircase is present in a model. However, it does create a situation where a shaft spanning over multiple storeys will be split over every storey, see Figure 5.22.

In Section 5.2, it was mentioned that the tool sometimes encounters objects that are flagged as having no representation while also not aggregating other objects that have a representation. For the processes described in this section, this also creates an issue. These objects are unable to be used to shape rough room shapes, see Figure 5.17. These objects also can not be used as split agents to refine the room shape. Finally, they can not be identified and used in the processes related to the topologic relations. In the `DigitalHub`, this results in *IfcSpace* objects that incorrectly occupy two spaces and in the mentioned missing related objects. These objects are the four *IfcDoor* objects for which no representation objects can be found.

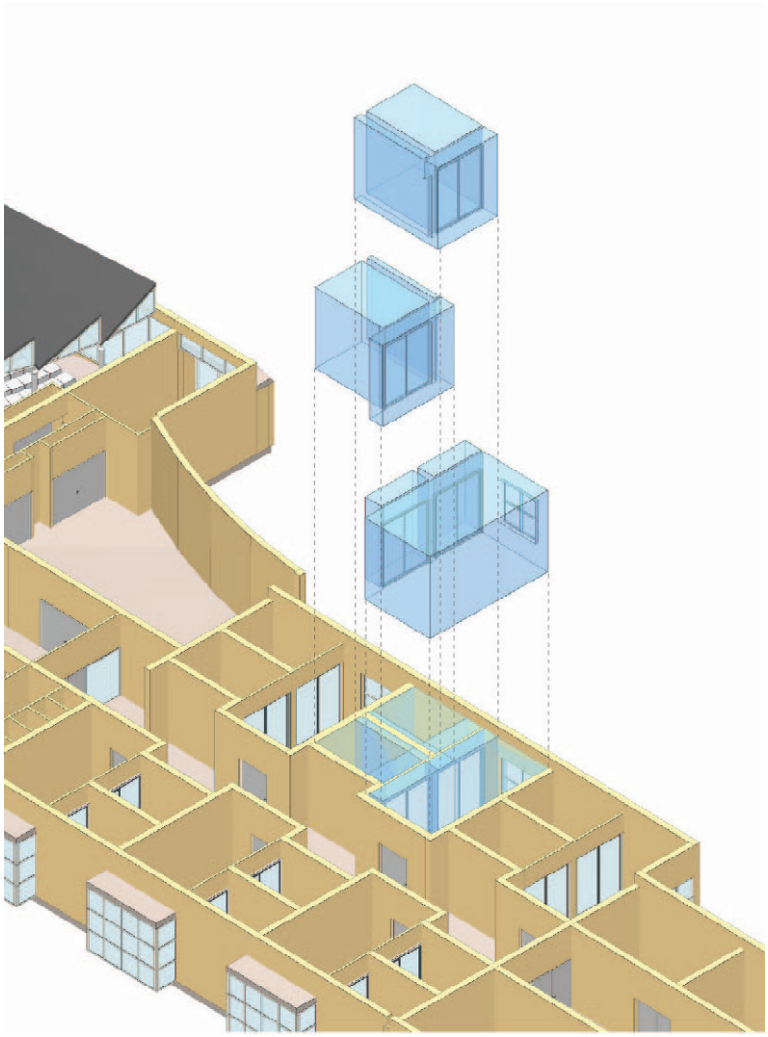


Figure 5.21.: Visualization of the incorrect output that can be caused by excessive curtain wall use. The CUVO model has a major part of its doors and windows stored as *IfcCurtainWall* objects. It can be seen that the processes do not deal with this properly, resulting in rooms bleeding through the curtain walls into other rooms.

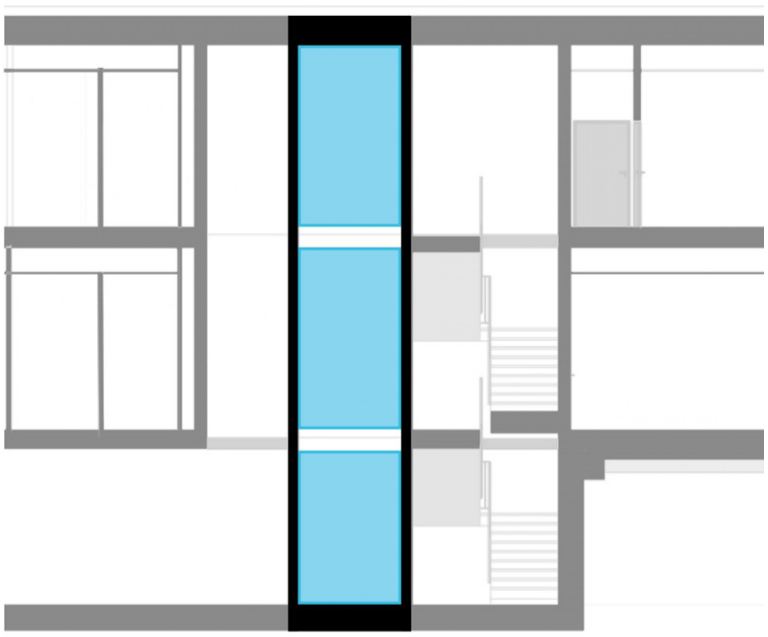


Figure 5.22.: Visualization of the shaft splitting that occurs due to the simplified *IfcSlab* use in the space detection process. In this section of the `DigitalHub` model, the shaft is highlighted with the resulting *IfcSpace* object shapes occupying this shaft highlighted in blue. It can be seen that the single shaft is split in three spaces separated by two voids equalling the height of the *IfcSlab* object next to it

Name	Value	Unit
Element Specific		
CompositionType	ELEMENT	
Description	Has 3 unique doors and 2 unique stairs. Connected to : 05.1.3, 5.2.3, 5.2.1, 5.2.2, 5.3.4, apartment: 4	
Guid	0W6KFFlzjFZfhgxG4ZMa5m	
IfcEntity	IfcSpace	
LongName	FLUR-5-3	
Name	5.2.4	

Figure 5.23.: Example of the connectivity data that is stored in the description of the *IfcSpace* objects that have been processed by the tool.

Model name	# of found connections	# incorrect connections	# missing connections	# floating spaces
FZK-Haus	5	0	0	0
Institute-Var-2	81	0	0	0 (2)
Smiley-West-10	147	0	3	3
RAC-sample-project*	11	0	3	14
DigitalHub	62	0	0	0 (8)
ON4 Building*	132	0	25	70 (85)
	158	0	2	6 (21)
CUVO Building	29	6	8	14 (32)
Projekt Golden Nugget*	45	8	37	79

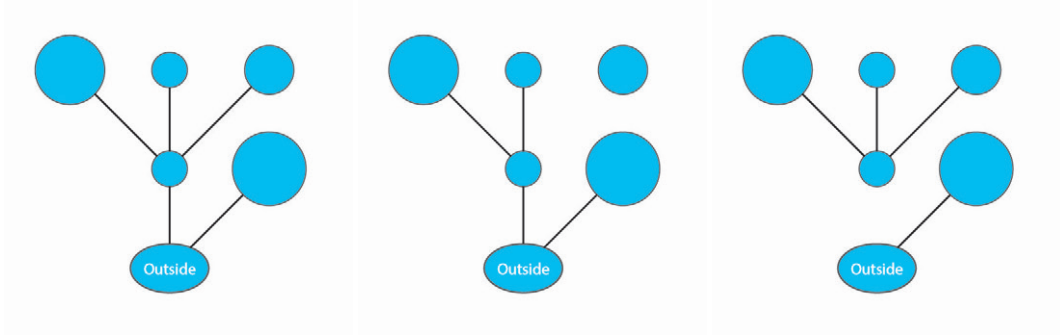
Table 5.12.: Summary of the room connection approximation. The numbers in brackets have the spaces included that have no connections to the rest of the building. These spaces are often shafts.

5.4. Detection and reconstruction of apartments

5.4.1. Overview of the results

To evaluate the performance of the room connectivity process, a combination of manual and automatic checking is done. With the help of Rhino 3D and grasshopper (with the Syntactic plugin), it is possible to display room connections in a graph and count the amount of connections. Although it can display the connections, it is only used to automatically count the number of connections. The accuracy of these connections is checked manually in the model itself. The tool writes the connectivity data to the description of the *IfcSpace* object, see Figure 5.23. This facilitates a reliable and fairly easy way to manually check the created connections between spaces. The result of these checks can be found in Table 5.12. The table shows the number of found connections and how many of those are incorrect. The table also covers the missing connections, these are connections that should be present based on the door and staircase placement but could not be detected by the tool. The final column covers the number of floating spaces. Floating spaces are spaces that do not have a connection to the outside. This can be both isolated spaces or groups of spaces, see Figure 5.23. The model names marked with an asterisk use the original *IfcSpace* objects stored in the IFC model due to the tool being unable to create new valid *IfcSpace* objects in these models.

The performance of the apartment/section detection is evaluated completely manually. As was mentioned before, the tool writes the connectivity data to the description of the *IfcSpace*



(a) (b) (c)

Figure 5.24.: Examples of floating spaces in graph form. (a) has no floating spaces, all spaces are (correctly) connected to the outside. (b) has one floating space. (c) has 1 floating cluster / 4 floating spaces

Model name	# expected apartments	# of correct apartments/-sections	# incorrect apartments/-sections	# missing apartments/-sections
FZK-Haus	1	1	0	0
Institute-Var-2	1	1	0	0
Smiley-West-10	10	10	0	0
RAC-sample-project*	1	0	2	0
DigitalHub	1	1	0	0
ON4 Building*	20	13	4	3
	20	16	4	0
CUVO Building	1	0	4	0
Projekt Golden Nugget*	8	0	11	0

Table 5.13.: Summary of the apartment approximation.

objects, see Figure 5.23. This can be compared to the expected apartment groups. The results of this assessment can be found in Table 5.13. This table covers the amount of correctly and incorrectly found apartments/sections. It also covers apartments that are clearly missing from the output.

For both the tables, the `ON4 Building` model has two rows. The first row is the original model. The second row is a model that has all empty door openings replaced with *IfcDoor* objects and the two galleries of the model have been occupied with an *IfcSpace* object. These two *IfcSpace* objects are thus places outside of the building. The CUVO model has been excluded from the majority of the described evaluations because the tool was unable to create a large set of the expected *IfcSpace* objects. The source BIM model is not available so manually adding the *IfcSpace* objects and and using those for further processing is regarded as too challenging.

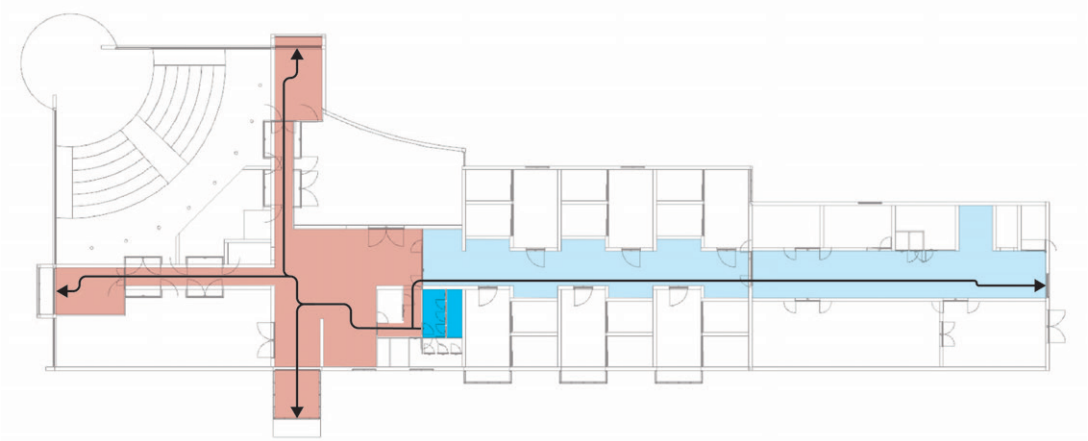


Figure 5.25.: Example of the required connections for a cluster in the `CUVO Building` model to be connected to the outside. The dark blue rooms are the evaluated cluster, the black pathways show all the connections to the outside. The blue spaces are the created spaces that are on this pathway, the red spaces are spaces that are on this pathway but have not been created. Although no pathway to the outside exists (every path passes through a non-existing space), the tool considers the cluster to be directly connected to the outside.

5.4.2. Discussion

Table 5.12 shows that the connections of rooms in the simple models are very accurately recovered. the `Smiley-West-10` building is however showing that three spaces are floating. These floating spaces are caused due to the incorrectly placed walls mentioned in Section 5.3.3. If the process is executed with the version of the tool that has fuzzy logic implemented, these floating spaces do not occur. The results based on the `DigitalHub` are also very accurate, the only floating spaces here are the shafts that are unconnected to other rooms in a horizontal manner. The original `ON4 Building` model is showing some missing connections and a large amount of floating spaces. The tool performs better on the adjusted model where openings are replaced with doors and the external connecting rooms are added. As with the DitigalHub model, there are also a lot of shafts present here that cause floating spaces to be present. At the `Projekt Golden Nugget` model the performance is severely hampered. Only 7 of the 86 *IfcSpace* objects have a connection to the outside and many connections are missing. Also the merging of the limited amount of made apartments seem to be executed unsuccessfully.

The results of the `CUVO Building` model shed a light on the inner workings of the tool due to its data being very unsuited for the process. Surprisingly, the model incorrectly creates a very low amount of floating spaces. Almost every cluster of rooms is considered to be connected to the outside, this is not the truth however. In reality, most of these clusters are not connected to the outside. Due to the limited amount of created *IfcSpace* objects in earlier processes, the *IfcSpace* objects that should form the connection between these clusters and the clusters that are actually connecting to the outside are missing, see Figure 5.25. Aside from this, also a relatively large amount of connections that were expected to be present are also missing. The majority of these missing connections are spaces that were supposed to be connected via *IfcCurtainWall* objects.

Table 5.13 shows that although the implemented rules for the detection of the apartments are

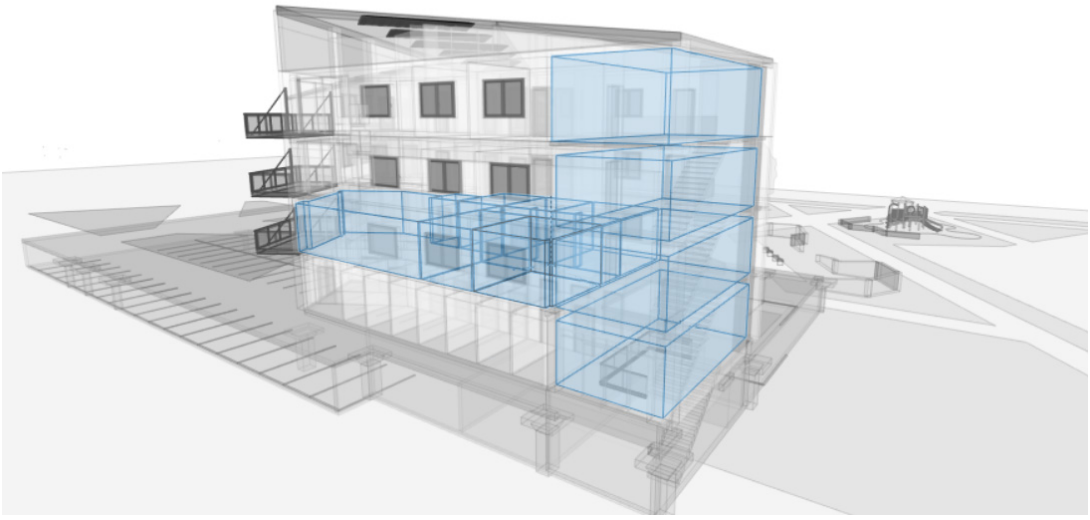


Figure 5.26.: Incorrect apartment growth due to the simplicity of the rule set. The apartment here has grown into the stairwell.

very simple, the tool is already able to approximate apartments with a decent success rate. The simple models show no incorrect apartment detections. The wall misplacement of the `Smiley-West-10` model does not have any effect on this process. Like the simple models, the `DigitalHub` model also has been processed with no inaccuracies. The `ON4 Building` model does show some inaccuracies. In both the adjusted and unadjusted model, the causes are primarily the same: incorrect or missing room connections. Aside from these there is one error that occurs in the model due to the simplicity of the rule set, see Figure 5.26. At this location, there is an apartment that has been grown into some of the hallways/connective spaces between other apartments. This growth has however not continued to other apartments but stayed inside the splitting spaces only. Therefore, it has not replaced other apartments or obstructed certain apartments to be created.

The choice to almost completely manually assess this process of the tool instead of relying on the graphs created with the help of Syntactic was taken due to Syntactic having issues with floating spaces. If the input data was not all connected, the output could be inaccurate or incorrect. In addition, complex models could give heavily cluttered graphs where apartments and even the normal connections were challenging to recognize. Syntactic was a very useful tool to help develop and implement the methods, but for the final results, it was not reliable and clear enough. The Syntactic graphs for the simple models can be found in Appendix F.

5.4.3. Limitations

The process that find the connections between spaces performs very well, however an issue has been introduced due to the way a preceding process has been implemented. Currently, there is no functionality to detect the connection between two *IfcSpace* objects that have their

boundary faces resting against (or close to) each other. This has been explicitly mentioned in the method as something that could not occur and, thus, no process regarding this has been developed. To an extent, this has not changed. However, in the preceding section, it has been mentioned that the current implementation of the methods splits spaces that span multiple storeys into multiple *IfcSpace* objects that only occupy a single storey. This occurs quite often for shafts for example, see Figure 5.22. These *IfcSpace* objects representing a single shaft often are only connected to other spaces via a face to face connection, and this connection cannot be found by the tool. As mentioned before, this is a limitation of the space creation process and, thus, it should be resolved in the space creation process and not in the processes related to connectivity of rooms.

However, this is not completely the end of this issue in the connectivity processes. By allowing the tool to generate results for models where it was unable to create new *IfcSpace* objects, this lack of room to room connectivity is again an issue. In original IFC models, there can be multiple *IfcSpace* that occupy one enclosed area while all being "Element" or "Partial" spaces, see Figure 5.27. These spaces should all be connected to each other but the tool is currently unable to signify them as doing so. Thus, although the room reconstruction method and implementation should be improved to prevent the need to use existing spaces, it would still be wise to implement a way to connect these kind of *IfcSpace* objects. This would increase the versatility of the tool. This is one of the issues that is present in the `ON4 Building` model and `Projekt Golden Nugget` model. In the `ON4 Building` model, occasionally, the rooms are connected via openings in the walls instead of *IfcDoor* objects. This is, for testing purposes, easily bypassed by replacing these openings with *IfcDoor* objects. This is done in the second entry of the `ON4 Building` model in Table 5.12 and 5.13. The `Projekt Golden Nugget` model has a lot of open spaces that are partitioned into *IfcSpace* objects. This model really showed how limiting this missing feature is. The tool is almost completely unable to reconstruct a connectivity graph. Due to this, it is subsequently also unable to separate the data it into apartments.

The connections to the outside are restricted to be only present at ground level. This prevents doors that lead to balconies, which are on the outside of the building, to be mistaken as normal ground floor exits. However, when an apartment complex has galleries that connect certain apartments to the ground floor and the outside, this can create issues. The tool is unable to detect an outside connectivity for those apartments. These apartments are not at a low enough level to be directly connected to the outside; consequently, the tool does not register an outside connections. This issue can be easily bypassed by creating an *IfcSpace* object in the model that occupies the space of these galleries, essentially becoming a hallway but outside of the building. This is done in the second entry of the `ON4 Building` model in Table 5.12 and 5.13.

The way connections to the outside space are determined create another issue. Currently when a door has only one singular connection to a space, the door is, when located near the ground floor, presumed to be a connection to the outside. However, when the code fails to create an *IfcSpace* object, a door object could incorrectly only have a single connection assigned. This can result in the creation of incorrect connections to the outside. This error occurs multiple times in the CUVO model. Clusters of rooms are marked as connected to the outside, while in reality, this is not true, see Figure 5.28.

The CUVO model shows another room connectivity issue. When the methods were developed the potential misuse of *IfcDoor* objects was not anticipated. The CUVO model uses three ways to model doors, with: *IfcDoor* objects, *IfcWindow* objects and *IfcPlate* objects. The *IfcDoor* objects function as intended because the process was developed to function on these

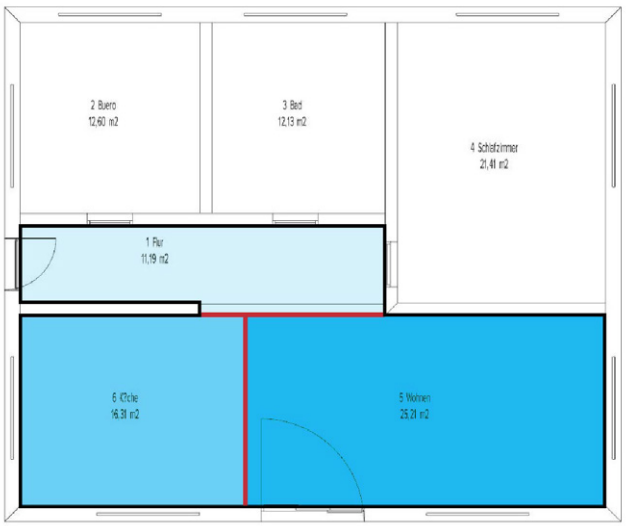


Figure 5.27.: Floorplan of the ground floor of the `FZK-Haus` model with the three *IfcSpace* shapes, occupying the largest space, highlighted. The faces of each *IfcSpace* shapes that rests against another is highlighted. These connections can not be detected by the tool.

Figure 5.28.: The room clusters incorrectly considered by the tool to be connected to the outside

objects. However, the use of *IfcWindow* and *IfcPlate* objects make these alternative "doors" to be completely discarded by the tool. If this misuse of objects was caused by the creator of the model (and not created by the software package when exporting), this might be one of the few issues that possibly should be addressed in the original BIM model. The only reason it is known that these objects should be considered as door is either due to the object's name or due to the separate use of *IfcAnnotation* objects. This is hard to reliably detect in an automated manner.

The amount of connecting doors and staircases is counted per *IfcSpace* object. Although not explicitly mentioned, this can be seen in Figure 5.23. This counting process occasionally over counts the actual amount of doors present in the model. This occurs when the door and the door frame are modelled as two separate *IfcDoor* objects. This door (and its frame), consisting out of two objects, is thus counted as two doors. A solution for this problem would be checking if an *IfcDoor* object completely encapsulates another *IfcDoor* object. If so, these two object could be considered as one single door.

Apartment buildings can be complex structures that are currently being generalized with extremely simple rules by the described methods. The generalization of the splitting agents (the hallways) creates at least one occurrence in the test models where the ending of an apartment is incorrectly detected, see Figure 5.26. This is due to the apartment being connected to a hallway that only has one door and two staircases. The staircases are not incorporated in the rule set, the rules only use the doors. However, adding the staircases to this rule would still not create a situation where the hallway is recognized as the ending of an apartment. The hallway has to have at least 5 connections to other spaces to be considered a hallway. Adding a rule that would make a room a splitting agent based on the presence of two staircases also will not function well because a room with two staircases can still be part of one apartment or section in another building. Examples of this would be the `Institute-Var-2`, `Smiley-West-10` and the `DigitalHub` models.

An improved apartment detection could be introduced with two additions. The first one would be the introduction of more, and/or more precise, rules that rely not only on the connectivity structure but also on other topologic relationships of the *IfcSpace* objects it tries to group. This could for example be the recognizing of different styles of doors that could signify the ending of an apartment or the wall thickness. This however has to rely partially on semantic data, which is considered unreliable. Secondly, the user could get more control over the rules. Being able to turn certain rules off and on or change variables would allow the user to get a more reliable output. However, it would also force the user to be more familiar with the model, which can cost a lot of time. This is also slightly error prone.

6. Conclusion & future work

This thesis showed potential methods to create and extract reliable data from IFC models for three subjects: storeys, rooms, and apartments. The development of these methods and their implementation was done based on the question:

- How can building features be automatically recovered from an IFC file in a reliable manner?

To answer this question completely and accurately one other question has to be covered first:

- What data in an IFC file can be considered accurate and/or reliable and is the minimal needed data to base the reconstruction upon?

The data in an IFC file that is considered accurate and/or reliable is the tangible geometry. This is geometry that is easily seen by the users and is of importance for a lot of involved parties. Thus, this geometry is getting a lot of passive and active accuracy checks. This geometry also implicitly stores a large quantity of other information that can be reconstructed based on it. However, not everything can be reconstructed based on tangible geometry. A subset of the semantic data has to be trusted at face value due to it not being implicitly stored in any other data. This subset should be kept as small as possible by ignoring any semantic data that can be extracted or recreated with the help of the tangible geometry. Topologic data is all considered unreliable and can be recreated based on the tangible geometry in this research's test cases.

The overarching research question has been attempted to be answered by focusing on three subjects: storey elevations, rooms, and apartments.

The detection/recovery of storey elevations can be done by grouping *IfcRoof* and *IfcSlab* objects and extracting the z-values of these groups. The classification of the IFC objects to a storey can be done based on the z-distance between a chosen base point on/in the object and the storey elevation. Although the detection process is very crude, it is able to accurately approximate the amount of storeys and their elevation in simple models very accurately. Buildings with more complex storey structures show that the presented method currently is still too unsophisticated to be trusted blindly. Similarly, the process that classifies the objects to storeys performs well on simple models but starts to become unreliable in buildings with more complex storey structures.

The rooms shapes can be reconstructed by a voxelization process that is followed by a boolean refinement. The voxelization and refinement process can be based on the room bounding geometry[1]. The semantic data can be recovered from the original *IfcSpace* object that the new *IfcSpace* object replaces via a simple point in solid operation. The related topologic relations can be created by a ray-casting process. This room reconstruction process is often able to

[1] *IfcWall*, *IfcSlab*, *IfcCovering*, *IfcRoof*, *IfcCurtainWall*, *IfcColumn*, *IfcBeam*, *IfcPlate*, *IfcMember*, *IfcDoor*, and *IfcWindow* objects

accurately voxelize most geometry shapes. However, the refinement process that creates a refined room shape can be severely hampered by complex geometry and incorrectly placed objects. In simple models, the refinement performs very well, but more complex models show that not all and occasionally no room shapes at all can be reconstructed. However, the process that recreates the topological relationships of the rooms performs well.

Finally, apartments can be detected by constructing a graph based on room connecting objects[2]. This graph can be split into apartments via a simple set of rules. It was observed that the apartment detection process performs very well considering the simplicity of the rule set that was used. However, it does show occasionally that this rule set requires extra refinement to improve performance and reliability.

In conclusion, this thesis has been unable to give a definite answer to the research question. All three subjects that were covered showed promising results, but still the output of the tool can not be relied upon blindly. For some subjects, the output can range from completely correct to completely missing. Therefore, the research showed how to automatically recover features, but not how to do it in a reliable manner. The research did show promise; the tool was able to recover/create very accurate data from some models. If the tool was unable to recover the data reliably, it is often known how to resolve this issue. For future works, it is suggested to address or look into these issues. The considered most important focus points are:

- Improving the geometry handling process, so that the tool is able to recover more complex data structures. Currently not all the geometry is recoverable, creating situations where not all objects are evaluated in the processes and their data is effectively being lost.
- Implementing an xy-domain into the storeys recovery and object classification code. This could increase the performance in models that have complex storey and roofing structures.
- Improved geometry simplification process and the incorporation of selective fuzzy logic in the room recovery process.
- Addition of more complex rules and evaluation methods in the apartment detection process.
- Improving functionality that is not part of the core code. E.g. the implementation of code that is able to detect the connection between *IfcSpace* objects that rest with their faces against each other.

[2] *IfcDoor* and *IfcStair* objects

Due to the presence of these issues and, additionally, due to the limited scope of the research, the correct functioning of the created tool is still heavily dependent on the quality of the input IFC file. This means that there are requirements for the input IFC files to ensure good functioning. The list of requirements is:

- The IFC file has to be in the IFC4 or IFC2x3 format.
- If multi-file models are used, one single file should have all the structural floors available.
- Only use single phase files.
- It is not recommended to use models that have room bounding elements that are *IfcBuildingElementProxy* objects.
- The chosen voxel size should be smaller than half the smallest distance between two room bounding objects.
- It is recommended to only process models with simple flooring structures, e.g. no half elevations.
- Geometry has to be watertight. e.g. no gaps between walls that should rest against each other.
- No direct openings from interior spaces to the outside are allowed.
- No objects representing doors that are not of type *IfcDoor* are allowed.
- It is recommended to avoid the processing of files that have extensive use of curtain walls.
- It is heavily discouraged to use curtain wall objects to replace window or door objects.
- *IfcSpace* objects should either not encapsulate or completely encapsulate other *IfcSpace* objects. No partial encapsulation is allowed.

A. Reproducibility self-assessment

A.1. Marks for each of the criteria

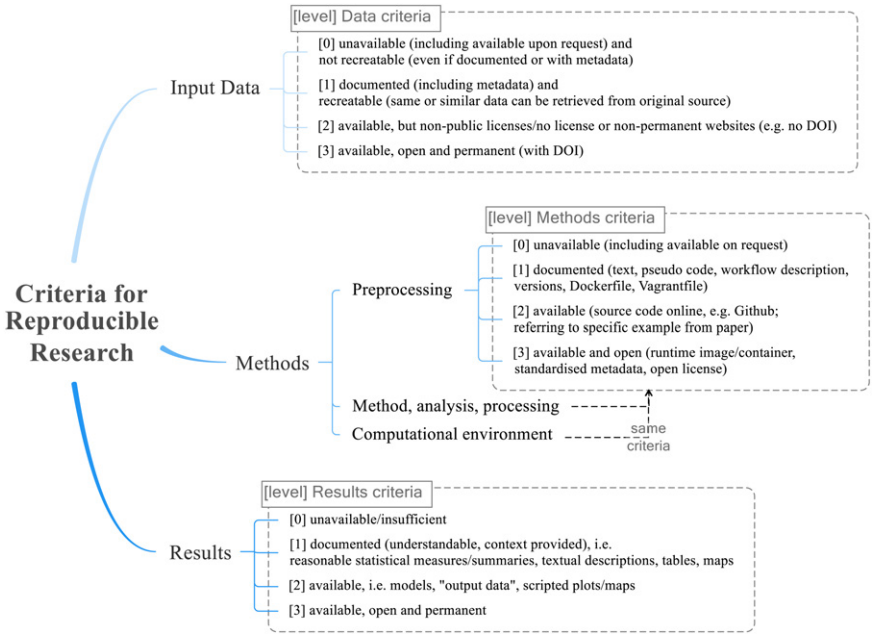
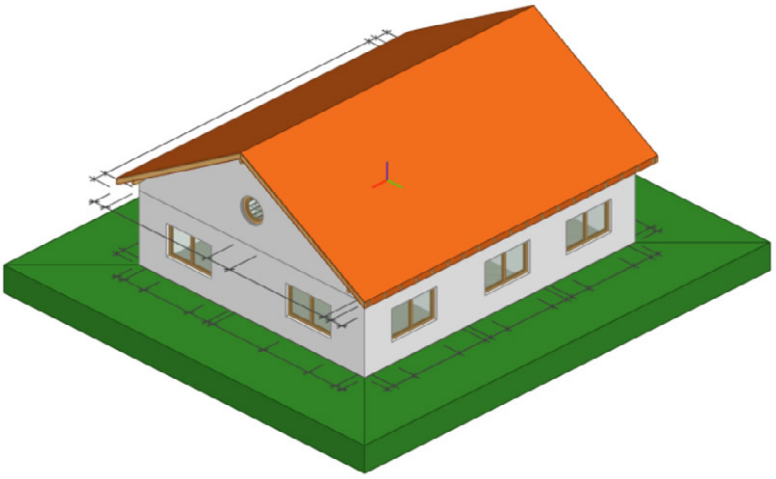


Figure A.1.: Reproducibility criteria to be assessed.

Criteria	Level	reflection
Input data	0 - 3	Ranges from publicly available data from third parties and myself to one private dataset that has not been made public
Preprocessing	(3)	Aside from exporting a model to IFC no preprocessing is required
Methods	3	Publicly available from the Github repository but does not contain metadata or DOI yet.
computational environment	3	All free and open source libraries were used.
results	0/3	Aside from a small subset, publicly available from the Github repository but does not contain metadata or DOI yet.

Table A.1.: The reproducibility criteria assessment.

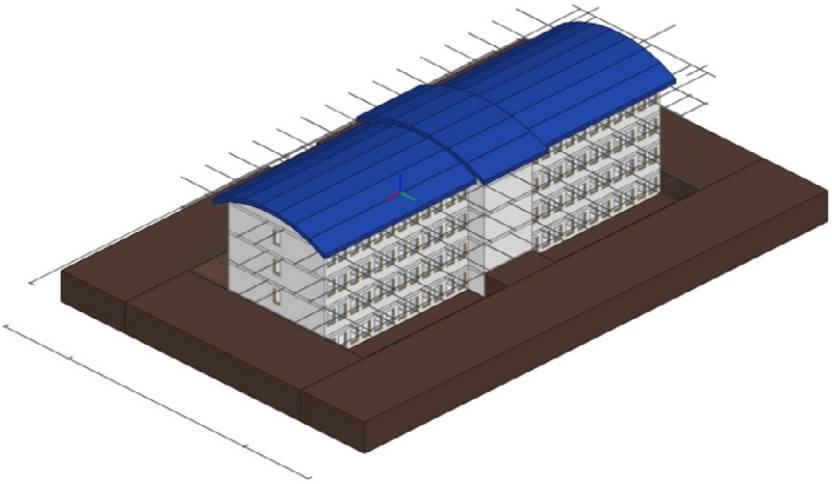
B. Used ifc models



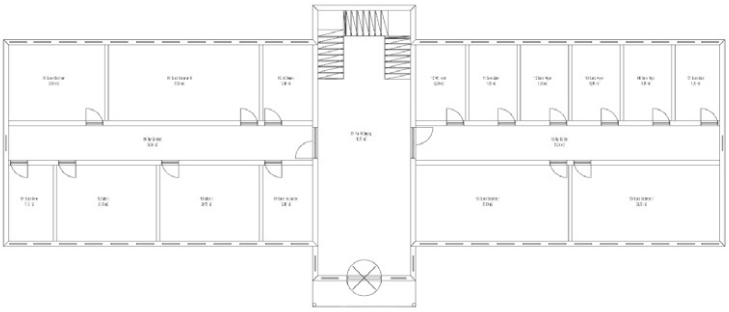
(a)

(b)

Figure B.1.: Visual overview of the `FZK-Haus` model. (a) isometric view of the model. (b) Floor plan of the ground floor.

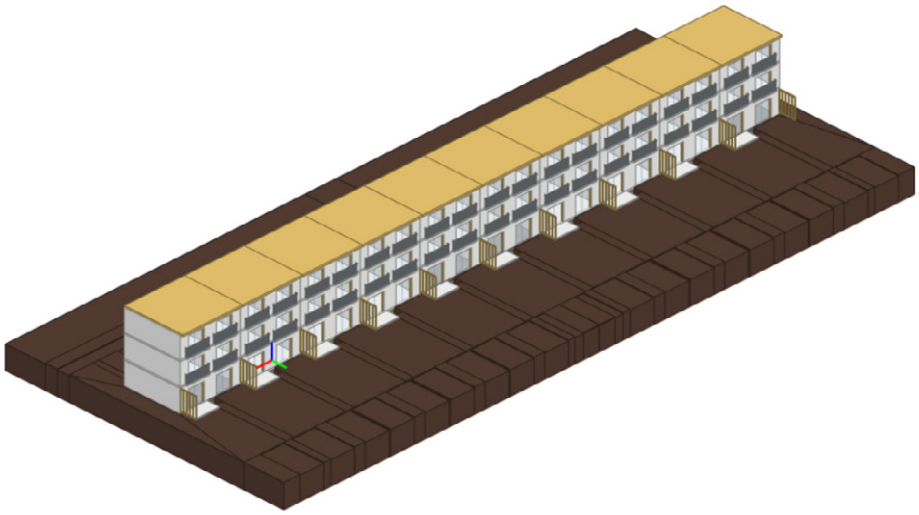


(a)



(b)

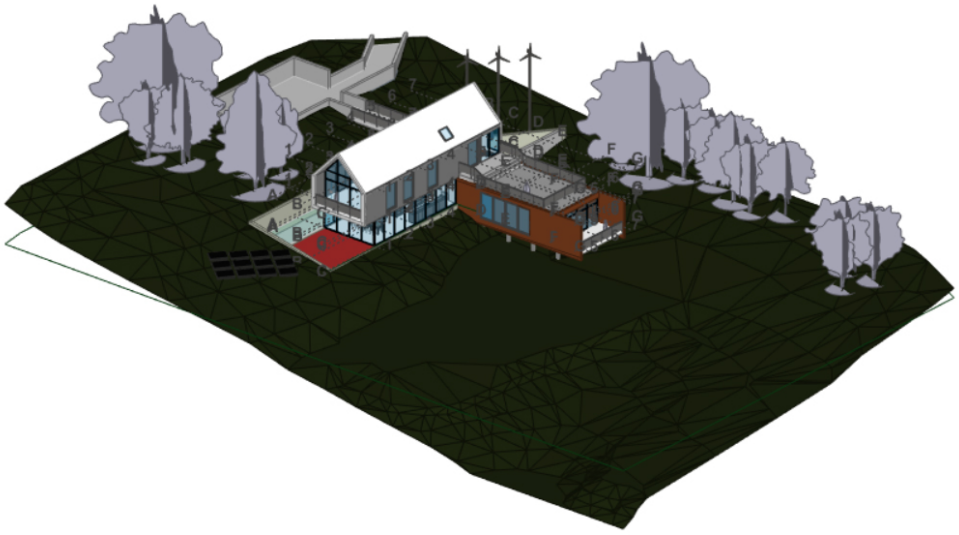
Figure B.2.: Visual overview of the `Institute-Var-2` model. (a) isometric view of the model. (b) floor plan of the ground floor.



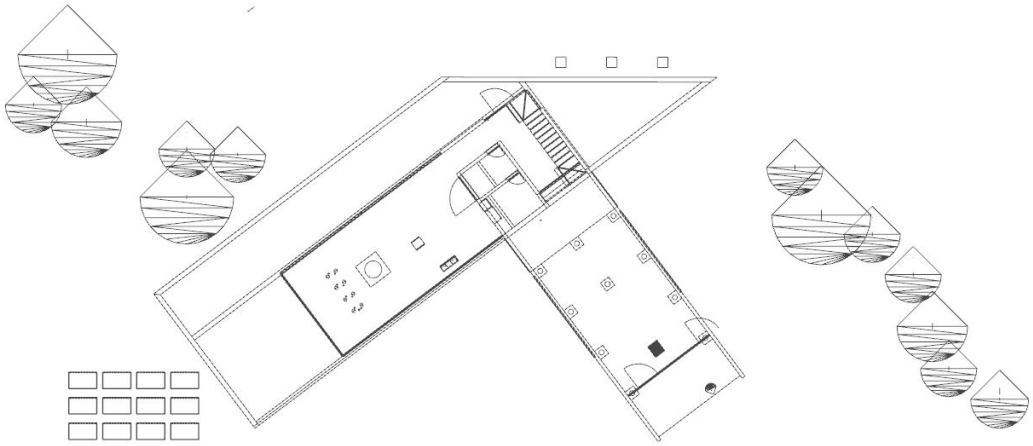
(a)

(b)

Figure B.3.: Visual overview of the `Smiley-West-10` model. (a) isometric view of the model. (b) floor plan of the ground floor.

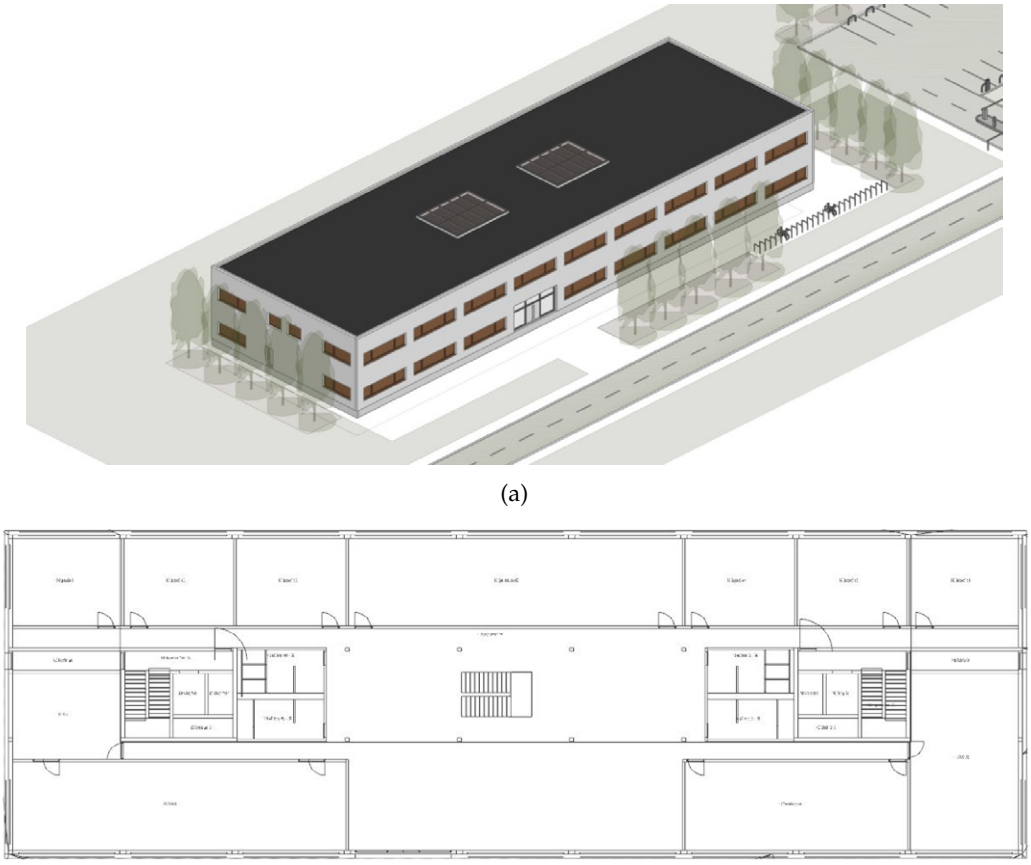


(a)



(b)

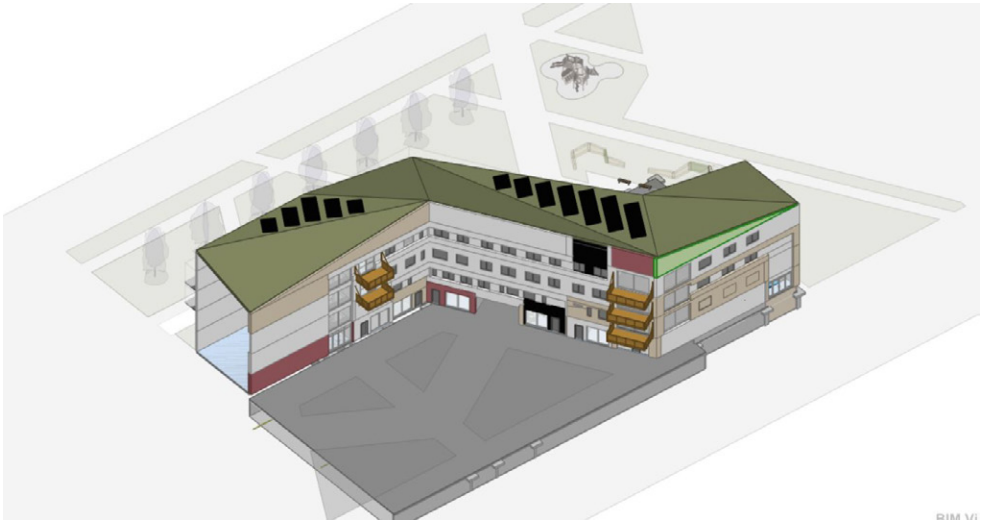
Figure B.4.: Visual overview of the `RAC-sample-project` model. (a) isometric view of the model. (b) floor plan of the two ground floor levels and partial site.



(a)

(b)

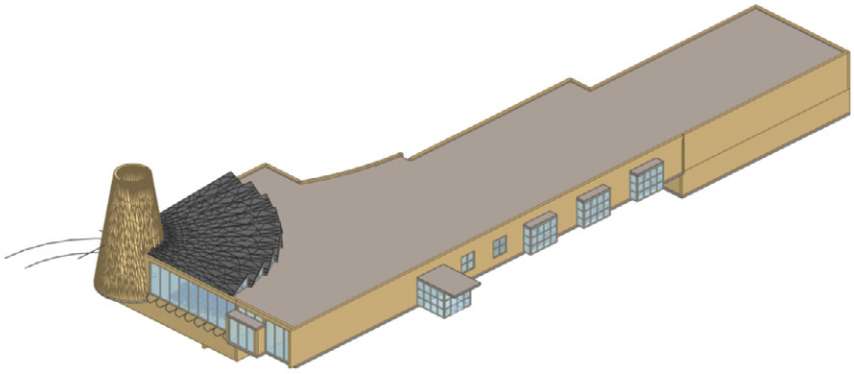
Figure B.5.: Visual overview of the `DigitalHub` model. (a) isometric view of the model. (b) floor plan of the ground floor.



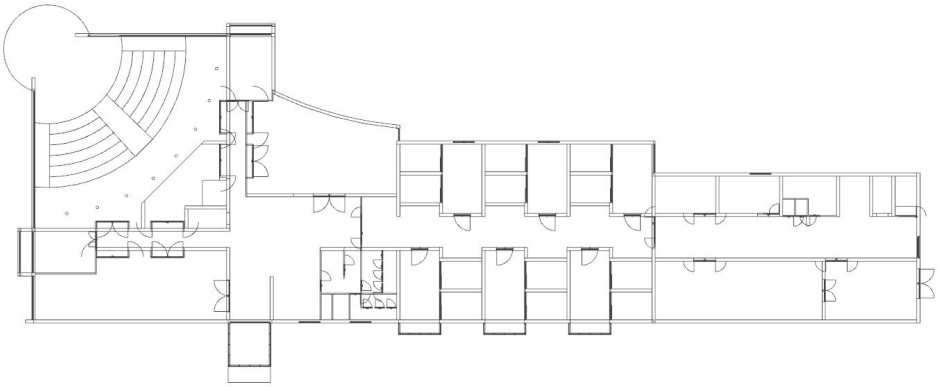
(a)

(b)

Figure B.6.: Visual overview of the `ON4 building` model. (a) isometric view of the model. (b) floor plan of the ground floor.



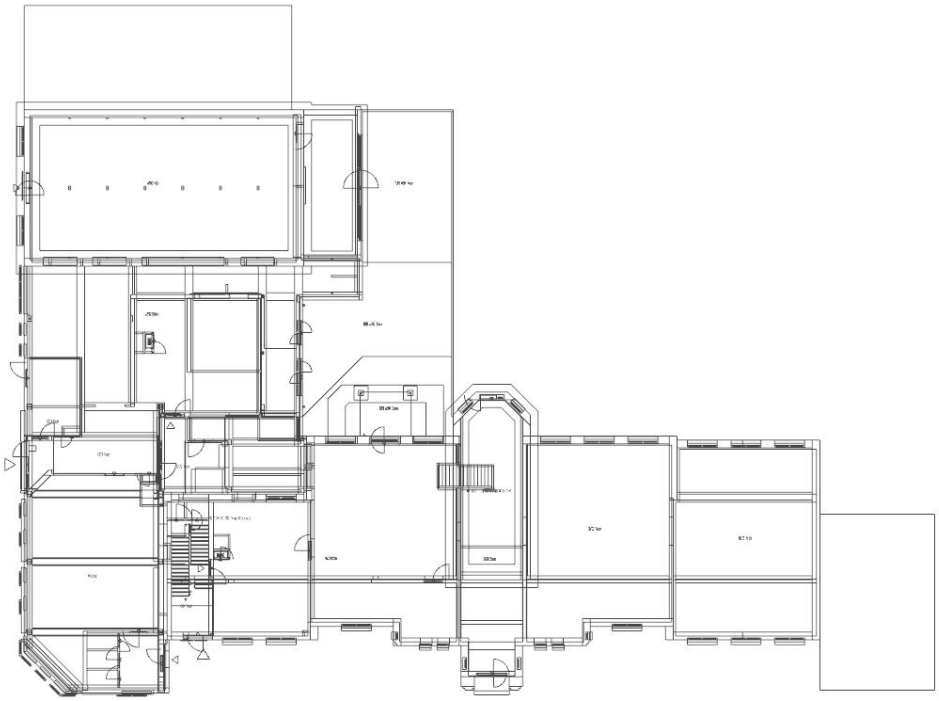
(a)



(b)

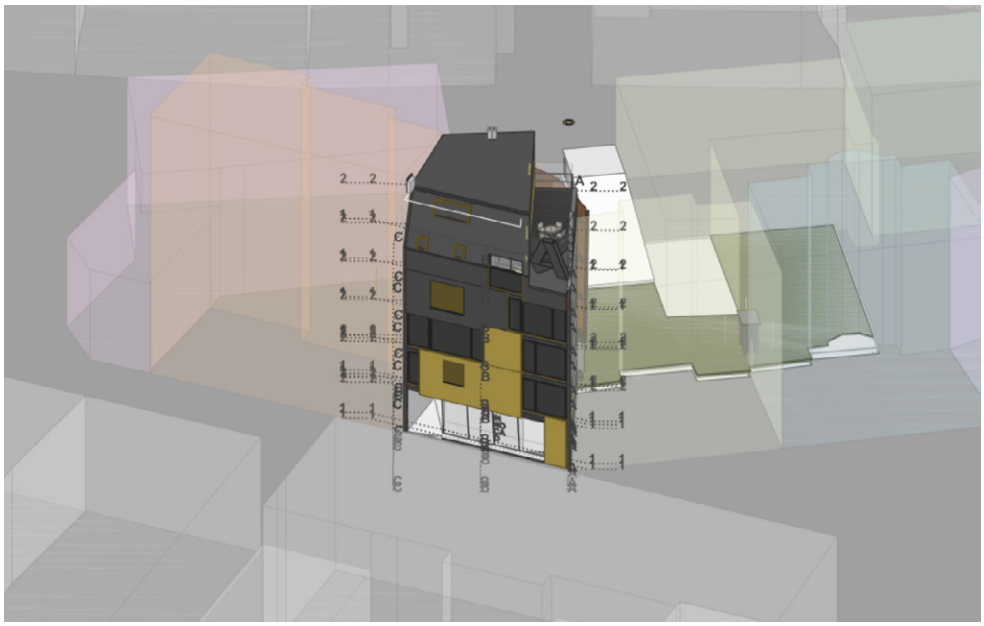
Figure B.7.: Visual overview of the `CUVO building` model. (a) isometric view of the model. (b) floor plan of the ground floor.

(a)

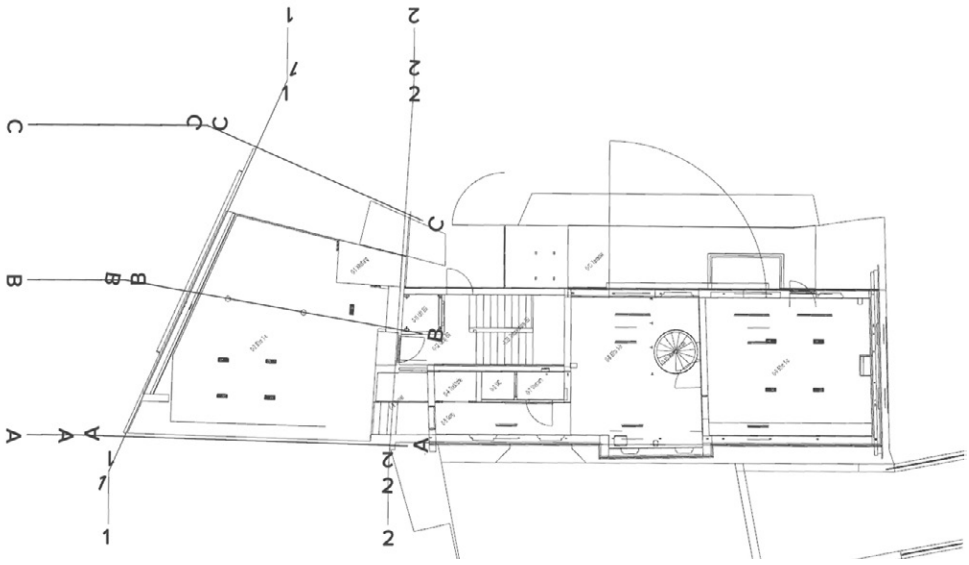


(b)

Figure B.8.: Visual overview of the `Witte_de_Withstraat` model. (a) isometric view of the model. (b) floor plan of the ground floor.

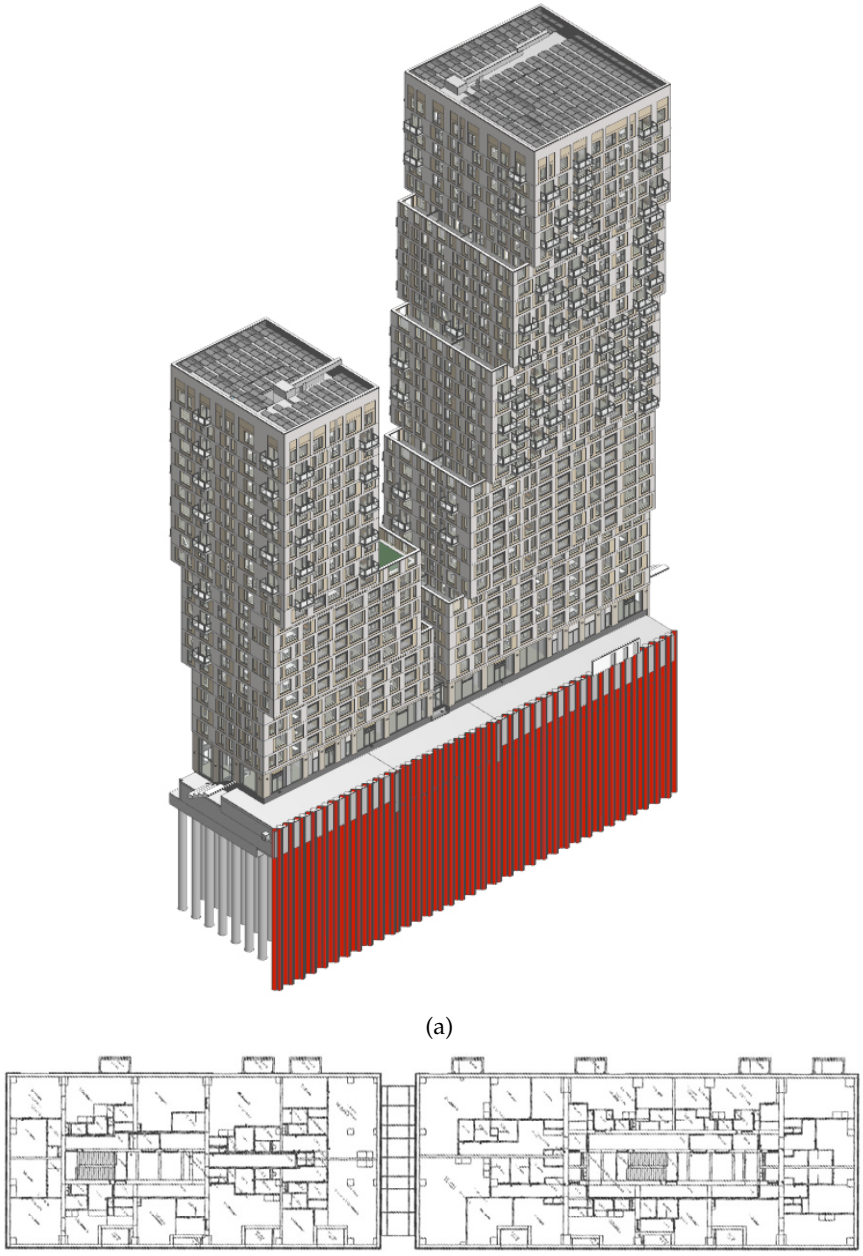


(a)



(b)

Figure B.9.: Visual overview of the `Projekt Golden Nugget` model. (a) isometric view of the model. (b) floor plan of the ground floor.



(a)

Figure B.10.: Visual overview of the the `Boompjes` model. (a) isometric view of the model. (b) floor plan of the first storey.

C. Results of the storey elevation detection process

FZK-Haus

Storey	Expected elevation (m)	# Original elevation (m)	# Approximated elevation (m)
0 Ground floor	0	0	0
1	2.70	2.70	2.70

Table C.1.: Full comparison between the stored in file storey elevations and the approximated elevations for the `FZK-Haus` model.

Institute-Var-2

Storey	Expected elevation (m)	# Original elevation (m)	# Approximated elevation (m)
-1	-3	-3	-3
0 Ground floor	0	0	0
1	3	3	3
2	6	6	6
3	9	9	9

Table C.2.: Full comparison between the stored in file storey elevations and the approximated elevations for the `Institute-Var-2` model.

Smiley-West-10

Storey	Expected elevation (m)	# Original elevation (m)	# Approximated elevation (m)
-1	-2.81	-2.81	-2.81
0 Ground floor	-0.25	-0.25	-0.25
1	2.53	2.53	2.53
2	5.23	5.23	5.23
3	7.75	7.93	7.95

Table C.3.: Full comparison between the stored in file storey elevations and the approximated elevations for the `Smiley-West-10` model.

RAC-sample-project

Storey	Expected elevation (m)	# Original elevation (m)	# Approximated elevation (m)
-1	-1.5	-0.80	-1.5
0 Ground floor	-0.55	-0.55	-
1	0	0	-
1b	-	2.70	-
2	2.53	-	-
2b	3	3	2.58
3	5.73	6	6.01

Table C.4.: Full comparison between the stored in file storey elevations and the approximated elevations for the `RAC-sample-project` model.

DigitalHub

Storey	Expected elevation (m)	# Original elevation (m)	# Approximated elevation (m)
-1	-3.67	-3.37	-3.67
0 Ground floor	-0.15	-0.15	-0.2
1	3.75	3.75	3.75
2	7.65	-	7.65

Table C.5.: Full comparison between the stored in file storey elevations and the approximated elevations for the `DigitalHub` model.

ON4 Building

Storey	Expected elevation (m)	# Original elevation (m)	# Approximated elevation (m)
-1	-3.50	-3.50	-3.50
0 Ground floor	0	0	-0.90
1	3.20	3.20	3.20
2	6.40	6.40	6.40
3	9.60	9.60	9.60
4	12.80	12.80	12.80
5	-	16.00	16.00

Table C.6.: Full comparison between the stored in file storey elevations and the approximated elevations for the `ON4 Building` model.

CUVO building

Storey	Expected elevation (m)	# Original elevation (m)	# Approximated elevation (m)
kelder	-2.9	-2.9	-2.9
bk. fundering	-	-1.4	-
ok BG	-0.7	-0.3	-
Peil=0	0	0	0
-	3.35	-	3.35
ok 1e	4	4	-
1e verdieping	-	4.50	-
ok 2e	-	7.8	-
3e verdieping	-	13.00	-

Table C.7.: Full comparison between the stored in file storey elevations and the approximated elevations for the `CUVO building` model. The original storey names are used where possible.

Witte_de_Withstraat

Storey	Expected elevation (m)	# Original elevation (m)	# Approximated elevation (m)
-	-2.5	-	-2.5
BK VLOER 00	-800	0	-0.8
BK VLOER 00 (R)	-	0.5	-
BK VLOER 01 (W)	3.453	3.5	3.453
BK VLOER 01 (R)	-	5.105	-
BK VLOER 01 (GZ)	-	5.5	-
BK VLOER 02 (W)	6.78* (?)	6.78	-
BK DAK (W)	9.9* (?)	9.6	-
BK VLOER 02 (R)		10.01	-

Table C.8.: Full comparison between the stored in file storey elevations and the approximated elevations for the `Witte_de_Withstraat` model. The original storey names are used where possible. The values marked with the asterisk are unsure. The structure of the building is complex and the storey elevation could be either one of those or both.

Projekt Golden Nugget

Storey	Expected elevation (m)	# Original elevation (m)	# Approximated elevation (m)
GN_UG 1_OK FFB	-2.5	-2.25	-2.5
TG_Garage_UG_OK FFB	-	-1.95	-
HG_UG 1_OK FFB	-	-1.948	-
GN_EG_OK FFB	0.14	0.3	0
HG_EG_OK FFB	-	0.75	-
HG_OG 1_OK FFB	3.33	3.50	3.33
GN_OG 1_OK FFB	-	3.79	-
Pfette Rechts	-	4.06	-
GN_OG 2_OK FFB	6.59	6.77	-
GN_OG 3_OK FFB	9.59	9.77	9.59
GN_OG 4_OK FFB	12.59	12.77	12.59
-	-	-	14.24
GN_OG 5_OK FFB	15.66	15.85	15.66
-	18.02	-	18.20

Table C.9.: Full comparison between the stored in file storey elevations and the approximated elevations for the `Projekt Golden Nugget` model. The original storey names are used where possible.

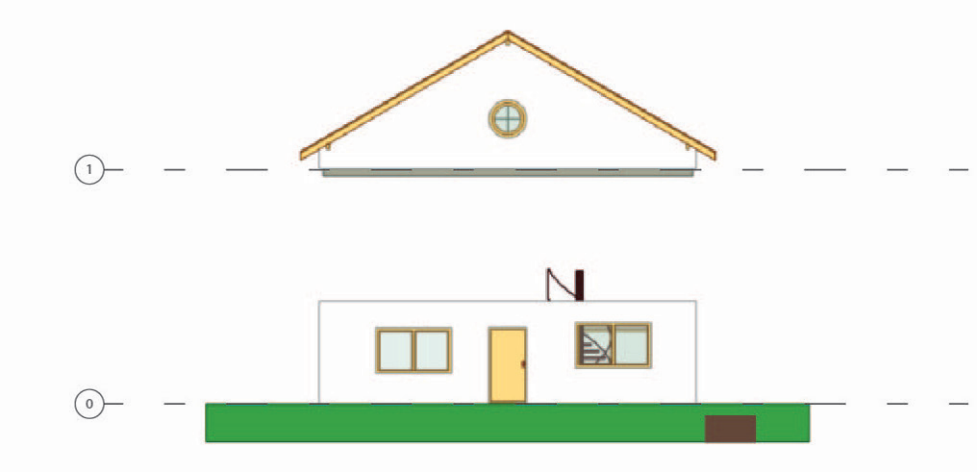
Boompjes

Storey	Expected elevation (m)	# Original elevation (m)	# Approximated elevation (m)
-3	-8.935	-8.935	-9.235
-2	?	-6.935	-6.935
-1a	?	-4.13	-4.13
-1	?	-2.7375	-1.37
0 Ground floor	-80	0	-
0a Ground floor	0.83	0.9125	0.83
0b	?	2.555	-
1	5.03	5.11	5.03
2	8.09	8.17	8.09
3	11.15	11.23	11.15
4	14.21	14.29	14.21
5	17.27	17.35	17.27
6	20.33	20.41	20.33
7	23.39	23.47	23.39
8	26.45	26.53	26.45
9	29.51	29.59	29.51
10	32.57	32.65	32.57
11	35.63	35.71	35.63
12	38.69	38.77	38.69
13	41.75	41.83	41.75
14	44.81	44.89	44.81
15	47.87	47.95	47.87
16	50.93	51.01	50.93
17	53.99	54.07	53.99
18	57.05	57.13	57.05
19	60.11	60.19	60.11
20	63.17	63.25	63.17
21	66.23	66.31	66.23
22	69.29	69.37	69.29
23	72.35	72.43	71.71
24	75.41	75.49	75.41
25	78.47	78.55	78.47
26	81.53	81.61	81.53
27	84.59	84.67	84.59
28	87.65	87.73	87.65
29	90.71	90.79	90.71
30	93.77	93.85	93.77
31	96.83	96.91	96.83
32	99.89	99.97	99.89
33	102.97	-	102.31

Table C.10.: Full comparison between the stored in file storey elevations and the approximated elevations for the `Boompjes` model. ”?” values are at locations too complex to accurately approximate an expected elevation manually. ”-” values do not have a matching elevation for that storey.

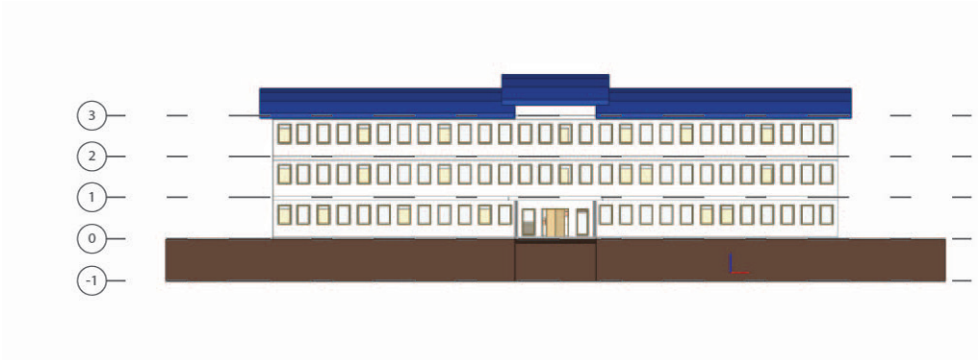
D. Visual results of the storey elevation detection and object sorting process

(a)

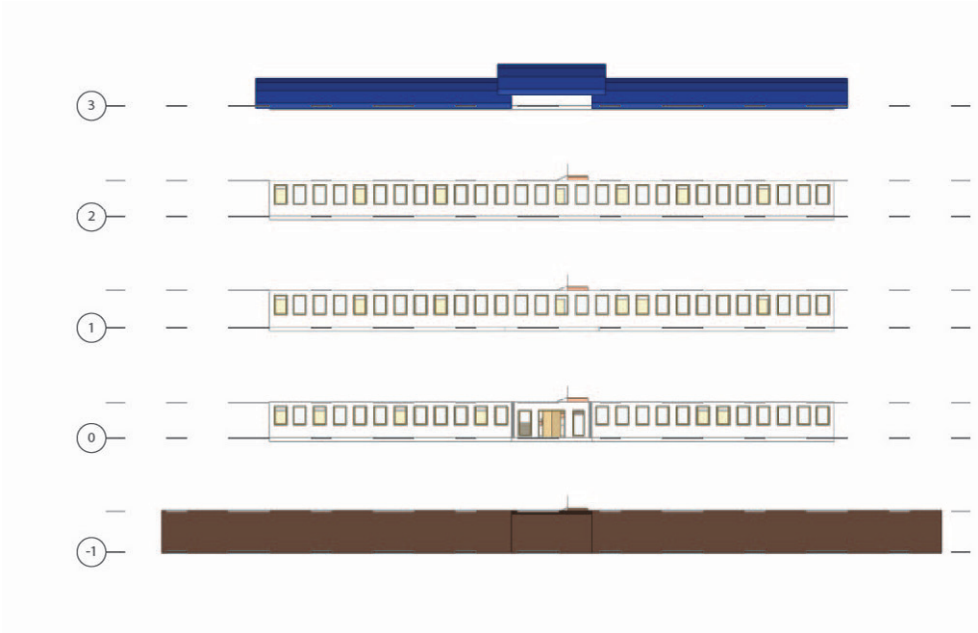


(b)

Figure D.1.: Visual result of the elevation detection process on the `FZK-Haus` model. (a) the front isometric view. (b) an exploded view based on sorting process based on these elevations.

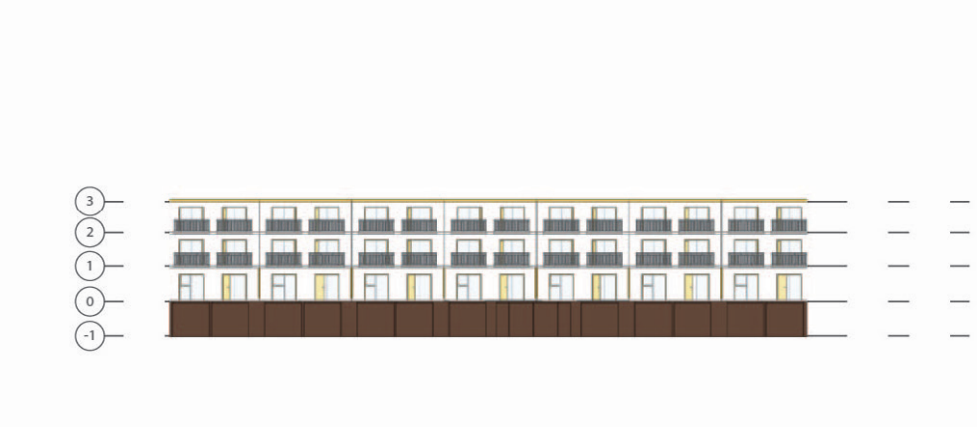


(a)

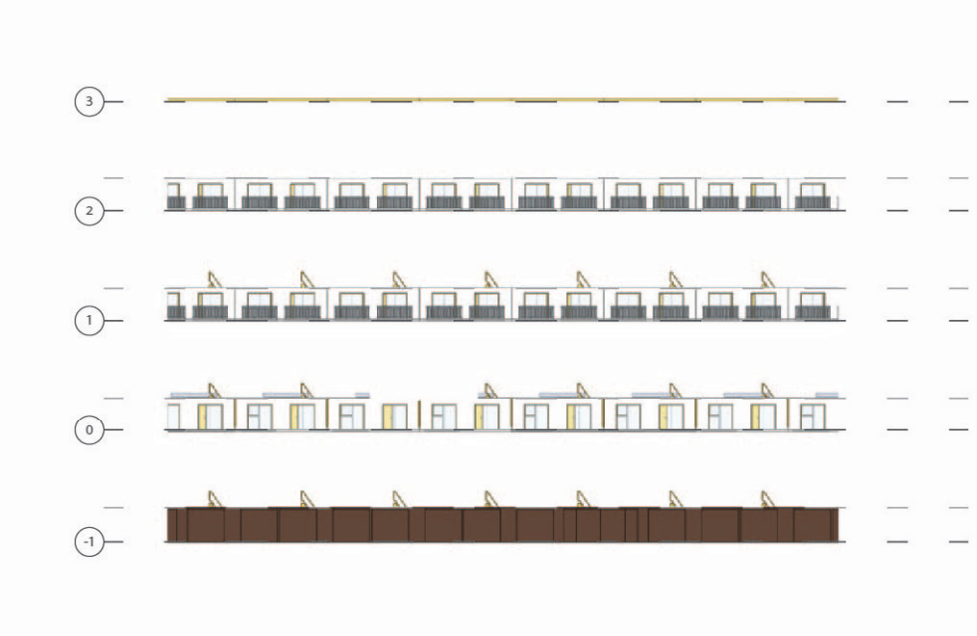


(b)

Figure D.2.: Visual result of the elevation detection process on the `Institute-Var-2` model. (a) the front isometric view. (b) an exploded view based on sorting process based on these elevations.



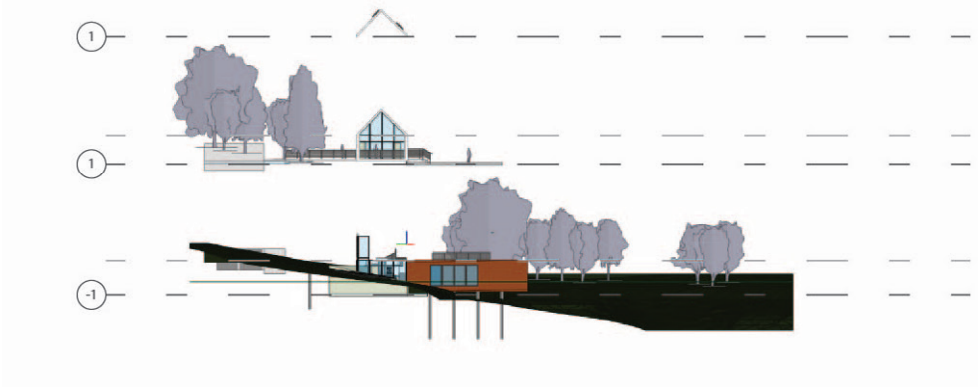
(a)



(b)

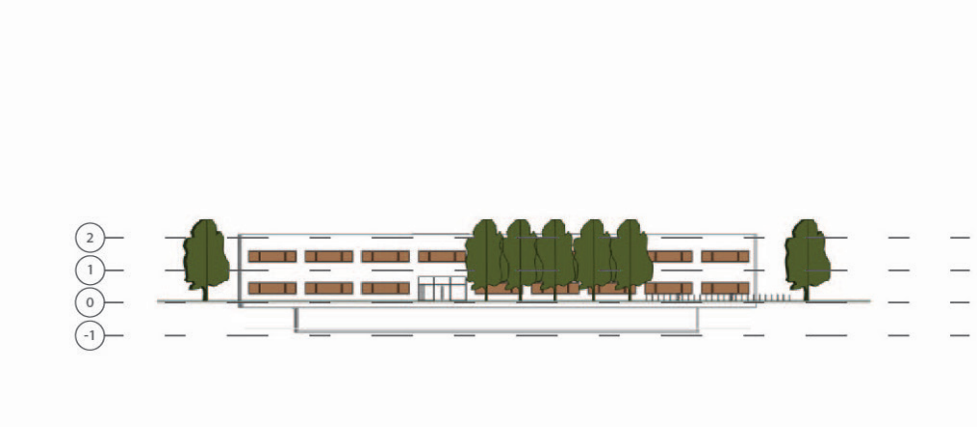
Figure D.3.: Visual result of the elevation detection process on the `Smiley-West-10` model. (a) the front isometric view. (b) an exploded view based on sorting process based on these elevations.

(a)

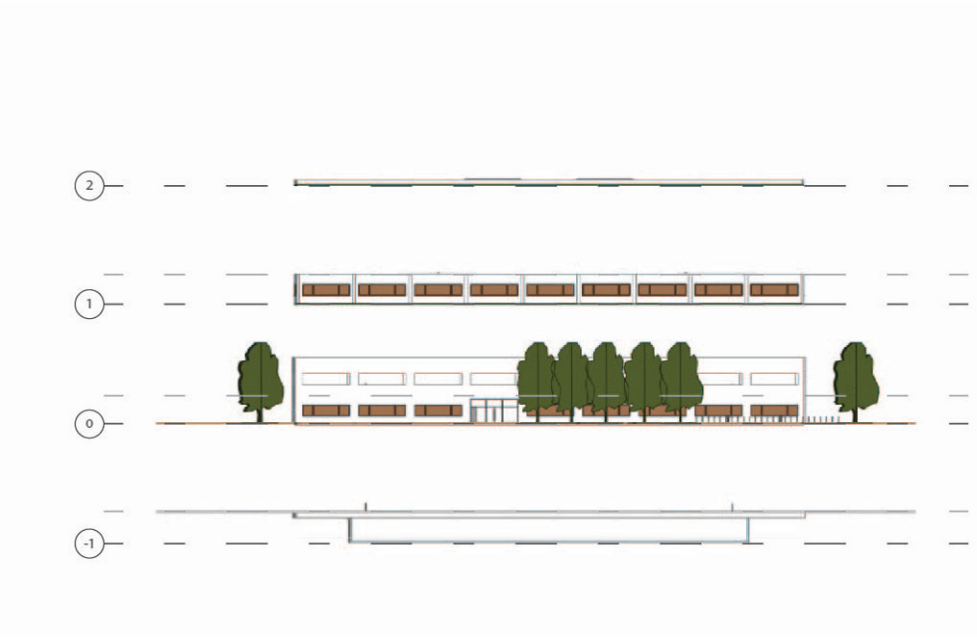


(b)

Figure D.4.: Visual result of the elevation detection process on the `RAC-sample-project` model. (a) the front isometric view. (b) an exploded view based on sorting process based on these elevations.

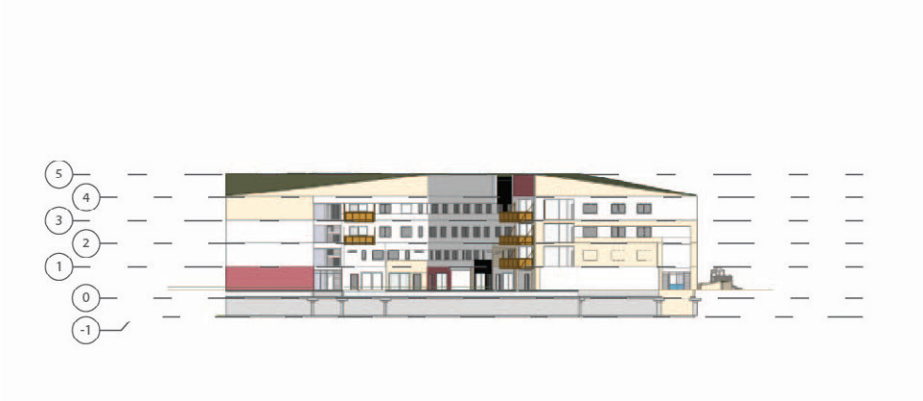


(a)

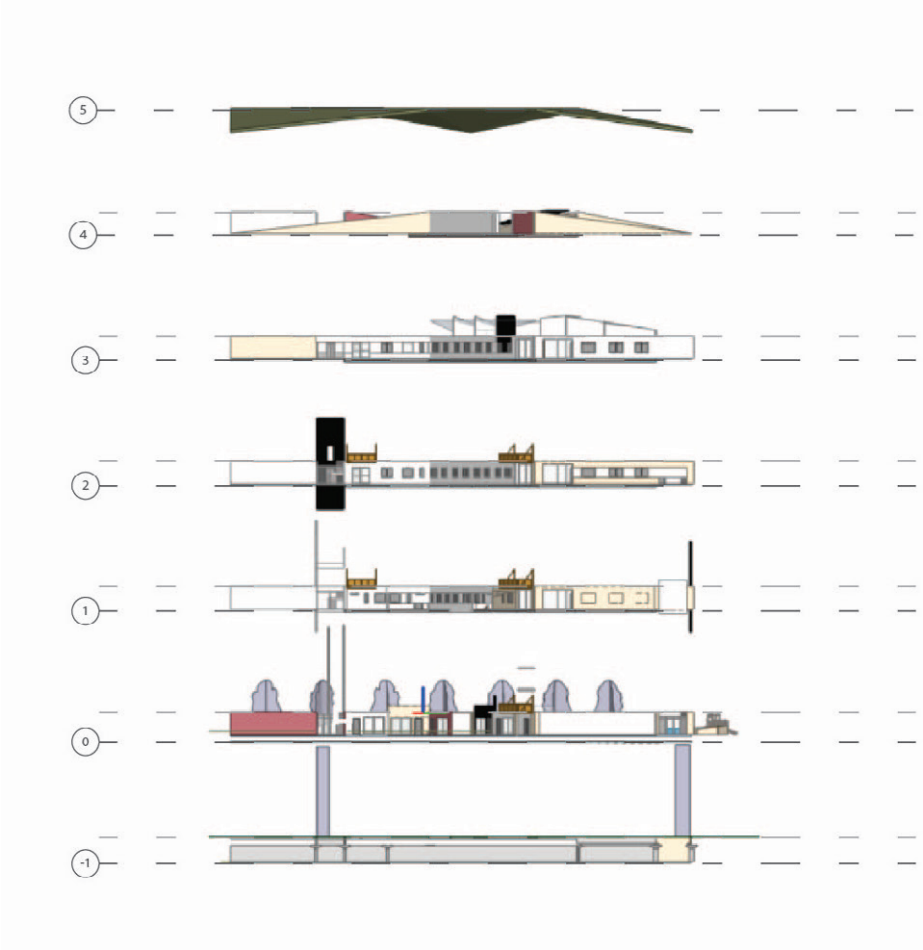


(b)

Figure D.5.: Visual result of the elevation detection process on the `DigitalHub` model. (a) the front isometric view. (b) an exploded view based on sorting process based on these elevations.

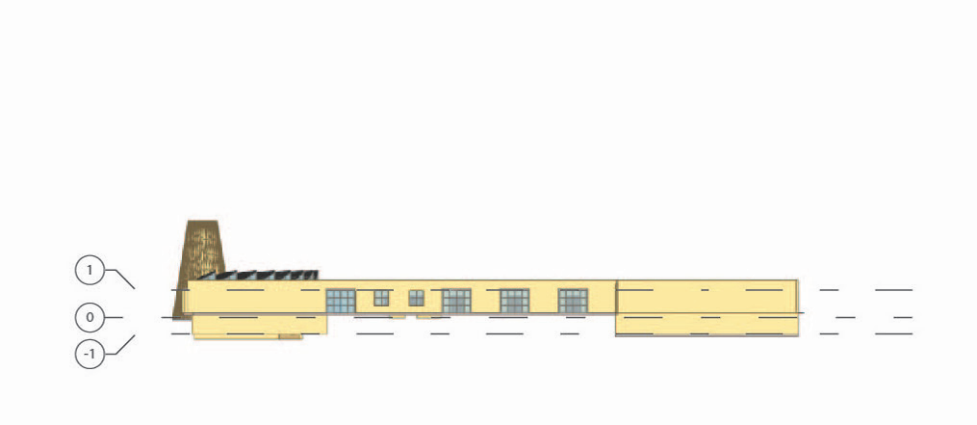


(a)

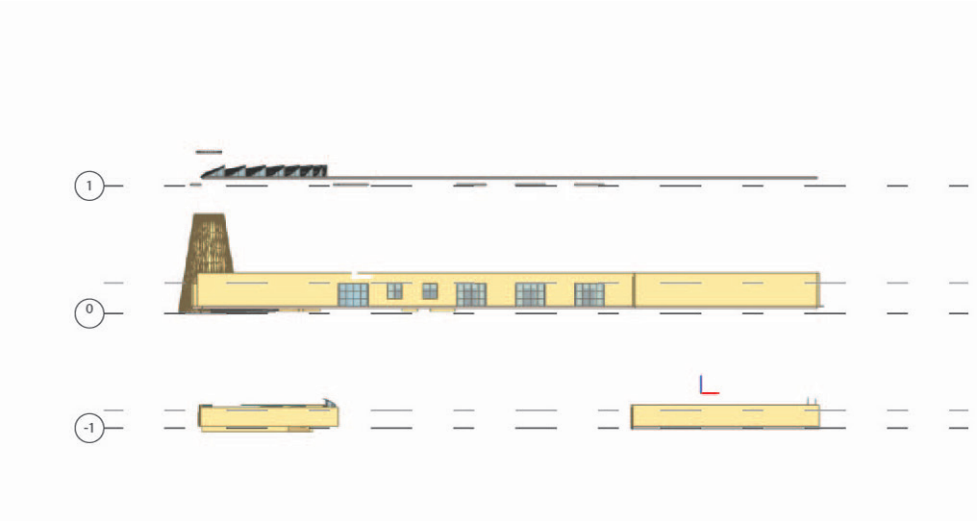


(b)

Figure D.6.: Visual result of the elevation detection process on the `ON4 building` model. (a) the front isometric view. (b) an exploded view based on sorting process based on these elevations.



(a)



(b)

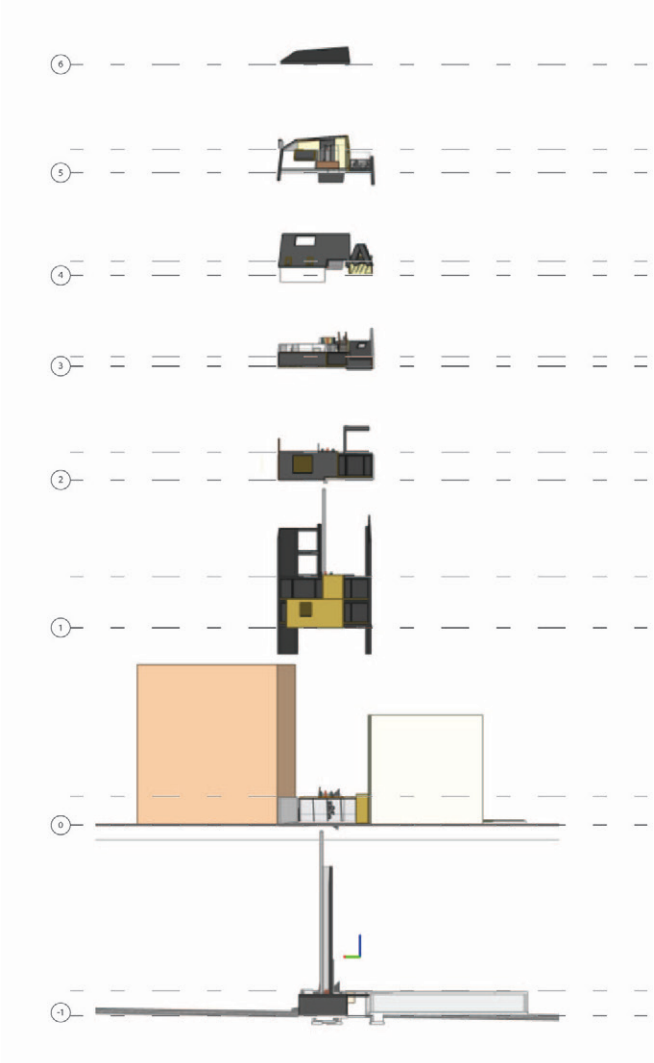
Figure D.7.: Visual result of the elevation detection process on the `CUVO building` model. (a) the front isometric view. (b) an exploded view based on sorting process based on these elevations.

(a)

(b)

Figure D.8.: Visual result of the elevation detection process on the `Witte_de_Withstraat` model. (a) the front isometric view. (b) an exploded view based on sorting process based on these elevations.

(a)



(b)

Figure D.9.: Visual result of the elevation detection process on the `Projekt Golden Nugget` model. (a) the front isometric view. (b) an exploded view based on sorting process based on these elevations.

(a)

(b)

Figure D.10.: Visual result of the elevation detection process on the top section of the `Boompjes` model. (a) the front isometric view. (b) an exploded view based on sorting process based on these elevations.

E. Adjusted Results of storey elevation detection process of the Boompjes model

Boompjes			
Storey	Expected elevation (m)	# Original elevation (m)	# Approximated elevation (m)
-3	-8.935	-8.935	-9.235
-2	?	-6.935	-6.935
-1a	?	-4.13	-4.13
-1	?	-2.7375	-1.37
0 Ground floor	0	0	-
0a Ground floor	0.93	0.9125	0.83
0b	?	2.555	-
1	5.11	5.11	5.03
2	8.17	8.17	8.09
3	11.23	11.23	11.15
4	14.29	14.29	14.21
5	17.35	17.35	17.27
6	20.41	20.41	20.33
7	23.47	23.47	23.39
8	26.53	26.53	26.45
9	29.59	29.59	29.51
10	32.65	32.65	32.57
11	35.71	35.71	35.63
12	38.77	38.77	38.69
13	41.83	41.83	41.75
14	44.89	44.89	44.81
15	47.95	47.95	47.87
16	51.01	51.01	50.93
17	54.07	54.07	53.99
18	57.13	57.13	57.05
19	60.19	60.19	60.11
20	63.25	63.25	63.17
21	66.31	66.31	66.23
22	69.37	69.37	69.29
23	72.43	72.43	71.71
24	75.49	75.49	75.41
25	78.55	78.55	78.47
26	81.61	81.61	81.53
27	84.67	84.67	84.59
28	87.73	87.73	87.65
29	90.79	90.79	90.71
30	93.85	93.85	93.77
31	96.91	96.91	96.83
32	99.97	99.97	99.89
33	102.97	-	102.31

Table E.1.: The adjusted full comparison between the stored in file storey elevations and the approximated elevations for the `Boompjes` model. The expected elevation values are based upon all the files comprising the model not only the construction model. ”?” values are at locations too complex to accurately approximate an expected elevation manually. ”-” values do not have a matching elevation for that storey.

F. Results of the apartment detection process

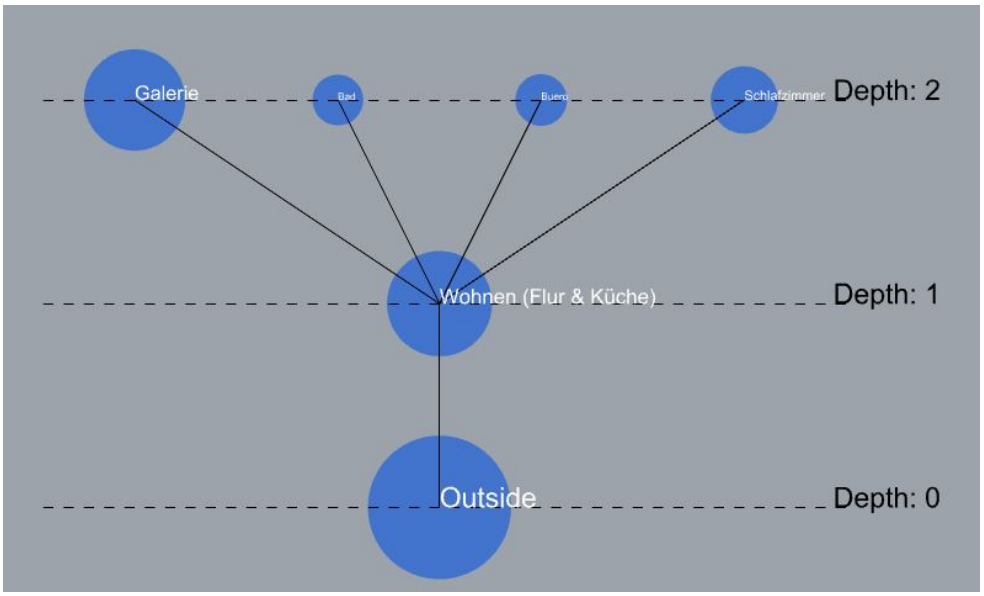


Figure F.1.: Connectivity graph of the `FZK-Haus` model.

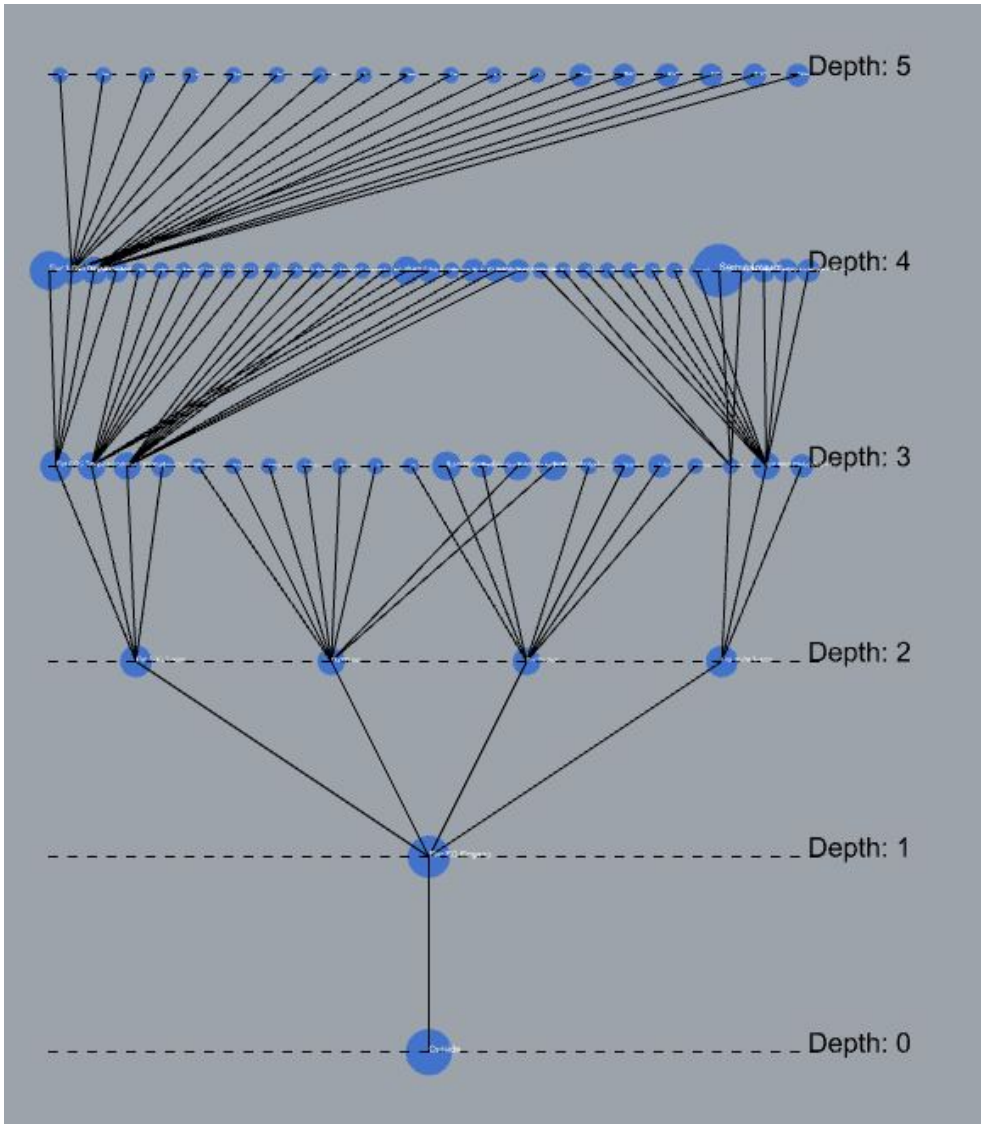


Figure F.2.: Connectivity graph of the `Institute-Var-2` model.

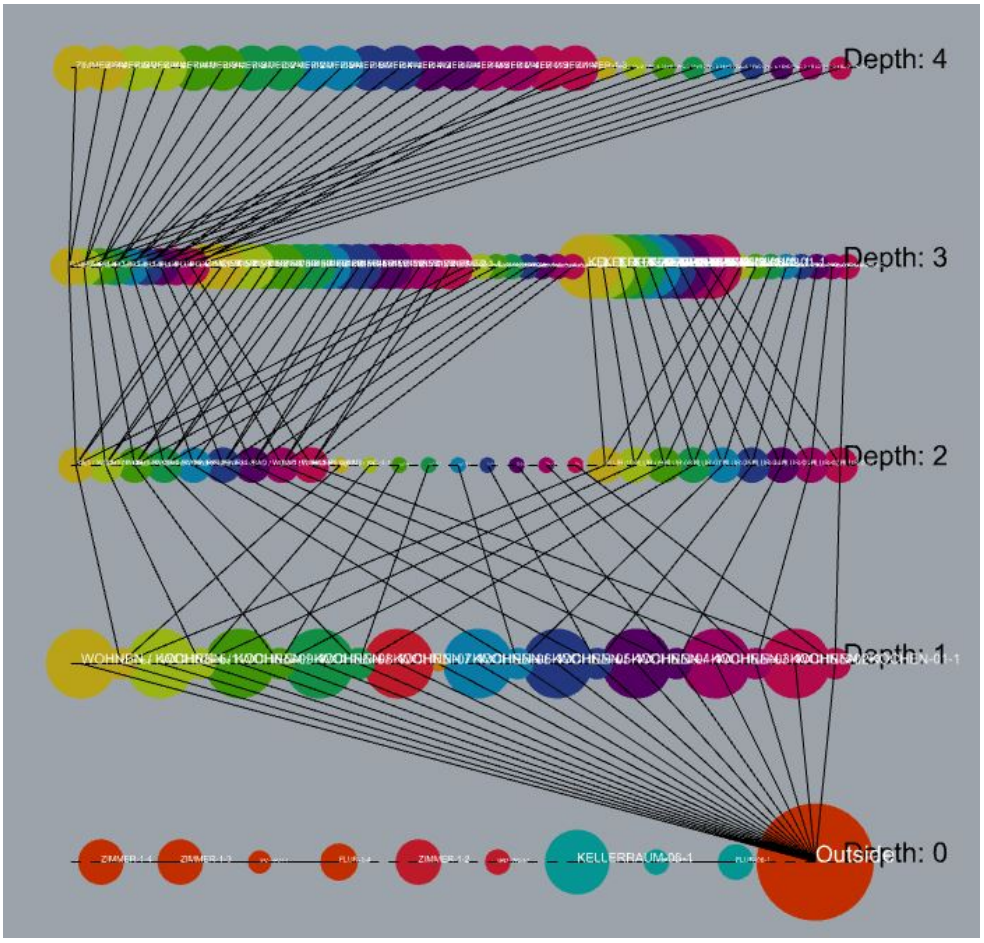


Figure F.3.: Connectivity graph of the `Smiley-West-10` model. The colors indicate different apartments/sections.

Bibliography

Ahmed, S., Liwicki, M., Weber, M., and Dengel, A. (2011). Improved automatic analysis of architectural floor plans. In *2011 International conference on document analysis and recognition*, pages 864–869. IEEE.

Ahmed, S., Liwicki, M., Weber, M., and Dengel, A. (2012). Automatic room detection and room labeling from architectural floor plans. In *2012 10th IAPR international workshop on document analysis systems*, pages 339–343. IEEE.

Arroyo Ohori, K., Diakité, A., Krijnen, T., Ledoux, H., and Stoter, J. (2018). Processing bim and gis models in practice: experiences and recommendations from a geobim project in the netherlands. *ISPRS International journal of geo-information*, 7(8):311.

Bouchlaghem, D., Shang, H., Whyte, J., and Ganah, A. (2005). Visualisation in architecture, engineering and construction (aec). *Automation in construction*, 14(3):287–295.

Hillier, B. and Tzortzi, K. (2006). Space syntax: the language of museum space. *A companion to museum studies*, pages 282–301.

Hu, Z.-Z., Zhang, X.-Y., Wang, H.-W., and Kassem, M. (2016). Improving interoperability between architectural and structural design models: An industry foundation classes-based approach with web-based tools. *Automation in Construction*, 66:29–42.

Jeong, S. K. and Ban, Y. U. (2011). Computational algorithms to evaluate design solutions using space syntax. *Computer-Aided Design*, 43(6):664–676.

Katsigarakis, K., Lilis, G. N., Giannakis, G., and Rovas, D. (2019). An ifc data preparation workflow for building energy performance simulation. In *European Conference on Computing in Construction', EC3*.

Kim, H., Jun, C., Cho, Y., and Kim, G. (2008). Indoor spatial analysis using space syntax, the. In *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences. Vol. XXXVII, Part B2*. Citeseer.

Krijnen, T., Noardo, F., Ohori, K. A., Ledoux, H., and Stoter, J. (2020). Validation and inference of geometrical relationships in ifc. In *Proceedings of the 37th International Conference of CIB W*, volume 78, pages 98–111.

Lilis, G., Giannakis, G., and Rovas, D. (2015). Detection and semi-automatic correction of geometric inaccuracies in ifc files. In *14th International Conference of IBPSA-Building Simulation 2015, BS 2015, Conference Proceedings*, pages 2182–2189. IBPSA.

Lilis, G. N., Giannakis, G. I., and Rovas, D. V. (2017). Automatic generation of second-level space boundary topology from ifc geometry inputs. *Automation in Construction*, 76:108–124.

Mura, C., Mattausch, O., Villanueva, A. J., Gobbetti, E., and Pajarola, R. (2014). Automatic room detection and reconstruction in cluttered indoor environments with complex room layouts. *Computers & Graphics*, 44:20–32.

Noardo, F., Arroyo Ohori, K., Krijnen, T., and Stoter, J. (2021). An inspection of ifc models from practice. *Applied Sciences*, 11(5):2232.

Noardo, F., Wu, T., Ohori, K. A., Krijnen, T., and Stoter, J. (2020). The use of ifc models towards automation of urban planning checks for building permit. *arXiv preprint arXiv:2011.03117*.

Nourian, P. (2016). Configraphics: Graph theoretical methods for design and analysis of spatial configurations.

Ozturk, G. B. (2020). Interoperability in building information modeling for aeco/fm industry. *Automation in Construction*, 113:103122.

Türkyılmaz, E. (2016). A method to analyze the living spaces of wheelchair users using ifc. *Procedia-Social and Behavioral Sciences*, 222:458–464.

Venugopal, M., Eastman, C. M., Sacks, R., and Teizer, J. (2012). Semantics of model views for information exchanges using the industry foundation class schema. *Advanced engineering informatics*, 26(2):411–428.

Venugopal, M., Eastman, C. M., and Teizer, J. (2015). An ontology-based analysis of the industry foundation class schema for building information model exchanges. *Advanced Engineering Informatics*, 29(4):940–957.

Xiong, Q., Zhu, Q., Du, Z., Zlatanova, S., Zhang, Y., Zhou, Y., and Li, Y. (2017). Free multi-floor indoor space extraction from complex 3d building models. *Earth Science Informatics*, 10(1):69–83.

Ying, H. and Lee, S. (2020). Automatic detection of geometric errors in space boundaries of ifc-bim models using monte carlo ray tracing approach. *Journal of Computing in Civil Engineering*, 34(2):04019056.

Ying, H. and Lee, S. (2021a). Generating second-level space boundaries from large-scale ifc-compliant building information models using multiple geometry representations. *Automation in Construction*, 126:103659.

Ying, H. and Lee, S. (2021b). A rule-based system to automatically validate ifc second-level space boundaries for building energy analysis. *Automation in Construction*, 127:103724.

Colophon

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