DIRECT ANALYSIS ON POINT CLOUDS

AUTOMATED WINDOW DETECTION FOR BUILDING ENERGY SIMULATIONS

Pantelis Kaniouras
Maria Moscholaki
Jordi van Liempt
Konrad Jarocki
Liyao Zhang

Msc Geomatics Synthesis Project

TU Delft

Gemeente Rotterdam

Direct Analysis on Point Clouds

Optional Subtitle

By

Pantelis Kaniouras
Maria Moscholaki
Jordi Van Liempt
Konrad Jarocki
Liyao Zhang

Synthesis Project 2019

Master of Science
in Geomatics

at the Delft University of Technology,
to be defended publicly on Friday June 28, 2019

Supervisors:	Ir. Edward Verbree	TU Delft
	Dr. Ir. Martijn Meijers	TU Delft

TUDelft

Preface

The Direct Analysis on Point Clouds project was performed within the framework of the Synthesis Project of the Master's Programme Geomatics for the Built Environment at the Technical University of Delft. The research was assigned by a formal executive client, in this case, the Municipality of Rotterdam (Netherlands) to a team consisting of five Master students. The diverse backgrounds of these members (Rural and Surveying Engineering, Remote Sensing, Human Geography and Urban Planning) enabled the conceptualization and enactment of the study from different perspectives. The study was conducted in 8 weeks, during which the focus was placed upon the outcome of the research itself, but also on the improvement of the learning curve and the personal development of the group. The implementation of a real-world project under time pressured circumstances, in which the knowledge gained throughout the first of part of this studying track was used, was the most apposite closure to our first year as students at the TU Delft Geomatics Programme.

In the process of realizing this project, acknowledgements should be expressed to:

Ir. Edward Verbree and Dr.ir. Martijn Meijers, for their guidance, encouragement and useful critiques.

Dr. Liangliang Nan for providing us with part of his research algorithm, namely Random sample consensus (RANSAC) and Easy3D library.

Christian Wisse, Maarten Vermeij, and Christian Veldhuis of the Municipality of Rotterdam, for their participation and cooperation in the Synthesis Project of Geomatics Masters.

Dr. Iman Zolanvari (Postdoctoral Research Fellow at Trinity College Dublin) for his expert advices and generous support.

Pantelis Kaniouras
Maria Moscholaki
Jordi van Liempt
Konrad Jarocki
Liyao Zhang

Delft University of Technology
Delft, June 2019

Contents

1 Introduction **10**
1.1 Project overview . . . 11
1.2 Problem definition . . . 11
1.3 Requirements of the Analysis . . . 13
1.3.1 MoSCoW Rules: . . . 13
1.3.2 Functional/Non Functional Requirements . . . 14

2 Theoretical Concepts **15**
2.1 Point cloud data acquisition . . . 15
2.1.1 Laser scanning techniques . . . 15
2.1.2 Photogrammetry techniques . . . 16
2.2 Point cloud indexing . . . 17
2.2.1 Kd-tree . . . 17
2.2.2 Nearest neighbour search (NNS) . . . 18
2.2.3 R-tree . . . 19
2.3 Segmentation . . . 19
2.3.1 Segmentation methods . . . 19
2.3.2 Random Sample Consensus segmentation (RANSAC) . . . 20
2.4 Morphological filtering . . . 22
2.5 Raster filtering . . . 23
2.5.1 Median . . . 23
2.5.2 Gaussian . . . 24
2.6 Web Map and Web Feature Service . . . 25
2.7 BAG . . . 26
2.8 Calibration of camera . . . 26
2.9 Software . . . 31
2.9.1 Fiona . . . 31
2.9.2 Shapely . . . 31
2.9.3 OWSLib . . . 32
2.9.4 Opencv . . . 32
2.9.5 PDAL . . . 32
2.9.6 Psycopg2 . . . 32
2.9.7 DBSCAN . . . 33
2.9.8 RANSAC . . . 33

3 Methodology **33**
3.1 Overview . . . 33
3.2 Dataset Filtering . . . 34
3.3 Linking points to BAG . . . 36
3.4 Segmentation . . . 36

3.4.1 Histogram-Based Segmentation Method 37
3.4.2 RANSAC Segmentation 38
3.5 Window Detection 39
3.5.1 Projection of 3D points in 2D plane 39
3.5.2 Slicing method 41
3.5.3 2D bitmap occupancy map method 43

4 Quality assessment **46**
4.1 Dataset filtering assessment 46
4.1.1 Results 46
4.1.2 Overall quality 47
4.2 Window detection and surface ratio calculation assessment 49
4.2.1 General Principles of Assessment 50
4.2.2 Benchmark data preparation 52
4.2.3 Assessment results 55
4.2.4 Assessment results - general conclusions 66

5 Applications **67**
5.1 Benefits of a direct point cloud analysis 67
5.2 Energy usage estimations 68
5.3 Window detection used in tunneling construction for risk assessment 70
5.4 Window detection for the simulation of high-rise building fires 70
5.5 Indoor applications 71

6 Conclusions **71**
6.1 Capabilities and Limitations 72
6.2 Recommendations - Future Work 74

7 Project Plan **75**
7.1 Meetings 75
7.2 Organizational and Work Breakdown Structure 75
7.3 Work Plan 77
7.4 Rich Picture 79

References **83**

A Appendix - Morphological filter comparison **85**

B Appendix - Point cloud filtering quality **89**

C Appendix - Project management **95**

D Appendix - Essential parts of code for bitmap method **96**

E Appendix - Detailed tables for specific floors **99**

F Appendix - Dataset filtering code **100**

G Appendix - BAG-linking code **103**

H Appendix - Bitmap points generation code **106**

I Appendix - Extreme points extracting by slicing method code **111**

J Appendix - Clustering of extreme points from slicing **121**

K Appendix - Bitmaps before and after windows detection. **125**

List of Figures

1 Research area: Noordereiland, an island within the municipality of Rotterdam. 12
2 Moscow Rules 14
3 Scanning Techniques 16
4 Dense Image Matching 17
5 Kd-tree indexing 18
6 Nearest Neighbour Search [9] 18
7 R-tree structure and overlapping of rectangles 19
8 RANSAC plane fitting 21
9 Median Filtering Example 24
10 Gaussian Curves for σ = 11, σ = 7, σ= 2 25
11 Example of BAG information for building 26
12 Pinhole camera model 27
13 Transformation of object to camera coordinates principles 30
14 Workflow 34
15 Dataset filtering workflow 34
16 The created equally spaced points on BAG features. Each point colour represents a different BAG ID. 36
17 Facades of buildings detected by using the Histogram-Based Segmentation Method 37
18 Planes segmented by refined RANSAC algorithm 39
19 Example of plane projection in 2D 40
20 Horizontal Slicing and resulting extreme points 42
21 Vertical Slicing and resulting extreme points 42
22 Extreme points derived from both horizontal and vertical slicing 43
23 The diagram showing workflow of the bitmap preparation. The numbers refers to sigma for Gaussian filter and size of kernel for median filter 44
24 An example of a part of the street that has not been removed by PMF (left), but has been removed by SMRF (right) 47
25 The difference between before (top) and after (below) removal of inside points 48
26 Subset of clusters of facade points coloured per unique BAG ID property, with on the background the boundaries of BAG features. 49
27 Clustered windows retrieved after applying DBSCAN. The polygons that are created will be used to calculate the windows-facade area ratio. 52
28 Facade 2 with fake window frames 54
29 Facades 4,5 and 6 - similar in shape and size of windows. 54
30 Obtained data from benchmark dataset and bitmap method. 55
31 Required results neccessary to provide final assessment. 55
32 Final results of the assessment. 56

33 Plot of differences of height and width in % of benchmark width and height for every window class. Classes 1.2 and 3.3 were removed because the algorithm failed to find any example of window for every of them. 56
34 Plot of differences of area in % of benchmark area for every window class. . . 57
35 Table with results for facade 2. 59
36 Table with results for facade 3. 60
37 Table with results for facade 4. 60
38 Table with results for facade 5. 61
39 Table with results for facade 6. 61
40 Plot of differences of area in % of benchmark area for every window - alpha shape areas 62
41 Plot of differences of area in % of benchmark area for every window - bounding box areas. 63
42 Plot of differences of area in % of benchmark area for every facade. Two approaches of area estimation were taken into consideration: area of bounding box around the shape and area of the shape. 64
43 Plot of differences of area in % of benchmark area for every floor. Two approaches of area estimation were taken into consideration: area of bounding box around the shape and area of the shape. 65
44 Comparison of the results obtained with Bitmap and Slicing Method. The numbers represent the different in % of benchmark facade. For Slicing Method, two sub-methods were assessed separately and for every facade the most and at least accurate results were marked (green and red colour respectively). . . 66
45 Organizational and Work Breakdown structure 77
46 Rich Picture 79
47 Point cloud subset before ground filtering 85
48 Point cloud subset after PMF filtering, with notable spots marked in the red boxes 86
49 Point cloud subset after SMRF filtering, with notable spots marked in the red boxes 87
50 Some points of undefinable objects are present close to these facades 89
51 Part of a car that is close to a facade is not entirely removed 90
52 A few objects in front of the corner building are not removed 91
53 A part of the original cloud before filtering 92
54 A part of the original cloud after filtering 93
55 Workplan 95
56 Detailed tables for specific floors 99
57 Facade 1 - bitmap before and after windows detection. 125
58 Facade 2 - bitmap before and after windows detection. 125
59 Facade 3 - bitmap before and after windows detection. 126
60 Facade 4 - bitmap before and after windows detection. 126
61 Facade 5 - bitmap before and after windows detection. 127

62 Facade 6 - bitmap before and after windows detection. 128

ACRONYMS

LIDAR	Light Detection And Ranging of Laser Imaging Detection And Ranging
DIM	Dense Image Matching
PostGIS	Spatial and Geographic objects for PostgreSQL
PostgreSQL	Free and open-source relational RDBMS emphasizing
SQL	Structured Query Language
RDBMS	Relational Database Management System
BAG	Basisregistratie Adressen en Gebouwen
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
OGC	Open Geospatial Consortium
LoD	The Level of Detail
LASER	Light Amplification by Stimulated Emission of Radiation
NTA	National Testing Agency
PDAL	Point Data Abstraction Library
XML	Extensible Markup Language
GEOS	Geometry Engine OpenSource
RAM	Random-Access Memory
RANSAC	Random Sample Consensus
DIM	Dense Image Matching
DIM	Dense Image Matching
PMF	Progressive Morphological Filter
SMRF	Simple Morphological Filter
NNS	Nearest neighbor search
UAV	Unmanned Aerial Vehicle

Summary

With the rapid growth in point cloud acquisition technologies the recent years we have the ability to measure large quantities of 3D points of significantly detailed and geometrically composite scenes such as urban environments [1]. This advantage can be exploited and used for direct analysis on point clouds. A direct point cloud analysis has several advantages over for example 3D surface reconstruction, such as the end result having more details and the computation being less expensive. In order to make a point cloud representation a suitable alternative for other types of 3D city models, they need to be semantically enriched, resulting in a rich point cloud.

One element of this enrichment is the detection of objects, such as windows. Extracting these from facades is specifically what this research revolves around, which can be done by taking advantage of the fact that they show up as holes, since lasers of the point cloud scanner do not properly reflect on them. Two different general approaches are taken to detect windows in a by mobile laser scanner obtained point cloud of Noordereiland, Rotterdam, The Netherlands.

The first method is the so-called 2D bitmap occupancy map method. Here, the point coverage of a facade is approximated in a bitmap. A Gaussian or median filter is then applied, to remove noise. Subsequently, holes can be extracted from this bitmap, which would represent windows. The boundary pixels of the holes are tied to original points, which means that these will be selected to be window points.

The other method is the slicing method, derived from an existing paper, in which a facade is cut into both horizontal and vertical slices. All points that belong to a slice are projected to 1 dimension, and based on a distance threshold between the points on the line, holes can be detected. The extreme points of these holes are extracted, and after doing this for every slice, the boundary points of all holes in the facade are detected. These points would then represent the outlines of windows, and are classified as such. The quality of both methods is assessed and compared.

Besides the enrichment of the point cloud with classified window points, the ratio between the surface of a facade and the total surface of its windows is calculated and added as an attribute as well, which is useful for the estimation of the energy usage of a building. Moreover, other applications for which this analysis can be used, are presented. Window detection in tunneling risk assessment and fire propagation simulation in high-rise buildings are two of these applications, which were the study objective of other research papers.

1 Introduction

In recent years, laser scanning technology became crucial in surveying and urban managing. The increasing possibilities driven by developing technologies, give the opportunity to acquire, measure and interpret the surrounding environment on detailed level and with high accuracy. However, big possibilities are always coming with big challenges – and one of them is computationally- demanding processing of point clouds. Obviously, the simplest solution for deriving information are basing on geometrically reconstructed objects. However, the effectiveness of this approach decreased dramatically together with increasing density or size of the datasets, forcing the scientist to find more suitable solution to face fast growing demands. The basic idea behind most of them, is to derive information of interest directly from a point cloud – without intermediate steps requiring expensive hardware - which was also the main assumption behind the idea of creating this project. With the help of Municipality of Rotterdam, the general idea of this analysis was related to a specific problem - assessment of energy effectiveness of buildings with respect to window coverage in facade surfaces. The size of the provided datasets (forcing to find solution without geometrical reconstruction), as well as a lot of emphasis in put on ecological issues, made the problem interesting from the point of view of the group.

Building energy performance results are becoming increasingly important for design decisions concerning building sustainability that have to be made in the early design stage [2]. However it is equally important to reduce the energy consumption of existing buildings. Windows are a significant source of heat loss in a building. Therefore, any effort to significantly improve building energy efficiency and reduce environmental impacts [3] is a step towards being more compliant with regulations, as the governments stipulates a minimum efficiency requirement for both new build and existing buildings. In general it would be of bigger importance for the occupants to have temperature control and thermal comfort, but mainly a further sustainable decrease in energy and utility costs which in turn, would unlock further potential for growth due to the extra purchasing power made available this way. In this context, undertakings may profit from both new market opportunities opening up in sectors engaged in rationalizing energy use, and more predictable and stable energy costs [4].

The primary objective of this study is to provide a preliminary solution that automatically and rapidly detects windows in existing buildings from LIDAR point cloud data that can be further utilized for building performance simulation applications. We process the provided point cloud data from a terrestrial and airborne laser scanner system to recognize and distinguish the windows as an individual building component that will be later on used by the client, for the calculation of applications like the overall heat loss coefficient of a building from windows, as mentioned above.

1.1 Project overview

This report provides an overview of the ”Automatic detection of windows from point clouds” research, as part of the Geomatics Synthesis Project 2019. The aim of this course is to apply the knowledge, skills and insight acquired in prior mandatory courses of the programme into practice, making it possible to combine scientific research with practical work. This way the opportunity was given to combine and implement the knowledge gained from the following courses:

– Digital Terrain Modelling (GEO1015)

– Geo Database Management Systems (GEO1006)

– Positioning and Location Awareness (GEO1003)

– 3D Modelling of the Built Environment (GEO 1004)

– Geoweb Technology (GEO1007)

– Geo Datasets and Quality (GEO1008)

– Python Programming for Geomatics (GEO1000)

– Sensing Technologies for the Built Environment (GEO1001)

The following research starts with providing the problem definition and also the requirements of the project as a whole (Chapter 1). Hereafter the theoretical background of every methodology milestone is presented thoroughly (Chapter 2), followed by the applied methodology that is necessary to derive to the results (Chapter 4). The next chapter is devoted to the quality assessment of the developed algorithm. Finally the conclusions (Chapter 5) as well as the project plan framework (Chapter 6) are provided .

1.2 Problem definition

The Municipality of Rotterdam has to perform various kinds of analysis by using point cloud data, acquired with different methods. Its goal is to use these for various applications. In this concept, it is of their interest to try to explore for potential types of analysis on point cloud data that can be used to their benefit, in terms of accuracy, accessibility, time management, cost savings in the long term and for several current or prospective projects and applications.

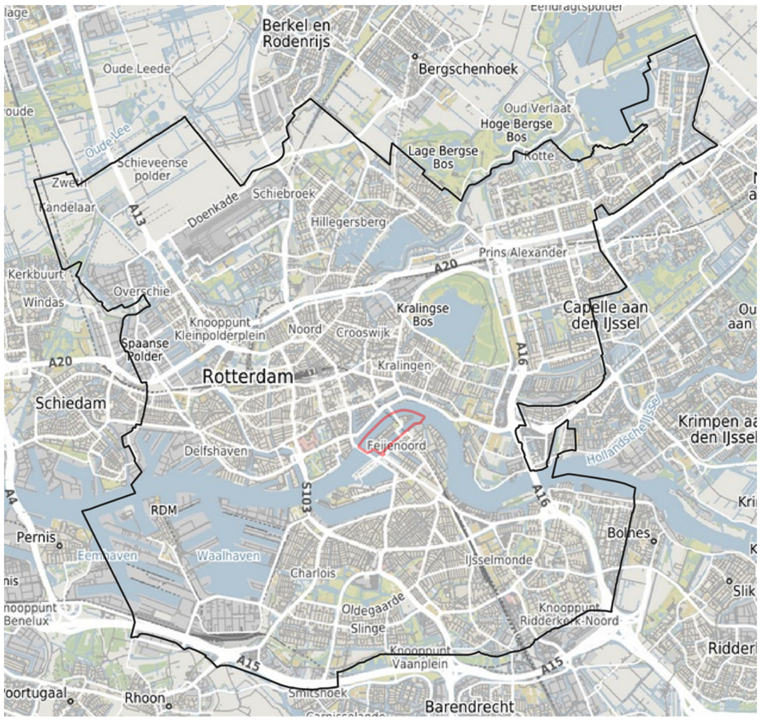


Figure 1: Research area: Noordereiland, an island within the municipality of Rotterdam.

Objective: To create a window detection algorithm that takes as input point cloud data and detects windows without the need of 3D surface reconstruction. Moreover it should be able to calculate the ratio between the window surfaces and total surface of buildings facades and also identify individual windows per facade.

Research question:

The research question defines the scientific goals that the team would like to explore and try to provide an answer to. Through this, a general direction is chosen on which the main focus will be.

To which extent can the immediate use and analysis of a point cloud be used for the specific purpose of window detection without the need of 3D Surface Reconstruction?

Therefore the main research question for this project is composed of the following sub-questions:

- What is the accuracy of the implemented algorithms?
- Is this analysis truly usable and practical for the current but also other applications?

- What is the efficiency level of the implemented algorithm?
- Can the result of this analysis be used in combination with other datasets (e.g BAG) or applications (e.g estimation of ratio between window - building surface areas per BAG ID)?

The research of the team will be focused on the detection of windows since this is the most important part of the application. Trying to achieve the best accuracy, in terms of the exact shape of the window and therefore the area of it will lead to a better and more precise outcome for the algorithm which will result in a more reliable application for the calculation of the window coverage ratio in facades.

1.3 Requirements of the Analysis

1.3.1 MoSCoW Rules:

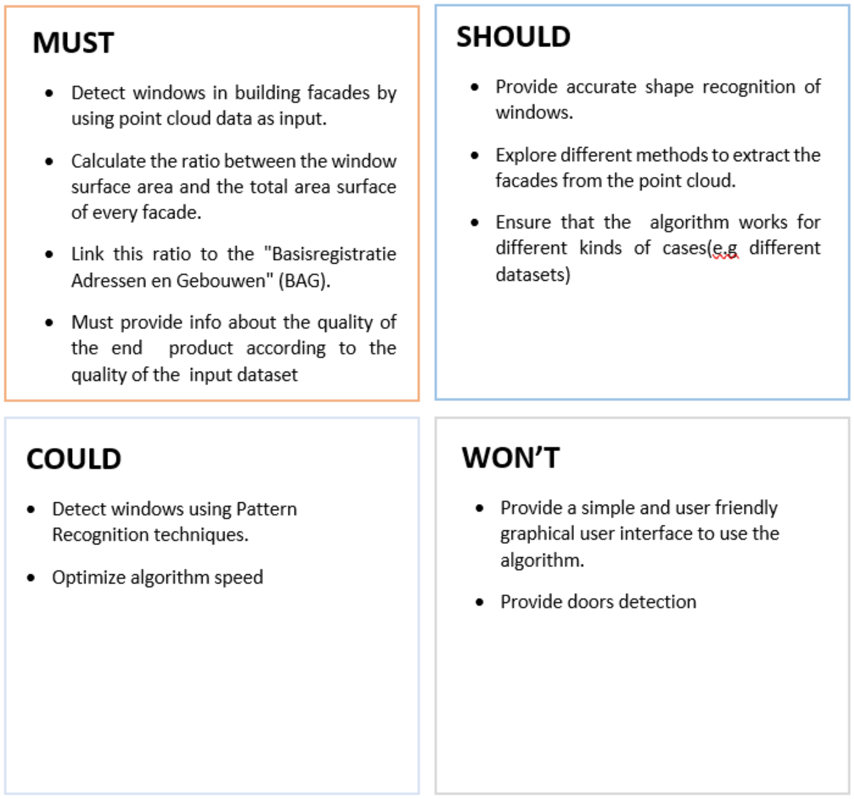
MoSCoW is a technique used in management and software development which aims to ensure that all stakeholders have a common understanding on the importance they place on the delivery of each requirement [5]. Every requirement is characterized as a MUST, SHOULD, COULD and WON'T depending on its importance within the projects realization.

Requirements labelled as MUST are the ones that have to be satisfied to acquire a final result, otherwise if absent, the implementation of the project might be considered as well as a failure.

SHOULD requirements are also important additional requisites, but are not vital for the success of the end product. They can be characterized as secondary priority, as the they cannot create a big risk for the project itself in case this requirements are not met.

COULD requirements are the ones which that are desirable or interesting to implement but not significant for the outcome. They do represent an enhancement of certain features used in the process that are not necessarily important but can be put into practise if the project is not highly - constrained in time just to increase client satisfaction.

WON'T requirements are concepts that the team will not be able to deliver. However they can be labelled as possible future work as the stakeholders are actually interested in this requirements but just have agreed that they will not be implemented as they are not achievable within the framework of the current application.



MUST

- Detect windows in building facades by using point cloud data as input.
- Calculate the ratio between the window surface area and the total area surface of every facade.
- Link this ratio to the "Basisregistratie Adressen en Gebouwen" (BAG).
- Must provide info about the quality of the end product according to the quality of the input dataset

SHOULD

- Provide accurate shape recognition of windows.
- Explore different methods to extract the facades from the point cloud.
- Ensure that the algorithm works for different kinds of cases(e.g different datasets)

COULD

- Detect windows using Pattern Recognition techniques.
- Optimize algorithm speed

WON'T

- Provide a simple and user friendly graphical user interface to use the algorithm.
- Provide doors detection

Figure 2: Moscow Rules

1.3.2 Functional/Non Functional Requirements

Apart from defining the main objectives of the project, it is necessary to determine the conditions and needs that must be met by an end product based on the requirements of the involved stakeholders. Although there are various types of requirements, the most important ones are the functional and non-functional. The functional requirements identify the necessary task, action or activity that must be accomplished for the project to run successfully and the non-functional requirements specify the criteria for assessing the outcome.

Functional Requirements: Data Processing

- Read point cloud datasets from different sources
- Clean/Filter Data
- Perform analysis on point cloud data
- Visualize Results

Functional Requirements: Softaware Tools

- QGIS: Manipulation of data
- Python 3.7: Algorithm development
- C++ (Visual Studio Compiler 2017): Algorithm development
- CloudCompare: Visualization and Processing
- Libraries Python: Algorithm development
- Libraries C++ : Algorithm development
- Image Master: Calibration of camera

Non-Functional Requirements:

- Accuracy: To try to ensure that the project outcome offers accurate enough results.
- Usability/Efficiency: To try to ensure that the implemented algorithm for the window detection is easy to understand, user friendly and most importantly performance and time wise efficient

2 Theoretical Concepts

2.1 Point cloud data acquisition

There are a number of well established surveying techniques, active or passive, for the acquisition of (geo)spatial information. A wide variety of instruments is commercially available, and a large number of companies operationally use these techniques, accompanied by many dedicated data acquisition, processing and visualization software packages [6]. A thorough inside in these techniques was given during the GEO101, GEO1004 and GEO1001 courses.

2.1.1 Laser scanning techniques

Typically, active techniques measure 1) the distance to the surface they are scanning and 2) their own position and orientation relatively to a global coordinate system. By combining these, the 3D coordinates of the measured location on the target surface can be computed [7]. Sensors/scanners, either airborne or terrestrial, are categorized as active methods. A LIDAR system measures distance to a target by generating and casting upon it pulses of laser light and then using a sensor to capture and measure the refleced pulse. (see Figure 3). It is generally considered to be the most accurate acquisition technique. By measuring the time-of-flight, i.e the difference in time between sending a pulse and sensing its return, the distance to the target that reflected the pulse can be found by multiplying the time-of-flight with the speed of light, which is a known constant [7]. Another way for measuring 3D points is with Tachymetry. This method does not provide point clouds in the sense defined before. In this case specific points of interest are measured. However, also in this case the accuracy that is provided is very high. The following figure (3) was derived from [8].

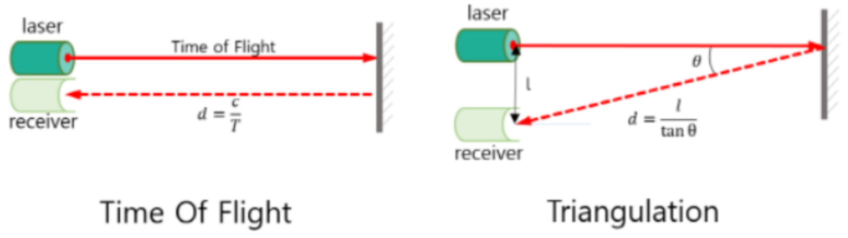


Figure 3: Scanning Techniques

2.1.2 Photogrammetry techniques

Photogrammetry (aerial or terrestrial) gives us as the possibility to measure 3D points. Due to the recent development of computer vision, we are able to acquire detailed and of good quality point clouds from images that were captured with uncalibrated simple cameras [6]. Photogrammetry is based on creating the geometry of a 3D feature by capturing the same object from different perspectives. Every separate image that is taken consists of pixels, each of which is connected to the center of the camera through a line, i.e the are collinear. The 3D position of the point can be identified by finding the intersection point location of these lines from different images. This however requires a further processing called Dense Image Matching (DIM) (Figure 4) [9]. With dense image matching it is attempted to find a match for every pixel in an image [6]. The following figure (4) was derived from [9].

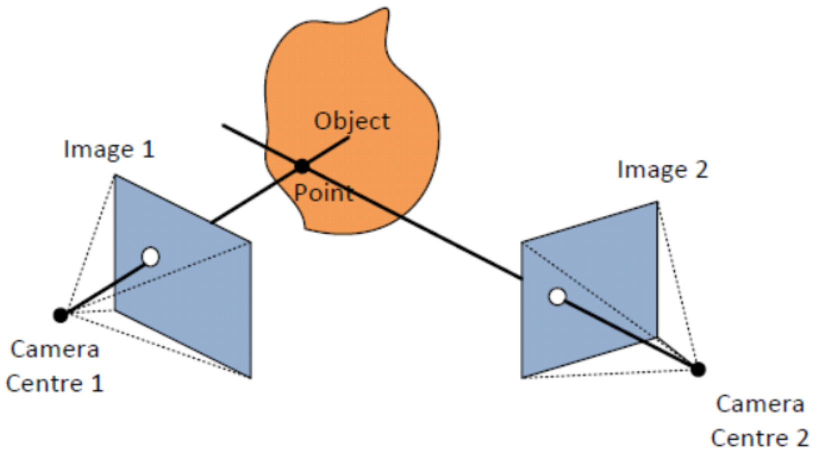


Figure 4: Dense Image Matching

2.2 Point cloud indexing

A spatial index is a data structure that was created to enable access and query operations in spatial databases. Otherwise the response time of a specific spatial feature would require a lot more time, as every record would be checked consecutively [10]. Therefore data structures are designed to enable fast access to spatial data. The used data structures for indexing and searching point clouds in the implementation of this projects algorithm are presented hereafter. Point cloud indexing was introduced during the courses GEO1015 and GEO1006.

2.2.1 Kd-tree

A k-dimensional tree is a data structure to organize points in a k-dimensional space [11]. Through this, the extended is also divided in different regions [11]. It is based on the concept of a binary search tree in which each node is used to partition one specific dimension. More analytically, each of its nodes represents an axis-aligned hyper-rectangle, as Figure 5 shows. Moreover, every node designates a specific axis and the points of the dataset are then separated and placed, by comparing their coordinate in this axis to a certain value in terms of being bigger or smaller, such as a coordinate median. This way, all points are indexed.As described in [11] and the hyperplanes are created in the following way: First, we choose one dimension as split axis and a splitting point which is the middle of the existing values in this axis. Afterwards, every point is positioned in the tree according to its specific axis-value and relatively to the middle point. This, as stated in [10], makes ”points with a lower coordinate value than the node along that dimension (corresponding to ‘left’ or ‘under’ the hyperplane) being put into the left subtree of the node, and the other ones into the right subtree” [11]. The following figure (5) was derived from [11].

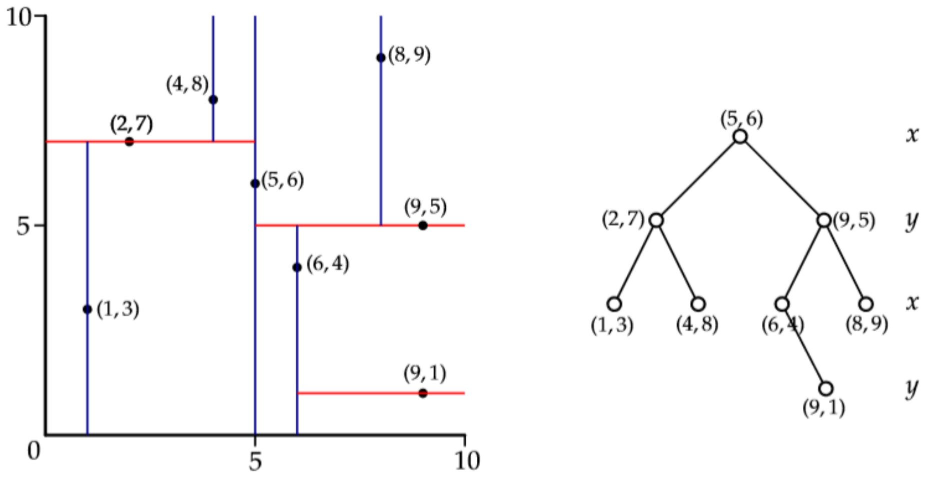
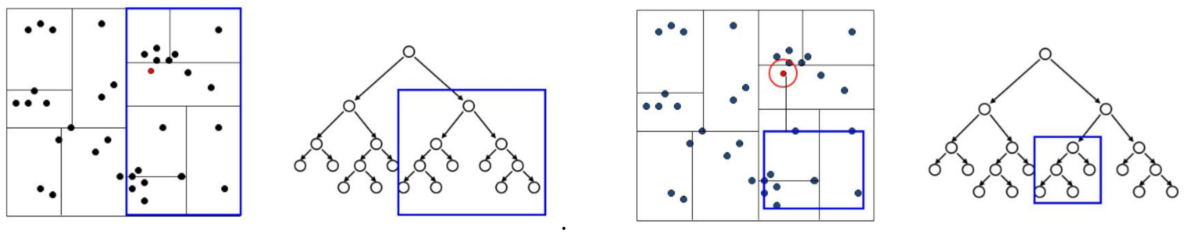


Figure 5: Kd-tree indexing

2.2.2 Nearest neighbour search (NNS)

Kd-trees, are a compact spatial data structure, hence they are well suited for important tasks like the location of nearest neighbours (NN). As mention in [11] ”the nearest neighbour query aims to find the point c in a set S that is the nearest (according to the Euclidean distance) to a query point q” and for this ”the whole tree has to be traversed (in depthrst order)”. The properties of the kd-tree offer the possibility to eliminate large portions of the tree. based on their bounding boxes. A generic k nearest neighbours search algorithm begins at the root of the kd-tree and adds points to a list if they are within a certain distance. For the k nearest neighbours, the list is sorted such that the point that most distant can be eradicated if the list contains k points and a new closer one is found [11]. More analytically, the first step is to locate the location where the point would be located if it were added to the tree (Figure 6). As the tree is traversed the distance between the point and the current node is recorded. Next, we go back up the tree evaluating each branch of the tree that could have points within the current minimum distance. The following figure (6) was derived from [10].



(a) Exploration of branch closest to query point (b) Elimination of part with no NN

Figure 6: Nearest Neighbour Search [9]

2.2.3 R-tree

R-tree is another index structure for partitioning space. Also here, the principle of partitioning data into axis-aligned tree nodes, is shared. In [12] it is stated that "the key idea of this specific data structure is to group nearby objects and represent them with their minimum bounding rectangle in the next higher level of the tree; the "R" in R-tree is for rectangle". Compared to the kd-tree which can usually contain only points, the R-tree can have coordinates, rectangles or polygons as objects. Another property of the R-tree is that its rectangles might overlap (Figure 7) and do not always cover the whole data space. Empty areas may be uncovered. The process starts with partitioning the data into a specific number n of rectangular boxes. Then each box is sorted into the same n number of smaller boxes. The same process is repeated until the final box contains n number of rectangles at most. R-tree structures are very useful since most normal range queries don't work for geometry (only for point data) so some spatial operations (e.g. intersect) can be enabled through the use of the R-tree. The following figure (7) was derived from [12].

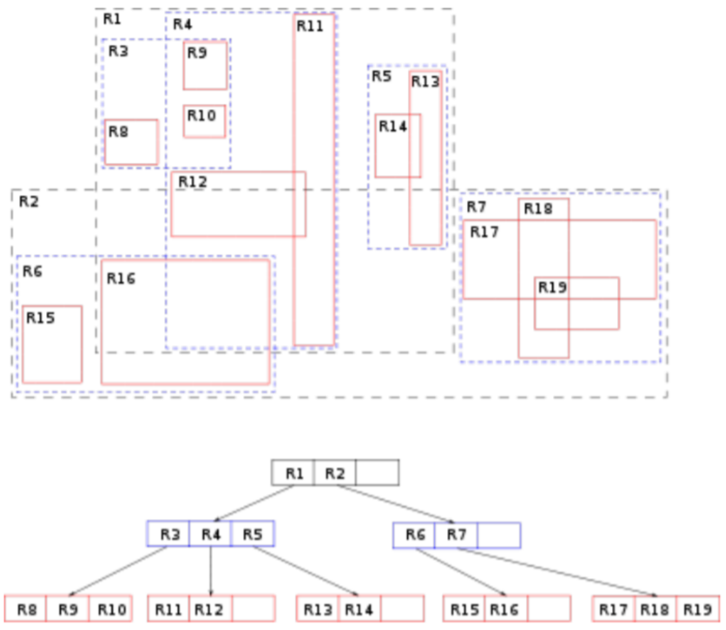


Figure 7: R-tree structure and overlapping of rectangles

2.3 Segmentation

2.3.1 Segmentation methods

Segmentation is one of the most important processes in the direct analysis of unorganized point clouds, since normally point clouds are unstructured 3D data (with few exceptions). It utilizes the generation of classified/ segmented point clouds. The main idea is to group points that share the same properties into homogeneous regions. It is considered a very challenging process as it has to deal with various matters such as uneven point density, high redundancy and absence of specific structure. There are different segmentation techniques used for different primitive extraction purposes (e.g edges, smooth regions). As described by Anh Nguyen and Bac Le in [13], these segmentation methods can be assorted in the following categories:

1. **Edge based methods**: The boundaries of objects are detected and points which belong to these are grouped together.
2. **Region based methods**: Points in a specific search neighbourhood with similar characteristics are detected and grouped in segments that share the same similarity.
3. **Attributes based methods**: The classification of the points is based on attributes that are computed for all points.
4. **Model based methods**: Points that can be represented with the same mathematical surface e.g plane, cylinder etc., are grouped together. RANSAC is one well known example of shape fitting algorithms.
5. **Graph based methods**: Segmentation is performed with a graph in which the nodes represent the points and the edges the connection of points that are spatially close.

2.3.2 Random Sample Consensus segmentation (RANSAC)

Random Sample Consensus (RANSAC) is used as one of the most common segmentation methods to detect planes. It provides robust models with a certain probability, meaning that it is able deal with the presence of outliers (values that differ abnormally in a random sample from the the population). Most parameter estimation techniques are based on the assumption that the noise is following a Gaussian distribution (mean is at zero) and therefore it would be smoothed after optimizing these parameters which are taken as an average of all the data. This does not apply in the case of gross erroneous samples. The Ransac paradigm, however is capable of interpreting data that contain a big percentage of gross errors [14] as it trains the model only with inliers according to an optimization method and the distance function of the model [15]. However as mentioned in [16], it suffers from the spurious-plane problem when noise and outliers exist, due to the uncertainty of randomly sampling the minimum subset with 3 points.

The main idea of the algorithm according to George Vosselman and Hans-Gerd Maas in [6], is based on the notion of minimal sets. Minimal set, is considered the minimum number of points that are required to define a plane (three points). The process starts by choosing a number of subsets of 3 points for each of which a plane is fitted. Afterwards, all the points that are inliers, i.e. are within a specific tolerance (distance) from the plane, constitute the consensus set as named in [6]. All the points that do not belong in the set are gross errors and are considered outliers, as they do not fit in the model. Once all the consensus sets have been formed, the model with the biggest consensus set, is the preponderant plane. For even better results, least square adjustment can be performed on the prevailing consensus set.

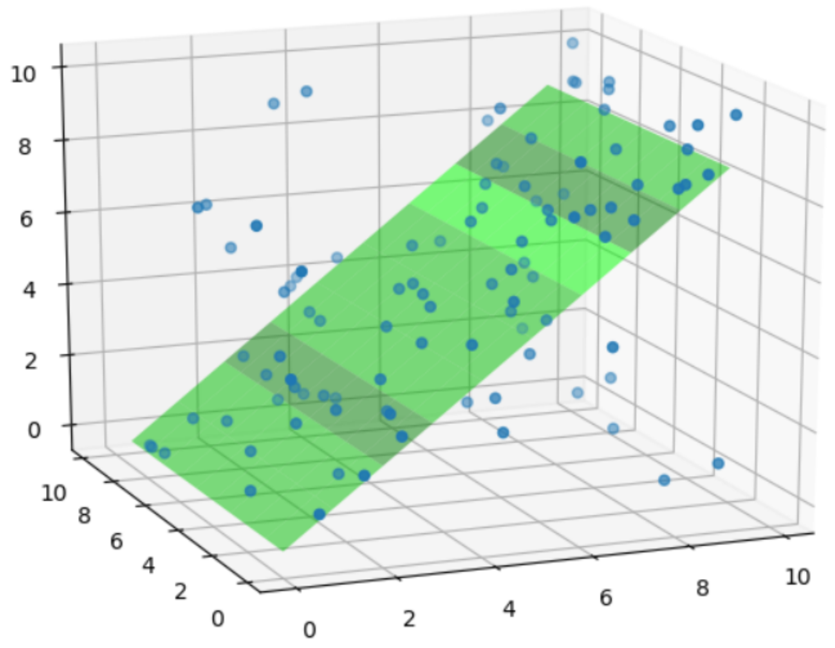


Figure 8: RANSAC plane fitting

As described in [17] RANSAC needs some predefined parameters that have to be chosen during the implementation of the algorithm:

1. **Number of minimal subsets:** Is estimated depending on the expected probability success and the (possibly) known number of outliers in the dataset.

2. **Initial size of sample subset:** Minimum number of points to fit the model. More samples in the sample subset are not better as expected, as this would increase the search space for the subset selection so more subsets, This way more subsets would have to be tested which would increase the computational complexity.

3. **Error tolerance:** We choose the threshold so that we have a certain probability p for inliers. This parameter helps us deal with the noise in the data.

4. **Number of samples:** We choose it in a way so that with probability p, at least one random sample is free from outliers.

5. **Consensus set size:** Should respond to expected inlier ratio. This ratio might be unknown a priori, but at first, the worst case scenario can be picked and then adapted in as second phase.

6. **Number of Iterations:** Instead of following the deterministic approach and check for every possible subset to find the best one, something that is computationally expensive, RANSAC chooses a subset with no outliers and then calculates the necessary number of iteration to obtain a succesful run with a certain high probability (success).

Apart from the gross outliers, the datasets contain noise which is not considered in the computation of the probability for good minimal sets. The error threshold is used for handling this noise but in practise it is generally not possible to analytically capture the eect of noise in the minimal set on the instantiated model [6]. It is very important, though to choose a reasonable value for the error tolerance in order to achieve the result desirable with this algorithm, as small value could result in the omission of correct points in the consensus set or contrariwise a big value in the commission of incorrect points. Therefore it is useful if the error tolerances are determined empirically, for example by using a small multiple of the precision mentioned in the specifications of a laser scanning device for the acquisition of a point cloud [6].

Overall, the RANSAC paradigm has many advantages. As a method it can be applied in may different cases, it is easy to implement, it is robust to outliers and has good runtime performance in big datasets. Nonetheless when used the following things should be considered: it has a certain probability of success, it requires prior knowledge about data , many parameters have to be tuned number of iterations increases logarithmically with outlier percentage [15] and lastly in the case of low inlier ratios, too many iterations will be or worst case there is the danger of the algorithm failing entirely.

2.4 Morphological filtering

During the point cloud data acquisition, no interpretation of the scene is performed [6]. Therefore, in order to exploit the maximum of the information that a point cloud can offer, the classification of the point cloud points into ground and non ground points is essential. However removing non ground points from LIDAR datasets has proven to be a challenging task. This can be achieved through various kinds of filtering of point clouds. Progressive morphological filtering is based on the concept of mathematical morphology which is a set-theoretical method of image analysis providing a quantitative description of geometrical structures [6] based on two operators, erosion, which is used to reduce, and dilation used to enlarge size of features. These two operators utilize the simplification of the surface by performing two different operations named closing (erosion after dilation) and opening (dilation after erosion). The ultimate goal of this method is to identify and remove non ground points like cars, trees, buildings etc.

With the help of a structure element (window) which represents the neighborhood of a point, the maximum (dilation output) or minimum elevation (erosion output) of the neighbouring points is determined. The points within a threshold from this lowest elevation are considered ground points. The size of the window has to be carefully picked since a small window can result in removing only significantly small features like cars while preserving bigger ones like buildings, whereas big window can falsely remove ground points. Since it is very hard to pick a window size that is small enough to preserve the terrain points and large enough to remove non terrain objects at the same time in order to compensate for this limitation, Kilian et al. [18] proposed to apply a progressive morphological filter.

In this type of filtering, the size of the window of the morphological filter is increasing gradually starting from the smallest size. One important property of this method is the fact that the opening operation preserves the features that are larger that the window size. However by using windows that grow progressively, we can acquire a smoother surface as the building measurements are removed and replaced by the minimum elevation of previous filtered surface in each iteration [18]. At the end of this process an approximation of the terrain is acquired since only the lowest height values are used. Therefore the points can be filtered according to a height tolerance from this lowest surface.

2.5 Raster filtering

A raster image is a matrix of intensity values. Filtering operations are often used to modify or enhance certain features in a raster image. They are performed on a neighborhood of pixels resulting to the alteration of the values of the pixels in the output image. For this, a kernel, which is an odd nxn array/ matrix, or a mask composed of weights is used, to compute the weighted average in a neighborhood. The kernel represents the weights of the pixels. Raster filtering was part of the education material for GEO1001.

2.5.1 Median

The median filter belongs to the class of non linear filters. It is used to remove impulse noise without blurring the image [19]. This is achieved trough the modification of the intensity values of each pixel by utilizing its neighbors. The principal of the median filter is to take the median value in a neighborhood of pixel instead of a weighted average. The median is different that the average as it is surrounded half by smaller values in the neighborhood by larger. Therefore as mentioned in [20], ”it is a stronger ”central indicator” than the average, so it is hardly affected by a small number of discrepant values among the neighbouring pixels”. The steps of the process are described as follows: first the kernel size is specified, then the pixel values that are covered by the kernel are listed in an ascending order, the median value is determined and finally the pixel at the center is replaced by the median of all of these pixel values inside the window. If the kernel covers an even number of pixels, the average of two median values are used [21]. The kernel is shifted until all elements are replaced. The final output is a linear/nonlinear function of the input, and more specifically an array of the same size containing the median filtered result.

The median filtering is very effective, to some extend, to distinguish between isolated random noise and actual features of the image (edges, lines etc) [19] but at the same time its main disadvantage is the that smoothing can not be performed in boundary regions. One way to compensate for this limitation is by beginning the median filtering by placing zeros around the row edge and column [22]. However this would still cause edge distortion near the image boundary. The median filter, as being an edge preserving smoothing filter [21], it is able to preserve sharp edges, but depending on the size of the feature, since if the size is less that half of the size of the filter, then pixel is replaced by the median as it is considered as a noisy element of the raster [23]. The following figure (9) was derived from [21].

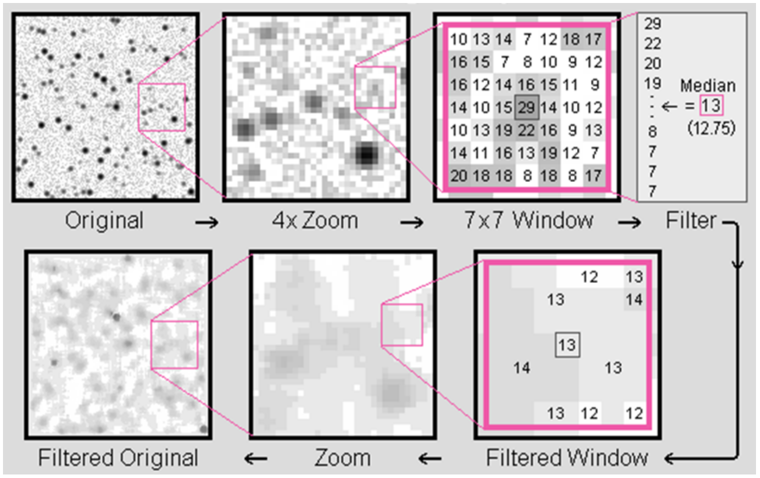


Figure 9: Median Filtering Example

2.5.2 Gaussian

Gaussian filtering is used to remove noise and most importantly to smooth images by eliminating detail (high frequency components) [24]. It is a convolution filter in which the values of the kernel are simulative to the shape of a Gaussian distribution [25], according to which the distribution of the values decreases the further we move away from the peak of the curve (Figure 10). Same applies in the Gaussian filtering where the weight values decline with the distance from the center pixel which has the highest weight [25]. Therefore the center and its immediate neighbours, lend heavily weight to the weighted average, which is also the main reason for the persistence of edges after this operation.

The two basic characteristics of the Gaussian filtering process are the kernel size and the standart deviation(s). The shape of the Gaussian kernel function is depends on the value of σ which is defined relatively to a decimal number of standart deviations. A higher value of sigma would cause higher levels of smoothing and less preservation of edges, as the Gaussian curve would be more broad and less peaked [24]. Increasing the standard deviation will affect likewise the kernel size. Overall, the kernel size can be normally limited to contain only values within specific standard deviations of the mean.The following figure (10) was derived from [26].

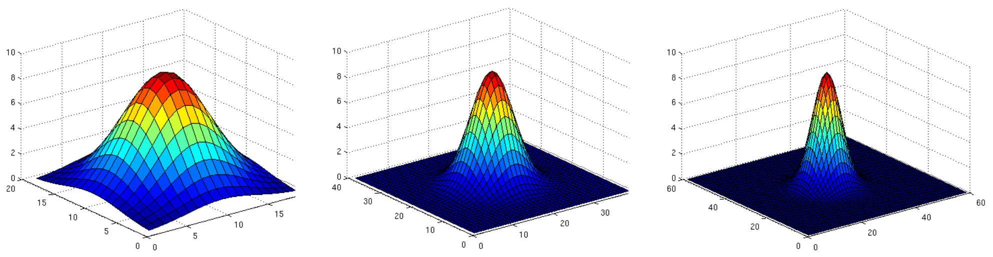


Figure 10: Gaussian Curves for σ = 11, σ = 7, σ= 2

2.6 Web Map and Web Feature Service

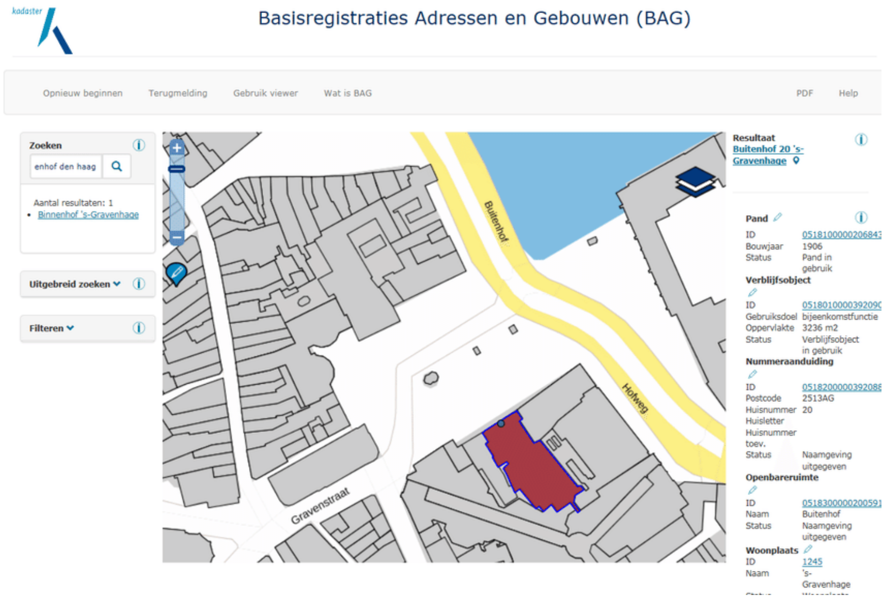
Geographic information infrastructures (GII) were created to facilitate the availability and access to geographic information for all possible stakeholders (government, ordinary citizens etc). Proliferation of geo spatial information and the need for interoperating applications led to the emergence of Web Services. Web Map and Web Feature services are considered a very useful new paradigm, as it provides access to spatial data online instead of direct connections to 'remote' databases [17]. This offers the indisputable advantages that no user login is required (as in the case of a database) and is not only for exclusive browser - server cases but also for more generic applications.

The Open Geospatial Consortium, which is an international organization leads the development of interoperable standards for geospatial and site-specific services [27]. The two geo - related standarts, namely Web Map and Web Feature Service, were created for viewing, modifying, and exchanging geographic information in raster and vector format in the Internet.

Web Map Service generates static maps in raster format for viewing purposes only. Web Feature Services is the OGC specification that defines the standart for exchanging geoinformation in digital format independently from the underlying data source [28]. It provides raw vector geodata comparable to shapefiles and PostGIS, without any particular portrayal. The output of the request is more than display. The user is able to edit and update the features itself. Apart from the geographic vector data, associated administrative information can be retrieved (e.g BAG IDs). During the GEO1007 course the possibilities of these web services were introduced.

2.7 BAG

The BAG (Basisregistratie Adressen en Gebouwen) is the Dutch governemnt system for registrations of buildings. It possesses information for all properties in Netherlands which can be municipal details for five different object types (buildings, residential objects, number designations, public spaces and places of residence) [29]. As mentioned in [29] the information itself are the purpose of use,construction year, status, surface area, geometry and location on the map (xy coordinates) as mention . These types of administrative data can be requested through the Web Feature Service (WFS). The following figure (11) was derived from [30].



Aantal resultaten: 1
- Binnenhof 's-Gravenhage

Resultaat
Buitenhof 20 's-Gravenhage

Pand

ID	051810000020684
Bouwjaar	1906
Status	Pand in gebruik

Verblijfsobject

ID	051801000039209
Gebruiksdoel	bijeenkomstfunctie
Oppervlakte	3236 m2
Status	Verblijfsobject in gebruik

Nummeraanduiding

ID	051820000039208
Postcode	2513AG
Huisnummer	20
Huisletter	
Huisnummer toev.	
Status	Naamgeving uitgegeven

Openbareruimte

ID	051830000020059
Naam	Buitenhof
Status	Naamgeving uitgegeven

Woonplaats

ID	1245
Naam	's-Gravenhage

Figure 11: Example of BAG information for building

2.8 Calibration of camera

Since the picture is only 2-dimensional representation of the reality, it is not possible to preserve all of the information about feature of the interest. However, some of the details about reality, especially the one which were distorted not occluded, are possible to restore, with the usage of camera parameters. To calculate them, camera calibration is needed [31], as a necessary step in order to extract metric information from 2D images [32]. It includes the determination of intrinsic and extrinsic camera parameters. More specifically, a mathematical problem is solved to find distortions of image, in other obtain real world measurements . These distortions are caused by the cameras physical flaws which are internal characteristics that are specific for each camera.

Calibration parameters: As it was written above, main goal of the camera calibration is the establishment of parameters describing dependencies between camera's and real coordinate system as well as parameters describing physical characteristic of instrument and transformation of the perspective [31].

In general it is possible to distinguish two groups of parameters in every procedure:

- Extrinsic parameters – related to shift and rotation of the camera in respect to real situation,
- Intrinsic parameters – describing the characteristics of lens and other "internal" parameters.

The parameters of the first group are relatively easy to obtain because rotation and shift is simple in description with 6 parameters (3 related to movement in the direction of every axis and 3 related to rotation around every axis of coordinate system). However in the second group situation differs; to describe them, the theoretical model of the camera needs to be developed.

Pinhole Camera Model Commonly, the pinhole camera model is used as the reference for geometrical relations (Figure 12 derived form [31]).

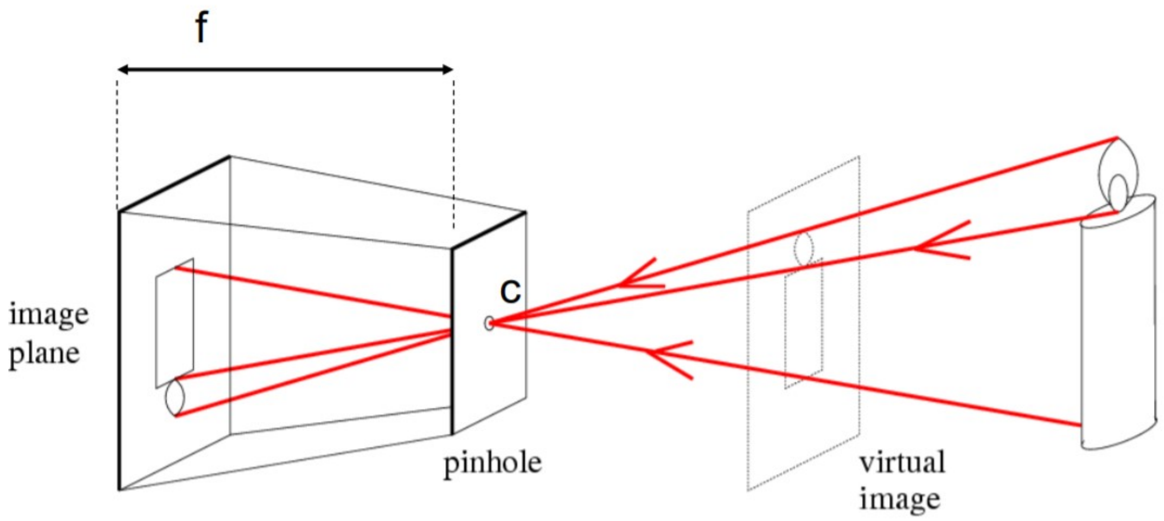


Figure 12: Pinhole camera model

The model presented is leading to equation: [31]

$$-x/f = X/Z,$$

Thus,

$$-x = f * X/Z.$$

After exchanging the surface of picture and camera from the picture, we can transform the equation as follows:

$$x/f = X/Z.$$

Unfortunately in real situation the assumption that the principal point of the camera is exactly in the middle of the image plane is not always preserved. Because of that it is important to add to the model two additional parameters: C_x and C_y. The final result is a simple model in which the point in the real world with 3 coordinates (X,Y,Z) is captured on the image plane in the place described with the equations as follows [31]:

$$x_{imageplane} = f_x(X/Y) + C_x$$

$$y_{imageplane} = f_y(X/Y) + C_y$$

where f_x is focal length

Parameters f_x, f_y, C_x, C_y are considered as intrinsic parameters and stored in matrix know as intrinsic parameters matrix and looks as follows:

$$M_i = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

Extrinsic parameters – rotation matrix and shift vector To describe the external parameters of camera it is necessary to introduce the rotation matrix and shifting vector. Location of the object (in real world) can be describe in respect to the coordinate system of the camera with the usage of them. Because of that is it possible to obtain the mathematical description about coordinates transformation between coordinate system of the object of interest and coordinate system of the camera [33]. The rotation of the camera can be described as matrix multiplication in 2 dimensions (Figure 13 derived from [31]). Thus, as I was mentioned in introduction to calibration, the rotation in 3-dimensional space can be decomposed to 3 2-dimensional rotation around every axis, which is leading to the equation: $R = R(\psi) + R(\phi) + R(\theta)$
Where:

$$R_x(\psi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\psi) & sin(\psi) \\ 0 & -sin(\psi) & cos(\psi) \end{pmatrix}$$

$$R_y(\phi) == \begin{pmatrix} cos(\phi) & 0 & cos(\phi) \\ 0 & 1 & 0 \\ cos(\phi) & 0 & cos(\phi) \end{pmatrix}$$

$$R_z(\theta) == \begin{pmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

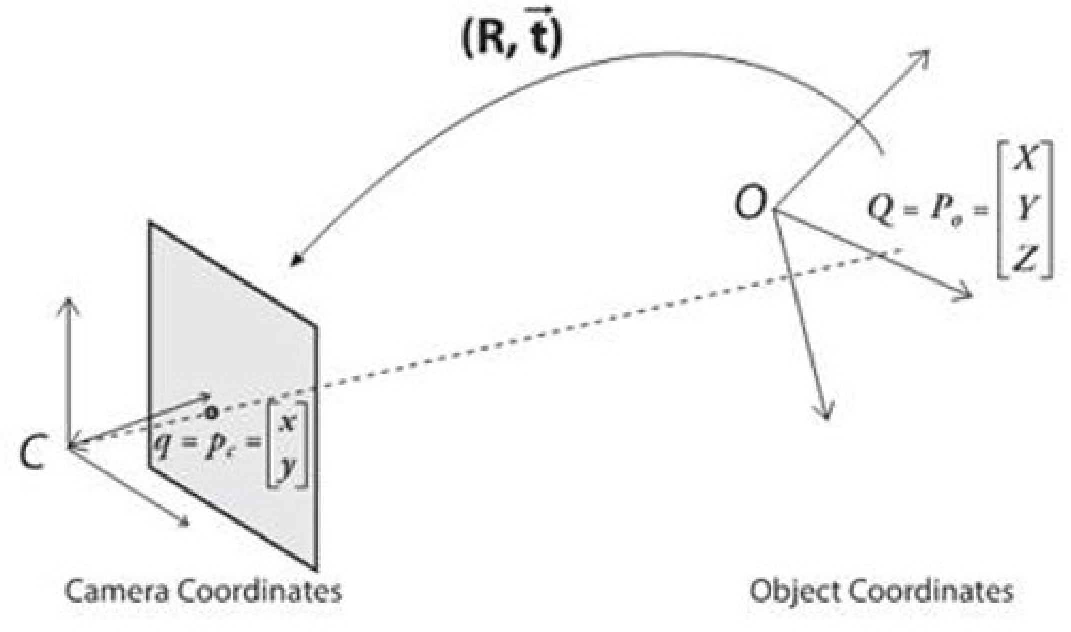


Figure 13: Transformation of object to camera coordinates principles

The second subgroup of extrinsic parameters is shift parameters, merged together as shifting vector. In other words, it is the vector which is describing the shift between origins of coordinate systems involved in transformation. With the previously obtained matrix of rotation and assumption that P_o is the actual position of the point and P_c is the position of the point on the image plane, the equation looks as follows:
$P_c = R(P_o - T)$,
where R is rotation matrix and T is shift vector.
Together with previously obtained information, which is intrinsic matrix:

$$M_i = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

and extrinsic parameters matrix M_e (together with shift parameters and all the rotations):

$$M_e = \begin{pmatrix} r_{11} & r_{12} & r_{13} & -R_1^T T \\ r_{21} & r_{22} & r_{23} & -R_2^T T \\ r_{31} & r_{22} & r_{23} & -R_3^T T \end{pmatrix}$$

Final relation between matrices can be written as follows:

$$\begin{pmatrix} x_o \\ w_o \\ w \end{pmatrix} = M_i M_e = \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix}$$

Where M is the projection matrix. With all the information it is possible to use the collinearity equation and transform it to the form which can be used to derive pixel coordinates

$$x_{px} = \frac{x_h}{w} = \frac{m_{11}X_w + m_{12}Y_w + m_{13}Z_w + m_{14}}{m_{31}X_w + m_{32}Y_w + m_{33}Z_w + m_{34}}$$

$$y_{px} = \frac{x_h}{w} = \frac{m_{21}X_w + m_{22}Y_w + m_{23}Z_w + m_{24}}{m_{31}X_w + m_{32}Y_w + m_{33}Z_w + m_{34}}$$

Which is the goal of image coordinate reconstruction – to relate 2-dimensional image plane to 3-dimensional reality. With all the parameters mentioned above as well as derived equations, he formal calibration camera became trivial and simplified to estimation of the parameters basing on the object with known size and shapes called calibration benchmark [43]. The specific procedure describing how to obtain the parameters have to be chosen depending on user experience and application. There are various techniques to solve that – most popular approaches are usually basing on non-linear square adjustment with altered version of Gauss-Newton algorithm and realised with pre-prepared solutions.

2.9 Software

2.9.1 Fiona

Fiona is a Python library used to read and write data. It mostly focuses on GIS vector spatial data that represent discrete fields (roads, boundaries of properties etc). It makes use of the OGR Simple Features Library which supports various data formats. The reason it is essential for data exchange in GIS applications is that, this way more types of data can be supported and converted in different GIS formats. The intermediary for input/output of Fiona, is a format called Well -Known Text (WKT), which is a text mark up language [12] regulated as a standard by the OGC, that represents the geometry of vector objects.

2.9.2 Shapely

Shapely is another Python tool essential for geospatial programming that analyses and manipulates data. It supports spatial operations of vector features (polygons, lines, points) such as intersections, centroids, convex hulls [34] and more. Shapely's functional operation is based on another OGC Simple Features standard, the GEOS (Geometry Engine Open Source) library, which executes spatial operations, including validation and topology functions. GEOS is the geometry engine of the PostGIS spatial extension for the PostgreSQL

RDBMS (relational database management system), hence Shapely is able to perform PostGIS type geometry operations outside of an RDBMS, avoiding the need for the data to be imported into a spatial RDBMS [35]. This is the main advantage of the Shapely library if we consider that, not all spatial data within a database are equitably well processed using SQL [35].

2.9.3 OWSLib

OWSLib is python package that enables the use of OGC Web Feature Service (WFS). The format of the response to a request for a selection of features from a data (geometry and attributes) values is by default text/xml. Other output formats (including older versions of GML, non-XML, binary and vendor specific formats)[36]. However for this, the appropriate values for the output format need to be mentioned in the capabilities file.

2.9.4 Opencv

OpenCV (Open source computer vision) is a programming library originally developed in C++ environment but was later started used as an extension of python. It has many different moduls such as Image Processing and Object detection and it supports many applications (e.g. segmentation and recognition)[37], but in the case of this project it was used for the processing of raster files to detect contours. Contours are considered all the continuous pixels with the same intensity or color along a boundary. They are a very useful for shape analysis and object detection and recognition [38].

2.9.5 PDAL

As described in the PDAL documentation [39], the Point Data Abstraction Library is used for translating and manipulating point cloud data. It is a C++ open source library meaning that it is available online. The main advantage of is that it can operate on point cloud data of any format. For this every operation that is performed on the dataset, is organized in pipelines that pertain to different stages of the processing. These pipelines are written in JSON (JavaScript Object Notation) format.

2.9.6 Psycopg2

Psycopg2 is a PostgreSQL database adapter for Python. It supports the insertion of the point cloud data in databases and then the access of it for retrieval (quering of data), update and deletion purposes through the processing in python environment. For the use of this library the installation of a PostGIS extension is necessary.

2.9.7 DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) is Python library that performs clustering of data. Clustering means grouping points together depending on some sort of similarity. The most important parameters defined in DBSCAN are the maximum distance between two points (eps) in order to be considered a part of a cluster and minimum number of samples (min_samples) to form a dense region [40]. This type of clustering has very good results with datasets high density and is capable to deal with the presence of outliers. However when similar densities occur in a dataset it has difficulty in discriminating between different clusters.

2.9.8 RANSAC

The RANSAC implementation in C++ environment was used as provided by Dr. Liangliang Nan for the course GEO1004.

3 Methodology

3.1 Overview

Our approach developed two different solution for the purpose of window detection. One of them was based on the scientific paper titled “Slicing Method for Curved facade and Window Extraction from Point Clouds”, written by S. M. Iman Zolanvari and Debra F.Laefer [41] and makes the assumption that all windows are uniformly represented as holes (lack of points). The second solution is derived from the idea mentioned in the research ”Automatic detection and feature estimation of windows for refining building facades in 3D ubran point clouds” written by Aijazi A. et al. and it is relying on assumption slightly different from the previously mentioned one - which is the fact, that windows in the points cloud are represented by lack of points [42].

Our approach contains 3 main steps. During the first step, the point cloud used as input is ”filtered”, meaning that the point cloud will be processed to fulfill the requirements for the purposes of the chosen task. During the second step, a segmentation algorithm is applied in order to classify each point to its corresponding building facade. The third and final step is the window detection, from which window and facade boundary points will be collected. Moreover, all points belonging to a specific window, are clustered together. A step-by-step implementation description is given in the following sections. At final part, the results are evaluated and conclusion about applications and accuracy is made.

The fundamental steps of the approach are shown on the diagram as follows:

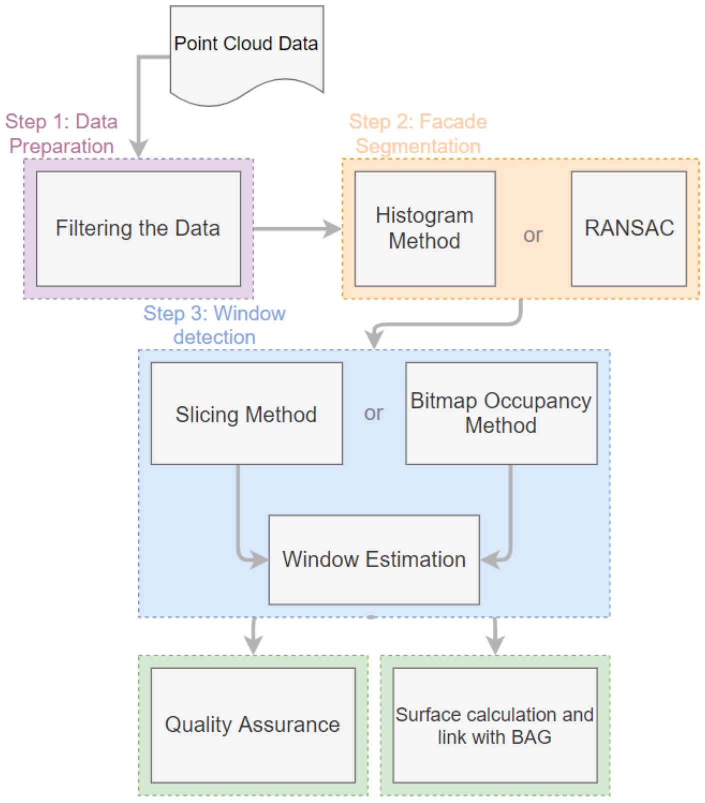


Figure 14: Workflow

3.2 Dataset Filtering

The cleaner the input dataset is for the window detection algorithm, the better the results will turn out. For this reason, filtering the dataset such that only facade points are left is an important step. There are multiple approaches for this. First of all, a point cloud that has been obtained with a terrestrial laser scanner is ideal to use for the application of this project, because its point density on facades is normally higher than of an airborne laser scanner point cloud. The steps of which the methodology of this part consists, are shown in the diagram below, and the used code is present in appendix G.

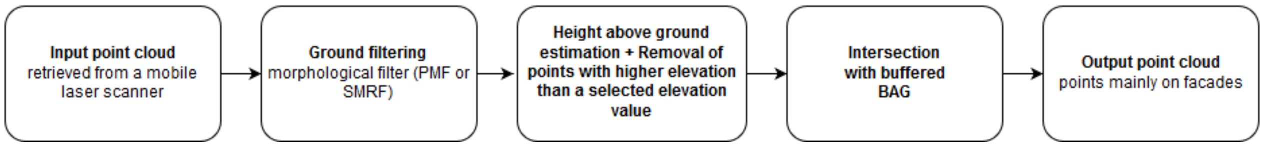

Input point cloud retrieved from a mobile laser scanner
Ground filtering morphological filter (PMF or SMRF)
Height above ground estimation + Removal of points with higher elevation than a selected elevation value
Intersection with buffered BAG
Output point cloud points mainly on facades

Figure 15: Dataset filtering workflow

The approach starts with filtering out the ground points from the dataset. For that purpose itself there are multiple approaches, but this research considers two of them - performing either a Simple Morphological Filter (SMRF, derived from [43]) or Progressive Morphological Filter (PMF, derived from [44]). The reason for this partially is because they are implemented in the Point Data Abstraction Library (PDAL), a library for processing point clouds that will be used for this part of the methodology.

The basic idea behind both morphological filters are as follows. Since the point cloud is assumed to have an irregular point distribution, a regular grid is imposed on it, of which every cell has the value of the lowest elevation point that intersects with the cell. That is unless there is no original point intersecting with the cell - it then gets the height value of the nearest point assigned to it. Subsequently, a morphological filter using an opening operation is performed on the previously created minimum elevation surface. Then, a window with a specified radius that defines a search neighbourhood is used as input for the filter, together with the surface. The surface is now smoothed, also because non-ground points are detected based on an elevation difference threshold. This step is performed iteratively with the new surface and an enlarged filter window as input. This iteration is terminated when the filter window has become larger than a set maximum value, which is marginally larger than the maximum size of buildings [44].

Ultimately, only points that are deemed as ground points are left. These can now be filtered out of the original point cloud. As the ground points are removed, the building facades will be left, along with other objects such as vegetation, cars and sidewalks. These objects should be removed to clean the point cloud further. Again, there is one proposed method to do so.

After the ground filtering, it is possible to derive a height above ground estimation for every point. By filtering points that are just above the ground, the sidewalks can be eliminated. Then, because the BAG is available, an intersection can be made between it and the points, to keep only those that are supposed to lie on a building. This will leave only the facades, as objects are generally located on a minimum distance from the facades.

However, facades are normally not straight, having parts that stick out such as balconies. In addition to that, the point cloud will contain points that have been reflected through windows from objects that are located inside buildings. While these points can potentially be used for window detection, as has been done in [45], they are only a hindrance in the methodology that is proposed in this report. For these reasons, rather than intersecting the point cloud with original BAG features, a buffer ring is created around every individual polygon. This is achieved by both creating a buffer with a positive and with a negative distance around every feature, resulting in separate polygons that are respectively bigger and smaller than the originals. By performing a difference operation with the smaller shapes on the bigger ones, only a ring remains, which would intersect with all facade points, and with just few other points (of outside and inside located objects).

3.3 Linking points to BAG

In order to link calculated areas of facades and windows (and the ratio between them) to BAG features, every point in the cloud should get the ID attached to it of the BAG feature from which it is reflected. The largest challenge for this part is the computation time of the algorithm. A simple intersection check is not sufficient since points of facade parts that stick out will not intersect with any polygon, and a closest polygon to point calculation for example is expensive, which is why another solution is sought. The final code is included in appendix F.

Firstly, points that intersect with a BAG feature will get the id of that feature attached to it. To increase the speed of point in polygon calculations, the R-tree is used to index the envelopes of all polygons. For every point, it is then checked with which envelopes it intersects. Then, the intersection with the real polygons that correspond to these envelopes are tested, to filter out false positives.

The second step is to assign an id to every point that does not intersect with any BAG feature. Because a closest point algorithm is faster than a closest polygon algorithm, equally spaced points are created on the boundaries of the polygons as shown on figure 16. With the help of a kd-tree, the points that have not yet been classified will receive the id of the polygon that corresponds to the closest one of the newly created points on the polygon boundaries.

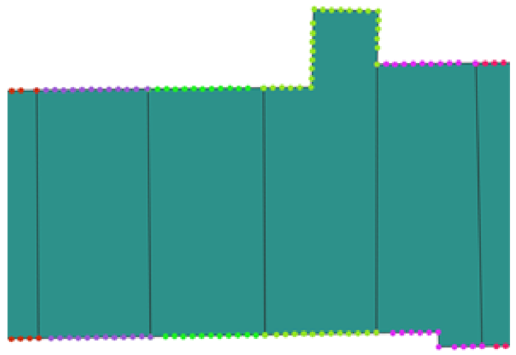


Figure 16: The created equally spaced points on BAG features. Each point colour represents a different BAG ID.

3.4 Segmentation

Plane segmentation is a usual task in the automatic 3D modelling reconstruction from unorganized point clouds acquired by different technologies such as airborne or mobile LiDAR. Since our window detection algorithm is iterating through different building facades and is not treating them as one, every point needs to be assigned to one of the facades. This is achieved with the execution of a point cloud segmentation algorithm to identify individual facades.

For this purpose, an iterative technique called Random Sample Consensus (RANSAC), which is easier to implement but has limited possibilities, is applied. Another approach to separate points into different facades is the Histogram-Based Segmentation Method, which, however, will not be used in our implementation due to the demanding time frame of the project. For the sake of comprehensiveness, a description of both methods will be given in the following subsections.

3.4.1 Histogram-Based Segmentation Method

The Histogram-Based Segmentation Method is based on quantitative measurements and one basic assumption, that points acquired by a laser scanner are evenly distributed in the 3D space. Density differences in the point cloud could be caused by differences in materials of objects (e.g. mirrors, glass), or simply, due to the absence of objects (smaller number of points observed at a local region). The main advantage of this approach is that it can also detect non-planar surfaces, like curved building facades, overcoming the possibilities of the RANSAC Segmentation approach, which can only detect planar surfaces.

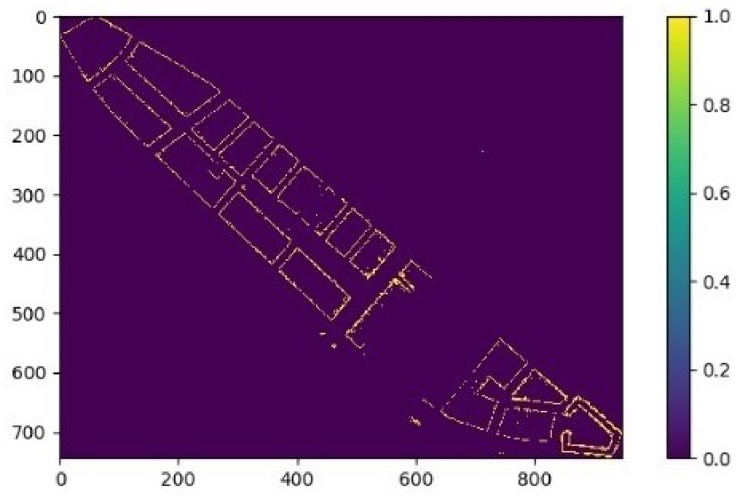


Figure 17: Facades of buildings detected by using the Histogram-Based Segmentation Method

The algorithm is "dropping" every point of the point cloud to 0-elevation, applies a regular grid on the same level and assigns the number of points inside every cell of the grid as the cell's value. The plot of the grid shows a three dimensional histogram of the points' distribution. Thus, searching for a building facade corresponds to choosing a threshold value for number of points inside a cell and searching for connected cells with equal or higher cell value than the threshold. High concentration of points in one of the histogram's cells can only occur for objects that consist of a high number of points - high density of points in a small neighborhood- and also, of high vertically distributed density of points. Even in cases of inclined building facades, the concentration of points on the histogram's cell for that region will still remain higher than the concentration of points that describe other objects than buildings. The threshold value selection can be based on the density of the point cloud, as well as the minimum building height of the study area. Obviously limitation related to quality of measuring equipment as well as not always perpendicular facades are forcing small buffer around the place of interest, defined as the cell size. In principles, the points filtered with conditions mentioned about should be used for plane estimation with non-linear solution, since the assumption is that this approach is mentioned to be used to recognize curved facades. However, due to limited time and lack of possibilities to find suitable data for this solution, it is only mentioned in the report as the proposition for solving the problem related to non-rectalinear facades.

3.4.2 RANSAC Segmentation

Since the previously described method is experimental and requires more time for testing and implementation, a different approach was proposed to ensure the fulfillment of the main project requirements in the given time frame, the RANSAC Segmentation approach. The major disadvantage of this approach is the fact that it is not robust to non-planar surfaces - the Random Sample Consensus algorithm is searching for linear solutions.

The RANSAC algorithm starts with selection of random points to build random subset and estimate the required model parameters basing on them. Secondly, it is assigning the points as outliers or inliers basing on provided threshold. By iterative repetition of this process, the algorithm is finding the solution that will mark highest possible number of points marked as inliers. Obviously, that kind of approach cannot guarantee finding optimal parametric model, hence the probability of finding optimal solution is increasing with every iteration. In the project, the RANSAC algorithm was chosen to estimate the optimal plane for specific facade representation according to the point coverage of each plane, only planes with point coverage larger than 30% are kept in the final result. Since the core part of the project is requiring to handle every facade separately, the necessity to cluster the points facade-wise is crucial for the algorithm.

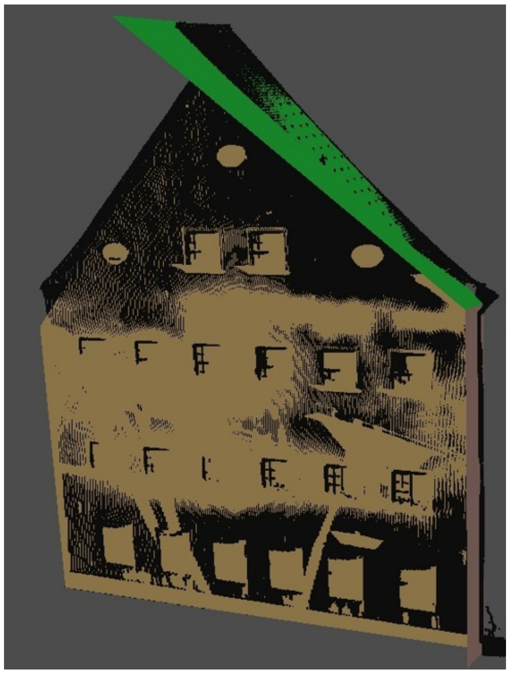


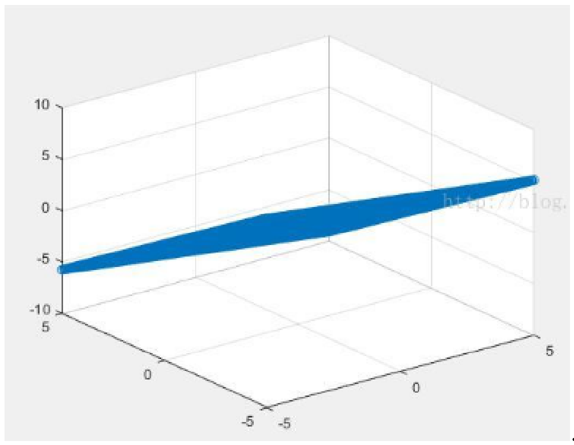
Figure 18: Planes segmented by refined RANSAC algorithm

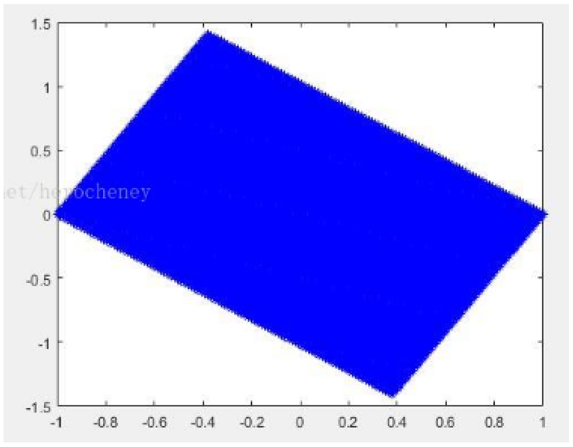
3.5 Window Detection

Window and facade boundary detection is challenging due to the fact, that all the solutions are ill-conditioned – which means not robust for big differences in dataset parameters (such as shape of clusters or robustness to mesh). Because of that, the characteristics of the dataset derived from previous steps have to be evaluated by the user depending on experience and assumptions. In the following subsections, two different window and facade boundary detection approaches will be described, the Slicing Method and the 2D bitmap occupancy map method. Our group focused on implementing two different approaches in order to be able to offer multiple solutions to our client, have an alternative plan in case one of the two methods fails and make an accuracy comparison of two different techniques.

3.5.1 Projection of 3D points in 2D plane

For the implementation of both window detection methods it was necessary to project the 3D points to a 2D plane while making sure that its shape remains unchanged. For this, a transform matrix for the projection and reprojection was used.


(a) Plane in 3D



(b) Projection in 2D

Figure 19: Example of plane projection in 2D

The direction of the z-axes is set same as the plane's normal direction, the direction of x-axes is parallel to the ground, and y-axes is perpendicular to these two axis. The transformation matrix is as follows:

$$T = \begin{bmatrix} vX.x & vY.x & vZ.x & CenterX \\ vX.y & vY.y & vZ.y & CenterY \\ vX.z & vY.z & vZ.z & CenterZ \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where:

vX: Direction vector of x-axes

vY: Direction vector of y-axes,

vZ: Direction vector of z-axes,

CenterX, CenterY, CenterZ: x, y, z value of centroid of point cloud

So, we can use the formula below to project the 3D points to 2D and vice versa:

(x1,y1,z1,1)*T=invT*(x0,y0,z0,1)*T

where:

x0, y0, z0: x, y, z coordinates of 3D points

x1, y1, z1:x, y, z coordinates of 2D points,

T: Transformation matrix

invT – Inverse of transform matrix

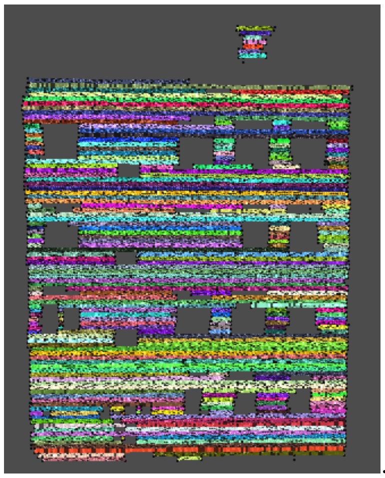
After projection, the z1 value of points on the same plane is almost same, so we can discard the z1 value and (x1, y1) is the 2D coordinates of each points.

3.5.2 Slicing method

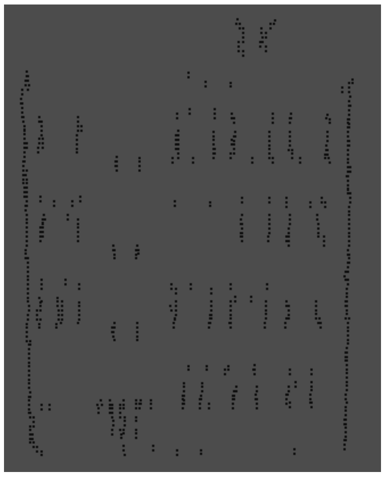
This method is based on the paper of Zolanvari and Laefer [41], and is able to detect any opening on a facade, including for example occluded parts or doors, but most importantly windows. As a prerequisite, it takes a segmented facade as input, of which non-planar points are removed. The facades are then sliced horizontally and vertically, with a set thickness that needs to be multiple times smaller than the smallest window, and detects points that describe boundaries of windows and facades. For every slice, clusters will be created based on the distances between the points inside. The used code for this method is in appendix I.

Points withing every slice are projected in 1-D, onto a locally defined x-axis, forming an alignment of points. Moving on the x-axis, from one side to the other, the distance between sequential points is used to delimit holes and surfaces. If the distance to the next point is larger than a selected threshold value, it means that there is a hole found between the current and the next point. The threshold used for data-sets in Rotterdam is usually determined as ten times the median distance between the points, which can be calculated as points are on a line, the x-axis. When this is done for a slice, points that were identified in the end or in the beginning of holes, are characterized as extreme points. Moreover, the first and the last points of the x-axis of every slice are also characterized as extreme points, which in the end are used as boundary points of the facade.

When the collection of extreme points is performed for every slice, every cluster, i.e. every window, is extracted by applying the Density-based spatial clustering of applications with noise (DBSCAN), proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996 [46]. Given a set of points, in our case the extreme points, it groups together points that are closely packed together, or different said, points with many nearby neighbors. In our implementation, the parameters we chose are the maximum distance between points and the minimum number of identified clusters. After completing the clustering of windows, a property is attached to the original 3D points of the point cloud, indicating which are part of a window boundary and which are not. The code for this part is included in appendix J.

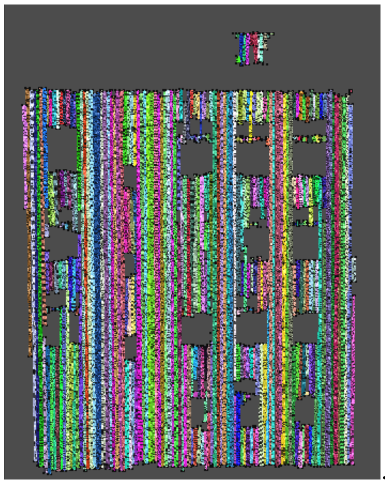

.

(a) Horizontal slicing

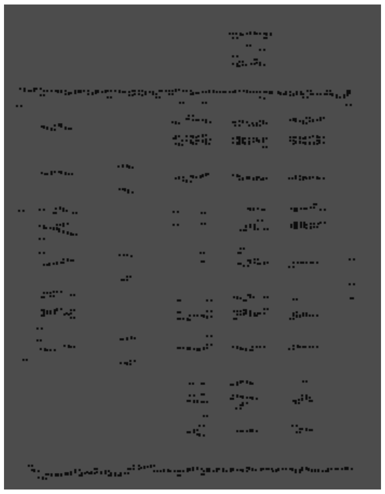


(b) Extreme points derived from horizontal slices

Figure 20: Horizontal Slicing and resulting extreme points


.

(a) Vertical slicing



(b) Extreme points derived from vertical slices

Figure 21: Vertical Slicing and resulting extreme points

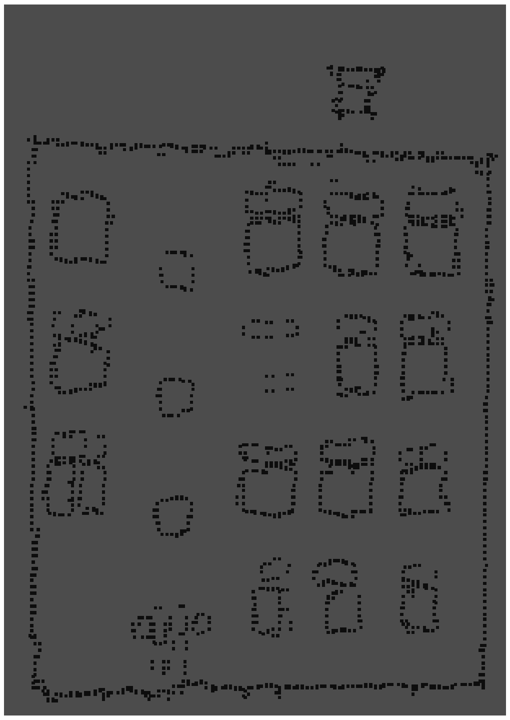


Figure 22: Extreme points derived from both horizontal and vertical slicing

3.5.3 2D bitmap occupancy map method

Data preparation: With all the points extrapolated to plane facade, the preparation of the raster can be done. Because every facade has it is own characteristic noise as well as geometry, it is necessary to specify several parameters during the whole process of bitmap preparation necessary to prepare the dataset for actual window detection. Since there are various reasons that can change them, starting with architecture characteristics, the weather, air condition and finally the quality of measuring equipment, it was concluded that the parameters have to be determined by trial and error. To perform that, various datasets were prepared basing on user experience and every of them was used in the next steps to conclude about application possibilities for every of them by comparing the results to measured benchmark data. Again, the code is included in the appendix (H).

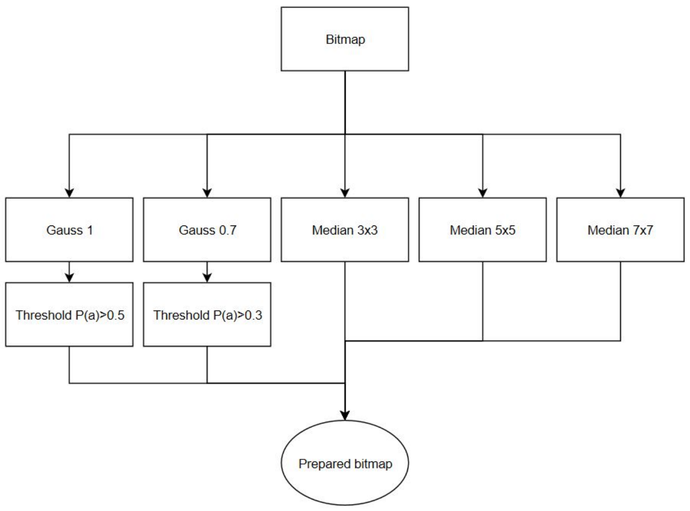


Figure 23: The diagram showing workflow of the bitmap preparation. The numbers refers to sigma for Gaussian filter and size of kernel for median filter

Obviously the main idea behind finding the parameters is to remove as much noise as possible and keep as much actually windows as possible. Regardless to the taken approach, some of the characteristics were expected and noticed on the bitmap, such as wood crosses in the middle of the window or curtains covering the window. Since the previous steps also resulted in multiple results (different cell size for projection and different treshold), at end 72 different results were tested. In most of the cases it was possible to assess with naked eye that the results of pre-processing will not lead to acquisition of windows shapes, position and size from point cloud - it was not possible to distinguish any windows in some cases. The datasets produced from 0.007 size of the cell bitmap looks to be the most accurate in the way of presenting real situation. The boundaries are clearly visible, better than in the two other approaches, and the middle cross is removed in most of the cases. Obviously, some artefact are still possible to notice. The only problem was related to poor density of the points on the facades which were measured with instrument located close to the building - the interval between scanning layers is increasing with distance and angle.

Bitmap - windows detection algorithm: The main idea driving this part of window detection, crucial in the workflow, is the fact that windows shapes are regular and there is no need to provide the exact boundary of it (if information about the centers of windows is present) and classify it to some "window template". Obviously, the variety of shapes and sizes is making this task challenging. To solve that, some assumptions are made.

At first, the window can be only distorted toward inside – it is not possible for the window to appear bigger than in real situation, because the most basic assumption is that the window is described as lack of points. The remaining parts of the buildings, because of the building materials, will reflect the light – which means, that every feature distorting actual shape of the window will cover it by adding additional points inside. This rule is fulfilled for curtains, crosses in the middle of the windows, stickers on the windows glass (such as souvenirs or Christmas decorations, balconies (the measurements were taken from the ground level so in specific situations it is even possible that the balcony floor can coven the whole window if it is width enough) and every other features than can interfere with proper window measurement. However this assumption is making necessary to take into consideration that density of points is dependent on vertical and horizontal distance to the measuring instrument – the cells of the bitmap with no points can be caused by low density if chosen size of cell does not take this fact into consideration.

The second assumption is that it is highly probable that the windows which are lying next to each other are similar in size and shape. Obviously, in one facade it is possible to detect variety of them; however, it is unlikely to have set up of 16 different windows on the facade with 16 of them. Besides that, as it was described above, it is not possible to detect the exact shape of the window – it will be always distorted. This information is leading to conclusion, that the windows will be not the same in shape and size on the bitmap even if they are in real situation – because of the random factor correlated with the chance for closed curtains and so on. Natural idea to solve this problem is to provide the factor which is adjusting the definition "the same" to "similar" with provision of numerical justification of this idea. The problem related to this approach is automatic assumption that windows considered as different does not have similarities described with those factors. This problem is neglaced due to the fact that the main goal of the research is to establish the ratio of the windows compare to facade – which is imposing usage of accumulated information together. Since parameters describing the size of the window in specific class are established basing on the parameters of the windows assigned to the class, there is no need to distinguish the classes which are highly similar, because the impact on results will be unsignificant. The decision tree for considering the window as similar was built as follows:

```
Data: Q(C), P(W) where
        P(X)$\subset (width, height, ratio), Q \subset \{$width, height, area, classid$\}, C \subset \{classes\}$
Result: Q(W)
for $\forall(C)$ do
    if $W == C \Leftrightarrow \forall(P)(P(W) == Q(C))$ where '==' denotes similarity. then
        $\forall(P)(P(W)$ assign as $P(C))$;
    else
        continue
    if $P(C) = \emptyset$ then
        CREATE C, WHERE $\forall(Q)(Q(C) = P(W)) and Q \subset \{classid\} = NEW$ else
            continue
```

Algorithm 1: Assign window to class

4 Quality assessment

4.1 Dataset filtering assessment

4.1.1 Results

As had been noted in the methodology chapter, the facades should be as clean as possible in order to retrieve the best window detection results. Therefore, testing has been done to find out what the quality is of the filtering, and what the influence is of some different choices. There are differences regarding the chosen ground filtering technique, and whether or not to remove inside points.

In the methodology, it was explained that the PDAL library gives the possibility to use both the Progressive and the Simple Morphological Filter (PMF and SMRF). After performing of these ground filters on a subset of the point cloud, some differences can be noticed in the results. It has to be kept in mind that other objects such as cars and trees are not removed during this step, so they are still included in the example images. The images are shown in appendix A (Figures 47, 48, 49), since they are too large. In addition to that, the execution time of both are compared.

The SMRF removes more points than the PMF, which can both be seen on table 1 below and the resulting figures. With the PMF 58,6 percent of the original amount of points is removed, and for the SMRF this is 64,4 percent. Parts where this is clearly shown are marked with red boxes on the figures. The SMRF removes the same points as the PMF, but also additional ones that seem to be part of the bottom of objects. Some lower parts of facades are removed, but this would be done in an additional step as well if PMF would be used - the height above ground estimation, and subsequent removal of points that are 20cm higher than the local estimated ground. This is needed to remove some pieces of pavement that are not removed, as can be seen in the snapshots of figure 24 below. Using the SMRF rather than the PMF would thus eliminate an extra step.

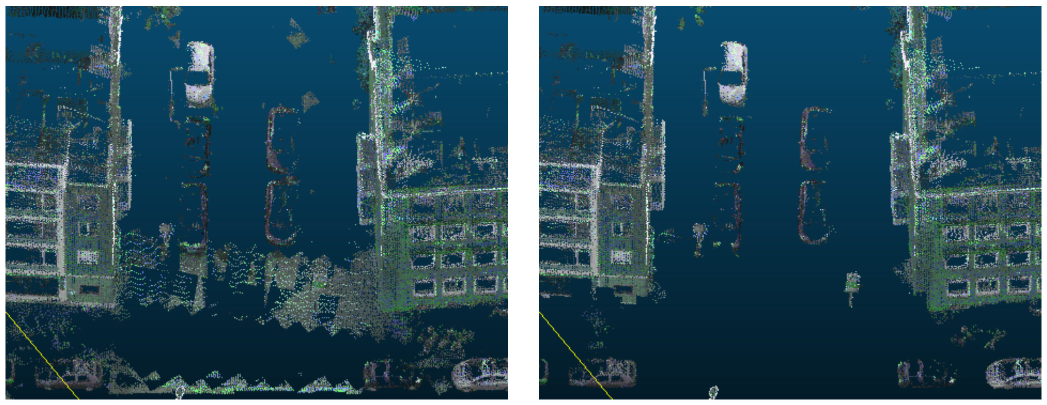


Figure 24: An example of a part of the street that has not been removed by PMF (left), but has been removed by SMRF (right)

Amount of points	Original subset	PMF subset	SMRF subset
Absolute	3.432.452	1.422.431	1.223.294
Relative	100%	41,4%	35,6%

Table 1: The amount of points that remain after the filtering

Adding to that, SMRF already has a shorter computation time. Both filters have been performed ten times on the same subset, and the computation times have been averaged. On average it took 4.82 seconds on the subset of 3.432.452 points for the SMRF, compared to 5.52 seconds for the PMF. This means that the former takes 12,7 percent less time. The specifications of the used system are listed in table 2 below. The SMRF thus is the best choice, as it is faster on itself and also diminishes one step to achieve the desired results.

Specification	Type
Operation system	Windows 10
Processor	Intel i5-4570 3.20GHz
RAM	8 GB

Table 2: The specifications of the computer used for the execution time benchmarking

4.1.2 Overall quality

For the removal of inside points, a simple method has been chosen as has been explained in the methodology. This means that inside points that are close to the facade itself remain.

This can be seen in figure 25 below - while most of them are removed, there is still noise behind the facade. To remove these points, a more sophisticated method has to be used. One if these is by using RANSAC. This has been noticed during the segmentation part, when using suitable parameters. This is also the used method in [45] for this purpose. However, another advantage of removing a part of the inside points is a reduced file size of the point cloud.

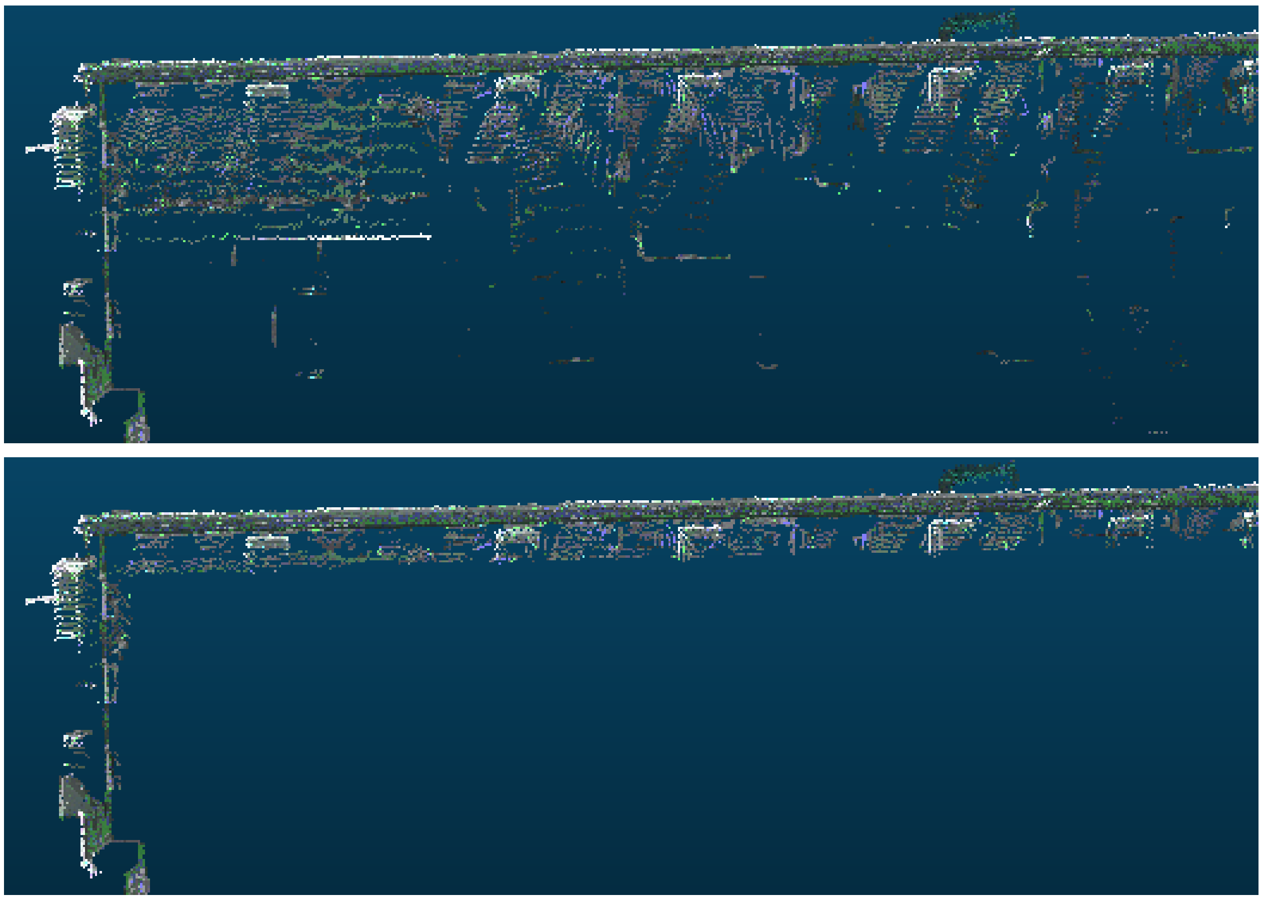


Figure 25: The difference between before (top) and after (below) removal of inside points

Close to the facades, some artefacts can be found. Examples of these are shown in appendix B, figures 50, 51, 52. The reason for this is that they are above the ground points and thus not removed in that way, and also are close enough to a building to fall within the buffers that are made around the buildings. While not ideal, these kind of artefacts are removed during the RANSAC segmentation part and therefore do not pose an issue.

Overall, 71 percent of the points of the original dataset are removed after the filtering step - 34.023.119 points are left, out of the 117.485.603 original ones. By lessening the file size and the amount of unneeded points, the computation speed and error proneness of the following steps are decreased. The problems that have been pointed out form no significant problems, and a large share of the resulting point cloud is clean. An example of a clean part of the final outcome is shown in appendix B, figures 53, 56. Lastly, from figure 26, it can be told that all points have been assigned to a proper corresponding BAG ID. This is the case for the complete dataset.

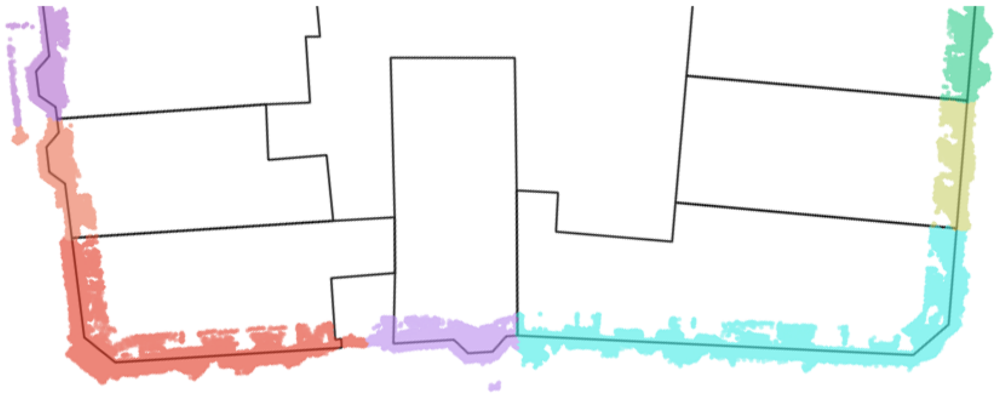


Figure 26: Subset of clusters of facade points coloured per unique BAG ID property, with on the background the boundaries of BAG features.

4.2 Window detection and surface ratio calculation assessment

Every research cannot be considered as done without proper assessment of the results with respect to the data wanted to perform benchmarking on. The dataset chosen for that role should have several characteristics which are making it reliable in a mentioned application. At first, it should cover all the aspect interesting from the point of view of benchmark. In this particular situation, the most interesting part from the researcher's point of view is the robustness to relatively clear input data (without major occlusion caused by balconies or features temporally placed between instrument and facades, appearing as artefacts too close to them to be automatically removed by filtering. Proposed algorithms are sensitive to distortions, and have limited possibilities for adjusting to poor quality data – the choice of low quality benchmarking data can cause difficulties in assessment of technical solutions, since major errors would be caused by that and recognition of minor mistakes would be impossible.

Because the proposed methods differ between each other, there is no point to provide unified way of benchmarking. The steps taken in this procedure should be chosen in respect to characteristics of specific method.

4.2.1 General Principles of Assessment

2D Bitmap Occupancy Map method:

Since the method is relying on several assumptions, it is important to prepare the benchmark procedure that will help with assessment of the approach on wide extent. Also, the benchmark should focus on crucial elements of the algorithms – related directly to the most important part of the results. In this particular case, it is more important to assess if windows/facade ratio is preserved than to assess if windows are detected in specific positions. Obviously, good quality of second information would be appreciated, however it is not crucial to obtain the results and should not be taken into consideration in conclusions assessing this part of the project.

Since the algorithm is following several steps to reach the final one with formal calculation of ratio, assessment of results that are obtained in middle part of the work flow can make possible to acquire information on more detailed level and explore specific problems. Crucial elements of the workflow, considering benchmarking, are listed as follows:

- Calculation of surface (area) and size of windows (height-width)
- Comparison of the results to classes and clustering the results to the groups with same class
- Assignation of the width, height and area to the window basing on the class characteristics
- Calculation of window-facade ratio
- Conclusion about differences in different direction of deviation (vertical and horizontal)

To conclude which parameters should be taken into account during assessment it is necessary to evaluate the dependencies between them. From one side, there is no point in assessment of area of the window compare to the benchmarking data, since it would be impossible to conclude if the problem is related to width or height calculation. From the other side, crucial functionality of the algorithm is related to area – and even with significant error in both of the dimensions the results can be still considered as accurate, as long as deviation of width and height is opposite. Because of that, to obtain most information possible, the benchmarking is performed on both of the approaches – with respect to height and width and area.

Slicing method:

The idea behind slicing method vary greatly from previously mentioned one – and that means, that there is a necessity to provide second proposition for benchmarking, suitable for this specific approach. The main idea is relying on the assumption, that extreme points are describing the boundaries. Obviously, the term extreme points requires some explanation. At the principles, the algorithm is identifying the point as extreme in specific slice, if the distance between the current point and the next one satisfies the threshold value that corresponds to a gap in the dataset and, therefore, a window. This assumption requires to take into consideration at least one important factor – density of the points. Point cloud is not continuous; without assumption about its sampling, the idea is worthless. However, the point cloud was collected from ground level with instruments with its own limitations – density of the point cloud is decreasing with increasing height of the building. It can lead to the problem, that assumptions correctly describing the facade at the ground level are making the whole analysis impossible on the higher floors.

Second problem is related with clustering – if density of the points is not high in respect to previous assumptions, it is problematic to conclude which points are related to which window – and lead to clustering of windows, which are in close distance to each other. Regardless to that, this problem should not be considered as crucial; the clustering of windows is only problematic in the situation when the frames are close to each other. Thus, area of them is comparable to sum of two separate one, and in respect to other problems driving the algorithm it is not essential to conclude about it robustness and application possibilities.

Since the algorithm is based on different principles than previous one, the way of deriving final ratio will differ as well. In this situation, there is no class assignment – all the windows are treated separately, without generalization. The area of them is considered as area of a polygon, which is created by the Density-Based Spatial Clustering of Applications with Noise. Besides that, the area of the bounding box around clusters was calculated, to provide addition information about possible approaches in derivation of solution.

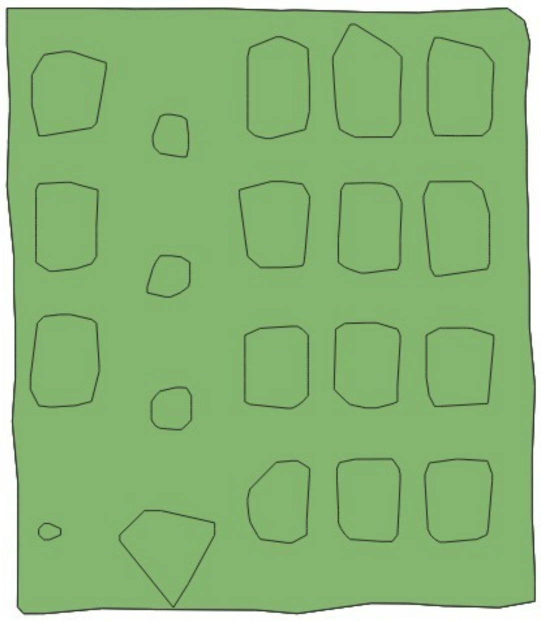


Figure 27: Clustered windows retrieved after applying DBSCAN. The polygons that are created will be used to calculate the windows-facade area ratio.

Because problem driving the crucial part of this method (finding threshold for extreme point recognition) is not constant on the whole facade, but it is dependent on the height in respect to height of measuring instrument, it is crucial to assess the results with respect to that information. Specific floors on the facade will be assessed separately. Besides that, the general comparison will be performed, with respect to windows/facade ratio.

4.2.2 Benchmark data preparation

To perform actual assessment, there is a necessity obtain data for which the scientists can be sure about the quality of them. For the purpose of the project, photogrammetrical solution was taken into consideration. At first, the calibration procedure was performed to obtain intrinsic and extrinsic parameters of the camera, making possible to transform coordinates from image plane to real world coordinates, necessary to obtain width and height of the windows. The calibration was made with usage of Image Master software and final results were derived in Python 3.7 with usage of collinearity equation, mentioned in previous sections. The facades for benchmarks were chosen from original dataset acquired in Rotterdam, to check it's robustness in this particular environment.

The requirements for both benchmarks were obtained as follows: Slicing method (for every window):

- facade id
- window id (local, global identification can be done together with facade id)
- height and width of the window's bounding box
- floor of the window
- area calculated with shoelace algorithm
- area calculated as height*width

Bitmap method (for every window class)

- facade id
- window class id
- height of window class
- width of window class
- number of windows in window class
- area calculated as height*width

Besides that, separate table with height and width of every facade was created, for control purposes. To confront all the requirements mentioned in previous sub chapters, 6 facades were chosen to perform the benchmark and group into 2 separate subgroups:

Group 1:

This part of the benchmark is focused on evaluation of the method on various type of input data. In the subgroup facades number 1, 2 and 3 were chosen, and every of them is representing different part of the assessment and can be characterized as follows:

- facade 1 - 2 rows of windows of same size (floor 2 and 3), 1 row of windows with various size and occlusions caused by cars - to compare the different types of windows in same facade
- facade 2 - unified type of windows frames with few windows covered - to focus on impact of that factor which is crucial for method
- facade 3 - facade with 3 different types of windows with diversity too little to be taken into consideration during class assignment - evaluation of that impact on the final ratio result.

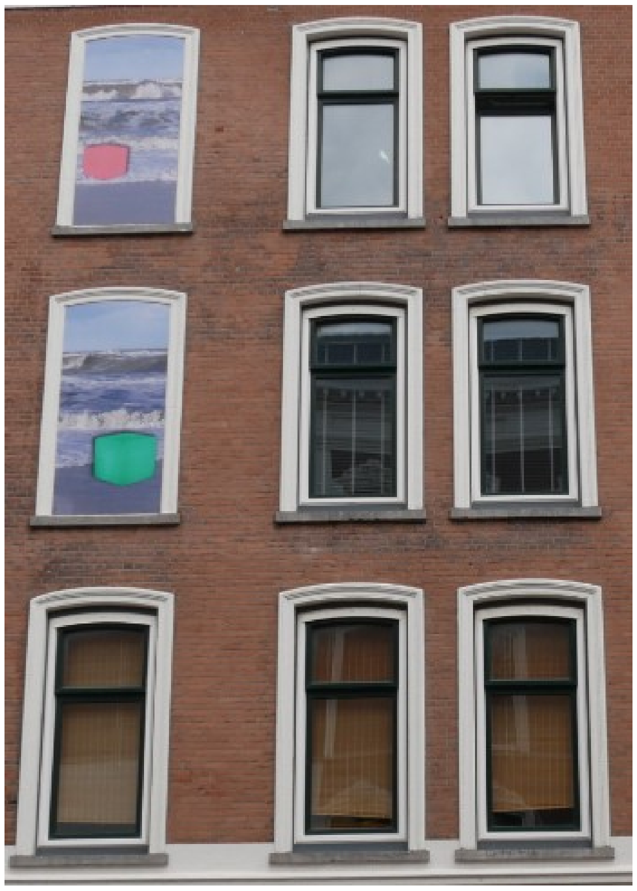


Figure 28: Facade 2 with fake window frames

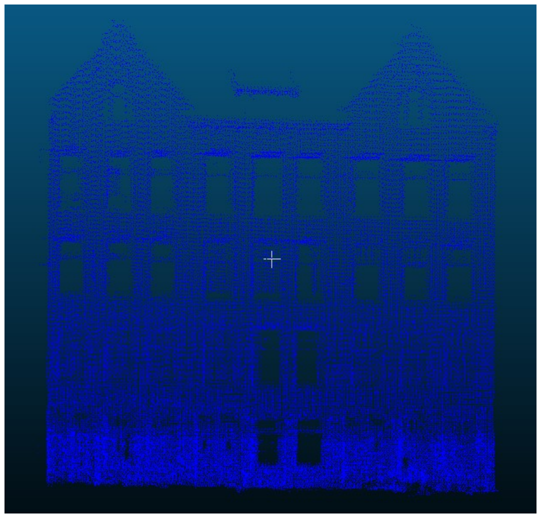


Figure 29: Facades 4,5 and 6 - similar in shape and size of windows.

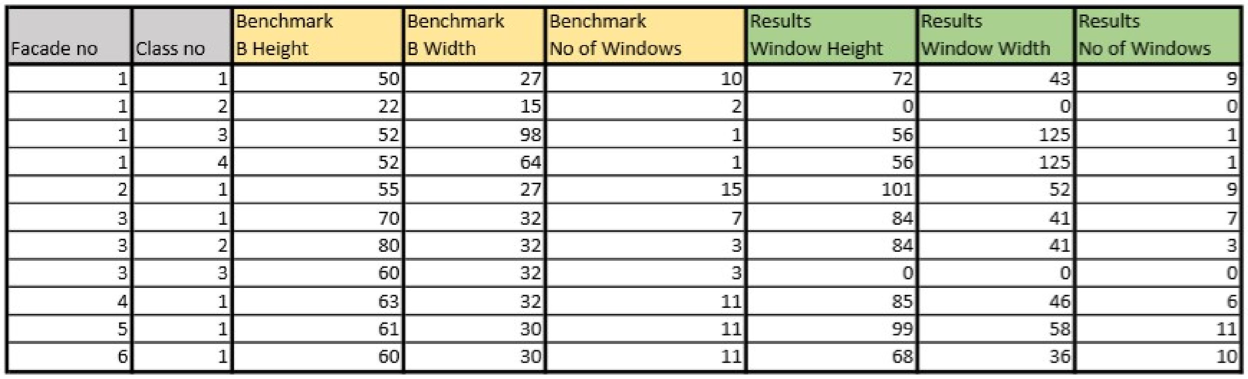
Group 2:

In this part the benchmark is focused on comparison of the results and robustness of the algorithm in the simplest case. The facades that were chosen are situated next to each other (to assume that the conditions during measurements were similar) with same type of window. Every one of them is tall compare to average height of the facades, which is making possible to conclude if robustness is constant regardless of density of points. Facade 5 was cleaned artificially to compare the results to "perfect" facade and provide more conclusions and reasoning.

4.2.3 Assessment results

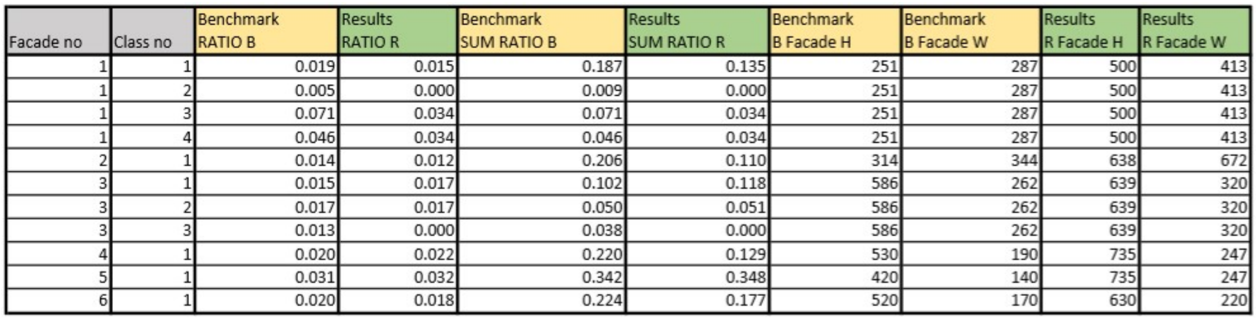
2D Bitmap Occupancy Map method:

The final results look as follows (bitmaps before and after windows detection are attached in appendix K):



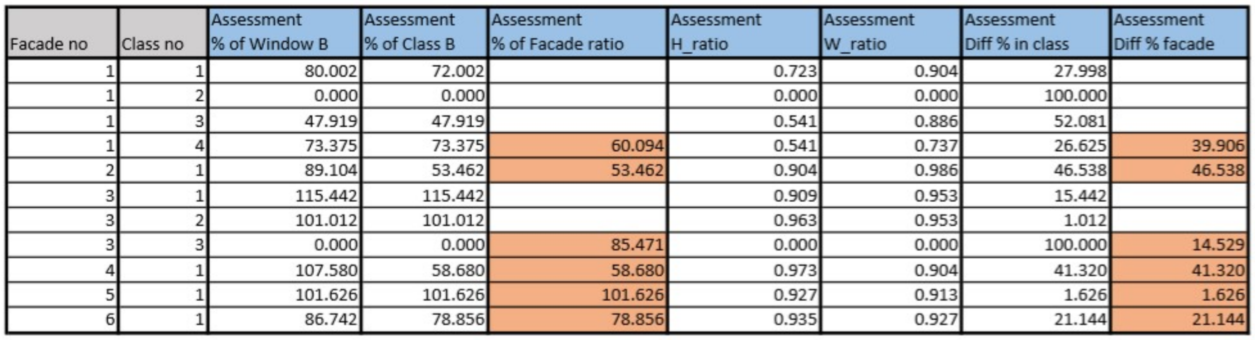
Facade no	Class no	Benchmark B Height	Benchmark B Width	Benchmark No of Windows	Results Window Height	Results Window Width	Results No of Windows
1	1	50	27	10	72	43	9
1	2	22	15	2	0	0	0
1	3	52	98	1	56	125	1
1	4	52	64	1	56	125	1
2	1	55	27	15	101	52	9
3	1	70	32	7	84	41	7
3	2	80	32	3	84	41	3
3	3	60	32	3	0	0	0
4	1	63	32	11	85	46	6
5	1	61	30	11	99	58	11
6	1	60	30	11	68	36	10

Figure 30: Obtained data from benchmark dataset and bitmap method.



Facade no	Class no	Benchmark RATIO B	Results RATIO R	Benchmark SUM RATIO B	Results SUM RATIO R	Benchmark B Facade H	Benchmark B Facade W	Results R Facade H	Results R Facade W
1	1	0.019	0.015	0.187	0.135	251	287	500	413
1	2	0.005	0.000	0.009	0.000	251	287	500	413
1	3	0.071	0.034	0.071	0.034	251	287	500	413
1	4	0.046	0.034	0.046	0.034	251	287	500	413
2	1	0.014	0.012	0.206	0.110	314	344	638	672
3	1	0.015	0.017	0.102	0.118	586	262	639	320
3	2	0.017	0.017	0.050	0.051	586	262	639	320
3	3	0.013	0.000	0.038	0.000	586	262	639	320
4	1	0.020	0.022	0.220	0.129	530	190	735	247
5	1	0.031	0.032	0.342	0.348	420	140	735	247
6	1	0.020	0.018	0.224	0.177	520	170	630	220

Figure 31: Required results neccessary to provide final assessment.



Facade no	Class no	Assessment % of Window B	Assessment % of Class B	Assessment % of Facade ratio	Assessment H_ratio	Assessment W_ratio	Assessment Diff % in class	Assessment Diff % facade
1	1	80.002	72.002		0.723	0.904	27.998	
1	2	0.000	0.000		0.000	0.000	100.000	
1	3	47.919	47.919		0.541	0.886	52.081	
1	4	73.375	73.375	60.094	0.541	0.737	26.625	39.906
2	1	89.104	53.462	53.462	0.904	0.986	46.538	46.538
3	1	115.442	115.442		0.909	0.953	15.442	
3	2	101.012	101.012		0.963	0.953	1.012	
3	3	0.000	0.000	85.471	0.000	0.000	100.000	14.529
4	1	107.580	58.680	58.680	0.973	0.904	41.320	41.320
5	1	101.626	101.626	101.626	0.927	0.913	1.626	1.626
6	1	86.742	78.856	78.856	0.935	0.927	21.144	21.144

Figure 32: Final results of the assessment.

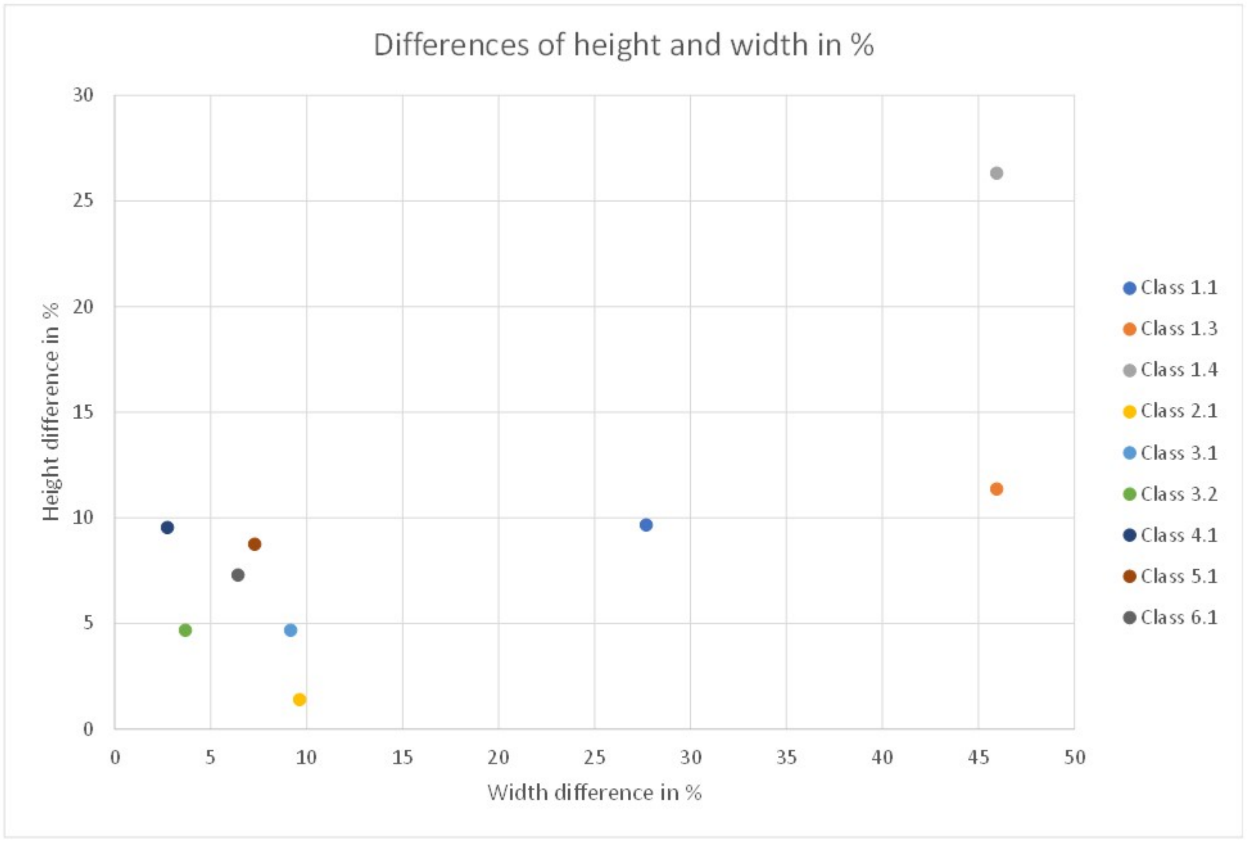


Figure 33: Plot of differences of height and width in % of benchmark width and height for every window class. Classes 1.2 and 3.3 were removed because the algorithm failed to find any example of window for every of them.

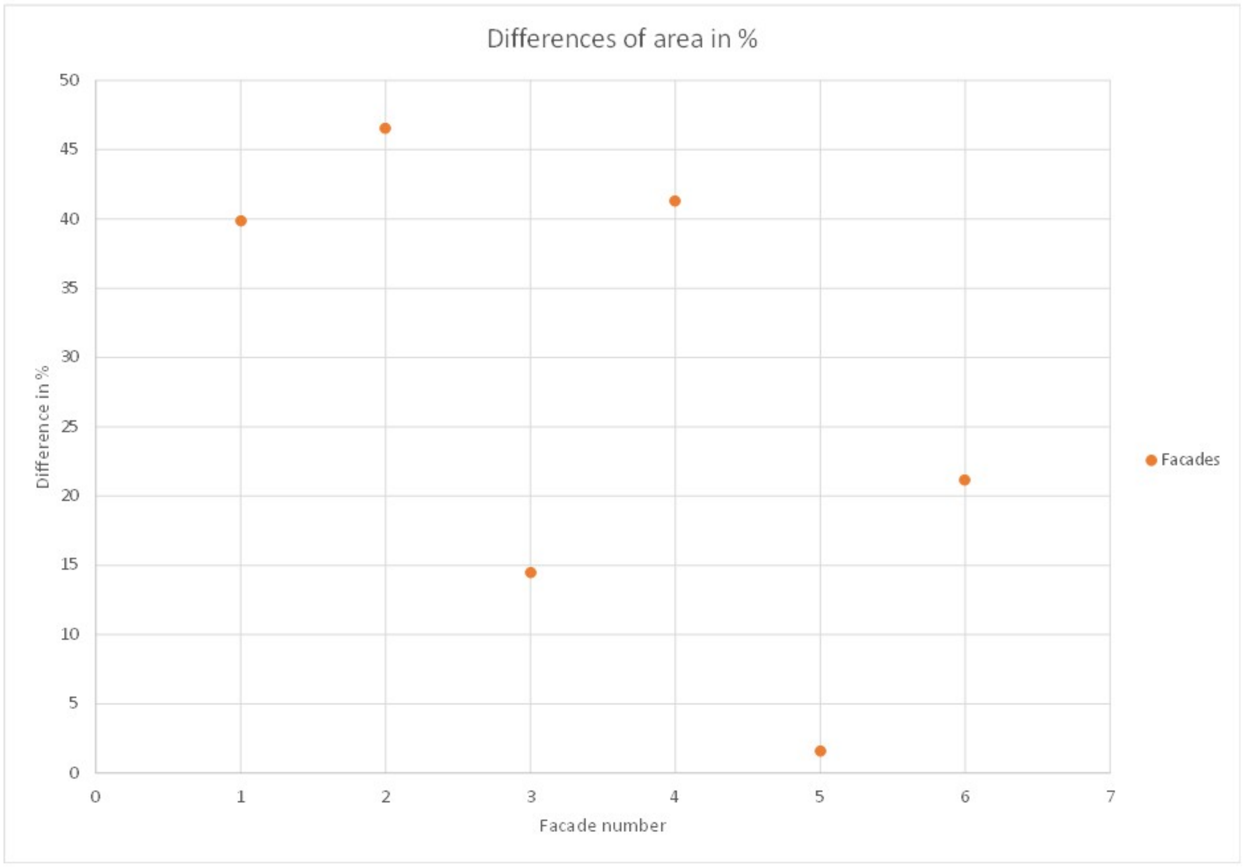


Figure 34: Plot of differences of area in % of benchmark area for every window class.

On the first facade it is possible to notice that main problem was driving the first floor of the building. Lots of occlusions, as well as too little windows (taken into consideration as noise and discarded) made the whole analysis low quality, even if some of the elements and most of the windows were found with acceptable accuracy.

In second facade the situation is different - the algorithm properly recognized the windows frame with wall inside. However, some windows were also counted as the wall (Figure 27). Direct analysis of point cloud is leading to conclusion, that it was caused by curtains which are close to the surface of the glass and were not remove by RANSAC as outliers. The problem is not directly related to approaches driving this algorithm and because of that it is not evaluated as the part of specific method assessment.

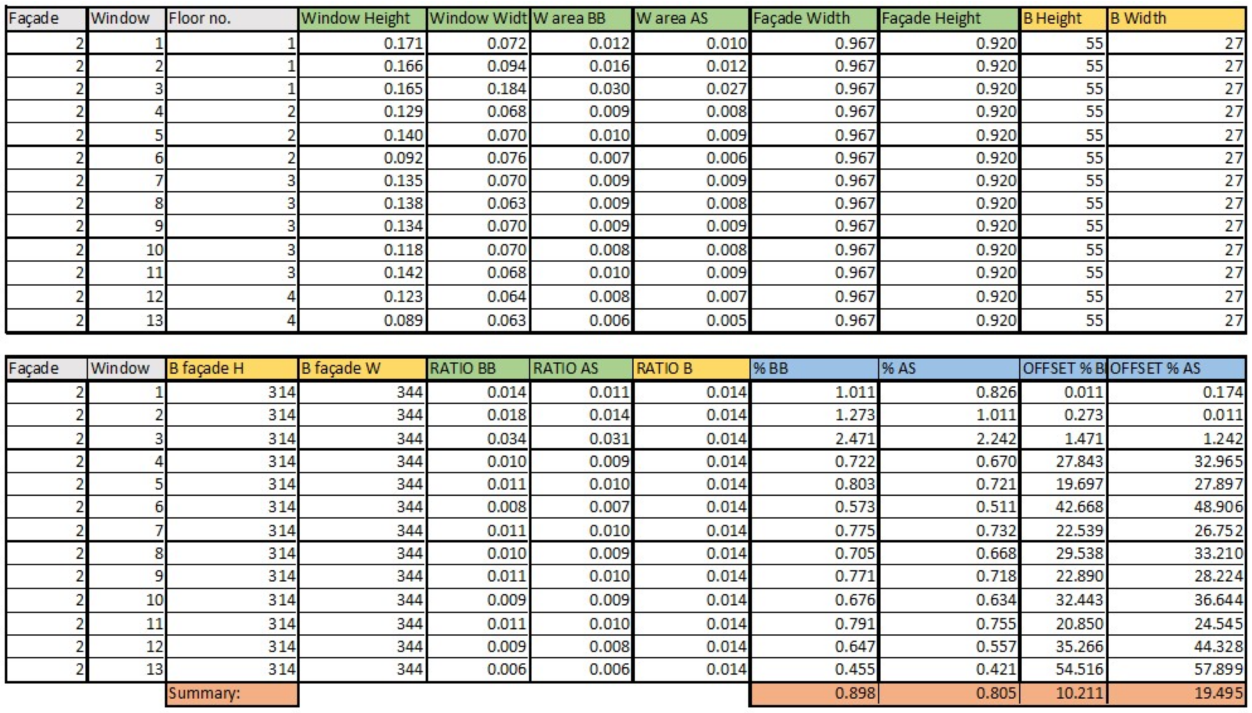
Third facade shows that (Figure 28), the windows were unified and only one type of them was reconstructed. However it is important to notice that the highest row of windows was not recognized at all - spread of points made the analysis impossible and shapes of windows were lost during pre processing.

The results of remaining part of the benchmark with facades with similar windows brought surprisingly accurate results when it comes to accuracy inside specific class - in all of them the accuracy was higher than 80%, with outstanding 98% for facade 5 (Figure 29). However, it is worth of noticing that ”dimensional accuracy” of the facade 5 was lower in both of the cases - which is leading to conclusion that 98% of the final accuracy is derived on random manner - one of the dimensions were overestimated and the other underestimated (Figure 30, Figure 31). Regardless of those facts, the overall accuracy of the results is still better than expected - low results of facade 4 were caused by covered windows (93% accuracy of window ratio, but only 6 out of 11 windows recognized - final ratio 59% accuracy).

The results show clearly that on every facade the algorithm was able to detect most of the windows. However, even with good accuracy in windows shape and size estimation, the number of windows found is low - in some cases 40% of windows were not detected at all. In this part of the assessment this effect is neglected due to the nature described above - it is not possible to improve the input dataset.

Slicing method:

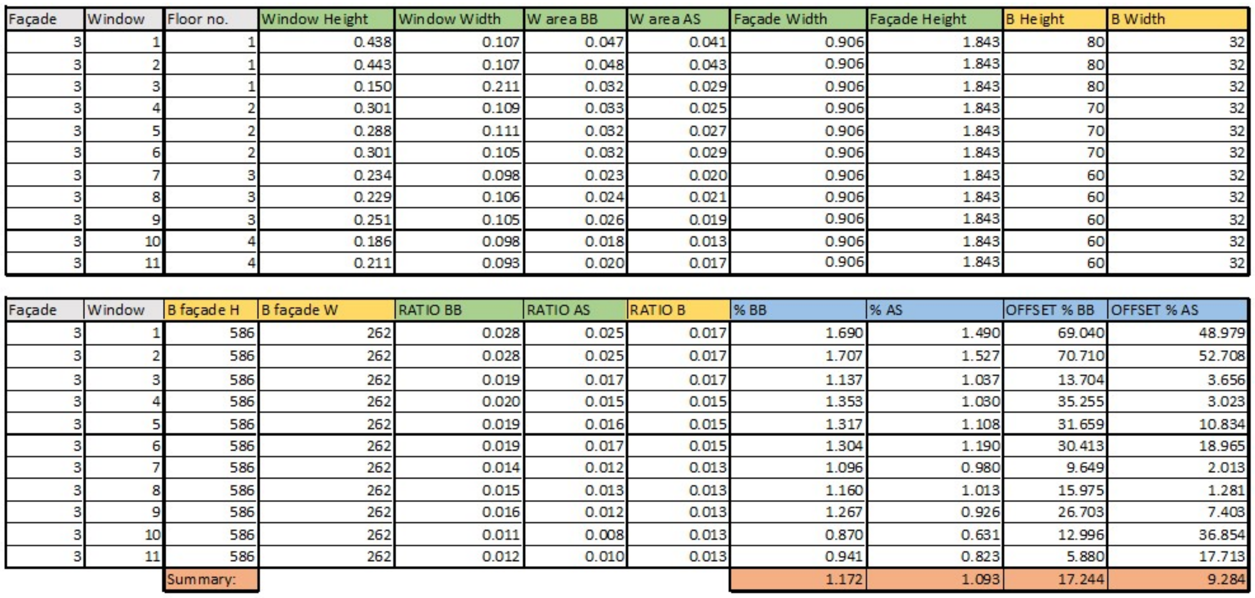
Facade 1 was discarded from set of facades assessed, because, due to the noise, it was not possible to derive results which could be used for benchmarking. The remaining results look as follows (detailed results for specific floors were provided as appendices):



Façade	Window	Floor no.	Window Height	Window Widt	W area BB	W area AS	Façade Width	Façade Height	B Height	B Width
2	1	1	0.171	0.072	0.012	0.010	0.967	0.920	55	27
2	2	1	0.166	0.094	0.016	0.012	0.967	0.920	55	27
2	3	1	0.165	0.184	0.030	0.027	0.967	0.920	55	27
2	4	2	0.129	0.068	0.009	0.008	0.967	0.920	55	27
2	5	2	0.140	0.070	0.010	0.009	0.967	0.920	55	27
2	6	2	0.092	0.076	0.007	0.006	0.967	0.920	55	27
2	7	3	0.135	0.070	0.009	0.009	0.967	0.920	55	27
2	8	3	0.138	0.063	0.009	0.008	0.967	0.920	55	27
2	9	3	0.134	0.070	0.009	0.009	0.967	0.920	55	27
2	10	3	0.118	0.070	0.008	0.008	0.967	0.920	55	27
2	11	3	0.142	0.068	0.010	0.009	0.967	0.920	55	27
2	12	4	0.123	0.064	0.008	0.007	0.967	0.920	55	27
2	13	4	0.089	0.063	0.006	0.005	0.967	0.920	55	27

Façade	Window	B façade H	B façade W	RATIO BB	RATIO AS	RATIO B	% BB	% AS	OFFSET % B	OFFSET % AS
2	1	314	344	0.014	0.011	0.014	1.011	0.826	0.011	0.174
2	2	314	344	0.018	0.014	0.014	1.273	1.011	0.273	0.011
2	3	314	344	0.034	0.031	0.014	2.471	2.242	1.471	1.242
2	4	314	344	0.010	0.009	0.014	0.722	0.670	27.843	32.965
2	5	314	344	0.011	0.010	0.014	0.803	0.721	19.697	27.897
2	6	314	344	0.008	0.007	0.014	0.573	0.511	42.668	48.906
2	7	314	344	0.011	0.010	0.014	0.775	0.732	22.539	26.752
2	8	314	344	0.010	0.009	0.014	0.705	0.668	29.538	33.210
2	9	314	344	0.011	0.010	0.014	0.771	0.718	22.890	28.224
2	10	314	344	0.009	0.009	0.014	0.676	0.634	32.443	36.644
2	11	314	344	0.011	0.010	0.014	0.791	0.755	20.850	24.545
2	12	314	344	0.009	0.008	0.014	0.647	0.557	35.266	44.328
2	13	314	344	0.006	0.006	0.014	0.455	0.421	54.516	57.899
		Summary:					0.898	0.805	10.211	19.495

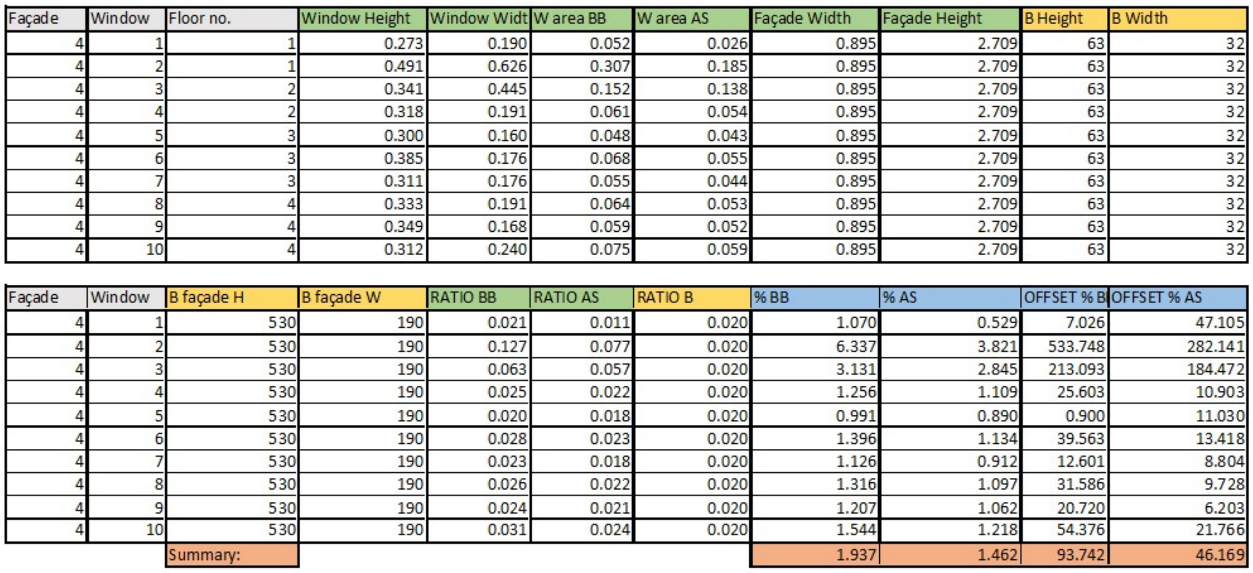
Figure 35: Table with results for facade 2.



Façade	Window	Floor no.	Window Height	Window Width	W area BB	W area AS	Façade Width	Façade Height	B Height	B Width
3	1	1	0.438	0.107	0.047	0.041	0.906	1.843	80	32
3	2	1	0.443	0.107	0.048	0.043	0.906	1.843	80	32
3	3	1	0.150	0.211	0.032	0.029	0.906	1.843	80	32
3	4	2	0.301	0.109	0.033	0.025	0.906	1.843	70	32
3	5	2	0.288	0.111	0.032	0.027	0.906	1.843	70	32
3	6	2	0.301	0.105	0.032	0.029	0.906	1.843	70	32
3	7	3	0.234	0.098	0.023	0.020	0.906	1.843	60	32
3	8	3	0.229	0.106	0.024	0.021	0.906	1.843	60	32
3	9	3	0.251	0.105	0.026	0.019	0.906	1.843	60	32
3	10	4	0.186	0.098	0.018	0.013	0.906	1.843	60	32
3	11	4	0.211	0.093	0.020	0.017	0.906	1.843	60	32

Façade	Window	B façade H	B façade W	RATIO BB	RATIO AS	RATIO B	% BB	% AS	OFFSET % BB	OFFSET % AS
3	1	586	262	0.028	0.025	0.017	1.690	1.490	69.040	48.979
3	2	586	262	0.028	0.025	0.017	1.707	1.527	70.710	52.708
3	3	586	262	0.019	0.017	0.017	1.137	1.037	13.704	3.656
3	4	586	262	0.020	0.015	0.015	1.353	1.030	35.255	3.023
3	5	586	262	0.019	0.016	0.015	1.317	1.108	31.659	10.834
3	6	586	262	0.019	0.017	0.015	1.304	1.190	30.413	18.965
3	7	586	262	0.014	0.012	0.013	1.096	0.980	9.649	2.013
3	8	586	262	0.015	0.013	0.013	1.160	1.013	15.975	1.281
3	9	586	262	0.016	0.012	0.013	1.267	0.926	26.703	7.403
3	10	586	262	0.011	0.008	0.013	0.870	0.631	12.996	36.854
3	11	586	262	0.012	0.010	0.013	0.941	0.823	5.880	17.713
		Summary:					1.172	1.093	17.244	9.284

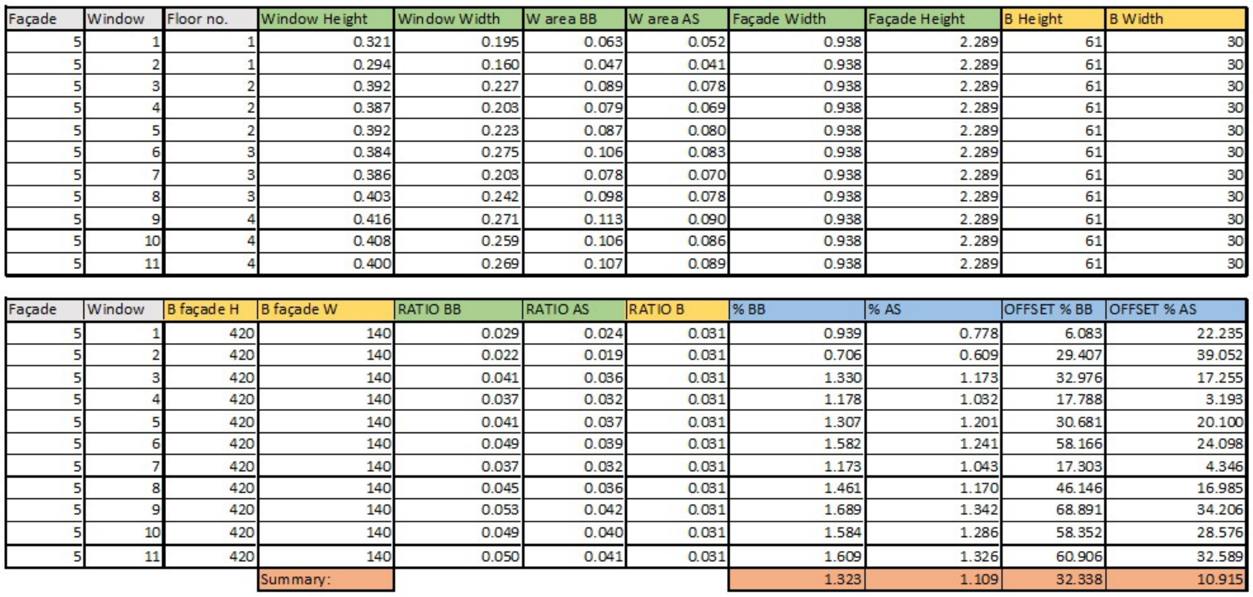
Figure 36: Table with results for facade 3.



Façade	Window	Floor no.	Window Height	Window Widt	W area BB	W area AS	Façade Width	Façade Height	B Height	B Width
4	1	1	0.273	0.190	0.052	0.026	0.895	2.709	63	32
4	2	1	0.491	0.626	0.307	0.185	0.895	2.709	63	32
4	3	2	0.341	0.445	0.152	0.138	0.895	2.709	63	32
4	4	2	0.318	0.191	0.061	0.054	0.895	2.709	63	32
4	5	3	0.300	0.160	0.048	0.043	0.895	2.709	63	32
4	6	3	0.385	0.176	0.068	0.055	0.895	2.709	63	32
4	7	3	0.311	0.176	0.055	0.044	0.895	2.709	63	32
4	8	4	0.333	0.191	0.064	0.053	0.895	2.709	63	32
4	9	4	0.349	0.168	0.059	0.052	0.895	2.709	63	32
4	10	4	0.312	0.240	0.075	0.059	0.895	2.709	63	32

Façade	Window	B façade H	B façade W	RATIO BB	RATIO AS	RATIO B	% BB	% AS	OFFSET % B	OFFSET % AS
4	1	530	190	0.021	0.011	0.020	1.070	0.529	7.026	47.105
4	2	530	190	0.127	0.077	0.020	6.337	3.821	533.748	282.141
4	3	530	190	0.063	0.057	0.020	3.131	2.845	213.093	184.472
4	4	530	190	0.025	0.022	0.020	1.256	1.109	25.603	10.903
4	5	530	190	0.020	0.018	0.020	0.991	0.890	0.900	11.030
4	6	530	190	0.028	0.023	0.020	1.396	1.134	39.563	13.418
4	7	530	190	0.023	0.018	0.020	1.126	0.912	12.601	8.804
4	8	530	190	0.026	0.022	0.020	1.316	1.097	31.586	9.728
4	9	530	190	0.024	0.021	0.020	1.207	1.062	20.720	6.203
4	10	530	190	0.031	0.024	0.020	1.544	1.218	54.376	21.766
		Summary:					1.937	1.462	93.742	46.169

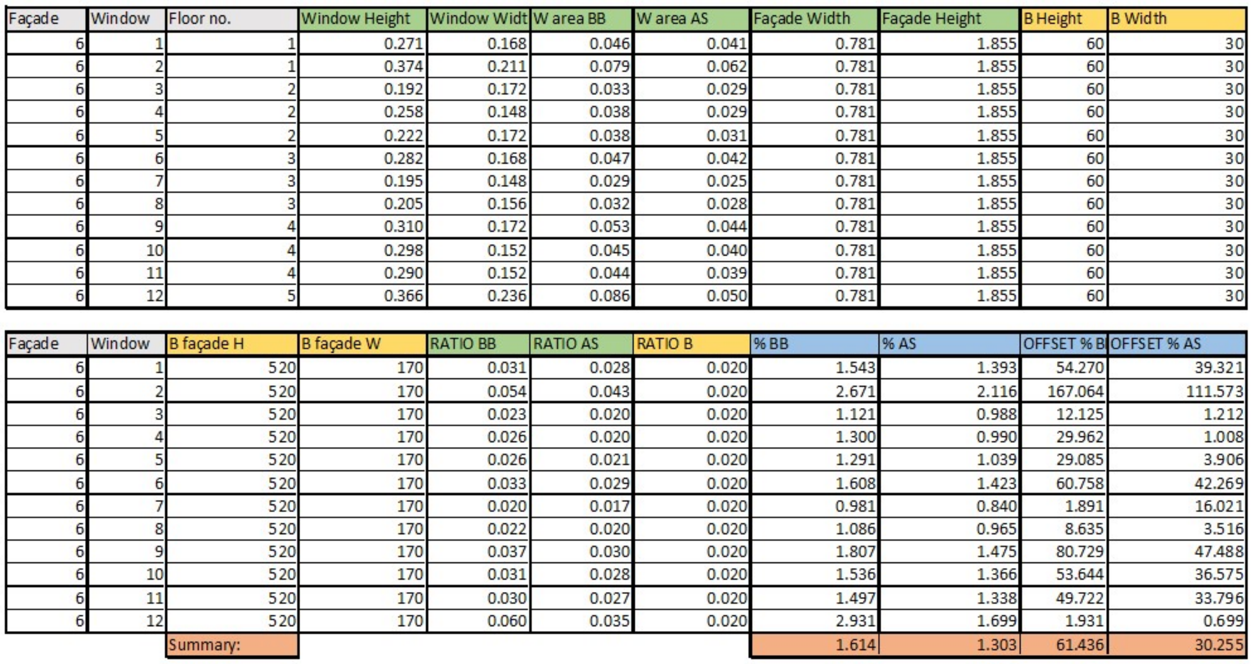
Figure 37: Table with results for facade 4.



Façade	Window	Floor no.	Window Height	Window Width	W area BB	W area AS	Façade Width	Façade Height	B Height	B Width
5	1	1	0.321	0.195	0.063	0.052	0.938	2.289	61	30
5	2	1	0.294	0.160	0.047	0.041	0.938	2.289	61	30
5	3	2	0.392	0.227	0.089	0.078	0.938	2.289	61	30
5	4	2	0.387	0.203	0.079	0.069	0.938	2.289	61	30
5	5	2	0.392	0.223	0.087	0.080	0.938	2.289	61	30
5	6	3	0.384	0.275	0.106	0.083	0.938	2.289	61	30
5	7	3	0.386	0.203	0.078	0.070	0.938	2.289	61	30
5	8	3	0.403	0.242	0.098	0.078	0.938	2.289	61	30
5	9	4	0.416	0.271	0.113	0.090	0.938	2.289	61	30
5	10	4	0.408	0.259	0.106	0.086	0.938	2.289	61	30
5	11	4	0.400	0.269	0.107	0.089	0.938	2.289	61	30

Façade	Window	B façade H	B façade W	RATIO BB	RATIO AS	RATIO B	% BB	% AS	OFFSET % BB	OFFSET % AS
5	1	420	140	0.029	0.024	0.031	0.939	0.778	6.083	22.235
5	2	420	140	0.022	0.019	0.031	0.706	0.609	29.407	39.052
5	3	420	140	0.041	0.036	0.031	1.330	1.173	32.976	17.255
5	4	420	140	0.037	0.032	0.031	1.178	1.032	17.788	3.193
5	5	420	140	0.041	0.037	0.031	1.307	1.201	30.681	20.100
5	6	420	140	0.049	0.039	0.031	1.582	1.241	58.166	24.098
5	7	420	140	0.037	0.032	0.031	1.173	1.043	17.303	4.346
5	8	420	140	0.045	0.036	0.031	1.461	1.170	46.146	16.985
5	9	420	140	0.053	0.042	0.031	1.689	1.342	68.891	34.206
5	10	420	140	0.049	0.040	0.031	1.584	1.286	58.352	28.576
5	11	420	140	0.050	0.041	0.031	1.609	1.326	60.906	32.589
			Summary:				1.323	1.109	32.338	10.915

Figure 38: Table with results for facade 5.



Façade	Window	Floor no.	Window Height	Window Widt	W area BB	W area AS	Façade Width	Façade Height	B Height	B Width
6	1	1	0.271	0.168	0.046	0.041	0.781	1.855	60	30
6	2	1	0.374	0.211	0.079	0.062	0.781	1.855	60	30
6	3	2	0.192	0.172	0.033	0.029	0.781	1.855	60	30
6	4	2	0.258	0.148	0.038	0.029	0.781	1.855	60	30
6	5	2	0.222	0.172	0.038	0.031	0.781	1.855	60	30
6	6	3	0.282	0.168	0.047	0.042	0.781	1.855	60	30
6	7	3	0.195	0.148	0.029	0.025	0.781	1.855	60	30
6	8	3	0.205	0.156	0.032	0.028	0.781	1.855	60	30
6	9	4	0.310	0.172	0.053	0.044	0.781	1.855	60	30
6	10	4	0.298	0.152	0.045	0.040	0.781	1.855	60	30
6	11	4	0.290	0.152	0.044	0.039	0.781	1.855	60	30
6	12	5	0.366	0.236	0.086	0.050	0.781	1.855	60	30

Façade	Window	B façade H	B façade W	RATIO BB	RATIO AS	RATIO B	% BB	% AS	OFFSET % B	OFFSET % AS
6	1	520	170	0.031	0.028	0.020	1.543	1.393	54.270	39.321
6	2	520	170	0.054	0.043	0.020	2.671	2.116	167.064	111.573
6	3	520	170	0.023	0.020	0.020	1.121	0.988	12.125	1.212
6	4	520	170	0.026	0.020	0.020	1.300	0.990	29.962	1.008
6	5	520	170	0.026	0.021	0.020	1.291	1.039	29.085	3.906
6	6	520	170	0.033	0.029	0.020	1.608	1.423	60.758	42.269
6	7	520	170	0.020	0.017	0.020	0.981	0.840	1.891	16.021
6	8	520	170	0.022	0.020	0.020	1.086	0.965	8.635	3.516
6	9	520	170	0.037	0.030	0.020	1.807	1.475	80.729	47.488
6	10	520	170	0.031	0.028	0.020	1.536	1.366	53.644	36.575
6	11	520	170	0.030	0.027	0.020	1.497	1.338	49.722	33.796
6	12	520	170	0.060	0.035	0.020	2.931	1.699	1.931	0.699
		Summary:					1.614	1.303	61.436	30.255

Figure 39: Table with results for facade 6.

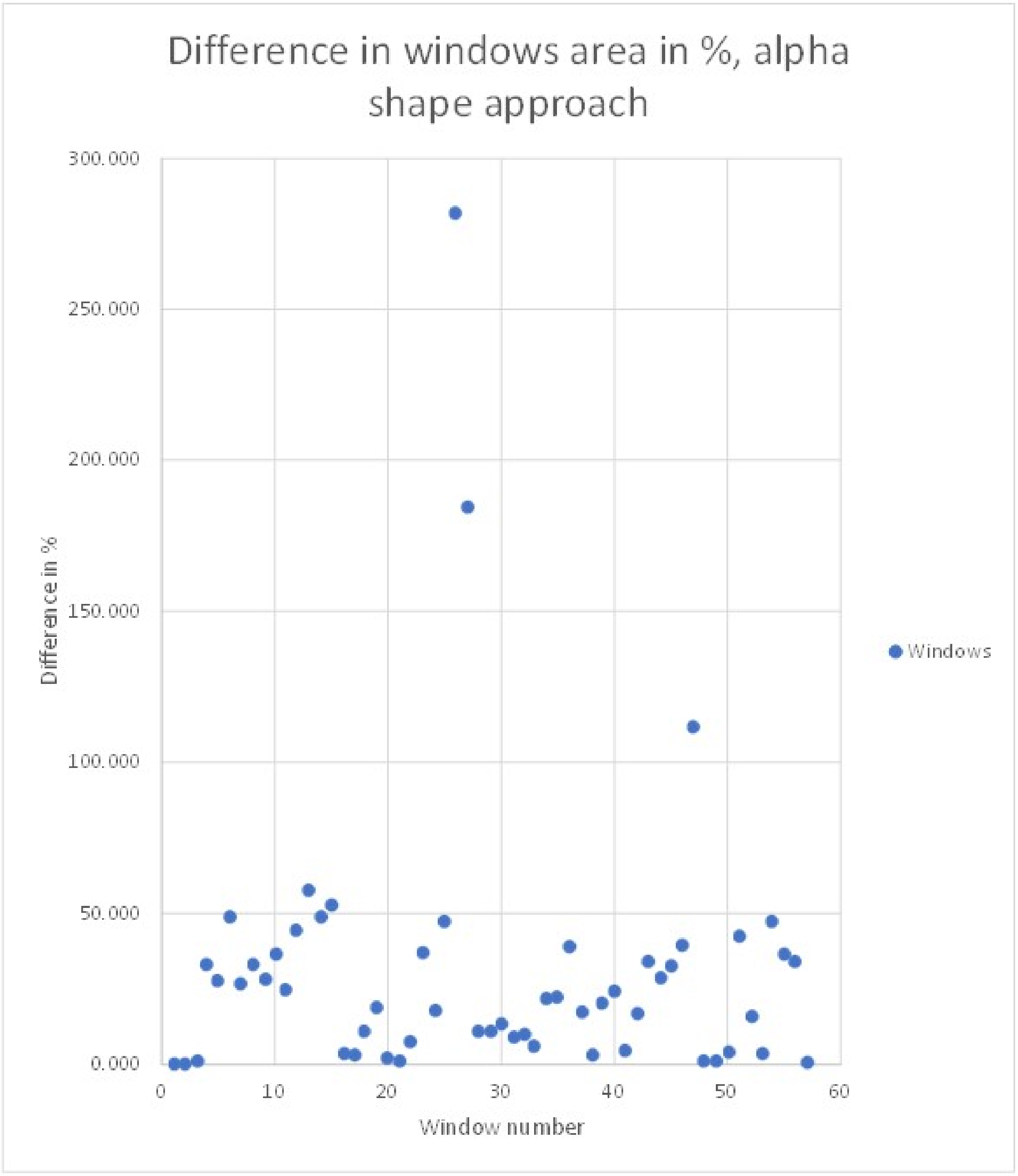


Figure 40: Plot of differences of area in % of benchmark area for every window - alpha shape areas

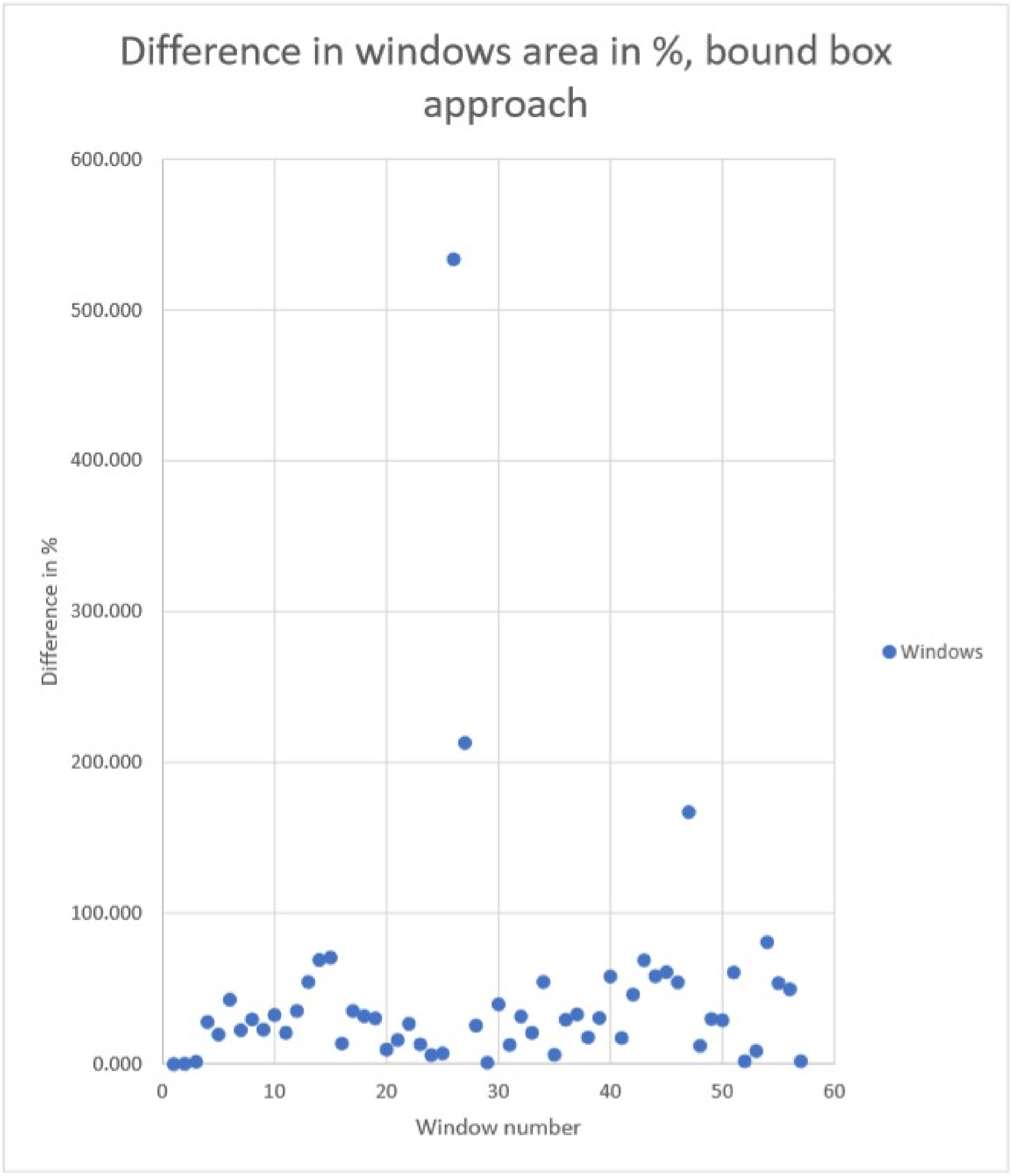


Figure 41: Plot of differences of area in % of benchmark area for every window - bounding box areas.

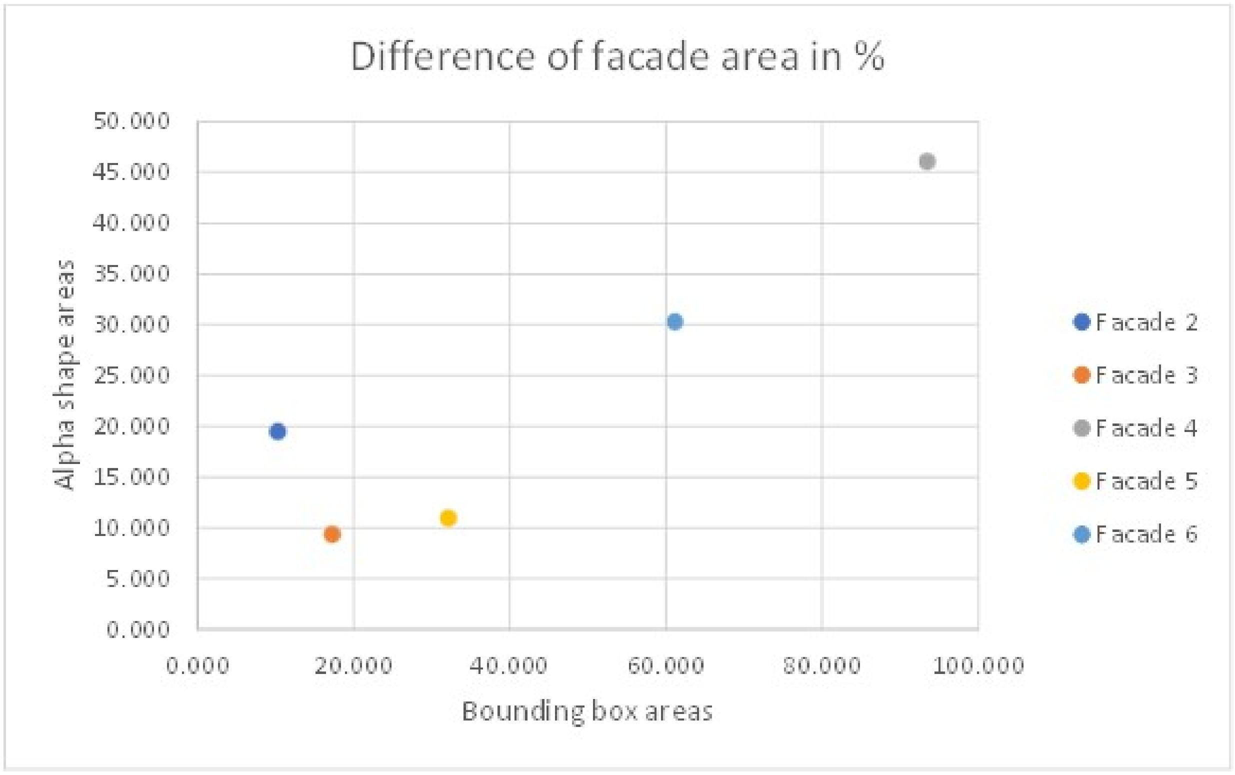


Figure 42: Plot of differences of area in % of benchmark area for every facade. Two approaches of area estimation were taken into consideration: area of bounding box around the shape and area of the shape.

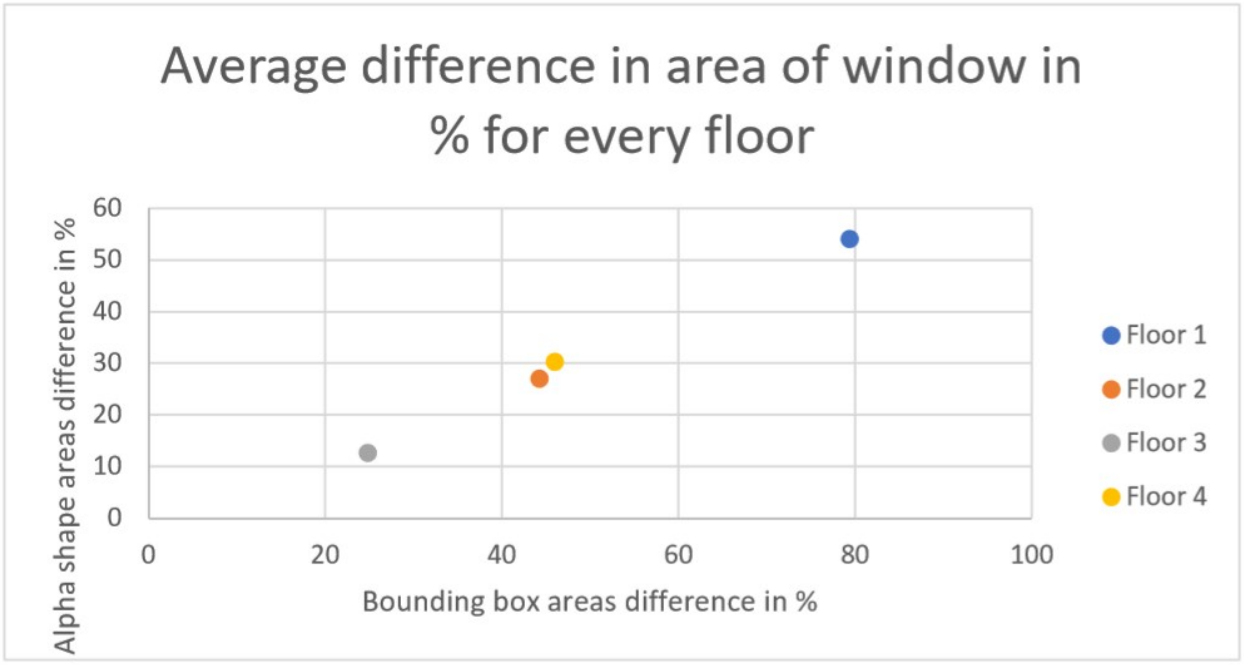


Figure 43: Plot of differences of area in % of benchmark area for every floor. Two approaches of area estimation were taken into consideration: area of bounding box around the shape and area of the shape.

In the first group on the second facade (Figure 32) the algorithm was able to reproduce most of the windows possible to see. It is worth of noticing, that it successfully distinguished empty window frames (with no windows inside). The windows on third floor seems to be reconstructed on the most accurate way. The algorithm reconstructed also the door, which is leading to conclusion that door frame have to be noticeably thicker than window frame and the points representing door were counted as interior (curtain etc.). Also, facade 2 is the only facade that bounding box approach reconstructed windows better, than alpha shape approach. On the third facade (figure 33), the algorithm was the most successful among all of the facades, with high accuracy in both of the methods. The windows were reconstructed accurately and no artefacts were considered as windows. The algorithm had problem with windows on the highest floor, which can be explainable by low density of the points cloud.

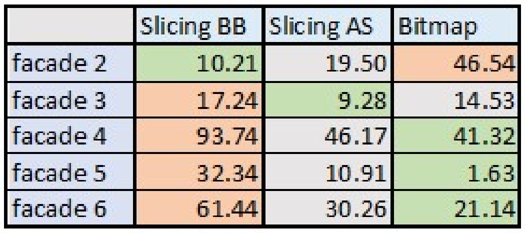
On the fourth facade, as first facade from the second group, (Figure 34) the situation differs; the slicing method could not reproduce windows properly, especially on the ground floor (the most probable reason for that is occlusion – making the shapes irregular). It is worth of noticing that the problem is increased in the case of bounding box – because of irregularity of the alpha shapes (Figure 39). On the fifth facade (Figure 35), the algorithm was successful especially with alpha shape approach. Obviously, facade 5 was prepared and "cleaned" - the expectations for the results were comparable to actually derived ones. Similar to facades

3 and 4, bounding box approach was not accurate enough to assume it as successful – the difference in ratio estimation was more than 30%. The last facade shows comparable results to facade 4 – both of the algorithms had problems with windows reconstruction (Figure 36). It is worth of noticing, that it is the only facade with clearly visible artefact on the top of it – the algorithm found the window in the place where there is no window in the real situation. The most probable reason is the same as previously mentioned – low density of points.

With information about all the facades, the benchmark results were also divided dependent on floors of the building (understand as horizontal rows of windows) and plotted on the diagram (Figure 40). The biggest errors are caused on the first level (ground floor), which can be explained by occlusions and irregular shapes of windows due to storefronts and irregular doors. The remaining floors are not making possible to conclude about the influence of point density on the windows estimation on specific facade. The assumption is that variance caused by different factors made this part of the assessment impossible to finish.

4.2.4 Assessment results - general conclusions

The final comparison of the results looks as follows:



	Slicing BB	Slicing AS	Bitmap
facade 2	10.21	19.50	46.54
facade 3	17.24	9.28	14.53
facade 4	93.74	46.17	41.32
facade 5	32.34	10.91	1.63
facade 6	61.44	30.26	21.14

Figure 44: Comparison of the results obtained with Bitmap and Slicing Method. The numbers represent the different in % of benchmark facade. For Slicing Method, two sub-methods were assessed separately and for every facade the most and at least accurate results were marked (green and red colour respectively).

The results show that both of the method have strong and weak sides (Figure 44). The slicing approach obtained more accurate results in the first group of the facades. However, it is important to underline, that first facade was discarded - which is leading to conclusion, that robustness of this method (especially in alpha shape version) should not be taken as granted. The algorithm has problems with major occlusions, pure point cloud density and missing information, which is leading to artefacts and wrongly reconstructed windows.

In the second group, bitmap approach seem to be more suitable. The algorithm had no problem with decreasing density of the points - but only to certain extend. on One of the facades last floor was totally ignored by the algorithm and no windows were detected. The conclusion is, that relation is not linear - as long as the density of points is high, which means that the distance between the points is not bigger than size of the cell, the problem is not present and correct results are possible to obtain.

5 Applications

To make a link between the technical parts of the window detection that has been performed during this project and the real world, it is necessary to explain why and how the end results of these algorithms are useful in practice. The major application of the detection of windows that has been taken in mind during the project, is the estimation of the energy usage of buildings. Besides this, the use for tunneling, fire propagation and indoors will be explained in this chapter. But firstly, the advantages and added use of direct point cloud analysis will be examined.

5.1 Benefits of a direct point cloud analysis

Since the project revolves around direct point cloud analysis, the window detection algorithms are supposed to result into a rich point cloud, which is a point cloud with added information that would act as a 3D city model on its own. These added semantics can be generated with automated classification of points and the detection of objects, such as windows. This means that for example 3D surface reconstruction is not involved, which usually takes point clouds as input for information with which generalized shapes are created [47]. Creating a rich point cloud is a conscious choice, since there are several advantages to both performing the analysis directly on a point cloud and having a point cloud as result, rather than transforming the data to another representation.

Point clouds are more detailed, since they are less generalized than other data types (vector, raster). This means that the representation of objects in a point cloud can be more realistic, such as trees with better leaves. Furthermore, the method of acquisition is mostly automated. When maintaining the point cloud representation, there is also no conversion time involved to transform the data to other types. Reconstructed surfaces on the other hand are expensive to compute, need relatively much human intervention, and information is lost as shapes are generalized by making assumptions, such as that they are planar or smooth [48][47].

The drawbacks are that a point cloud consists of more data, as many individual points are used rather than fewer points that are connected edges to form boundaries. They are thus more expensive to store and load. As for the visualization, there are empty spaces between the points. Additionally, current algorithms are not always suitable to use on a point cloud, and standards and existing software are also mostly developed for 3D surfaces rather than points. These need to be redeveloped, which is a reason for this project to look into this specific analysis on object detection. Lastly, the density in a point cloud varies because of variations in the geometry and surface of objects, and the sampling process with objects farther from the scanner returning less points due to diverting laser beams [48][47]. This not only causes issues for the visualization, but also during analysis as has been noted during this project.

Some examples of applications in which detected windows are useful but specifically in combination with direct point cloud analysis, are the inclusion of visible indoor points through a window from specific perspectives (viewshed analysis), assessing with multi-temporal point clouds if parts of a building have been damaged (change detection, such as in [49]), and the automatic blurring or censoring of windows for privacy reasons (not only in the point cloud itself, but also on corresponding 3D imagery) [48].

5.2 Energy usage estimations

As has been introduced in the first chapter of this report, the windows of buildings are a large source of heat transfer, which thus can either be gain or loss. The detection of windows can thus contribute in the estimation of the energy usage of a building, since temperature control in buildings is a large consumer of energy, be it air conditioning or central heating. The size of the windows of a building, their locations and the ratio of openings (windows and doors) to the complete facade surface all have a share in the heat balance of the building [2][3]. In this subsection, a deeper look is taken into the influence of windows on the energy usage of a building, and how knowing the dimensions of windows helps in making a heat transfer estimation.

Firstly, to understand why windows are important for this cause, the three ways in which energy can transfer through windows - temperature driven heat transfer, solar heat gain, and infiltration - are explained, with information from [50] and [51]. These processes result in heat loss or heat gain, which both can be an advantage or disadvantage, depending on the desired temperature in a building.

Temperature driven heat transfer is the energy transfer that happens when the outside temperature differs from the inside temperature. This can go both ways, so heat can also come in through a window in this way. It happens through the window frame and the window's glazing. An important factor for this is the size of a window. Within this type of heat transfer, there are three processes that play a role. Conduction, which means heat that is transferred through solids, convection, which indicates energy transfer through the movements of gases or liquids, and radiative heat transfer, which is energy transfer that is done through space, but not using conduction through or movement of the air. With these three processes in mind, the U-factor can be calculated to measure the insulation quality of a window, that indicates the total heat transfer “through the fenestration product”. Environmental properties are taken into account as well in the U-factor. The smaller it is, the less heat transfer would take place [50].

Solar heat gain is the most significant factor in the energy transfer, which is caused by solar radiation being transmitted through a window. This can both directly originate from sun beams, or indirectly by it being reflected from other surfaces. These beams also include near-infrared, far-infrared and ultraviolet electromagnetic radiation, which are thus not visible. The glazing of a window determines which wavelengths to what extent are able to pass through, reflect back or absorbed by the material. Knowing this property of a window, and again the environmental properties at a moment, you can calculate how much heat is transferred in this way. But for this kind of calculation, the size of a window is also of importance, since a larger surface area means more interaction with solar beams. This effect can also be advantageous in the winter, when buildings are heated up [50].

Lastly, heat can be gained or lost by infiltration. This regards the travel of air that can occur through small holes around the window. The extent to which this happens, again depends on environmental circumstances such as the wind outside and the temperature difference, impacting the air pressure. Infiltration can be reduced by double glazing, which also improves the insulation of a window [50].

Ultimately, the impact of these three energy transfer processes for windows depend much on the type of glass/glazing and the window frame that is used. Environmental circumstances are important as well, and these are not constant. While the size of a window is also of significance in energy usage estimations, there are many more parameters that need to be known in order to do accurate calculations. Even this explanation regarding heat transfer through and around windows leaves out many complexities.

Then, there are other factors too such as walls and indoor heat sources [51]. The Dutch government uses the NTA 8800 energy performance calculation specifications [52] to do such estimations, which is explained in over 900 pages. This indicates how complex the topic is. All these parameters can not be determined from such a point cloud as that has been used for this project, mainly because laser beams do not reflect on windows.

However, being able to automatically determine the size of windows for buildings is a useful step for this application, as is specifically noted in [3]. When this can be combined with the locations of windows and the (general) types of windows of a building, it is already possible to do estimations.

5.3 Window detection used in tunneling construction for risk assessment

As mentioned in [53] the risk assessment of ground subsidence is one of the major possible threats when creating a tunnelling network underground. In order to evaluate it, we need information about the building environments they cover. Building models is one way to make this process more automated but very often they are very simplified. Detected windows can be used to make these models more detailed. With direct analysis on point clouds the detection of windows can be done with less computational expense. Despite regional and architectural variation in the facade's and window's appearance [53], the provided information would still be enough to estimate the possible risk.

Underground construction elements can cause damage in above ground structures. For this not only the tunneling hypothetical routes are enough. The goal is to find which route is the optimal to minimize risk. But this has to be done in large scale, meaning that since the type of construction networks might be big and can cover whole districts, we need information that enables the analysis in terms of accuracy, time and sources efficiency [53].

Windows affect the risk assessment as they are related to the stiffness of a building construction. Building structures are made of various materials, some of them are more fragile (e.g. glass) and others less (e.g. concrete) but in any case they are directly related to the structural robustness of the building. There is a relation between the settlement and the structure of the building at each location, which can result in some cases for example in ground depressions. Therefore it is crucial to calculate the buildings opening ratios. For this purpose a window detection algorithm can be implemented for settlement prediction as a preprocessing step and subsequently a damage risk evaluation [53].

5.4 Window detection for the simulation of high-rise building fires

Building fires especially in urban environments can cause huge disasters, as described thoroughly in the research paper of [54]. The way the building areas are build and designed, having high density and population, are increasing the fire vulnerability of it and consequently the appearing damaging effects. Windows are directly related to the way the fire propagates. Flames and hot gases can be ejected through openings openings and flow along building walls accelerating this way the fire towards upper floors [54]. The window glasses play a big role in this as many buildings don't use thick fire-rated glazing. Instead they shatter amongst other reasons due to explosions or the absence of glass temperature uniformity causing a "thermal shock" making different parts of it expand in different amounts and eventually crack. In the case of high- rise buildings especially the situation can go easily out of hands as it would be severely difficult for firefight teams to control it.

Window configurations, in other words its shape and dimension, were proven in [55] to be of crucial importance in the spread of fire. A specific model was developed that could describe the geometry (height) of the flames and the amount of heat that flows to upper floors through the exterior wall. In this implementation of fire simulation, the results of several practical tests with various types of windows of different width/height and shape, showed that the route the fire is following in wide windows does not reach far away for the facade but climbs instead to higher heights in the wall, something that does not happen in the case of narrow windows. Moreover, smaller windows create a higher heat and smoke flux and seem to cause oppressive heat temperatures at the wall. Finally, they concluded that heat emission exposure to the exterior wall is high for wide windows, but lower for square windows and less for tall windows[55].

5.5 Indoor applications

Algorithms to extract windows can be useful for indoor point clouds as well, which are utilized to create BIMs. These models can be used for architectural purposes and indoor navigation. Currently, it is still a slow and expensive procedure, while also demanding human intervention to retrieve good results. Better automated object extraction methods can thus aid in improving this process. However, there is yet no uniform method to detect windows in all three different situations - outdoor, roof and indoor datasets (both sources). A reason for this, is that indoor point clouds have different properties from exterior and top view ones, which are used in this report [56] [57].

For example, holes in a wall perceived from indoors can be the result of indoor objects that block the laser beams of the scanner. While exterior point clouds also can have problems with occlusion caused by for example trees, it will there rather create holes of such a size that it could not realistically be a window. Indoors, it is more likely that occluded spots have a size similar to regular windows. Because of this, it is challenging to create one window detection method that works well in both perspectives [56] [57]. The used window detection methods in this report focus on the detection on outdoor point clouds, and would have to be altered in order to yield results of similar quality, if possible at all. Additionally, the tested methods can be combined with indoor ones, both to increase each others' quality and as a step in combing outdoor information with indoor (GeoBIM).

6 Conclusions

As technology grows every day, point clouds are easily and quickly derived and provided to end users. However, in terms of applicability and usability, it is of most significant interest not only to acquire point clouds but also to find ways to enrich point clouds with attributes that can be useful to end-users for particular domain applications.

Our research ties in with the above mentioned goal of enrichment of point clouds, as it adds to each point of a dataset the property of being part of a window or not. However there are several issues that need to be addressed for the applicability of this algorithm and the requirements related to the input datasets. Through a benchmarking process it was possible to answer to the subquestions of the problem definition of this research.

Evaluation of acquired results showed, that both of the method are promising when it comes to mentioned application. However, a lot of improvements and robustness need to be added. Without that it is impossible to perform it on automated way – the algorithms required setting the parameters by the user, sometimes even by trial and error, with conclusion about the best setup based on examination of the results. Obviously this way of providing solutions is not suitable for points cloud processing in large extend. The main conclusion of the research was done and further examination of the method sounds to be promising way to develop efficient way for windows detection, without expensive geometry reconstruction.

6.1 Capabilities and Limitations

In conclusion the provided end product is able to detect windows in limited extend. Major problem leading to biggest differences in comparison to benchmark data, was related to occlusions and hard covers behind window glass. However, those problems are related to quality of the point cloud and the algorithms have no possibility to enrich it. The second problem identified during assessment was related to the low point cloud density. The assumption is, that changing the way of performing measurement can solve this issue; in measurement recorded from longer distance (for example, from airborne LIDAR) the difference in point density will be significantly lower; thus, it should be easier to find parameters which are suitable for the top and the bottom of the facade on the same manner. Besides that, the other conclusion from the assessment is changing the focus of searching for improvement - the best results were acquired on facade, that was prepared manually to fulfill tough requirements. The conclusion is, that the algorithm is able to provide accurate solution, but it is sensitive to occlusions and artefacts. Focusing on data filtering can also provide noticeable improvement for the whole application.

To answer concisely to the subquestions of our problem definition, the overall accuracy is highly dependent on number of occlusions and other artefacts. For clear facade it was possible to reach more than 90% of correctly detected windows, however on most noisy facades the number was dropping to 50%. Also, the actual percentage of detected windows is dependent on number of covered windows which are physically covered from the inside and it is not possible to determine the mean value of this number for general assessment. In terms of processing time during the tests on individual facades the resulting duration was reasonable. Moreover, considering always the above mentioned factors it is possible to combine the result this analysis with other datasets such as the BAG for extracting the ratio between window - building surface areas per BAG ID. However, the quality of the results depends highly on the quality of the input dataset.

There are three major groups of problems that have to be faced during the whole process of window detection project. First, the limited time frame – the complexity of the task is too high to test all of the interesting methods and elaborate about quality of them. The purpose of the project from the perspective of stakeholders from Municipality of Rotterdam is to derive the best possible solution for windows detection, and from that perspective it is crucial to test various approaches to conclude about differences between them. Unfortunately, the other algorithms were moved to optional part of the project with note, that it should be taken into consideration if the project will proceed without noticeable delays, to avoid cross the specified deadline.

The second group, is the most common problem for most of the spatial analysis – inconsistent quality of the data. The LIDAR technology is not robust to all kind of shapes and surfaces it is scanning. The appearance of unwanted artefacts has to be taken into consideration to avoid misclassification of outliers as the points of the interest.

The third group are "artificial" limitations, which are not deriving directly as natural corollary of chosen approaches and equipment, but are set up by the users to make the project more challenging and focus on finding the specific kind of the solutions. The two major ones are the limited computational power, to avoid time-consuming and computationally ineffective solutions and taken approach, to provide the computation directly on point cloud (if possible).

More analytically:

- Our method cannot be fully automated, because additional information about the building facades is required. Building materials, shape and size of facades, architectural style, etc. play a major role in our window detection algorithm. For example, a building which has a wide facade will require higher amount of vertical slices than a narrow facade.

- Our methodology is based on the assumption that a window in the point cloud will be recognized as missing information, because the laser beam emitted from the laser scanner penetrated the glass surface and didn't create a return. Objects that are located on that surface, like curtains, posters, furniture, ornamental, etc. affect the final point cloud output of the laser scanning measurement and windows cannot be identified.

- The scanner's location affects both the point cloud's density and completeness. The bigger the distance between the scanner and an object, the more sparse the point cloud becomes. As a result, the point cloud is sparser for higher elevations, and in extent, in higher floors of a building. Also, due to the fact that a mobile scanner is usually placed on top of a car, a lot of obstacles prevent the return of laser beams back to the scanner, creating empty regions in the point cloud or regions with not enough information to describe the target object. Missing information or less-dense point cloud in one region means that our algorithm might consider it as a gap of information and therefore, falsely identify a window.

- The computation time of our methodology is not optimal, due to the fact that the methodology was divided into multiple tasks and implemented with different programming languages and packages. The used programming languages are C++ and Python, but it would be quicker to be able to use one streamlined workflow in C++. In addition to that, it has not always been attempted to optimise the speed of algorithms. For it to be used for an application, it is desirable to do as much optimisation as possible, especially so that it is also feasible to use on larger datasets.

6.2 Recommendations - Future Work

Especially since direct point cloud analysis is a relatively new research area and the scope of this project has been limited, multiple recommendations and ideas for future work can be noted. Firstly, the quality of the used dataset is of importance for the results, and possible improvements on that regard will be discussed. Lastly, the application side of the topic will be considered.

By performing laser scanning measurements on regions where data is missing, it will become more complete. Without enough data, no proper additional information can be derived from a point cloud. In such a case, it can only be predicted according to statistical reasoning (e.g. patterns), which the tested methods in this report do not take into account. Atlernatively to improving the dataset quality, the methods can be adapted to retrieve better results for occluded spots.

Adding onto the topic of dataset quality, its density could be increased by merging different types of point clouds, like airborne laser scanning measurements (with UAVs or airplanes), point clouds created by Structure from Motion techniques, or even terrestrial laser scanning measurements. Specifically, this can prevent problems that arise when dealing with varying point densities within objects. What has been noted as well in the mobile laser scanner dataset, is that parts of rooftops can show up as artefacts. Windows can be present on here as well, but these parts are currently removed as outliers during the performance of the methodology.

Moreover, filtering of the dataset could be improved by using more sophisticated methods. Potentially, the computation time of the filtering can be made better, and a cleaner input for the window detection algorithms can be derived. For example, inside points that have been registered by the scanner through a window can be identified properly. These can cause problems during the gap detection, as they can be seen as part of a plane during RANSAC segmentation when they lie too close to the actual window, which is often the case when curtains are closed.

Moving on to the application part, knowing the dimensions of the windows of a building and the ratio between windows and full facade surfaces are not sufficient to make reliable energy usage estimations of a building. While window detection definitely benefits this application, it should be combined with other data and techniques. GeoBIM could be suitable in this regard, as it could contain the type of glazing of a window for example. Weather simulations are another addition that is necessary to improve the estimations. Future work can attempt to combine these aspects.

Also, the two algorithms have only been tested on the exterior of buildings, but from a BIM perspective, methods need to be created that are suitable for indoor point clouds [57]. It could be tested whether these methods work with such a dataset, which alterations should be done, or if they are not useful at all for indoor applications.

7 Project Plan

The purpose of the project plan is to strategically define the process of the implementation of the project and all of its different components. The plan of the project is created in its infancy containing info about schedule baselines, project requirements and communication of stakeholders. It is a crucial part of the project as a whole, as the absence of is often connected to a bad outcome in the execution of the project such as mediocre results or even failure.

7.1 Meetings

Weekly team meetings are on Monday, Wednesday and Friday. During these meetings the daily tasks were discussed and the team got informed on the progress made by other team members. The stand-up meeting on Wednesday was usually attended by mentors Edward Verbree and Martijn Meijers.

7.2 Organizational and Work Breakdown Structure

Successful project management requires a highly organized planning. This includes a very detailed specification of the tasks, the related responsibilities and the timeline of the process. The Organizational Breakdown Structure (OBS) is a model that is used to describe the responsibilities assigned to each task according to the created Work Breakdown Structure (WBS). WBS aims to describe the organizational framework by subdividing the project in smaller parts. These two models combined, will give a clear indication of the different and also the corresponding contribution of each team member.

The team:
Liyao Zhang (Technical Manager), Maria Moscholaki (Report Manager), Konrad Jarocki (Quality Manager), Pantelis Kaniouras (Coordinator) and Jordi van Liempt (Communicator).

- Coordinator: Coordinates resources and information, defines meetings, monitors the progress.

- Communicator: Is responsible for the communication between the team, client and the supervisors, sets meetings.

- Quality Manager: Evaluates the overall project quality, defines accuracy/quality results of implemented project method.

- Report Manager: Is responsible for the organization and evaluation of report and document requirements.

- Technical Manager: Is responsible for technical issues.

In order to make certain that every part of the project would be of good quality, responsibilities were divided. This would ensure that every section of the implementation would be enoughly checked and evaluated. Of course since the main goal of this project was that all team members would work equally troughout the project while learning as much as possible, every person of the team got involved with all work packages. However, for each work package the person first mentioned had the main responsibility.

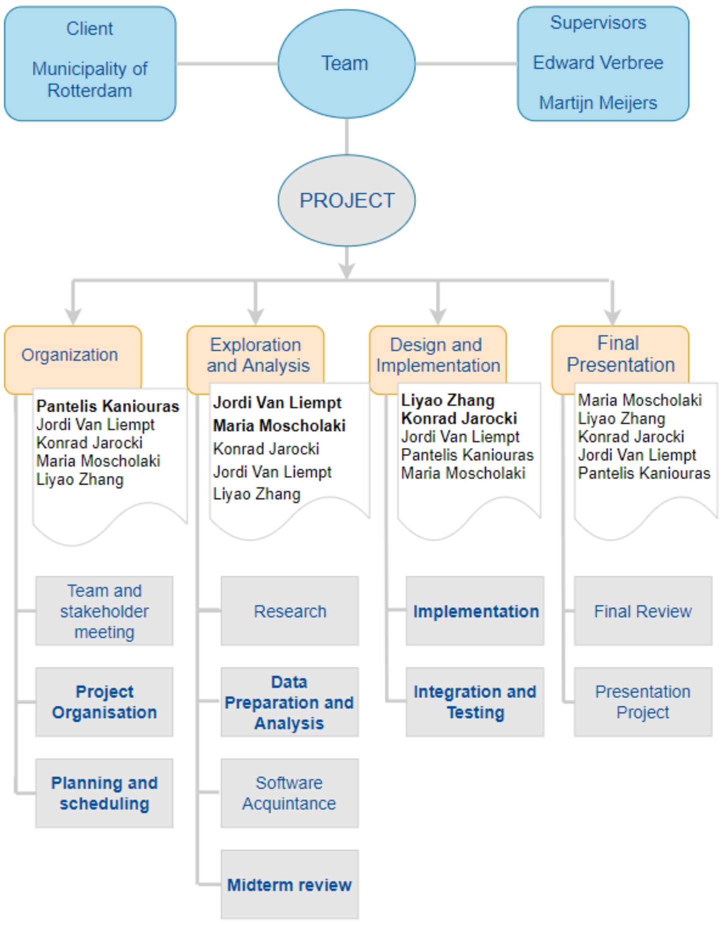


Figure 45: Organizational and Work Breakdown structure

7.3 Work Plan

The Work Plan is used to organize big projects. It represents all work phases and the included work packages together with the time -related progress of each of the project parts. The deliverables of the report were organized in 3 parts. They are namely: the Inception Report, the Midterm/Technical Review and the Final Technical Review. The purpose of these were to inform the supervisors and client about the progress of the project while giving the chance for some fruitful feedback. The dates of deliverables (reports and presentations) are shown in the Work plan located in the Appendix C.

Below is a brief description of each of the tasks presented in the **Work Packages**.

Organization:

- Team and stakeholder meeting as held after the kick off meeting of the Synthesis 2019.

- The Municipality of Rotterdam explained the goal of the project and their expectations.
- Another meeting took place with the group supervisor, Edward Verbree where a brief discussion about the management and the technical details of the project was held.
- Discussion among the team members about a rough organization of the entire project, from its start until the finish date.
- A few specific responsibilities were assigned to the different team members for the start of the implementation of the project.

Exploration and Analysis:

- During the research several external sources like papers and information gained through the communication with relevant professors and professionals of the field, were taken into consideration and studied thoroughly.
- Data preparation by filtering the provided datasets for the removal non building points like ground points, vegetation, cars etc.
- Exploration and familiarization of the possible software, programming languages and tools (libraries etc) for the implementation of the algorithm.
- A Midterm presentation of the project to inform about the implemented steps and the general course of the project so far.

Design and Development:

- Design and Development of a specific methodology for the project.
- Implementation of the steps of the methodology by using more than one datasets, software and programming languages.
- Integration and Testing of the algorithm by evaluating the results for different datasets or different chosen parameters in the implemented code.

Final Presentation:

- Creation of the the final report.
- Presentation of the Project at the Geomatics Day Symposium.

7.4 Rich Picture

In order to get an insight on the goals, the important aspects and possible risks and concerns of the project, a rich picture was created. A rich picture is visual means to describe the main elements of a project, the related stakeholders and the relationships between them. It consists of a triangle formed by our team and the project coaches and the client (Municipality of Rotterdam).

The project is the core of a triangle in the peaks of which, the stakeholders are placed. Each group has different motivation, interests and needs with respect to the current project. Some of these are presented for each interest group with a green frame. The results of the team's research will assist in the decision making process the municipality officials for different applications of the implemented algorithm. As in every situation that involves many stakeholders, questions or concerns about what is perceived as risks are also present. The rich pictures indirectly imply that, every decision and action within this "stakeholder system" will have some sort of impact on the existing relationships and individual initiatives and motivations.

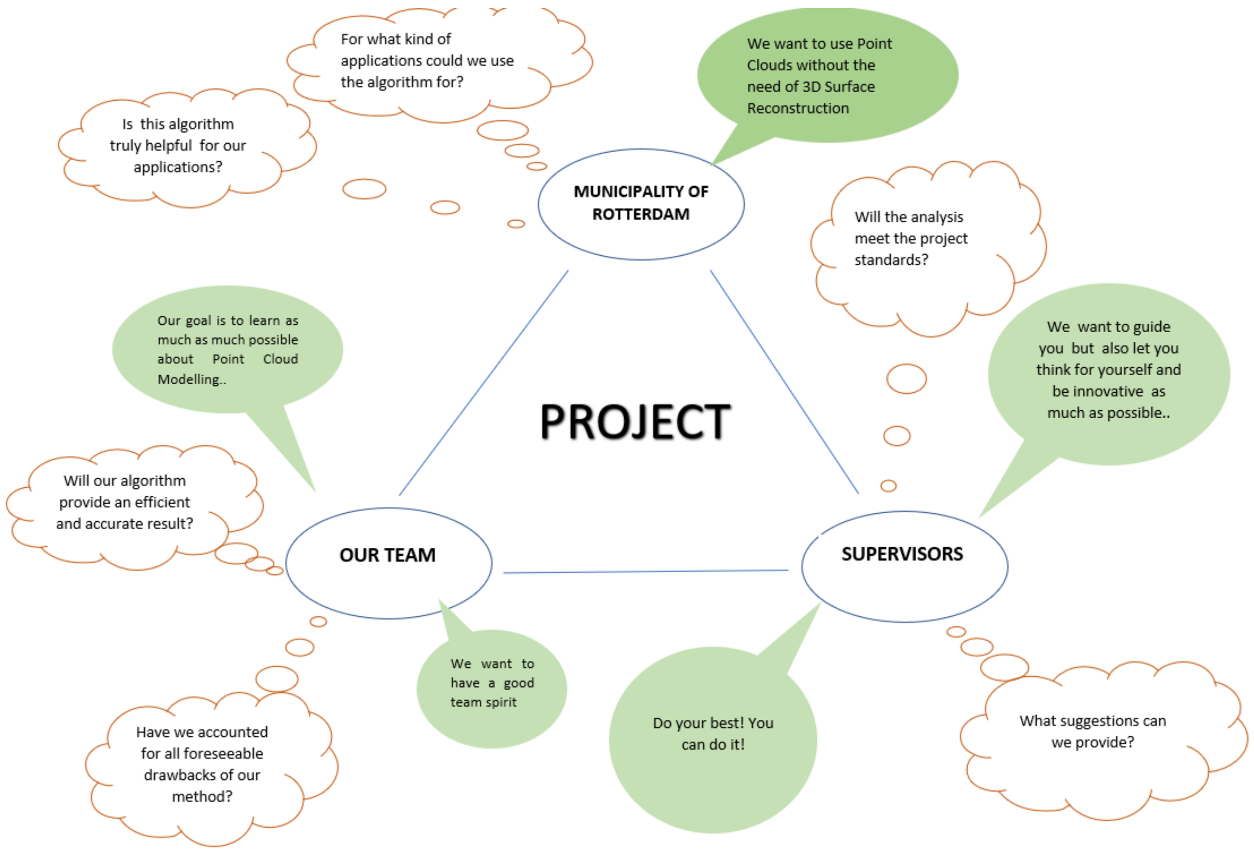

For what kind of applications could we use the algorithm for?
Is this algorithm truly helpful for our applications?
We want to use Point Clouds without the need of 3D Surface Reconstruction
MUNICIPALITY OF ROTTERDAM
Will the analysis meet the project standards?
Our goal is to learn as much as much possible about Point Cloud Modelling..
PROJECT
We want to guide you but also let you think for yourself and be innovative as much as possible..
Will our algorithm provide an efficient and accurate result?
OUR TEAM
SUPERVISORS
We want to have a good team spirit
Have we accounted for all foreseeable drawbacks of our method?
Do your best! You can do it!
What suggestions can we provide?

Figure 46: Rich Picture

References

[1] Agis Mesolongitis and Ioannis Stamos. Detection of windows in point clouds of urban scenes. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 17–24. IEEE, 2012.

[2] Chao Wang, Yong K Cho, and Changwan Kim. Automatic bim component extraction from point clouds of existing buildings for sustainability applications. *Automation in Construction*, 56:1–13, 2015.

[3] Chao Wang and Yong Cho. Automated 3d building envelope recognition from point clouds for energy analysis. In *Construction Research Congress*, pages 1155–1164, 2012.

[4] National building energy performance strategy, 2015.

[5] Moscow method, 2019.

[6] George Vosselman and Hans-Gerd Maas. *Airborne and terrestrial laser scanning*. Whittles Publishing, 2013.

[7] Ken Arroyo Ohori Ravi Peters, Hugo Ledoux. Acquisition and artefacts.

[8] Low cost solid-state 2d lidar.

[9] Martin Kodde. Dense image matching, 2016.

[10] Zhang Xiaoyi and Zhenghong Du. Spatial index. *Geographic Information Science & Technology Body of Knowledge*, 2017(Q4), October 2017.

[11] Ken Arroyo Ohori Hugo Ledoux, Ravi Peters. Handling massive terrains.

[12] R-tree, wikipedia, 2019.

[13] Anh Nguyen and Bac Le. 3d point cloud segmentation: A survey. In *2013 6th IEEE conference on robotics, automation and mechatronics (RAM)*, pages 225–230. IEEE, 2013.

[14] H Cantzler. Random sample consensus (ransac). *Institute for Perception, Action and Behaviour, Division of Informatics, University of Edinburgh*, 1981.

[15] Jana Kosecka. Model fitting, ransac.

[16] Lin Li, Fan Yang, Haihong Zhu, Dalin Li, You Li, and Lei Tang. An improved ransac for 3d point cloud plane segmentation based on normal distribution transformation cells. *Remote Sensing*, 9(5):433, 2017.

[17] Theo Tijssen, Martijn Meijers, Peter van Oosterom, and Marian de Vries. Web map/feature services.

[18] Johannes Kilian, Norbert Haala, Markus Englich, et al. Capture and evaluation of airborne laser scanner data. *International Archives of Photogrammetry and Remote Sensing*, 31:383–388, 1996.

[19] James Broesch. Median filter, 2009.

[20] Information Resources Management Association. *Computer Vision, Concepts, Methodologie and and Applications*. IGI Global, 2018.

[21] PENG Lei. Adaptive median filtering. In *Seminar Report, Machine Vision*, volume 140, 2004.

[22] Li Tan and Jean Jiang. *Fundamentals of analog and digital signal processing*. AuthorHouse, 2007.

[23] Median filter, computer based learning unit.

[24] Auckland university. `https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures/Gaussian%20Filtering_1up.pdf`, 2010.

[25] Spatial filters, gaussian smoothing.

[26] 2d/3d plot of image processing filters. `https://stackoverflow.com/questions/12606048/2d-3d-plot-of-image-processing-filters`.

[27] Geo services en apis. `https://pdok-ngr.readthedocs.io/services.html`.

[28] Wfs reference. `https://docs.geoserver.org/stable/en/user/services/wfs/reference.html`, journal=WFS reference - GeoServer 2.15.x User Manual.

[29] Basisregistratie adressen en gebouwen (bag). .

[30] `https://www.researchgate.net/profile/Frederika_Welle_Donker2/publication/315670696/figure/fig1/AS:476859600248832@1490703596370/Figuur-31-Uitsnede-uit-de-Basisregistratie-Adressen-en-Gebouwen-BAG.png`.

[31] P Guzik. *Camera calibration method and estimation methods in 3-dimentional space*. PhD thesis, 2015.

[32] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22, 2000.

[33] Thomas Luhmann, Stuart Robson, Stephen Kyle, and Jan Boehm. *Close-range photogrammetry and 3D imaging*. Walter de Gruyter, 2013.

[34] Tom MacWright. Gis with python, shapely, and fiona, 2012.

[35] The shapely user manual, 2018.

[36] Owslib. https://geopython.github.io/OWSLib/#wfs.

[37] Opencv wikipedia. https://en.wikipedia.org/wiki/OpenCV, 2019.

[38] Contour approximation method, 2019.

[39] What is pdal?, 2019.

[40] Kelvin Salton do Prado. How dbscan works and why should we use it?

[41] SM Iman Zolanvari and Debra F Laefer. Slicing method for curved façade and window extraction from point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 119:334–346, 2016.

[42] Ahmad Kamal Aijazi, Paul Checchin, and Laurent Trassoudaine. Automatic detection and feature estimation of windows for refining building facades in 3d urban point clouds. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2(3):1, 2014.

[43] Thomas J Pingel, Keith C Clarke, and William A McBride. An improved simple morphological filter for the terrain classification of airborne lidar data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 77:21–30, 2013.

[44] Keqi Zhang, Shu-Ching Chen, D. Whitman, Mei-Ling Shyu, Jianhua Yan, and Chengcui Zhang. A progressive morphological filter for removing nonground measurements from airborne LIDAR data. *IEEE Transactions on Geoscience and Remote Sensing*, 41(4):872–882, April 2003.

[45] S Tuttas and U Stilla. Window detection in sparse point clouds using indoor points. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38:3, 2011.

[46] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[47] Stephan Nebiker, Susanne Bleisch, and Martin Christen. Rich point clouds in virtual globes–a new paradigm in city modeling? *Computers, Environment and Urban Systems*, 34(6):508–517, 2010.

[48] Peter van Oosterom. Web-based 3d viewer for massive point clouds, 2019.

[49] Diogo Duarte, Francesco Nex, Norman Kerle, and George Vosselman. Damage detection on building façades using multi-temporal aerial oblique imagery. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2/W5, 2019.

[50] Regina Bokel. Building physics energy: Glazing properties and their effect on the insulation and heat admittance in a building.

[51] Aart Schuur. Energie en warmteproblemen in gebouwen: Energiegebruik van gebouwen.

[52] Nederlandse Norm. Energy performance of buildings - determination method.

[53] Marcel Neuhausen, Markus Obel, Alexander Martin, Peter Mark, and Markus König. Window detection in facade images for risk assessment in tunneling. *Visualization in Engineering*, 6(1):1, 2018.

[54] KOHYU Satoh and KUNIO Kuwahara. A numerical study of window-to-window propagation in high-rise building fires. *Fire Safety Science*, 3:355–364, 1991.

[55] Mohamed Ibrahim, Abdelsalam Sharaf eldin, Mohamed Fayek Abdrabbo, and Mostafa Ayoub. Effect of window configurations on fire spread in buildings. 07 2013.

[56] Kourosh Khoshelham, Lucía Díaz-Vilariño, Michael Peter, Zhi-Zhong Kang, and Debaditya Acharya. The isprs benchmark on indoor modelling. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W7, 2017.

[57] Rami Assi, Tania Landes, Hélène Mache, and Pierre Grussenmeyer. Energy function algorithm for detection of openings in indoor point clouds. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W13, 2019.

A Appendix - Morphological filter comparison

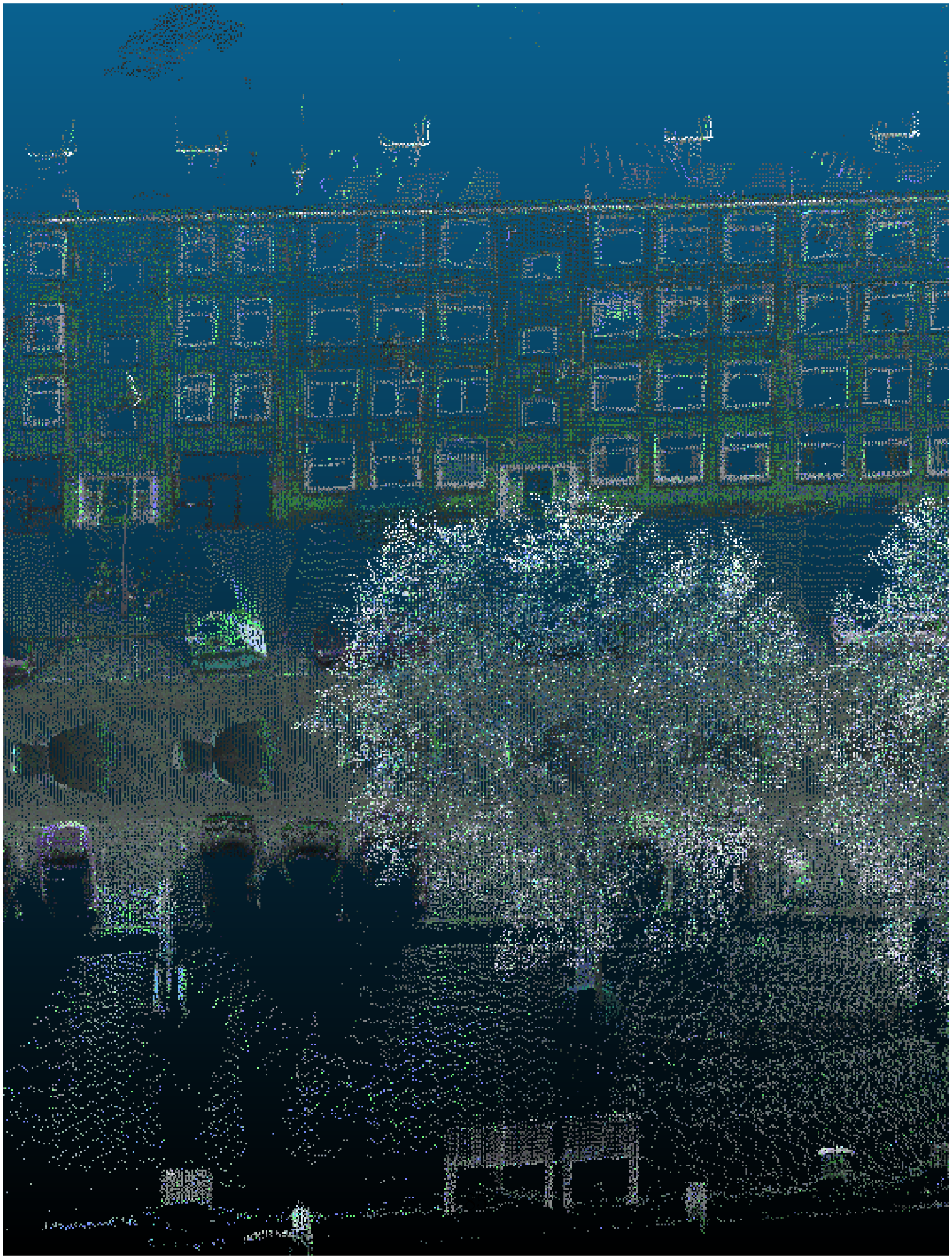


Figure 47: Point cloud subset before ground filtering

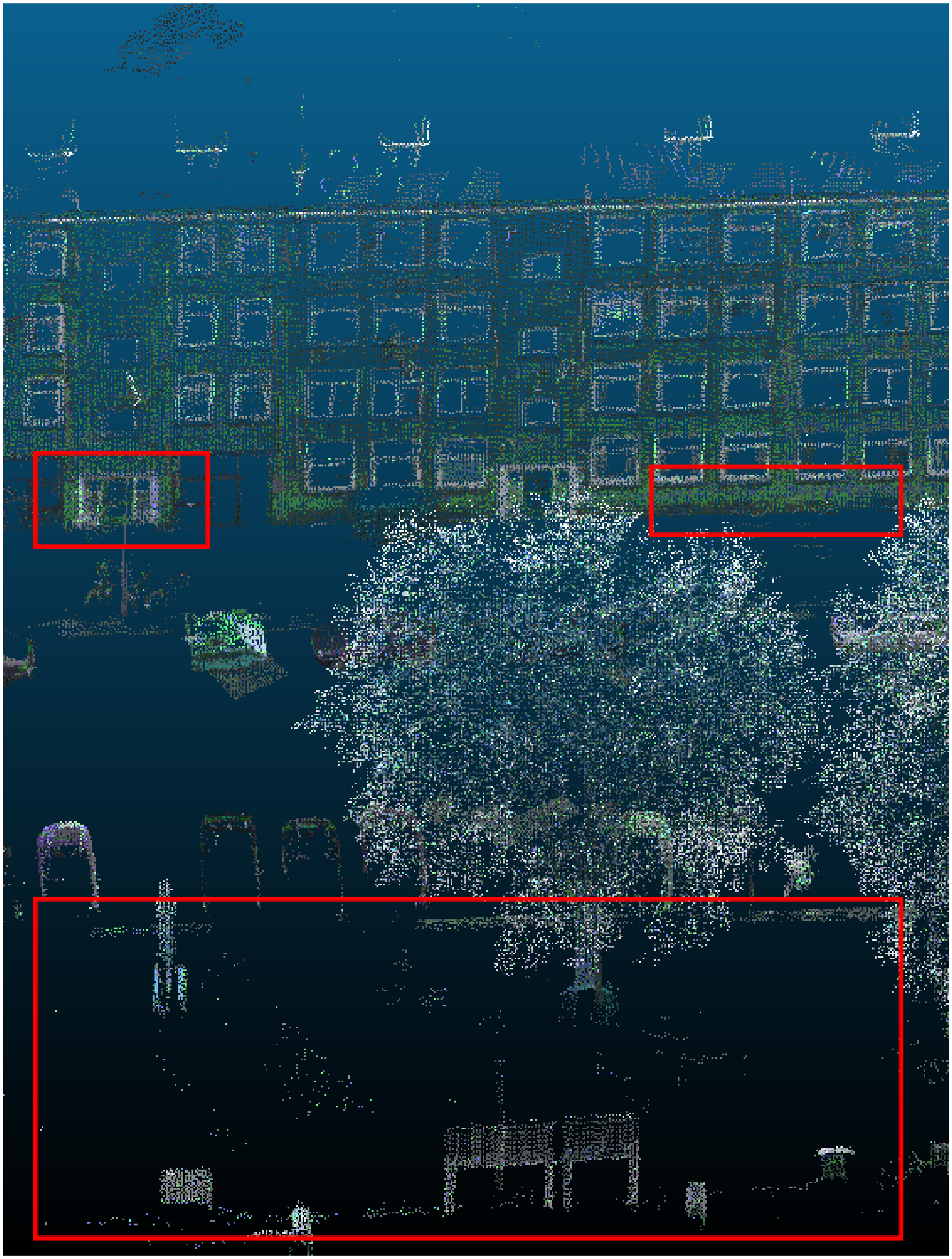


Figure 48: Point cloud subset after PMF filtering, with notable spots marked in the red boxes

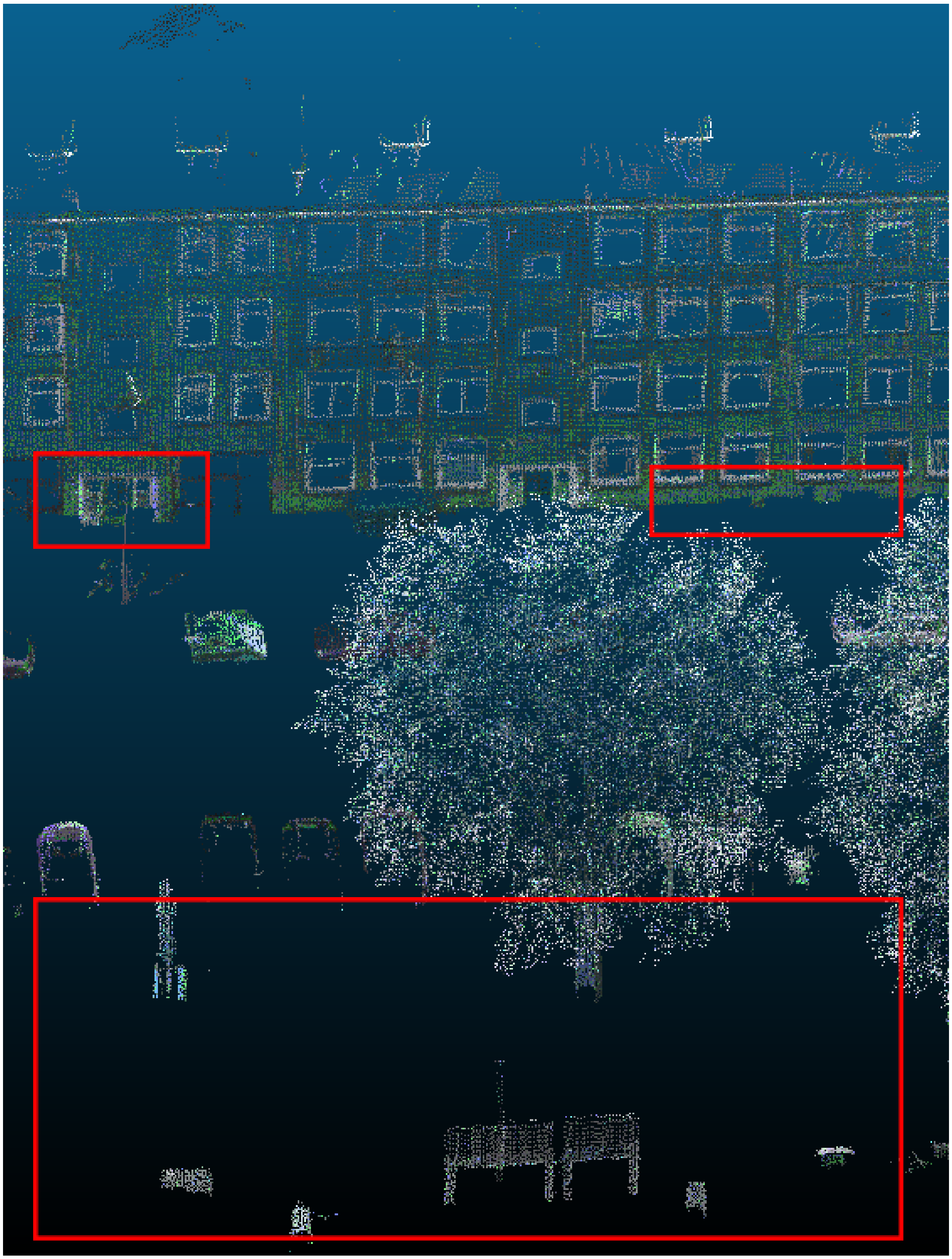


Figure 49: Point cloud subset after SMRF filtering, with notable spots marked in the red boxes

B Appendix - Point cloud filtering quality

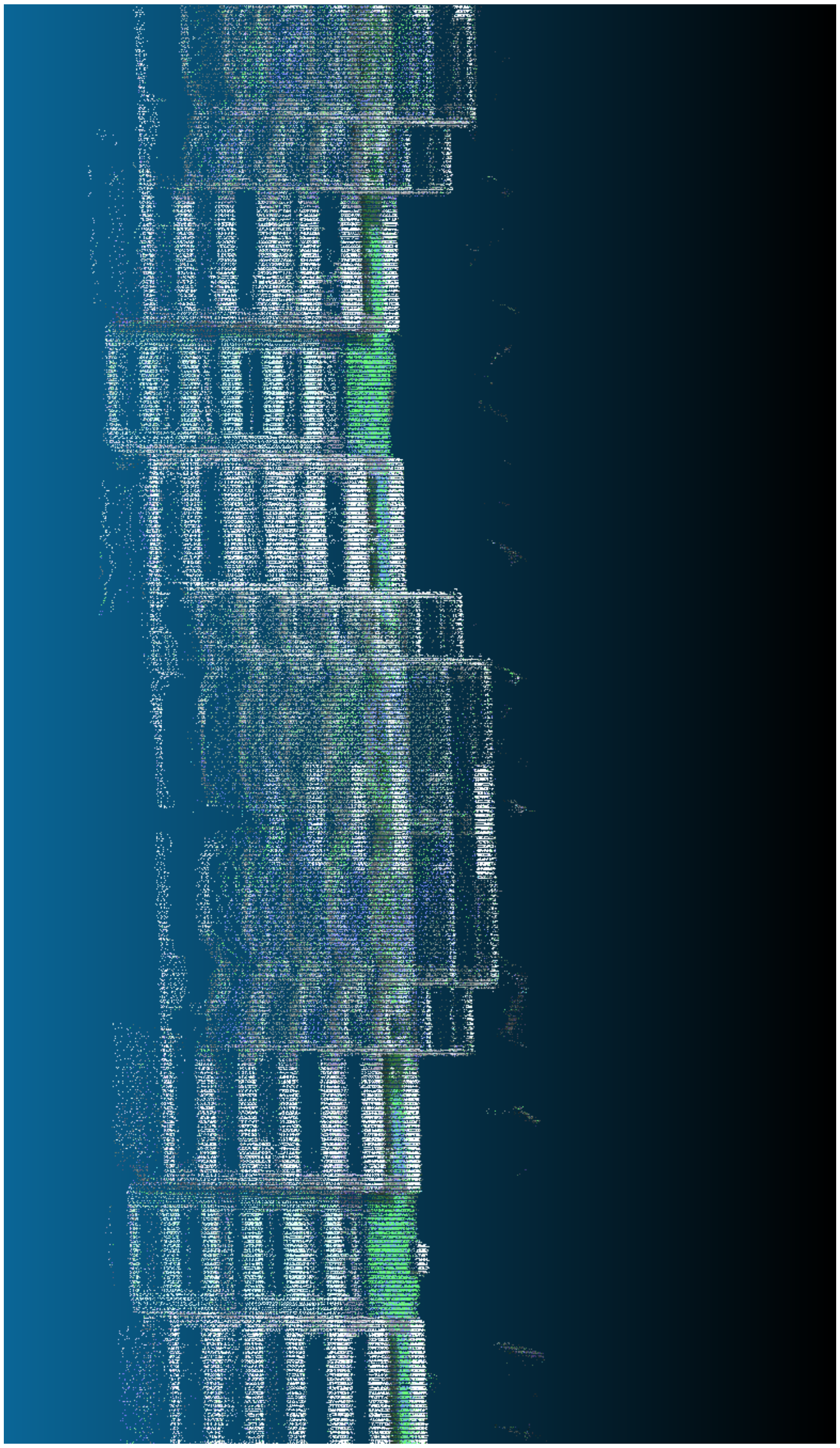


Figure 50: Some points of undefinable objects are present close to these facades

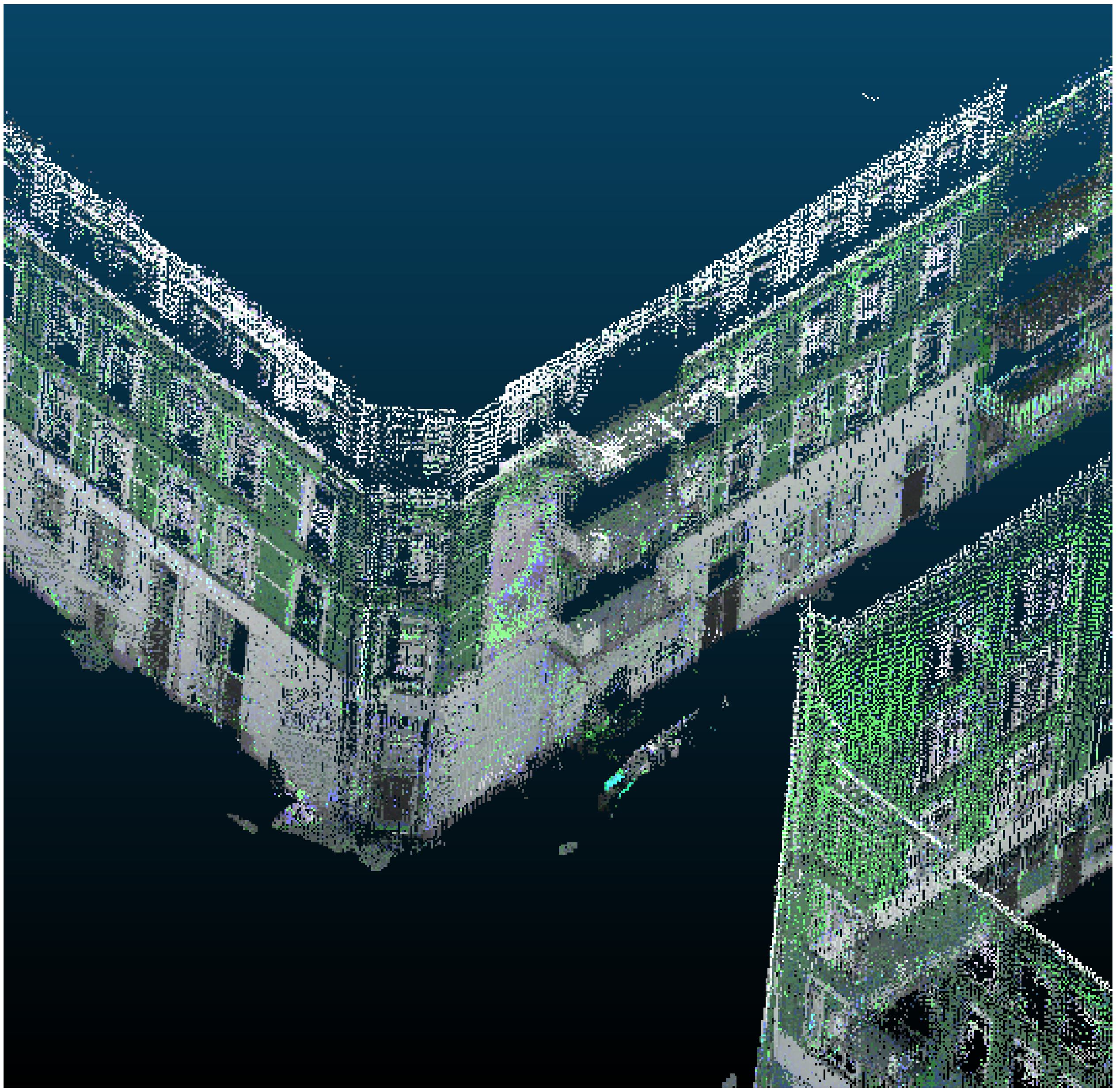


Figure 51: Part of a car that is close to a facade is not entirely removed

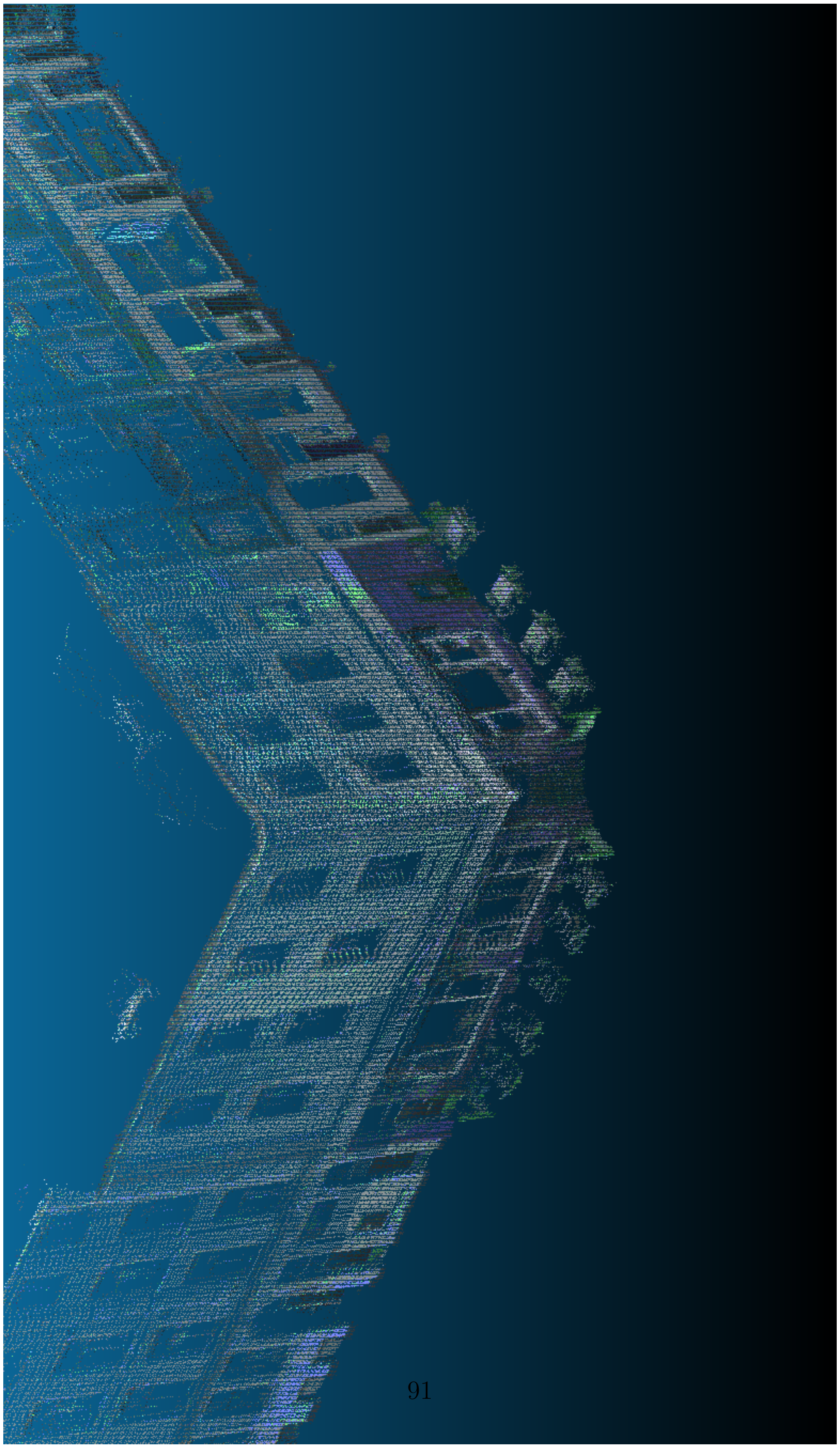


Figure 52: A few objects in front of the corner building are not removed

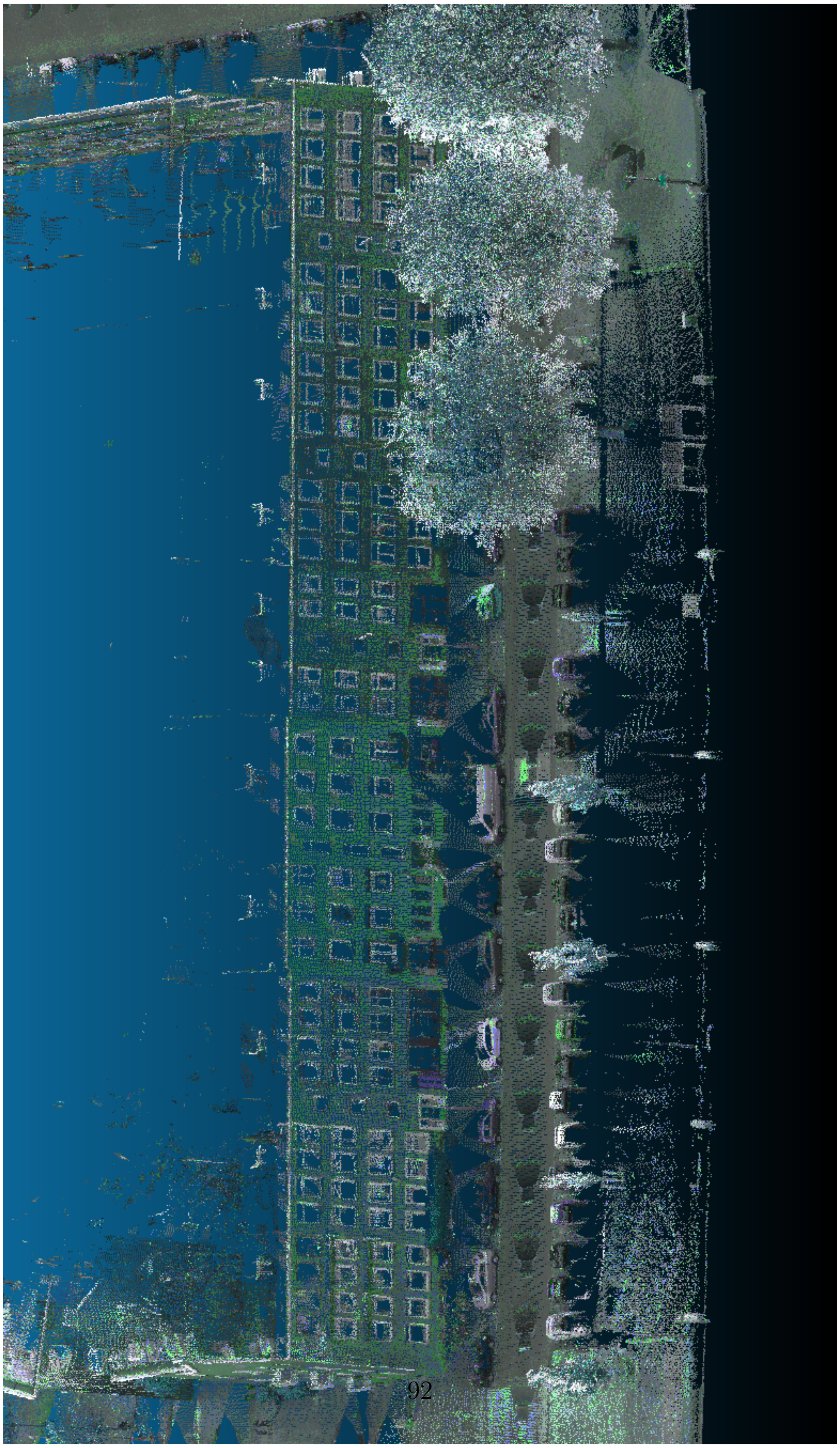


Figure 53: A part of the original cloud before filtering

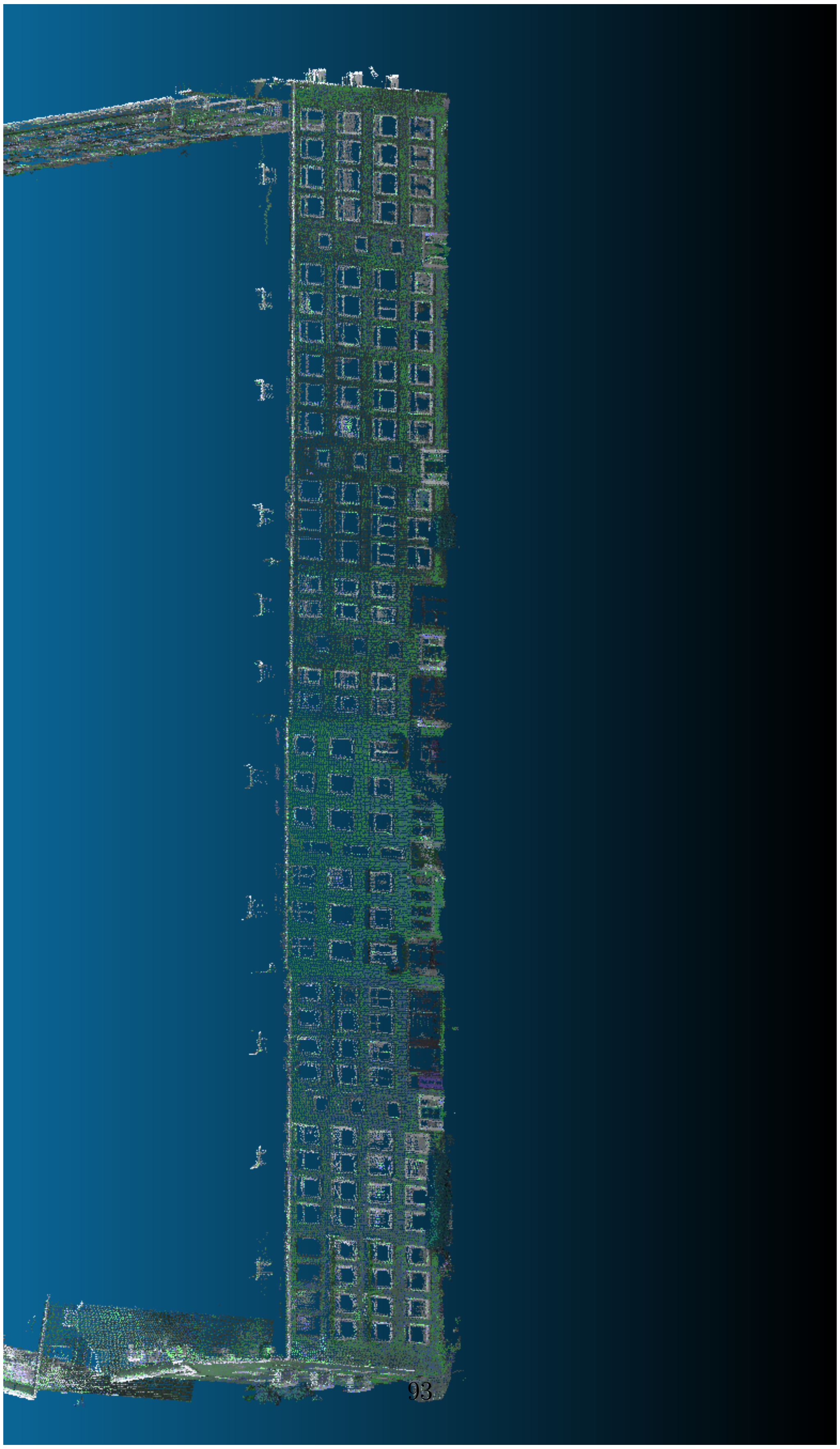


Figure 54: A part of the original cloud after filtering

C Appendix - Project management

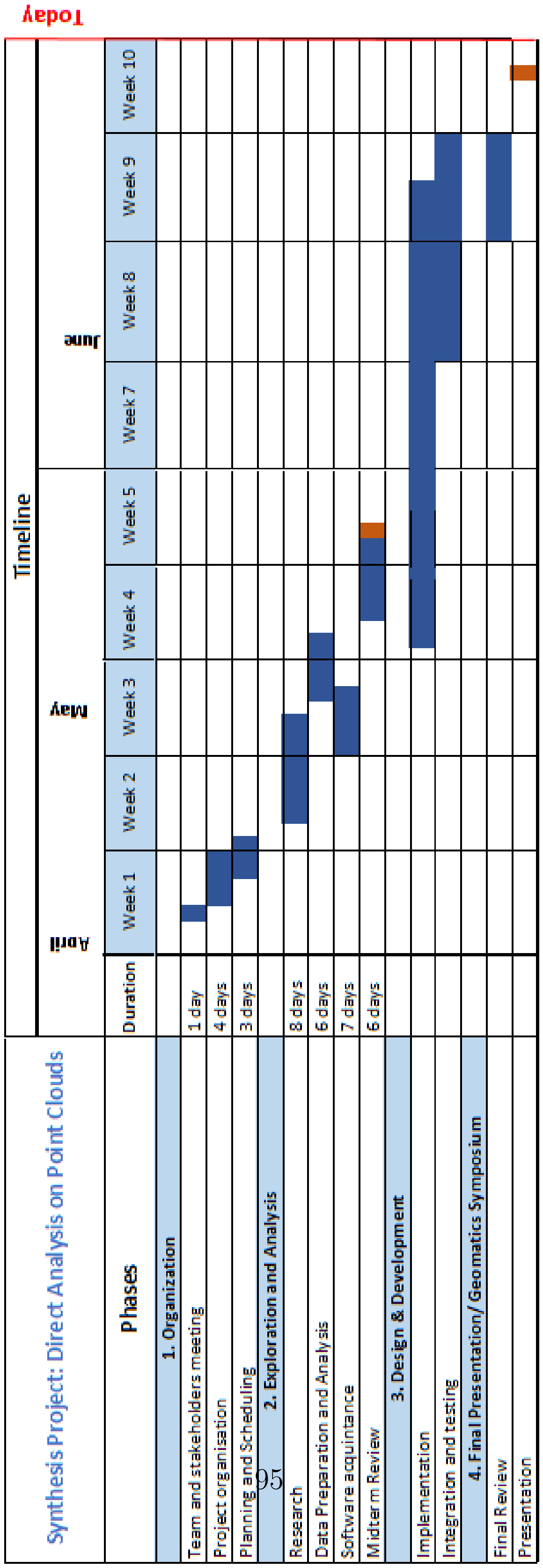


Figure 55: Workplan

D Appendix - Essential parts of code for bitmap method

```python
#Merge clusters in certain distance together
def find_if_close(cnt1,cnt2):
    row1,row2 = cnt1.shape[0],cnt2.shape[0]
    for i in range(row1):
        for j in range(row2):
            dist = np.linalg.norm(cnt1[i]-cnt2[j])
            #distance in pixels
            if abs(dist) < 10 :
                return True
            elif i==row1-1 and j==row2-1:
                return False

#function for finding maximum value in the group
def groupvaluemax(cont_areas, group_numbers, group_number):
    counter2 = 0
    area = 0
    for i in group_numbers:
        if i==group_number:
            if area<cont_areas[counter2]:
                area = cont_areas[counter2]
        counter2+=1
    return area

#function for finding average value in the group
def groupvalue(cont_areas, group_numbers, group_number):
    counter=0
    counter2 = 0
    area = 0
    for i in group_numbers:
        if i==group_number:
                area += cont_areas[counter2]
                counter+=1
        counter2+=1
    return area/counter

#importing the facade
img = cv2.imread('facade6.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
ret,thresh = cv2.threshold(gray,150,255,cv2.THRESH_BINARY)
contours,hier = cv2.findContours(thresh,1,2)
LENGTH = len(contours)
status = np.zeros((LENGTH,1))
```

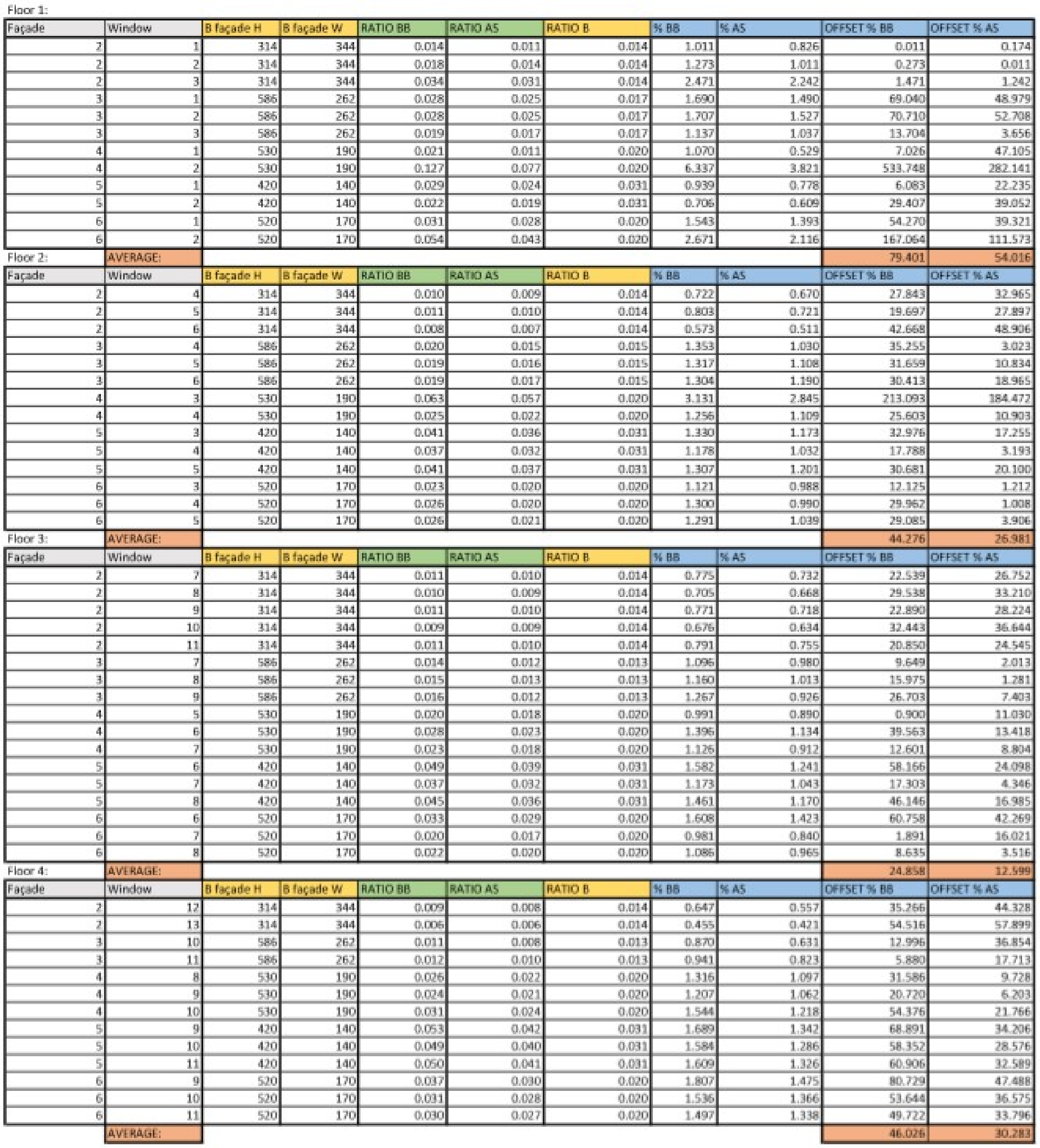
```
group_numbers = np.zeros((LENGTH,1))
cont_areas = np.zeros((LENGTH,1))
cont_w = np.zeros((LENGTH,1))
cont_h = np.zeros((LENGTH,1))
contours_saved = []

#iterating every cluster to find neighbors
for i,cnt1 in enumerate(contours):
    x = i
    if i != LENGTH-1:
        for j,cnt2 in enumerate(contours[i+1:]):
            x = x+1
            dist = find_if_close(cnt1,cnt2)
            if dist == True:
                val = min(status[i],status[x])
                status[x] = status[i] = val
            else:
                if status[x]==status[i]:
                status[x] = i+1
#assignation to the group and calculation of parameters
for i in range(maximum):
    kupa = []
    pos = np.where(status==i)[0]
    if pos.size != 0:
        cont = np.vstack(contours[i] for i in pos)
        x,y,w,h = cv2.boundingRect(cont)
        c_area = w*h
        #condition for max and min size to avoid noise
        if c_area<0.4*area and c_area>0.005*area and w>10 and h>10:
            cont_areas[counter3] = c_area
            cont_h[counter3] = h
            cont_w[counter3] = w
            counter1 = 1
            for group_area in groups_area:
                if group_area >0:
                    #similarities conditioning
                    if c_area>group_area*(1-similarity) and c_area<group_area*(1+
                        similarity):
                        if w>groups_w[counter1]*(1-similarity_d) and w<groups_w[
                            counter1]*(1+similarity_d):
                            if h>groups_h[counter1]*(1-similarity_d) and h<groups_h[
                                counter1]*(1+similarity_d):
                                group_numbers[counter3] = counter1
                    counter1+=1
            #if non group assigned
            if group_numbers[counter3] == 0:
                counter2+=1
                group_numbers[counter3] = counter2
            #calculation of new parameters for the group
            groups_area[int(group_numbers[counter3])] = groupvalue(cont_areas, group_numbers, group_numbers[counter3])
            groups_w[int(group_numbers[counter3])] = groupvalue(cont_w, group_numbers, group_numbers[counter3])
            groups_h[int(group_numbers[counter3])] = groupvalue(cont_h, group_numbers, group_numbers[counter3])
            counter3+=1

            #Estimation position of the window and plotting the results
counter = 1
for i in range(maximum):
    pos = np.where(status==i)[0]
    if pos.size != 0:
        cont = np.vstack(contours[i] for i in pos)
        x,y,w,h = cv2.boundingRect(cont)
        c_area = w*h
        if c_area<0.4*area and c_area>0.003*area and w>10 and h>10:
            group_n = int(group_numbers[counter])
            M = cv2.moments(cont)
            cX = int((M["m10"] / M["m00"]))
            cY = int((M["m01"] / M["m00"]))
            x,y = cX-groups_w[group_n]/2, cY-groups_h[group_n]/2
            h,w = groups_h[group_n], groups_w[group_n]

            print(x,y,cX,cY,h,w,groups_h[group_n]/2)
            print('...')
            counter+=1
            ratio = round(int(w*h)/int(area),4)
            image = cv2.rectangle(img,(x,y),(x+w,y+h),(110,0,50),2)
            cv2.putText(image, 'G:'+str(group_n), (cX, cY), 2,0.5, (0, 120, 255), 1)
cv2.imshow('Windows␣Detection',image)
```

E Appendix - Detailed tables for specific floors



Floor 1:

Façade	Window	B façade H	B façade W	RATIO BB	RATIO AS	RATIO B	% BB	% AS	OFFSET % BB	OFFSET % AS
2	1	314	344	0.014	0.011	0.014	1.011	0.826	0.011	0.174
2	2	314	344	0.018	0.014	0.014	1.273	1.011	0.273	0.011
2	3	314	344	0.034	0.031	0.014	2.471	2.242	1.471	1.242
3	1	586	262	0.028	0.025	0.017	1.690	1.490	69.040	48.979
3	2	586	262	0.028	0.025	0.017	1.707	1.527	70.710	52.708
3	3	586	262	0.019	0.017	0.017	1.137	1.037	13.704	3.656
4	1	530	190	0.021	0.011	0.020	1.070	0.529	7.026	47.105
4	2	530	190	0.127	0.077	0.020	6.337	3.821	533.748	282.141
5	1	420	140	0.029	0.024	0.031	0.939	0.778	6.083	22.235
5	2	420	140	0.022	0.019	0.031	0.706	0.609	29.407	39.052
6	1	520	170	0.031	0.028	0.020	1.543	1.393	54.270	39.321
6	2	520	170	0.054	0.043	0.020	2.671	2.116	167.064	111.573
	AVERAGE:								79.401	54.016

Floor 2:

Façade	Window	B façade H	B façade W	RATIO BB	RATIO AS	RATIO B	% BB	% AS	OFFSET % BB	OFFSET % AS
2	4	314	344	0.010	0.009	0.014	0.722	0.670	27.843	32.965
2	5	314	344	0.011	0.010	0.014	0.803	0.721	19.697	27.897
2	6	314	344	0.008	0.007	0.014	0.573	0.511	42.668	48.906
3	4	586	262	0.020	0.015	0.015	1.353	1.030	35.255	3.023
3	5	586	262	0.019	0.016	0.015	1.317	1.108	31.659	10.834
3	6	586	262	0.019	0.017	0.015	1.304	1.190	30.413	18.965
4	3	530	190	0.063	0.057	0.020	3.131	2.845	213.093	184.472
4	4	530	190	0.025	0.022	0.020	1.256	1.109	25.603	10.903
5	3	420	140	0.041	0.036	0.031	1.330	1.173	32.976	17.255
5	4	420	140	0.037	0.032	0.031	1.178	1.032	17.788	3.193
5	5	420	140	0.041	0.037	0.031	1.307	1.201	30.681	20.100
6	3	520	170	0.023	0.020	0.020	1.121	0.988	12.125	1.212
6	4	520	170	0.026	0.020	0.020	1.300	0.990	29.962	1.008
6	5	520	170	0.026	0.021	0.020	1.291	1.039	29.085	3.906
	AVERAGE:								44.276	26.981

Floor 3:

Façade	Window	B façade H	B façade W	RATIO BB	RATIO AS	RATIO B	% BB	% AS	OFFSET % BB	OFFSET % AS
2	7	314	344	0.011	0.010	0.014	0.775	0.732	22.539	26.752
2	8	314	344	0.010	0.009	0.014	0.705	0.668	29.538	33.210
2	9	314	344	0.011	0.010	0.014	0.771	0.718	22.890	28.224
2	10	314	344	0.009	0.009	0.014	0.676	0.634	32.443	36.644
2	11	314	344	0.011	0.010	0.014	0.791	0.755	20.850	24.545
3	7	586	262	0.014	0.012	0.013	1.096	0.980	9.649	2.013
3	8	586	262	0.015	0.013	0.013	1.160	1.013	15.975	1.281
3	9	586	262	0.016	0.012	0.013	1.267	0.926	26.703	7.403
4	5	530	190	0.020	0.018	0.020	0.991	0.890	0.900	11.030
4	6	530	190	0.028	0.023	0.020	1.396	1.134	39.563	13.418
4	7	530	190	0.023	0.018	0.020	1.126	0.912	12.601	8.804
5	6	420	140	0.049	0.039	0.031	1.582	1.241	58.166	24.098
5	7	420	140	0.037	0.032	0.031	1.173	1.043	17.303	4.346
5	8	420	140	0.045	0.036	0.031	1.461	1.170	46.146	16.985
6	6	520	170	0.033	0.029	0.020	1.608	1.423	60.758	42.269
6	7	520	170	0.020	0.017	0.020	0.981	0.840	1.891	16.021
6	8	520	170	0.022	0.020	0.020	1.086	0.965	8.635	3.516
	AVERAGE:								24.858	12.599

Floor 4:

Façade	Window	B façade H	B façade W	RATIO BB	RATIO AS	RATIO B	% BB	% AS	OFFSET % BB	OFFSET % AS
2	12	314	344	0.009	0.008	0.014	0.647	0.557	35.266	44.328
2	13	314	344	0.006	0.006	0.014	0.455	0.421	54.516	57.899
3	10	586	262	0.011	0.008	0.013	0.870	0.631	12.996	36.854
3	11	586	262	0.012	0.010	0.013	0.941	0.823	5.880	17.713
4	8	530	190	0.026	0.022	0.020	1.316	1.097	31.586	9.728
4	9	530	190	0.024	0.021	0.020	1.207	1.062	20.720	6.203
4	10	530	190	0.031	0.024	0.020	1.544	1.218	54.376	21.766
5	9	420	140	0.053	0.042	0.031	1.689	1.342	68.891	34.206
5	10	420	140	0.049	0.040	0.031	1.584	1.286	58.352	28.576
5	11	420	140	0.050	0.041	0.031	1.609	1.326	60.906	32.589
6	9	520	170	0.037	0.030	0.020	1.807	1.475	80.729	47.488
6	10	520	170	0.031	0.028	0.020	1.536	1.366	53.644	36.575
6	11	520	170	0.030	0.027	0.020	1.497	1.338	49.722	33.796
	AVERAGE:								46.026	30.283

Figure 56: Detailed tables for specific floors

F Appendix - Dataset filtering code

```
from owslib.wfs import WebFeatureService
import xml.etree.ElementTree as ET
from shapely.geometry import Polygon
from shapely import ops
import numpy as np, pdal, json
import shapely.wkt

# retrieve BAG features
wfs11 = WebFeatureService("https://geodata.nationaalgeoregister.nl/bag/wfs?",
    version="1.1.0")

wfs_response = wfs11.getfeature(typename="bag:pand", bbox=(93020, 436021, 94263,
    436884))

bag_parse = ET.parse(wfs_response)
bag = bag_parse.getroot()

# parse BAG features into a list
bag_polygons = []
for geom in bag.iter("{http://www.opengis.net/gml}posList"):
    np_points = (np.array(geom.text.split())).astype(float)

    np_points = np_points[np_points != 0]

    np_points = np.split(np_points, len(np_points)/2)

    bag_polygon = Polygon(list(map(tuple, np_points)))

    bag_polygons.append(bag_polygon)

# for removing some BAG-features that we don't want
manual_bag_polygons = [shapely.wkt.loads( * insert some long WKT polygons *)
                 ]

# for removing other artefacts
manual_other_polygons = [shapely.wkt.loads("POLYGON␣((93655.320313␣436559.75,␣
    93654.617188␣436560.75,␣93571.359375␣436481.25,␣93568.539063␣436478.156250,␣
    93655.320313␣436559.75))"),
                        shapely.wkt.loads("POLYGON␣((93923.23␣436705.93,␣93923.23␣
                            436741.5,␣93954.78999999999␣436741.5,␣93954.78999999999␣
                            436741.18,␣93946.78999999999␣436741.18,␣
                            93946.78999999999␣436732.37,␣93954.78999999999␣
                            436732.37,␣93954.78999999999␣436693.44,␣93934.95␣
                            436693.44,␣93934.95␣436705.93,␣93923.23␣436705.93))")]

# remove the unwanted parts that we have just defined from the original BAG
manual_bag_polygons = ops.unary_union(manual_bag_polygons)
manual_other_polygons = ops.unary_union(manual_other_polygons)
bag_polygons = ops.unary_union(bag_polygons)
bag_polygons = bag_polygons.difference(manual_bag_polygons)
bag_polygons = bag_polygons.buffer(1.8, cap_style=3)
bag_polygons = bag_polygons.difference(manual_other_polygons).wkt


# the filtering steps that are described in the report
json = """
{
    "pipeline":
    [
        "filtered_output.las",
        {
            "type":"filters.reprojection",
            "out_srs":"EPSG:28992"
        },
         {
          "type":"filters.assign",
          "assignment" : "NumberOfReturns[0:0]=1"
        },
        {
            "type":"filters.pmf"
        },
        {
            "type":"filters.hag"
        },
        {
            "type":"filters.range",
            "limits":"HeightAboveGround[0.2:]"
        },
        {
            "type":"filters.crop",
            "polygon":%s
        },
        {
            "type":"writers.las",
            "filename":"bag_test_full3.las",
            "extra_dims":"all"
        }
    ]
}
""" %json.dumps(bag_polygons)

# PDAL performing the steps of the above defined pipeline
pipeline = pdal.Pipeline(json)
pipeline.validate() # check if our JSON and options were good
pipeline.loglevel = 8 #really noisy
count = pipeline.execute()
arrays = pipeline.arrays
metadata = pipeline.metadata
log = pipeline.log
```

G Appendix - BAG-linking code

```python
def bag_id(ins,outs): # ins and outs refer to the point cloud
    bag_polygons, bag_ids = retrieve_bag()

    # slightly erode BAG polygons
    polygons = []
    for polygon in bag_polygons:
        polygons.append(polygon.buffer(-0.001, cap_style=3, join_style=2))

    # arrange the points from the point cloud
    points = np.stack((ins['X'], ins['Y']), axis=-1)

    polygon_points = []
    polygon_points_ids = []
    polygon_lines_set = []

    # first, retrieve all the lines from the polygons
    for i, polygon in enumerate(polygons):
        x, y = polygon.exterior.coords.xy
        polygon_lines = []

        for c in range(len(x) - 1):
            polygon_lines.append(LineString((Point(x[c], y[c]), Point(x[c+1], y[c
                +1])))) # create line with current and next point
            polygon_lines_set.append(LineString((Point(x[c], y[c]), Point(x[c+1], y
                [c+1]))))

        # then, create points on the lines
        for l in polygon_lines:
            for distance in range(int(l.length / 0.5)):
                point = l.interpolate(0.5 * distance )
                polygon_points.append((point.x, point.y))
                polygon_points_ids.append(bag_ids[i])

    # create kd-tree
    kdtree = scipy.spatial.cKDTree(polygon_points)
    neigh_dist, neigh_i = kdtree.query(points, k=1)

    # we've just retrieved nearest neighbour indices, but we want the
        corresponding BAG-ids and put it in the .las
    neigh_bag_id = np.empty((0, 1))
    neigh_bag_id = np.append(neigh_bag_id, [ polygon_points_ids[i] for i in
        neigh_i ])


    # now, we need to change the BAG-id for points that actually intersect a BAG-feature, using rtree
    rtree = build_rtree(bag_polygons)
    func = get_intersection_func(rtree)

    # check for every point with which polygon it intersects
    for i, point in enumerate(points):
        my_intersections = func(point)
        if(len(my_intersections) > 0): # because if len = 0, there is no intersection
            neigh_bag_id[i] = bag_ids[my_intersections[0].id] # assign BAG-id


    # time to update the point cloud with the BAG-ids
    neigh_bag_id = neigh_bag_id.astype(np.dtype('Float64')) # a new dimension added (in the PDAL pipeline) always is of type float64

    outs['bag_id'] = neigh_bag_id # update bag_id dimension in point cloud

    return True # PDAL requires this for a self-defined function


# the next three functions are for the rtree. mostly copied from https://stackoverflow.com/questions/53224490/how-can-i-look-up-a-polygon-that-contains-a-point-fast-given-all-polygons-are-a
def get_containing_box(p):
    # this function is needed because the rtree only takes boxes, and not polygons
    xcoords = [x for x, _ in p.exterior.coords]
    ycoords = [y for _, y in p.exterior.coords]
    box = (min(xcoords), min(ycoords), max(xcoords), max(ycoords))
    return box

def build_rtree(polys):
    def generate_items():
        pindx = 0
        for pol in polys:
            box = get_containing_box(pol)
            yield (pindx, box, pol)
            pindx += 1
    return index.Index(generate_items())

def get_intersection_func(rtree_index):
    def intersection_func(point):
        pbox = (point[0], point[1],
                point[0], point[1])
        hits = rtree_index.intersection(pbox, objects=True)
        # Filter false positives:
        result = [pol for pol in hits if (pol.object).intersects(Point(point)) ]
        return result
return intersection_func
```

H Appendix - Bitmap points generation code

```
void MyViewer::bitmap(easy3d::PointCloud* cloud, std::vector<vec4> all_plane_vertices, std::vector<int> list, std::vector<vec3> vertex_list, std::vector<vec4> plane_list, std::vector<vec3> &all_vx, std::vector<int> &primitive_index, std::vector<vec4> &primitive_list, std::vector<vec3> &primitive_point_list) {
  auto points = cloud->get_vertex_property<vec3>("v:point");
  auto indices = cloud->vertex_property<int>("v:segment_index", -1);
  auto plane_normals = cloud->vertex_property<vec3>("v:plane_normals", vec3(0, 0, 0));

  std::vector<vec3> colors;
  std::vector<vec3> all_plane_list;
  SurfaceMesh* mesh = new SurfaceMesh;

  std::vector<vec3> candidate_points;
  std::cout << "Please␣enter␣path␣of␣output␣file:";
  std::string path;
  std::cin >> path;
  std::ofstream fout(path);

  int all_cols;
  int all_rows;
  float gap;
  std::cout << "Please␣enter␣the␣size␣of␣cell:";
  std::cin >> gap;

  float threshold;
  std::cout << "Please␣enter␣the␣threshold␣of␣bitmap:";
  std::cin >> threshold;
  if (!gap || threshold < 0) {
    std::cerr << "Warning:␣size/threshold␣is␣invalid!" << std::endl;
  }
  else {
    std::cout << "Bitmap␣is␣generating..." << std::endl;

    std::vector<int> points_number;
    for (int i = 0; i < list.size(); ++i) {
      std::vector<easy3d::PointCloud::Vertex> vertices_i;
      for (auto v : cloud->vertices()) {
        if (indices[v] == list[i])
          vertices_i.push_back(v);
      }
    points_number.push_back(vertices_i.size());
  }

  auto max_element = std::max_element(std::begin(points_number), std::end(
    points_number));
  int max_facade = list[std::distance(std::begin(points_number), max_element)];

  for (int i = 0; i < list.size(); ++i) {
    if (list[i] != max_facade)//only keep one primitive with most points, which
      is detected by RANSAC
      continue;

    vec3 n = vec3(plane_list[i][0], plane_list[i][1], plane_list[i][2]);
    if (abs(n[2]) > 0.02)//don't deal with the horizontal planes(i.e.,rooftops)
      continue;
    std::vector<vec4> plane_vertices;
    std::vector<int> plane_indices;
    vec3 c = random_color();
    for (int j = 0; j < all_plane_vertices.size(); ++j) {
      int index = all_plane_vertices[j][3];
      if (index / 100 == list[i]) {
        plane_vertices.push_back(vec4(all_plane_vertices[j][0],
          all_plane_vertices[j][1], all_plane_vertices[j][2], index % 100));//
          plane_vertices are vertors which store the vertices of each face
          polygon
        plane_indices.push_back(index % 100);
      }
    }
    plane_indices.erase(unique(plane_indices.begin(), plane_indices.end()),
      plane_indices.end());

    std::vector<easy3d::PointCloud::Vertex> vertices_i;
    for (auto v : cloud->vertices()) {//find points belongs to the facade
      if (indices[v] == list[i])
        vertices_i.push_back(v);
    }

    //calculate the transform matrix
    float sumX = 0;
    float sumY = 0;
    float sumZ = 0;
    for (int j = 0; j < vertices_i.size(); ++j) {
      sumX += points[vertices_i[j]][0];
      sumY += points[vertices_i[j]][1];
      sumZ += points[vertices_i[j]][2];
    }

    //coordinates of centroid of these points
    float centerX = sumX / vertices_i.size();
    float centerY = sumY / vertices_i.size();
    float centerZ = sumZ / vertices_i.size();

    sort(plane_vertices.begin(), plane_vertices.end(), compX);
    sort(plane_vertices.begin(), plane_vertices.end(), compY);
    sort(plane_vertices.begin(), plane_vertices.end(), compZ);
    vec3 p0 = plane_vertices[0];
    plane_vertices.erase(plane_vertices.begin());

    std::vector<vec4>dif_z;
    for (int k = 0; k < plane_vertices.size(); ++k) {
      float dz;
      dz = plane_vertices[k][2] - p0[2];
      dif_z.push_back(vec4(plane_vertices[k][0], plane_vertices[k][1],
          plane_vertices[k][2], dz));
    }
    sort(dif_z.begin(), dif_z.end(), compGradient);

    vec3 vy;//y-axes direction vector
    for (int k = 0; k <dif_z.size(); ++k) {
      if (plane_vertices[k][0] != p0[0] || plane_vertices[k][1] != p0[1] ||
          plane_vertices[k][2] != p0[2]) {
        vy = vec3(dif_z[k][0] - p0[0], dif_z[k][1] - p0[1], 0);
        break;
      }
    }
    vec3 vx = cross(vy, n);//x-axes direction vector
    vec3 vz = n;//z-axes direction vector

    mat4 T = mat4(vx[0], vy[0], vz[0], centerX,
      vx[1], vy[1], vz[1], centerY,
      vx[2], vy[2], vz[2], centerZ,
      0, 0, 0, 1);//transform matrix
    mat4 inv_T = inverse(T);
    fout << vx[0] << "␣␣" << vy[0] << "␣" << vz[0] << "␣␣" << centerX << std::
        endl;
    fout << vx[1] << "␣␣" << vy[1] << "␣" << vz[1] << "␣␣" << centerY << std::
        endl;
    fout << vx[2] << "␣␣" << vy[2] << "␣" << vz[2] << "␣␣" << centerZ << std::
      endl;
    fout << 0 << "␣␣" << 0 << "␣" << 0 << "␣␣" << 1 << std::endl;

    float sumz = 0;
    std::vector<vec3> projected_points;
    for (int j = 0; j < vertices_i.size(); ++j) {//project the points to 2d
      vec4 p = vec4(points[vertices_i[j]][0], points[vertices_i[j]][1], points[vertices_i[j]][2], 1);
      vec4 pp = inv_T * p;
      projected_points.push_back(vec3(pp[0], pp[1], pp[2]));
      sumz += pp[2];
    }
    float z = sumz / projected_points.size();
    fout << z << std::endl;

    int num = 0;

    //get number of cols and rows by dividing the length/width with cell size
    sort(projected_points.begin(), projected_points.end(), compX);
    float xmin = projected_points[0][0];
    float xmax = projected_points[projected_points.size() - 1][0];
    sort(projected_points.begin(), projected_points.end(), compY);
    float ymin = projected_points[0][1];
    float ymax = projected_points[projected_points.size() - 1][1];
    int cols = int((xmax - xmin) / gap) + 1;
    int rows = int((ymax - ymin) / gap) + 1;
    fout << cols << "␣" << rows << std::endl;//export number of cols and rows, in order to generate raster bitmap in the next step

    std::vector<vec3> candidate_points_j;
    for (int k = 0; k < cols; ++k) {
      for (int l = 0; l < rows; ++l) {
        int inside_point = 0;
        //p0, p1, p2, p3 are the vertices of each cell
        vec3 p0 = vec3(xmin + gap*k, ymin + gap* l, 1);
        vec3 p1 = vec3(xmin + gap*(k + 1), ymin + gap* l, 1);
        vec3 p2 = vec3(xmin + gap*(k + 1), ymin + gap* (l + 1), 1);
        vec3 p3 = vec3(xmin + gap*k, ymin + gap* (l + 1), 1);
        std::vector<vec3> grid;
        grid.push_back(p0);
        grid.push_back(p1);
        grid.push_back(p2);
        grid.push_back(p3);
        grid.push_back(p0);
```

```

          int is_hole = 1;//variable which shows if the cell is inside a hole (candidate windows)

          for (int m = 0; m < projected_points.size(); ++m) {
            if (in_polygon(projected_points[m], grid) == 1) {//determine if the point is inside the cell
              inside_point += 1;
            }
            if (inside_point > threshold) {
              is_hole = 0;
              break;
            }
          }

          vec4 grid_point = vec4((p0[0] + p1[0]) / 2, (p0[1] + p2[1]) / 2, z, 1);
          vec4 candidate = T * grid_point;//reproject it to 3d
          if (is_hole == 1) {
            fout << grid_point[0] << "␣" << grid_point[1] << "␣" << 0 << "␣␣" << list[i] << std::endl;
          }
          else if (is_hole == 0) {
            candidate_points.push_back(vec3(candidate[0], candidate[1], candidate[2]));
            fout << grid_point[0] << "␣" << grid_point[1] << "␣" << 1 << "␣␣" << list[i] << std::endl;
          }
        }
      }
      projected_points.clear();
      vertices_i.clear();
    }
    std::cout << "creating␣drawable..." << std::endl;
    easy3d::PointsDrawable* drawable = cloud->points_drawable("points");
    drawable->update_vertex_buffer(candidate_points);
    drawable->set_per_vertex_color(true);
    drawable->set_point_size(0.8f);
    update();
  }
}
```

I Appendix - Extreme points extracting by slicing method code

```
void MyViewer::slicing(easy3d::PointCloud* cloud, std::vector<vec4>
    all_plane_vertices, std::vector<int> list, std::vector<vec4> plane_list) {
  auto points = cloud->get_vertex_property<vec3>("v:point");
  auto indices = cloud->vertex_property<int>("v:segment_index", -1);
  auto plane_normals = cloud->vertex_property<vec3>("v:plane_normals", vec3(0, 0,
      0));

  std::vector<vec3> colors;
  std::vector<vec3> all_slicing_vertices;
  std::vector<vec3> boundaries;
  SurfaceMesh* mesh = new SurfaceMesh;

  std::cout << "Please␣enter␣BAG_id:";
  std::string BAG_id;
  std::cin >> BAG_id;

  std::cout << "Please␣enter␣window_ratio:";
  float ratio;
  std::cin >> ratio;

  std::cout << "Please␣enter␣path␣of␣output␣file:";
  std::string path;
  std::cin >> path;
  std::ofstream fout(path);

  std::cout << "Please␣enter␣type␣of␣threhold␣(1.␣average,␣2.␣median,␣3.constant):
      ";
  int threshold_type;
  std::cin >> threshold_type;

  std::cout << "Please␣enter␣the␣size␣of␣threhold␣(if␣type␣is␣average␣or␣median,␣
      then␣threshold␣will␣be␣x␣times␣of␣type␣above,␣if␣type␣is␣constant,␣then␣the␣
      size␣is␣threshold␣itself␣):";
  float threshold_size;
  std::cin >> threshold_size;

  float num;
  std::cout << "Please␣enter␣the␣number␣of␣slicing␣(veritical␣slicing):";
  std::cin >> num;

  float num2;
  std::cout << "Please␣enter␣the␣number␣of␣slicing␣(horizontal␣slicing):";
  std::cin >> num2;

  if (num < 0 || num2 < 0) {
    std::cerr << "Warning:␣num␣is␣invalid!" << std::endl;
  }
  else {
    std::cout << "Slicing␣is␣generating..." << std::endl;

    std::vector<int> points_number;
    std::vector<int> rows;
    std::vector<int> cols;
    std::vector<vec3> all_extreme_points;

    for (int i = 0; i < list.size(); ++i) {
      std::vector<easy3d::PointCloud::Vertex> vertices_i;
      for (auto v : cloud->vertices()) {
        if (indices[v] == list[i])
          vertices_i.push_back(v);
      }
      points_number.push_back(vertices_i.size());
    }

    auto max_element = std::max_element(std::begin(points_number), std::end(points_number));
    int max_facade = list[std::distance(std::begin(points_number), max_element)];

    for (int i = 0; i < list.size(); ++i) {
      if (list[i] != max_facade)//only keep one primitive with most points, which is detected by RANSAC
        continue;

      vec3 n = vec3(plane_list[i][0], plane_list[i][1], plane_list[i][2]);
      if (abs(n[2]) > 0.02)//don't deal with the horizontal planes(i.e. rooftops)
        continue;
      std::vector<vec4> plane_vertices;
      std::vector<int> plane_indices;

      for (int j = 0; j < all_plane_vertices.size(); ++j) {
        int index = all_plane_vertices[j][3];
        if (index / 100 == list[i]) {
          plane_vertices.push_back(vec4(all_plane_vertices[j][0], all_plane_vertices[j][1], all_plane_vertices[j][2], index % 100));//plane_vertices are vertors which store the vertices of each face
          polygon
        plane_indices.push_back(index % 100);
      }
    }
    plane_indices.erase(unique(plane_indices.begin(), plane_indices.end()),
        plane_indices.end());

    std::vector<easy3d::PointCloud::Vertex> vertices_i;
    Box3 box;
    for (auto v : cloud->vertices()) {
      if (indices[v] == list[i]) {
        vertices_i.push_back(v);//find points belongs to the facade
        box.add_point(points[v]);
      }
    }

    //since the easy3d library can only deal with 6-digits, and this will cause
        lose of decimal part of coordinates, so we first move them to upper left
        corner of all the points to keep the precision
    float minx = box.x_min();
    float miny = box.y_min();
    float minz = box.z_min();

    fout << minx << "␣" << miny << "␣␣" << minz << std::endl;

    //calculate the transform matrix
    float sumX = 0;
    float sumY = 0;
    float sumZ = 0;
    for (int j = 0; j < vertices_i.size(); ++j) {
      sumX += points[vertices_i[j]][0];
      sumY += points[vertices_i[j]][1];
      sumZ += points[vertices_i[j]][2];
    }

    //coordinates of centroid of these points
    float centerX = sumX / vertices_i.size();
    float centerY = sumY / vertices_i.size();
    float centerZ = sumZ / vertices_i.size();

    sort(plane_vertices.begin(), plane_vertices.end(), compX);
    sort(plane_vertices.begin(), plane_vertices.end(), compY);
    sort(plane_vertices.begin(), plane_vertices.end(), compZ);
    vec3 p0 = plane_vertices[0];
```

```
      plane_vertices.erase(plane_vertices.begin());

      std::vector<vec4>dif_z;
      for (int k = 0; k < plane_vertices.size(); ++k) {
        float dz;
        dz = plane_vertices[k][2] - p0[2];
        dif_z.push_back(vec4(plane_vertices[k][0], plane_vertices[k][1],
            plane_vertices[k][2], dz));
      }
      sort(dif_z.begin(), dif_z.end(), compGradient);

      vec3 vy;//y-axes direction vector
      for (int k = 0; k <dif_z.size(); ++k) {
        if (plane_vertices[k][0] != p0[0] || plane_vertices[k][1] != p0[1] ||
            plane_vertices[k][2] != p0[2]) {
          vy = vec3(dif_z[k][0] - p0[0], dif_z[k][1] - p0[1], 0);
          break;
        }
      }
      vec3 vx = cross(vy, n);//x-axes direction vector
      vec3 vz = n;//z-axes direction vector

      mat4 T = mat4(vx[0], vy[0], vz[0], centerX,
        vx[1], vy[1], vz[1], centerY,
        vx[2], vy[2], vz[2], centerZ,
        0, 0, 0, 1);//transform matrix
      mat4 inv_T = inverse(T);

      std::vector<vec3> projected_points;
      float sumz = 0;
      for (int j = 0; j < vertices_i.size(); ++j) {//get the 2d coordinates of
          points on the original plane i
        vec4 p = vec4(points[vertices_i[j]][0], points[vertices_i[j]][1], points[
            vertices_i[j]][2], 1);
        vec4 pp = inv_T * p;
        projected_points.push_back(vec3(pp[0], pp[1], 1));
        sumz += pp[2];
      }
      float avrz = sumz / vertices_i.size();

      sort(projected_points.begin(), projected_points.end(), compX);
      float xmin = projected_points[0][0];
      float xmax = projected_points[projected_points.size() - 1][0];
      sort(projected_points.begin(), projected_points.end(), compY);
      float ymin = projected_points[0][1];
      float ymax = projected_points[projected_points.size() - 1][1];

      float gapy = (ymax - ymin) / num;

      for (int j = 0; j < num; ++j) {//vertical slicing
        std::vector<double> slicing_x;
        std::vector<vec3> slicing_points;

        for (int k = 0; k < projected_points.size(); ++k) {
          if ((projected_points[k][1] >= ymin + gapy * j) && (projected_points[k
              ][1] <= ymin + gapy * (j + 1)))
            slicing_points.push_back(projected_points[k]);//find points which
                belong to current slice
        }

        if (slicing_points.size() == 0)//ignore the slice without any point
          continue;

        sort(slicing_points.begin(), slicing_points.end(), compX);
        std::vector<double> dis;
        double sumd = 0;
        for (int k = 0; k < slicing_points.size() - 1; ++k) {
          double d = slicing_points[k + 1][0] - slicing_points[k][0];
          sumd += d;
          if (d > 0)
            dis.push_back(d);
        }
        sort(dis.begin(), dis.end());//vector dis is to store the distance between
             each two neighbors

        double threshold;//determine the threshold according to the parameters
        if (dis.size() != 0) {
          if (threshold_type == 1)
            threshold = threshold_size * sumd / (slicing_points.size() - 1);
          else if (threshold_type == 2)
            threshold = threshold_size * dis[int(dis.size() / 2)];
        }
        else {
          if (threshold_type == 3)
            threshold = threshold_size;
          else
            threshold = 9999;
        }
```

```

slicing_x.push_back(slicing_points[0][0]);
fout << slicing_points[0][0] << "␣" << ymin + gapy * j << "␣␣" << 1 << "␣␣" << list[i] << std::endl;
fout << slicing_points[0][0] << "␣" << ymin + gapy * (j + 0.5) << "␣␣" << 1 << "␣␣" << list[i] << std::endl;
fout << slicing_points[0][0] << "␣" << ymin + gapy * (j + 1) << "␣␣" << 1 << "␣␣" << list[i] << std::endl;

for (int k = 0; k < slicing_points.size() - 1; ++k) {
  if ((slicing_points[k + 1][0] - slicing_points[k][0]) > threshold) {//if the distance between these two points is larger than the threshold, then they are extracted as extreme points
    slicing_x.push_back(slicing_points[k][0]);
    fout << slicing_points[k][0] << "␣" << ymin + gapy * j << "␣␣" << 0 << "␣␣" << list[i] << std::endl;
    fout << slicing_points[k][0] << "␣" << ymin + gapy * (j + 0.5) << "␣␣" << 0 << "␣␣" << list[i] << std::endl;
    fout << slicing_points[k][0] << "␣" << ymin + gapy * (j + 1) << "␣␣" << 0 << "␣␣" << list[i] << std::endl;

    slicing_x.push_back(slicing_points[k + 1][0]);
    fout << slicing_points[k + 1][0] << "␣" << ymin + gapy * j << "␣␣" << 0 << "␣␣" << list[i] << std::endl;
    fout << slicing_points[k + 1][0] << "␣" << ymin + gapy * (j + 0.5) << "␣␣" << 0 << "␣␣" << list[i] << std::endl;
    fout << slicing_points[k + 1][0] << "␣" << ymin + gapy * (j + 1) << "␣␣" << 0 << "␣␣" << list[i] << std::endl;
  }
}

slicing_x.push_back(slicing_points[slicing_points.size() - 1][0]);
fout << slicing_points[slicing_points.size() - 1][0] << "␣" << ymin + gapy * j << "␣␣" << 1 << "␣␣" << list[i] << std::endl;
fout << slicing_points[slicing_points.size() - 1][0] << "␣" << ymin + gapy * (j + 0.5) << "␣␣" << 1 << "␣␣" << list[i] << std::endl;
fout << slicing_points[slicing_points.size() - 1][0] << "␣" << ymin + gapy * (j + 1) << "␣␣" << 1 << "␣␣" << list[i] << std::endl;

for (int k = 0; k < slicing_x.size() / 2; ++k) {
  std::vector<vec3> slicing_vertices;
  vec3 c = random_color();
  vec4 p0 = vec4(slicing_x[2 * k], ymin + gapy * j, avrz, 1);
  vec4 p1 = vec4(slicing_x[2 * k + 1], ymin + gapy * j, avrz, 1);
```

```
        vec4 p2 = vec4(slicing_x[2 * k + 1], ymin + gapy * (j + 1), avrz, 1);
        vec4 p3 = vec4(slicing_x[2 * k], ymin + gapy * (j + 1), avrz, 1);

        vec4 p00 = T * p0;//reproject the points to 3d
        vec4 p11 = T * p1;
        vec4 p22 = T * p2;
        vec4 p33 = T * p3;

        slicing_vertices.push_back(vec3(p11[0], p11[1], p11[2]));
        slicing_vertices.push_back(vec3(p22[0], p22[1], p22[2]));
        slicing_vertices.push_back(vec3(p33[0], p33[1], p33[2]));

        boundaries.push_back(vec3(p00[0], p00[1], p00[2]));
        boundaries.push_back(vec3(p11[0], p11[1], p11[2]));
        boundaries.push_back(vec3(p22[0], p22[1], p22[2]));
        boundaries.push_back(vec3(p33[0], p33[1], p33[2]));

        //draw the mesh of slices
        for (int k = 0; k < slicing_vertices.size() - 1; ++k) {
          vec3 p0 = vec3(p00[0], p00[1], p00[2]);
          vec3 p1 = slicing_vertices[k];
          vec3 p2 = slicing_vertices[k + 1];
          all_slicing_vertices.push_back(p0);
          all_slicing_vertices.push_back(p1);
          all_slicing_vertices.push_back(p2);
          std::vector<SurfaceMesh::Vertex> face_vertices = {
            mesh->add_vertex(p0),
            mesh->add_vertex(p1),
            mesh->add_vertex(p2)
          };
          mesh->add_face(face_vertices);
          colors.push_back(c);
          colors.push_back(c);
          colors.push_back(c);
        }
      }
    }

    float gapx = (xmax - xmin) / num2;

    for (int j = 0; j < num2; ++j) {//horizontal slicing
      std::vector<double> slicing_y;
      std::vector<vec3> slicing_points;

      for (int k = 0; k < projected_points.size(); ++k) {
        if ((projected_points[k][0] >= xmin + gapx * j) && (projected_points[k][0] <= xmin + gapx * (j + 1)))
          slicing_points.push_back(projected_points[k]);//find points which belong to current slice
      }

      if (slicing_points.size() == 0)//ignore the slice without any point
        continue;

      sort(slicing_points.begin(), slicing_points.end(), compY);
      std::vector<double> dis;
      double sumd = 0;
      for (int k = 0; k < slicing_points.size() - 1; ++k) {
        double d = slicing_points[k + 1][1] - slicing_points[k][1];
        sumd += d;
        if (d > 0)
          dis.push_back(d);
      }
      sort(dis.begin(), dis.end());//vector dis is to store the distance between each two neighbors

      double threshold;//determine the threshold according to the parameters
      if (dis.size() != 0) {
        if (threshold_type == 1)
          threshold = threshold_size * sumd / (slicing_points.size() - 1);
        else if (threshold_type == 2)
          threshold = threshold_size * dis[int(dis.size() / 2)];
      }
      else {
        if (threshold_type == 3)
          threshold = threshold_size;
        else
          threshold = 9999;
      }

      slicing_y.push_back(slicing_points[0][1]);
      fout << xmin + gapx * j << "␣␣" << slicing_points[0][1] << "␣" << 1 << "␣␣" << list[i] << std::endl;
      fout << xmin + gapx * (j + 0.5) << "␣␣" << slicing_points[0][1] << "␣␣" << 1 << "␣␣" << list[i] << std::endl;
      fout << xmin + gapx * (j + 1) << "␣␣" << slicing_points[0][1] << "␣␣" << 1 << "␣␣" << list[i] << std::endl;

```

```
        for (int k = 0; k < slicing_points.size() - 1; ++k) {
          if ((slicing_points[k + 1][1] - slicing_points[k][1]) > threshold) {//if the distance between these two points is larger than the threshold, then they are extracted as extreme points
            slicing_y.push_back(slicing_points[k][1]);
            fout << xmin + gapx * j << "␣␣" << slicing_points[k][1] << "␣" << 0 << "␣␣" << list[i] << std::endl;
            fout << xmin + gapx * (j + 0.5) << "␣␣" << slicing_points[k][1] << "␣␣" << 0 << "␣␣" << list[i] << std::endl;
            fout << xmin + gapx * (j + 1) << "␣␣" << slicing_points[k][1] << "␣␣" << 0 << "␣␣" << list[i] << std::endl;

            slicing_y.push_back(slicing_points[k + 1][1]);
            fout << xmin + gapx * j << "␣␣" << slicing_points[k + 1][1] << "␣" << 0 << "␣␣" << list[i] << std::endl;
            fout << xmin + gapx * (j + 0.5) << "␣␣" << slicing_points[k + 1][1] << "␣␣" << 0 << "␣␣" << list[i] << std::endl;
            fout << xmin + gapx * (j + 1) << "␣␣" << slicing_points[k + 1][1] << "␣␣" << 0 << "␣␣" << list[i] << std::endl;
          }
        }
        slicing_y.push_back(slicing_points[slicing_points.size() - 1][1]);
        fout << xmin + gapx * j << "␣␣" << slicing_points[slicing_points.size() - 1][1] << "␣" << 1<< "␣␣" << list[i] << std::endl;
        fout << xmin + gapx * (j + 0.5) << "␣␣" << slicing_points[slicing_points.size() - 1][1] << "␣␣" << 1 << "␣␣" << list[i] << std::endl;
        fout << xmin + gapx * (j + 1) << "␣␣" << slicing_points[slicing_points.size() - 1][1] << "␣␣" << 1 << "␣␣" << list[i] << std::endl;

        for (int k = 0; k < slicing_y.size() / 2; ++k) {
          std::vector<vec3> slicing_vertices;

          vec3 c = random_color();
          vec4 p0 = vec4(xmin + gapx * j, slicing_y[2 * k], avrz, 1);
          vec4 p1 = vec4(xmin + gapx * j, slicing_y[2 * k + 1], avrz, 1);
          vec4 p2 = vec4(xmin + gapx * (j + 1), slicing_y[2 * k + 1], avrz, 1);
          vec4 p3 = vec4(xmin + gapx * (j + 1), slicing_y[2 * k], avrz, 1);

          vec4 p00 = T * p0;//reproject the points to 3d
          vec4 p11 = T * p1;
          vec4 p22 = T * p2;
          vec4 p33 = T * p3;
          slicing_vertices.push_back(vec3(p11[0], p11[1], p11[2]));
          slicing_vertices.push_back(vec3(p22[0], p22[1], p22[2]));
          slicing_vertices.push_back(vec3(p33[0], p33[1], p33[2]));

          boundaries.push_back(vec3(p00[0], p00[1], p00[2]));
          boundaries.push_back(vec3(p11[0], p11[1], p11[2]));
          boundaries.push_back(vec3(p22[0], p22[1], p22[2]));
          boundaries.push_back(vec3(p33[0], p33[1], p33[2]));

          //draw the mesh of slices
          for (int k = 0; k < slicing_vertices.size() - 1; ++k) {
            vec3 p0 = vec3(p00[0], p00[1], p00[2]);
            vec3 p1 = slicing_vertices[k];
            vec3 p2 = slicing_vertices[k + 1];
            all_slicing_vertices.push_back(p0);
            all_slicing_vertices.push_back(p1);
            all_slicing_vertices.push_back(p2);
            std::vector<SurfaceMesh::Vertex> face_vertices = {
              mesh->add_vertex(p0),
              mesh->add_vertex(p1),
              mesh->add_vertex(p2)
            };
            mesh->add_face(face_vertices);
            colors.push_back(c);
            colors.push_back(c);
            colors.push_back(c);
          }

          slicing_vertices.clear();
        }
      }
    }
    std::cout << "creating␣drawable..." << std::endl;
    easy3d::FacesDrawable* drawable = mesh->add_faces_drawable("surface");
    // transfer vertex coordinates and colors to the gpu.
    drawable->update_vertex_buffer(all_slicing_vertices);
    drawable->update_color_buffer(colors);
    drawable->set_per_vertex_color(true);
    Viewer::add_model(mesh);// add the model to the viewer
    easy3d::PointsDrawable* drawable2 = cloud->points_drawable("points");
    drawable2->update_vertex_buffer(boundaries);
    update();
  }
}
```

J Appendix - Clustering of extreme points from slicing

```python
import numpy as np
import alphashape #https://pypi.org/project/alphashape/ a=0 --> convex hull , a=2 --> concave hull, If you go too high on the alpha parameter, you will start to lose points from the original dataset.
from sklearn.cluster import DBSCAN #https://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html#sphx-glr-auto-examples-cluster-plot-dbscan-py
from shapely.geometry import Point, LineString, mapping
import scipy.spatial
import fiona
from fiona.crs import from_epsg

#Read the txt file in a list of lists
file_name = ""
input_path = "facade5.txt"
output_path = ""
file = open(input_path)
t_matrix = []
lines = []
for i, line in enumerate(file):
    # The rstrip method gets rid of the "\n" at the end of each line
    if i < 5:
        t_matrix.append(line.rstrip().split())
    else:
        lines.append(line.rstrip().split())
file.close()


# Distinguish window and facade points
facade = []
windows = []
for line in lines:
    if line[2] == "0":
        windows.append([float(line[0]), float(line[1])])
    else:
        facade.append([float(line[0]), float(line[1])])


#Facade area calculation
facade = np.array(facade)
alpha_facade = alphashape.alphashape(facade, alpha=2.0)
facade_surface = alpha_facade.area

```

```
#Clustering of windows
windows = np.array(windows)
clustering = DBSCAN(eps=0.055, min_samples=40).fit(windows)
unique_labels = np.unique(clustering.labels_)




#Collect indices of each individual window according to the clustering
window_indices = []
for label in unique_labels:
    window = np.where(clustering.labels_ == label)
    window_indices.append(window)

# create points on facade alpha shape
alpha_facade_points = []
x, y = alpha_facade.exterior.coords.xy
polygon_lines = []
for c in range(len(x) - 1):
    polygon_lines.append(LineString((Point(x[c], y[c]), Point(x[c+1], y[c+1])))) #
         create line with current and next point

for l in polygon_lines:
    for distance in range(int(l.length * 1000)):
        point = l.interpolate(((l.length / (int(l.length * 1000))) * distance))
        alpha_facade_points.append((point.x, point.y))

#collect final windows from indices
kdtree = scipy.spatial.cKDTree(alpha_facade_points)
window_facade_threshold = 0.000
windows_distance_threshold = 0.000
final_windows = []
alpha_shapes = []
window_surfaces = []
total_windows_surface = 0
for i, ind_window in enumerate(window_indices):
    if i == 0:
        continue
    final_window = windows[ind_window[0]]

    neigh_dist, neigh_i = kdtree.query(final_window, k=1)

    if min(neigh_dist) < window_facade_threshold: # if the window is too close to
        the facade, do not add it
```

```
            continue
        #elif (alphashape.alphashape(final_window, alpha=3.5)).area < 0.000:
        # continue
        else:
            combined_windows_indices = []
            deprecated_windows_indices = []
            polygon_window = alphashape.alphashape(final_window, alpha=3.5)

            # merge clusters based on the distances between their alpha shapes
            if(len(alpha_shapes) > 0):
                for j, window in enumerate(alpha_shapes):
                    if polygon_window.distance(window) < windows_distance_threshold:
                        #if polygon_window.area < 0.010 and window.area < 0.010:
                        final_window = np.concatenate((final_window, final_windows[j]))
                        polygon_window = alphashape.alphashape(final_window, alpha=3.5)
                        deprecated_windows_indices.append(j)



            final_windows.append(final_window)
            alpha_shapes.append(polygon_window)

            for i in sorted(deprecated_windows_indices, reverse=True):
                del final_windows[i]
                del alpha_shapes[i]

            # calculate surface of every window
            window_surfaces.append(polygon_window.area)
            #print("Current polygon area: ", polygon_window.area)
            total_windows_surface = total_windows_surface + polygon_window.area
            #print ("Total windows area: ",total_windows_surface)



#write the output WKT file
with open("alpha_shapes_test.txt", "w") as polygon:
    polygon.write("id|geom\n")
    polygon.write("0" + "|" + alpha_facade.wkt + "\n")
    for i, alpha_shape in enumerate(alpha_shapes, start=1):
        polygon.write(str(i) + "|" + alpha_shape.wkt + "|" + "\n")


with open("clusters.txt","w") as clustered:
    count = 0
        clustered.write("id|geom\n")
        for label in unique_labels:
            if label == -1:
                continue
            window = np.where(clustering.labels_==label)
            for index in window[0]:
                point_shapely = Point(windows[index][0],windows[index][1])
                clustered.write("1" + "|" + str(point_shapely) + "|" + str(count) + "\n")
            count +=1

ratio = total_windows_surface / facade_surface
with open("windows_labeled_test.txt" , "w") as labeled_points:
        for line in t_matrix:
            labeled_points.write('␣'.join(line) + "\n")

        for facade_point in facade:
            labeled_points.write(str(facade_point[0]) + "␣" + str(facade_point[1]) + "␣" + "1" + "␣" + str(ratio) + "\n")

        for i, window in enumerate(final_windows):
            for window_point in window:
labeled_points.write(str(window_point[0]) + "␣" + str(window_point[1]) + "␣" + str(i+2) + "␣" + str(ratio) + "\n")
```

K Appendix - Bitmaps before and after windows detection.

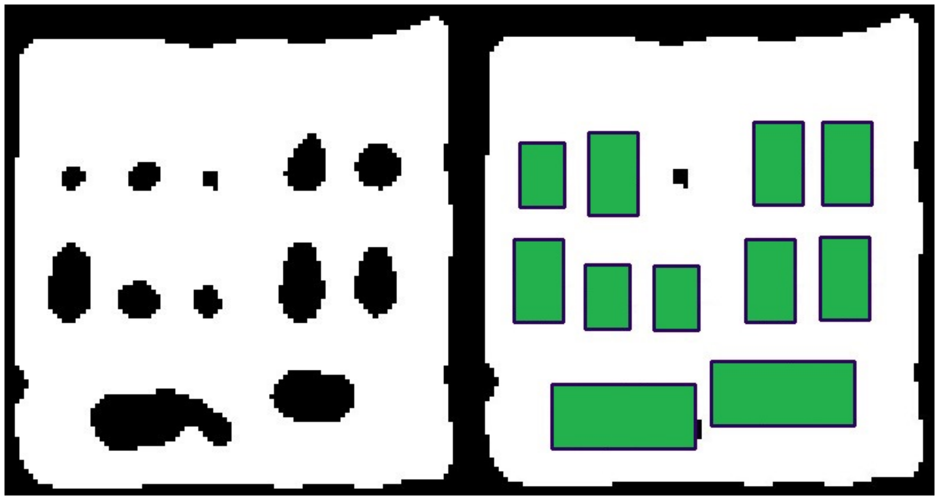


Figure 57: Facade 1 - bitmap before and after windows detection.

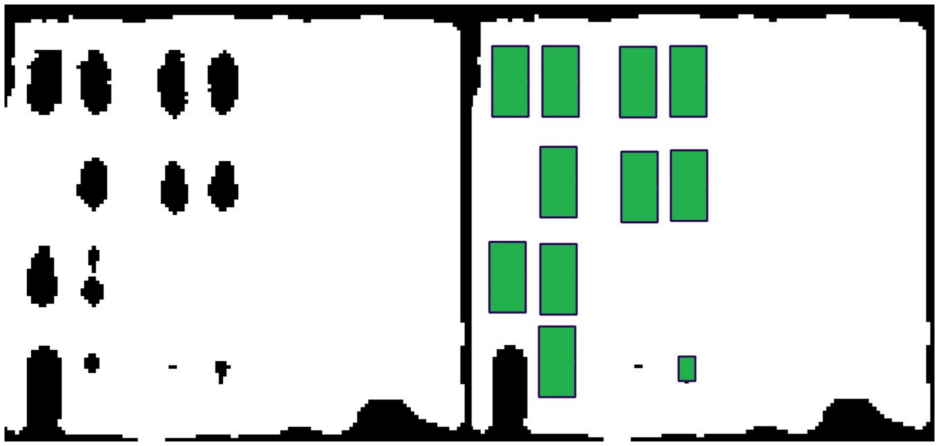


Figure 58: Facade 2 - bitmap before and after windows detection.

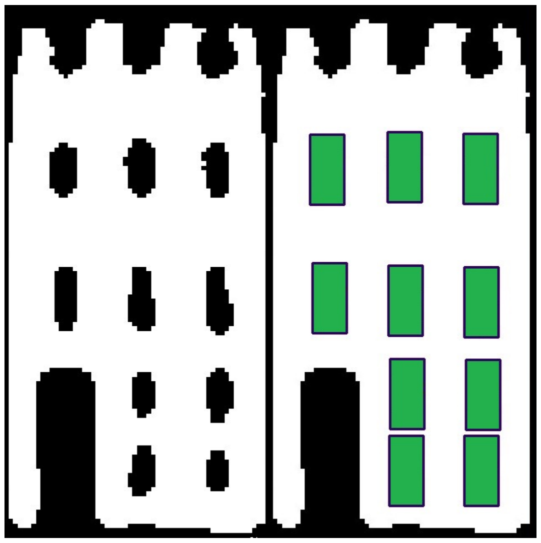


Figure 59: Facade 3 - bitmap before and after windows detection.

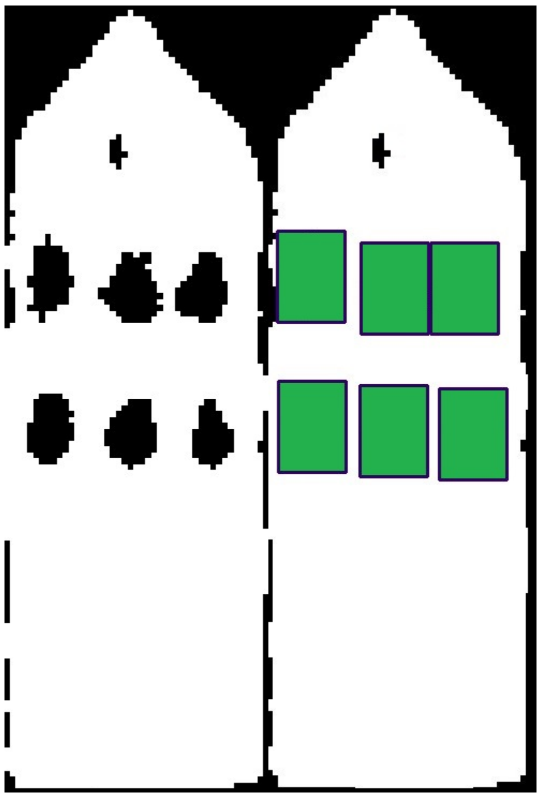


Figure 60: Facade 4 - bitmap before and after windows detection.

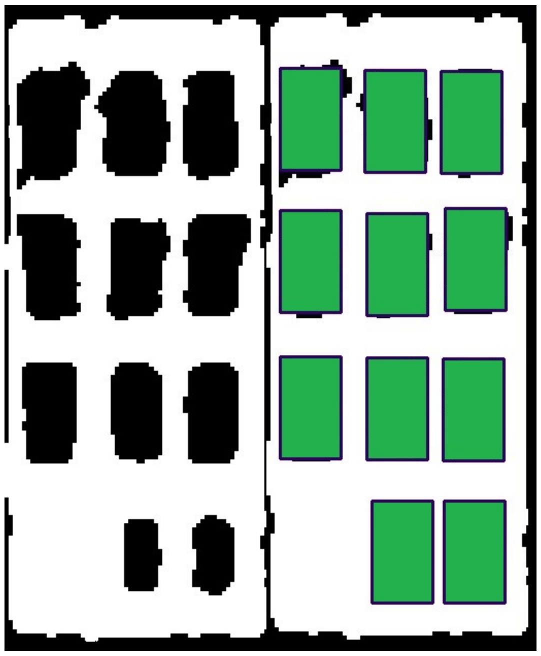


Figure 61: Facade 5 - bitmap before and after windows detection.

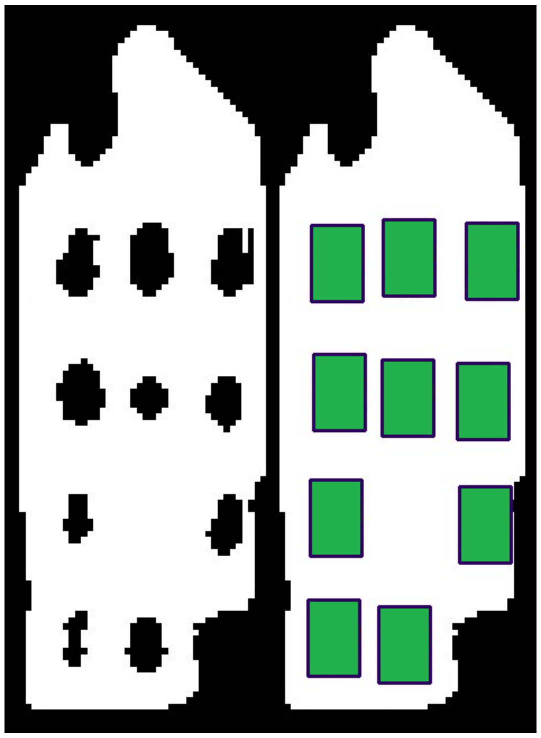


Figure 62: Facade 6 - bitmap before and after windows detection.