FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Image-Based Modeling with Polyhedral Primitives

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Stephan Zapotocky
Matrikelnummer 0426209

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Mitwirkung: Projektass.(FWF) Dr.techn. Dipl.-Mediensys.wiss. Przemyslaw Musialski

Wien, 3.12.2013

(Unterschrift Verfasser) (Unterschrift Betreuung)

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Image-Based Modeling with Polyhedral Primitives

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Stephan Zapotocky
Registration Number 0426209

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Associate Prof. Dipl.-Ing. Dipl.-Ing. Dr.techn. Michael Wimmer
Assistance: Projektass.(FWF) Dr.techn. Dipl.-Mediensys.wiss. Przemyslaw Musialski

Vienna, 3.12.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Stephan Zapotocky
Rudolf-Hochmayergasse 4, 2380 Perchtoldsdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

_____ _____

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

First, I want to thank my supervisor Przemyslaw Musialski for the valuable support and feedback during the work on this thesis. I also want to express my gratitude to my main supervisor Michael Wimmer and the entire staff at the Institute of Computer Graphics and Algorithms for offering highly interesting lectures and research opportunities for students in the computer science study programme at the University of Technology in Vienna.

This thesis is dedicated to Tini, without whose love and endless support this work would not have been possible.

Abstract

This thesis presents a method for converting a point cloud into a combination of polyhedral primitives. The process is supported by the use of digital imagery, which also forms the basis for the 3-dimensional (3D) dataset.

Modern techniques from the field of computer vision allow the reconstruction of 3D objects or entire scenes out of a series of 2-dimensional (2D) images. The different coordinates of corresponding features on 2 or more images define the position of the respective point in 3D space as well as the camera parameters of the source images. The result is a 3D point cloud, whose appearance is equal to the real scene. Post-processing of this kind of data may be necessary, but it challenges the user due to its large amount of samples and lack of structure. This thesis describes an approach for converting such point clouds into a combination of parameterized polyhedral primitives, whose appearance approximates the real object.

Part of the work on this thesis was the implementation of a tool for the reduction of point clouds to the previously mentioned combination of primitives. The fitting of those objects is automated, but the user defines the parts of the dataset to be reconstructed. The selection and displaying of the dataset's source images forms an important part in the navigation through the point cloud, accelerating the reconstruction task. The resulting set of primitives forms a strong reduction of manipulation tasks.

The main part of this thesis concentrates on the research, evaluation and implementation of an opimization technique for the fitting of parameterized polyhedral primitives to the point cloud. During the work both image-based and 3D-based optimization approaches have been studied, which form an important part of current work on this topic.

The principle is presented and evaluated with sample datasets. The results show, that the combination of automated optimization and manual user interaction leads to a simplified dataset with high accuracy.

Kurzfassung

Diese Arbeit präsentiert eine Methode zur Verarbeitung einer Punktwolke in eine Kombination aus polyhedralen Primitiven mithilfe von Bildern, die zugleich die Grundlage für den Datensatz bilden.

Moderne Verfahren aus dem Bereich der Computer Vision ermöglichen die Rekonstruktion dreidimensionaler Objekte oder ganzer Szenen aus einer Serie von zweidimensionalen Abbildungen heraus. Zwischen den einzelnen Bildern werden gemeinsame Punkte gesucht, deren unterschiedliche Positionen in den Abbildungen Aufschluss über die Koordinaten im Raum sowie über die den Bildern zugrundeliegenden Kameraparametern geben. Das Resultat ist eine dreidimensionale Punktwolke, deren räumliches Erscheinen der tatsächlichen Szene entspricht. Eine Nachbearbeitung dieses Datensatzes ist je nach Anwendung erforderlich, stellt den Benutzer allerdings vor Herausforderungen, da die Manipulation einen beträchtlichen Aufwand darstellt. Diese Diplomarbeit beschreibt ein Verfahren, um eine durch bildbasierte Methoden rekonstruierte Punktwolke in eine Kombination von parametrisierten polyhedralen Primitiven zu vereinfachen, deren Erscheinung der tatsächlichen Szene entspricht.

Parallel zu dieser Diplomarbeit entstand ein Tool für die Reduktion von Punktwolken-Datensätzen zu der oben genannten Kombination polyhedraler Primitive. Dabei ist die Anpassung einzelner Objekte in den Datensatz weitgehend automatisiert, sodass der Benutzer nur minimalen Bearbeitungsaufwand hat. Die dem Datensatz zugrundeliegenden Bilder sind wichtiger Bestandteil des Tools und erleichtern die Navigation durch die Punktwolke erheblich. Der durch die Umwandlung entstandene neue Datensatz aus Primitiven ermöglicht eine starke Reduzierung des Bearbeitungsaufwandes bei späteren Manipulationen oder Ergänzungen.

Das Hauptaugenmerk dieser Arbeit liegt auf der Recherche, Evaluierung und Implementierung eines Optimierungsalgorithmus zum Fitting von parametrisierten polyhedralen Primitiven in die Punktwolke. Im Rahmen der Forschungstätigkeit wurden dabei sowohl bildbasierte als auch 3D-basierte Optimierungen evaluiert, welche Bestandteil einiger aktueller wissenschaftlicher Arbeiten sind.

Anhand beispielhafter Datensätze wird die Funktionsweise demonstriert und die entstandenen Ergebnisse evaluiert. Es zeigt sich, dass die Kombination aus automatisierter Optimierung und manuellem Eingreifen zu einem vereinfachten Datensatz mit hoher Genauigkeit führt.

Contents

CHAPTER 1

Introduction

1.1 Introduction

The reconstruction of real-world objects or scenes plays an important role in many scientific and non-scientific areas. There are various examples for applications, where it is necessary to gather information on spatial structures and textures for a given scenario and convert it into digital data, which can be investigated easily. Archaeological scientists may want to examine an excavation site. For this purpose they have a choice of different scanning techniques to gather digital information on the object. By having a virtual reconstruction it is possible to do research on it on a variety of locations and devices, being independent from the availability of the actual object. Another example is the entertainment business, especially the movie industry. Here, a real-world scene is transformed into a digital representation, that can be manipulated and animated for many kinds of special effects. The majority of 3D scanning techniques produce a data output, that is comprised of a point cloud. This type of information consists of a large number of point measurements distributed in 3D space according to their position in the real-world scene. The spatial structure of an object is therefore represented by several samples spread across its surface. Fig. 1.1 provides an example for such a dataset. Depending on the resolution of the scanning method the number of points varies from a few hundred to several hundreds of thousands or even millions. An object is therefore not represented by a fixed set of parameters, e.g. position, rotation and dimensions, but by an absolute description of its surface using sampled points. Given a high resolution, this type of representation can store fine details of complex scenes.

1.2 Problem Statement

A drawback of point clouds is the great challenge of manipulating such data. In most cases the points are simply stored in the list, so that there is no information on the relation between individual samples. In this case it is difficult to make changes to the data, since every sample

Figure 1.1: A photo of the St.George dataset [STG] and the corresponding 3D point cloud viewed from the same angle.

must be edited separately. There are a number of tools for editing whole regions of a point cloud, but even then the spatial relation between individual points is missing. Another issue is the consumption of memory space, since every point needs to be stored separately in the file. A representation by a 3D point cloud is ideal for finely detailed objects and surfaces without regularity. On the other hand many scenes are comprised of a series of man-made objects, which often consist of a combination of regular simple objects. A good example would be an architectural scene, where the buildings usually have flat surfaces and regular shapes. Having a large number of samples on a flat surface would induce redundancy, and only storing corner points would be enough without losing spatial information. For these cases a different representation would fit better. The approach presented in this paper transforms a point cloud into a combination of polyhedral primitives and supports cuboids, cylinders, cones and spheres. In a theoretical sense, the latter three are not polyhedrons, since they are solids of revolution. However, their representation in the system is parameterized and the rendering is done by using a finite set of faces for the objects.

1.3 Contribution

This thesis presents a method for an approximation of a 3D point cloud dataset with a combination of simple polyhedral primitives. In the context of this work, the relation to images has 2 meanings. First, the dataset, that serves as the input for the presented method, is created out of a set of digital images. Second, the use of the same photos within the tool provides an intuitive navigation throughout the whole modeling process.

1.4 Structure

This thesis is organized in 3 main parts. The first chapter gives a state of the art report concentrating on approaches for point cloud processing and datasets created out of digital imagery.

The second chapter introduces the principles and algorithms used in this work and describes the methodology for fitting polyhedral primitives to a given point cloud. Additionally, technical details on the implementation are presented. The last chapter gives some results from the presented work and discusses the quality and performance of the algorithm. At the end of the thesis a summary and conclusion of the entire work is given.

CHAPTER 2

State of the Art

There exists a large number of work on the creation and processing of 3D datasets from real world objects. This chapter gives an overview on recent scientific activity in this research field. The summary will include both image-based 2D techniques, as well as approaches realized directly on the 3D dataset.

2.1 Urban Reconstruction

A very active topic is the reconstruction of urban areas. The work of Musialski et al. [MWA+12] provides a survey of techniques for this purpose. This summary concentrates on the parts of the paper associated with the scope of this thesis.

The input for reconstruction tasks can be sourced from a variety of methods. The authors of the paper focus on 2 kinds: digital imagery and Light Detection And Ranging (LiDAR). Using images has a lot of benefits; they can be stored and manipulated without great expenditure, and the availability of data has grown significantly since the introduction of various online databases and other web-based image services. 3D data has to be calculated out of a set of images, whereas LiDAR is a technique that directly produces a 3D point cloud. The system involves a scanner, that is usually carried onto a plane to produce aerial scans of regions. [VKH06]

The authors point out 3 main factors, that challenge the conception and development of urban reconstruction techniques. First is the grade of automation. Urban scenarios usually consist of a large number of different objects, and their manual reconstruction would cost a lot of time and effort. Depending on the underlying data, available computing power and desired accuracy, these tasks can be automated to a certain extent. The challenge is, that the context and structure of data is often not present or can only be gathered by human perception. The second factor concerns quality and scalability. The first is directly connected to the challenge of automation, which is a crucial element for measuring the quality of an output. It depends on the application

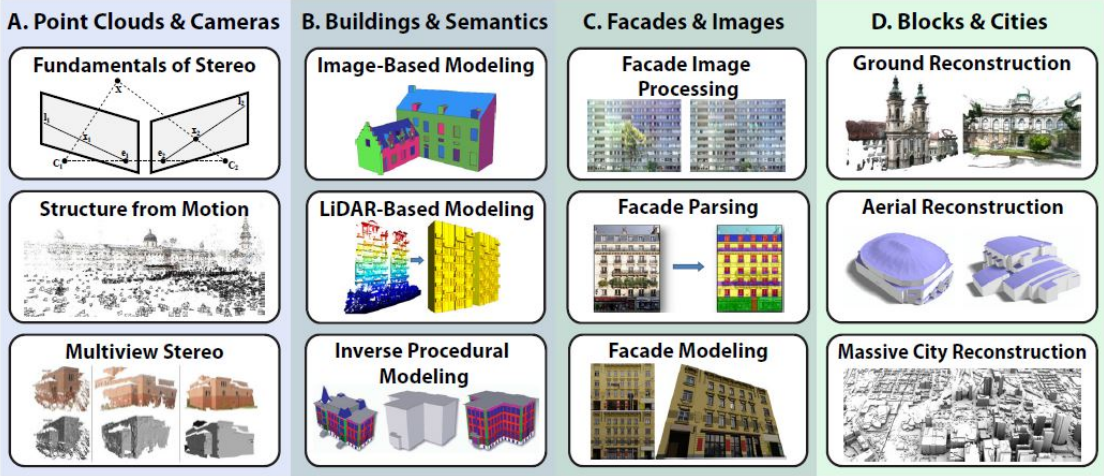


Figure 2.1: Various approaches for urban reconstruction. Image taken from the paper of Musialski et al. [MWA+12].

area, if the level of detail and accuracy of the reconstructed scene should be very high or if certain flaws are acceptable. A scene from a current action movie may call for sophisticated quality, normally requiring manual intervention of the user in the creation process. A casual game for a mobile device may allow a lower level of detail and accuracy. As for scalability, the question of application of a certain approach for a wide range of input scenarios is tested. For example, a solution for modern block-like buildings may not work properly for urban areas with historical buildings and variable object sizes. The third callenge concerns data acquisition. Urban reconstruction brings the challenge of large-scale objects. Depending on the recording technique, this may lead to the problem of complete data. Aerial scans, e.g. using the above mentioned LiDAR system, omit the majority of details of the buildings' side walls, including only the roofs, streets and other characters of the ground. Buildings of varying width or height may occlude each other, making it difficult to obtain a complete dataset. In such cases several scans need to be gathered and merged together in a post-processing step.

The reconstruction can have different levels of detail. One differentiates between sparse, semi-dense and dense reconstructions. The first one is usually an intermediate output of reconstruction algorithms, having a dense dataset as the final output. There exists a large number on previous work on that. One example for dense reconstruction is the use of $depthmapfusion$. This approach performs a pairwise dense matching of 2 registered views and combines the computed depth maps with each other. Dense reconstruction often introduces noise in the dataset, which - depending on the later use and quality constraints - has to be eliminated or reduced in a post-processing step.

Instead of relying on a 3D dataset, that has been reconstructed in a previous automated or semi-automated process, one can constructed objects and scenes directly from the images using sketching techniques. The meaning of *image based modeling* can be interpreted in different

Figure 2.2: Example for image-based modeling. Photos are used to sketch the outlines of shapes, which form objects in a 3D representation. This method can also be used to define regions for later texturing. Image taken from the work of Debevec et al. [DTM96].

ways. In the scope of the work presented in this thesis, it means the support of images for navigation throughout the modeling process, which is done in 3D space. Another approach to this term is the direct use of the source images to construct objects and scenes. The user is presented with a photo, on which he can sketch the outlines of shapes, that describe 3D objects in the reconstructed scene. This method can also be used to define regions on the photos for texturing. Figure 2.2 illustrates this technique. For a proper reconstruction, the desired object has to be captured in its entirety, so a series of images is required. Objects like the tower in the given figure usually consist of a combination of geometric primitives, like cuboids, cylinders and pyramid-like shapes. The user creates outlines for individual parts, which are later combined to form a seamless spatial representation. Tree structures can be used to store the hierarchical composition of primitives.

Another example presented is *Façade*, a tool for interactive multiview modeling. Similar to the approach of this thesis, the reconstruction is made out of a set of parameterized polyhedral primitives (called *Blocks*), which are stored in a hierarchical structure. This is depicted in Figure 2.3. In the scope of architecture, these elements have a number of benefits. First, their appearance is well suited for reconstructing man-made buildings. Second, the manipulation of geometric primitives can be done in an intuitive way, due to their simple shape. Third, the generation of a surface model is straight forward. Finally, the modeling task's complexity is reduced by the usage of parameterized primitives. The reconstruction process is done by selecting a set of photographies and marking corresponding edges. Using epipolar geometry one can transform these marked elements into a reconstructed 3D model, which can be fitted using non-linear optimization solvers. This method forms the basis for later work in this field.

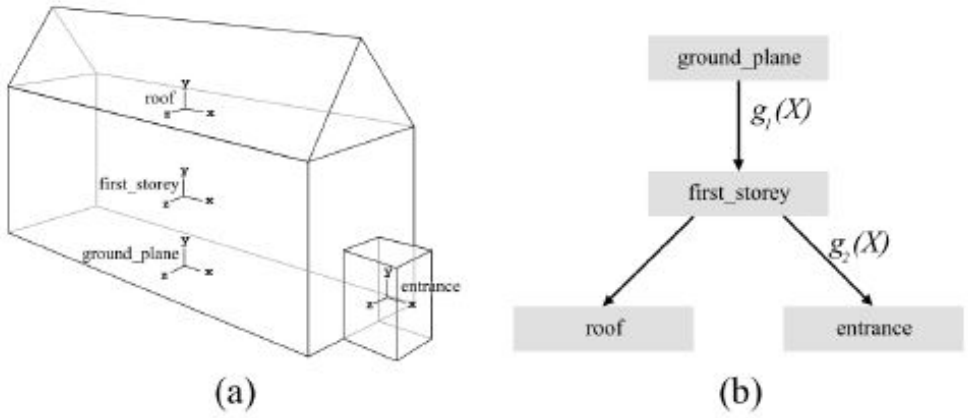


Figure 2.3: Façade modeling. The scene is made up of a series of polyhedral primitives (a) and stored in a hierarchical structure (b). Image taken from the work of Debevec et al. [DTM96].

Figure 2.4: Image-based modeling for videos. Objects like the bollard can be reconstructed and replicated, as depicted in the left image. The right one illustrates replication of a whole group of objects. Image taken from the work of Hengel et al. [vdHDT+07].

One of these examples is the usage of this technique for video material. The shape of certain elements in the scene can be modelled and transformed into replicable objects. Figure 2.4 gives an illustration of a sample scene. Bollards on the street can be modeled and replicated to form a new scenery.

Modeling and texturing is usually performed in 2 separate steps. One can also combine these tasks and create a sketch-based modeling process, that creates both shape and texture at the same time. The user marks corner points that define shapes, e.g. roofs, wall segments or windows, which are reconstructed to a textured 3D object. Figure 2.5 depicts the task of modeling a house by means of this technique.

So far these approaches are based on manual user interaction. Specific features of architectural scenes, like parallel or othogonal edges, are suitable for a variety of image-based and photogrammetric techniques. Statistics can also be the source for different techniques. The authors of the

Figure 2.5: The combined approach for modeling and texturing in 1 step. The user defines shapes according to the differently oriented segments in the image, which are transformed into textured surfaces. Image taken from the work of Sinha et al. [SSS+08].

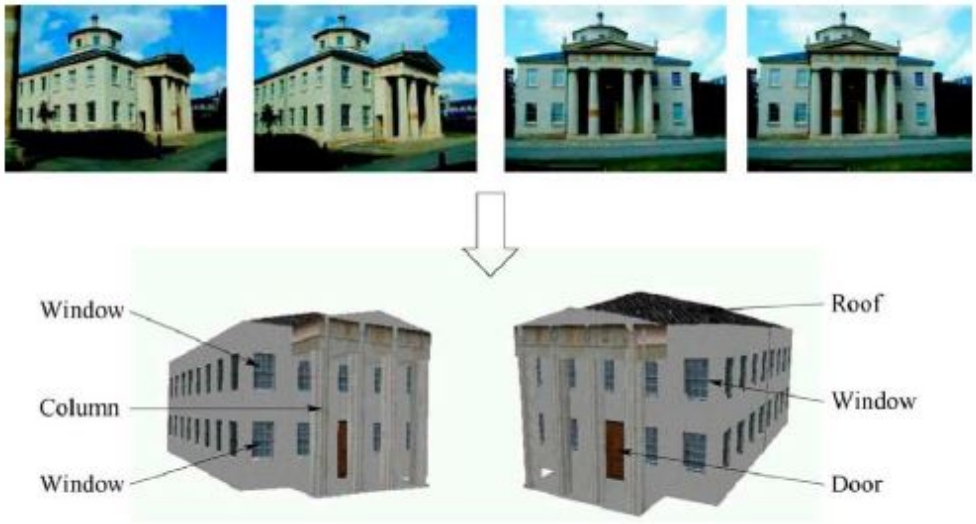


Figure 2.6: Fully automatic modeling example. A series of images from an architectural scenery are taken as the source for the automatic modeling process. The output is a labeled 3D model. Image taken from the work of Dick et al. [DTC04].

paper present an algorithm, that takes the probability of the presence of objects into account. Elements like doors, windows or other structures are assigned to a certain probabilistic distribution. After the basic wall element has been created through the usage of traning data, the elements can be added manually to build the final object. The procedure may output a series of solutions, from which the user can choose the optimal one. Figure 2.6 illustrates a sample reconstruction. Several images form the source for the automatic modeling process, generating a labeled 3D model.

In contrast to the usage of 2 or more images for reconstruction, approaches based on single views exist as well. Research work has been done on tools for creating objects out of a single 2D view by adding a simulated depth giving the impression of a 3D scene. There also exist automatic

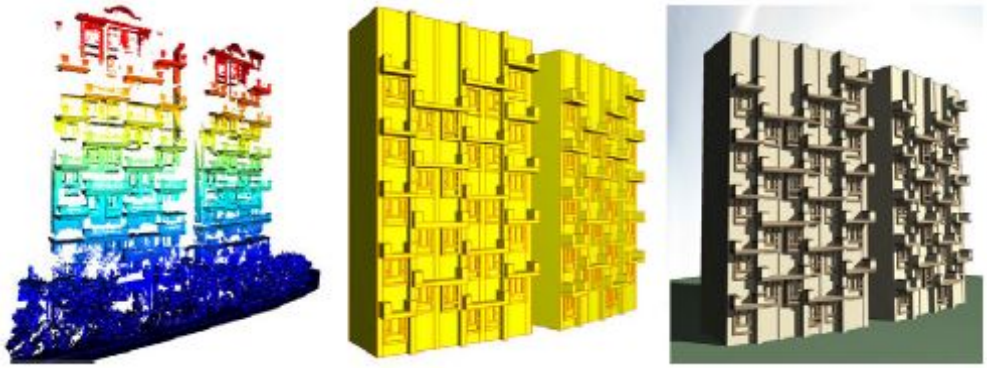


Figure 2.7: Incomplete and noisy LiDAR data (left) can be transformed into a set of geometric primitives (middle), in this case boxes. Texturing can finally be applied to the resulting model (right). Image taken from the work of Nan et al. [NSZ+10].

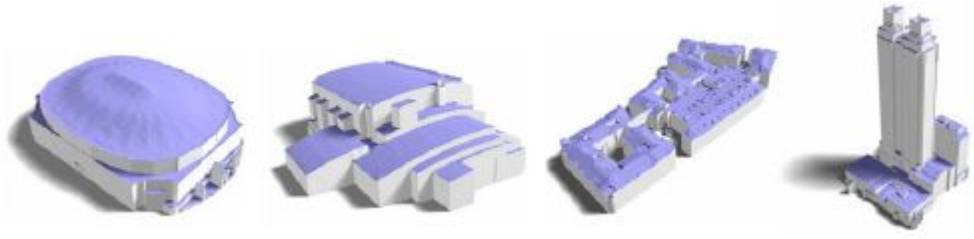


Figure 2.8: Automatic reconstruction method using Li- DAR segmentation. Image taken from the work of Zhou et al. [ZN10].

approaches, which find vanishing points to segment the image and add depth assignments.

Musialski et al. [MWA+12] point out, that fully automatic approaches still do not produce satisfying results. The intervention of manual user work is crucial for a certain level of quality, which is the case with the approach presented in this thesis.

The second part of the survey concentrates on the processing of point clouds created from LiDAR scans. These datasets can contain noise or holes due to incomplete scanning. By replacing the point cloud with box-shaped elements, the data can be transformed into a complete model. Figure 2.7 illustrates the process. It is possible to implement optimization techniques, that aid the user during the modeling process. For example, a snapping feature can be implemented, that aligns the fitting primitives according to features in the point cloud.

Instead of just fitting primitives to a point cloud, one can also take the relationships between individual samples of the dataset into account. One approach is presented, that takes aerial LiDAR data as an input and tracks the boundaries of arbitrarily shaped building roofs to form a model. Figure 2.8 illustrates this solution. The resulting models are either 3D or just 2.5D.

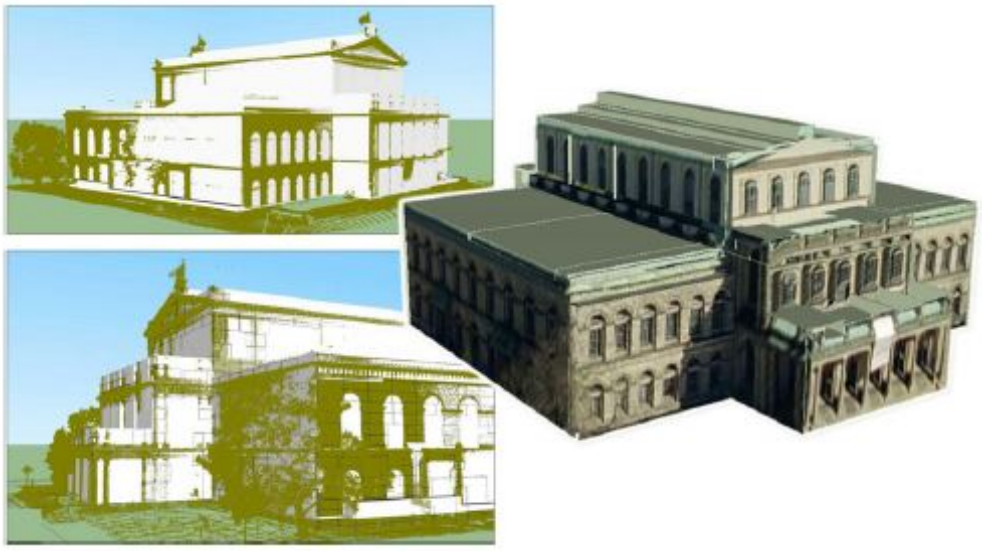


Figure 2.9: Automatic reconstruction of a building dataset using volumetric models (left). Using photographies a texturing can be applied to the resulting object (right). Image taken from the work of Vanegas et al. [VAB12].

More recent work involving LiDAR data is presented by Poullis et al. [PY11]. Their solution is able to compute photorealistic models of large-scale areas.

A recent approach uses the assumption of Manhattan World building geometry, which has proven to be a reliable principle in the field of urban reconstruction. The dataset is clustered, and relationships between the segments are determined. This information helps to determine the shape of individual parts of the object. If photographies are available, they can be used for final texturing of the object.

The survey includes several other techniques and approaches, such as inverse procedural modeling and various techniques for façade reconstruction. Since the presentation of all these solutions will exceed the scope of the paper, they are ommitted here.

2.2 Reconstruction of 3D Models from Images and Point Clouds with Shape Primitives

An image-based approach as well as a point cloud-based approach are part of the dissertation of Irene Reisner-Kollmann [RK13]. These 2 techniques are summarized in the following subchapters.

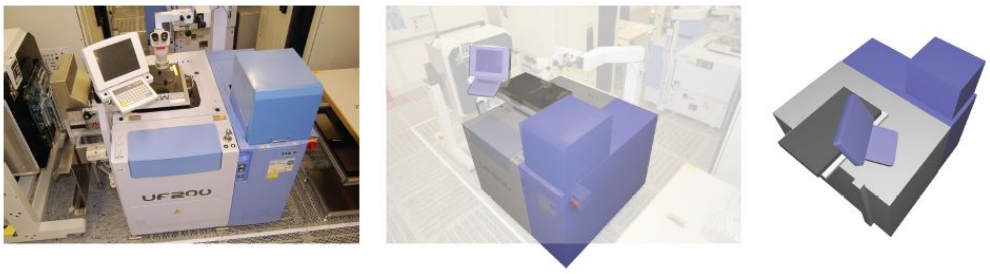


Figure 2.10: Image-based reconstruction of industrial environments. A source image (left) is used to create a reconstruction, which is overlaid by a second picture (middle). The perspective view of the model can finally by changed arbitrarily (right). Image taken from the dissertation of Reisner-Kollmann [RK13].

Image-based Modeling of Industrial Environments

It combines the definition of object outlines with the modeling of 3D objects for the reconstruction of industrial objects, such as machines and other technical devices. Figure 2.10 illustrates an example. The object is reconstructed from a 2D view. Different images are used to add missing parts to the reconstruction.

The procedure consists of 3 main parts: camera orientation, image segmentation and image-based modeling. In order to design and implement such a solution, 8 requirements are defined at the beginning, which are presented in the following listing.

- simple workflow
- wide camera baseline
- reflective surfaces may occur on the objects
- textureless objects should also be supported
- support of arbitrary lighting conditions
- fast and accurate camera orientation
- models comprised of simple geometry
- intuitive user interface

The first step is the camera calibration and orientation. This is important for all later reconstruction steps. The focus must lie on accuracy, since minimal errors lead to consecutive divergency from the correct result. In order to achieve the appropriate orientation, pre-calibrated cameras are used. The calibration is realized by using coded markers illustrated in Figure 2.11. This way,

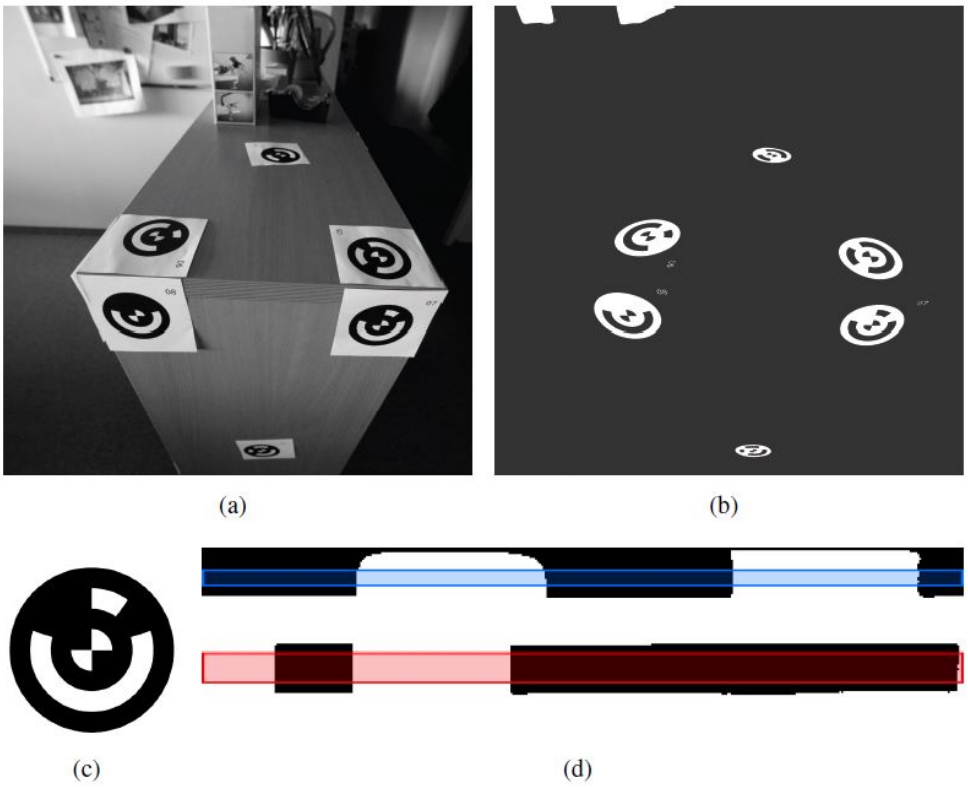


Figure 2.11: Camera calibration with coded markers. The different patterns allow the differentiation between them. The markers are placed onto the object (a) and extracted from the scan (b). Each marker region (c) is unwrapped in order to obtain its code (d). Image taken from the dissertation of Reisner-Kollmann [RK13].

a planar calibration pattern can be localized in the images. The intrinstic parameters together with radial and tangential distortion parameters can now be computed with the correspondences between 2D positions in the image and 3D projected points. The extrinsic camera parameters - position and orientation - are calculated in the next step. The coded markers are tracked across multiple images, and their corresponding positions are used to determine the missing camera parameters.

For a given pair of images, the relative position of the points is computed using the *5-point-algorithm*. After this has been done, all other images are added, and the reconstruction of corresponding points is done by the *3-point-algorithm*. The output is a camera network that has to be optimized using bundle adjustment. The final step is the application of the *Levenberg-Marquardt algorithm* to minimize the distance between the reprojected triangulated points and their corresponding position in the images.

The next step is model fitting, which uses predefined parameters for various shapes. The basic

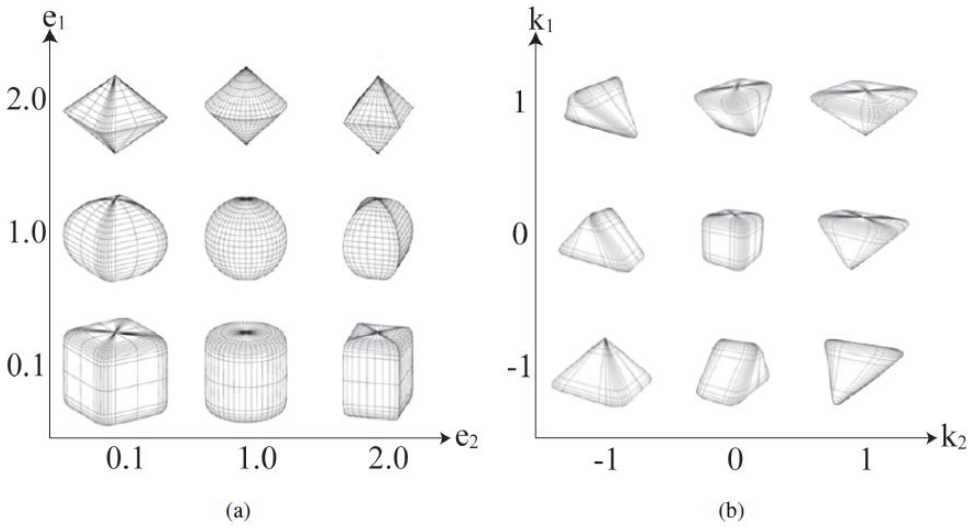


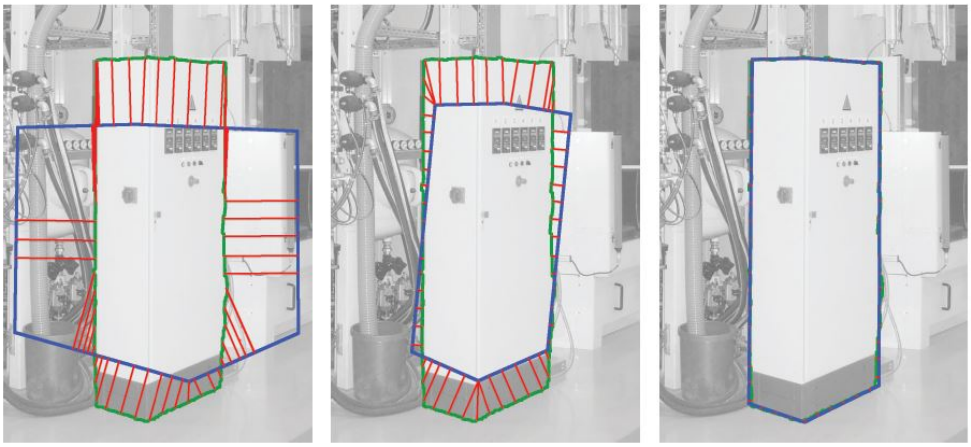
Figure 2.12: A listing of superellipsoids. Depending on the selected parameters, various shapes are possible, including spheres, cubes and cylinders. The factors e_1 and e_2 describe the shape of the superellipsoid and are added to the usual parameters position, scale and rotation. Image taken from the paper of Reisner.

primitives box, cube, sphere, cylinder, frustrum and pyramid are supported as well as extended objects made from superellipsoids, which are depicted in Figure 2.12. The latter are parameterized objects that can represent a variety of different shapes, as can be seen in the Figure. Complex objects can be reconstructed by using a combination of primitives. Position, scale and rotation of the objects is described by a number of parameters, which differ from the type of object. A cube has a position, a 3D scale vector, as well as a rotation around 3 axes. A cylinder can also be rotated in that way, but needs only a 2D scale vector for radius and height. On the other hand, a sphere is fully defined with a position and radius; rotation is irrelevant.

The next part of the algorithm deals with image constraints. In order to achieve a reconstruction, the 2D images are used to find the objects' edges and regions. They can be manually drawn or created with intelligent tools based on the image's properties. Figure 2.13 depicts this process. The result is a segmentation of the image into its individual parts. It defines the outlines of the reconstructed objects. In order to get a fitting, the segmentation has to be done on multiple images. The more constraints are set, the more accurate and complete the final model will get.

In addition to the constraints created in the image, the user can define relational constraints. They define dependencies between the individual objects, such as distance, up vector or a common ground plane.

When all constraints are defined, the optimization step is performed, which performns the Levenberg-Marquardt algorithm on the data. This is a non-linear optimization approach, that tries to find a local minimum. Therefore, a sufficiently fitting initial model has to be found in



(a) (b) (c)

Figure 2.13: Optimizing outline constraint by defining edges and regions.. The optimization process minimizes the distance between model and user-defined shape outline (a,b) until the difference is minimized (c). Image taken from the dissertation of Reisner-Kollmann [RK13].

order for the algorithm to converge to the correct result. The position can be estimated by triangulating the center of all segments. The scale can be initialized from the median distance from the region center to the outer contour of all segments in an image. This distance is projected back, and the average value is taken as the uniform scale value for the initial model.

The author of the dissertation also describes the support of tubular structures, whose possibilities for segmentation definition are depicted in Figure 2.14. The final representation consists of a 3D spline together with a diameter along the object. Segmentation can be done in 3 different ways, always producing a 2D curve and a value indicating the width. The first possibility is to define a user-drawn curve along the object and a line specifying the tube's width. The second way is to define the region, whose boundaries match the outlines of the tube. The skeleton is then calculated with the use of this information. The third method is to define the 2 edges left and right to the tube or to a segment of the tube. The central line is computed and smoothed.

The results can be seen in Figure 2.15. In this example, 9 photos have been taken from the machine, and 19 primitives have been reconstructed to make up the entire object. Image segmentation and other constraints led to the optimization of 376 parameters and 21075 measurements, which gives an idea of the scale of such a challenge.

In the conclusion the author points out the applicability of this approach for a variety of scenarios consisting of man-made objects. However, the drawback of the manual segmentation is highlighted. This problem can be eliminated with the introduction of automatic segmentation techniques in some cases, but this is highly dependent on the nature of the images.

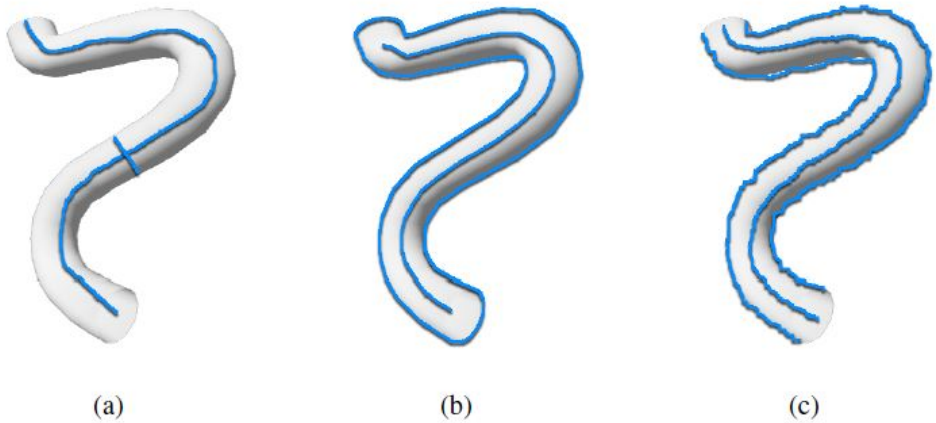


Figure 2.14: Tube segmentation. It can be realized by using user-drawn curves (a), regions with skeletons (b) or 2 edges in combination with an intermediate curve (c). Image taken from the dissertation of Reisner-Kollmann [RK13].

Automatic Building Reconstruction from Point Clouds

The last subchapter dealt with data solely coming from 2D images. In contrast to that, this presented technique additionally relies on the availability of a 3D unstructured point cloud. The benefit from the availability of images comes from the fact, that they store valuable information on edges, which may not be extractable from the point set. The author points out that this is a fully automatic approach with possibility of manual intervention at most stages of the process. The reconstruction pipeline is depicted in Figure 2.16. The first step is the segmentation of the point cloud. The data is separated into planar clusters. The second part of the pipeline is dependent on the availability of images. If only the point cloud is at hand, a polygonial surface is reconstructed using a rasterization procedure. If imagery is available, the reconstruction is done by extracted multi-view information from the pictures. Finally, the polygons are combined to a single model, that can be textured in the last step.

The segmentation procedure separates the point set into axis clusters. All points belonging to the surface normal of a specific axis are assigned to it and marked. If a point is unclustered, it is evaluated until a cluster can be found. This is the case with points in non-planar areas or other outliers. The result of this step is a set of clusters corresponding to the faces of the 3D object, as depicted in Figure 2.17. The axis fitting follows the common angles of 45°, 60° or 90°.

After the segmentation has be done, multiple 3D planes are fitted into the clusters. In order to find the correct positions of the planes, the clusters are projected into 1D space, as depicted in Figure 2.18. The projection creates a histogram indicating the number of points along the cluster axis. It can be observed, that the positions of the planes correspond to the peaks in the histogram. Unclustered points left from the previous step are fed into the next iteration of the clustering procedure with different axis orientations.

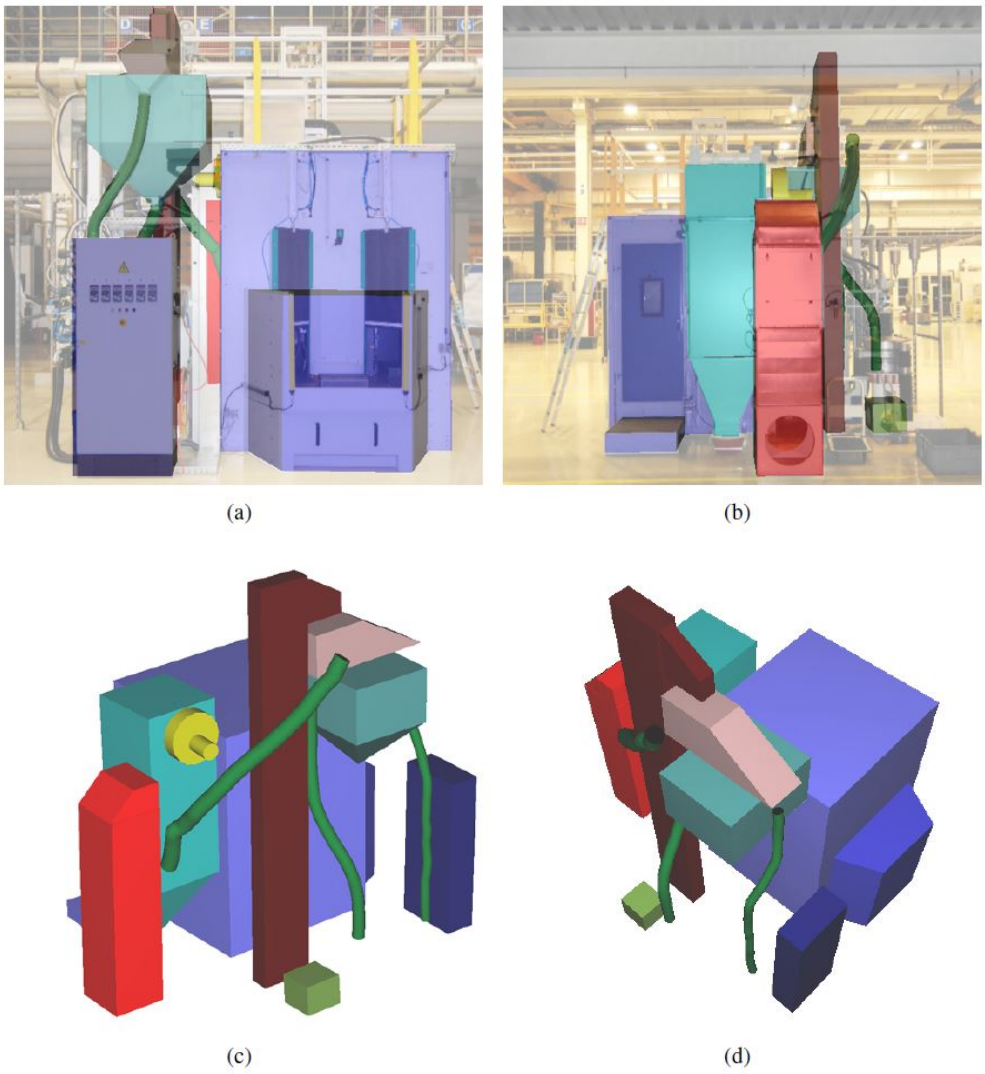


Figure 2.15: An reconstruction example of a machine based on 9 images and 19 defined regions. Image taken from the dissertation of Reisner-Kollmann [RK13].

The next step is the polygonization of the previously calculated planes. First, a point rasterization is performed. The point clusters are projected onto their fitting plane, leading to a 2D binary image. Small holes in the resulting map can be filled by the means of morphological closing techniques, as it is illustrated in Figure 2.19.

If images are available, the surface reconstruction can be started. For each cluster an appropriate image is selected automatically. The best choice is a photo, where the camera plane is parallel to the one of the respective cluster. The picture is then projected onto the plane to determine its correct boundaries.

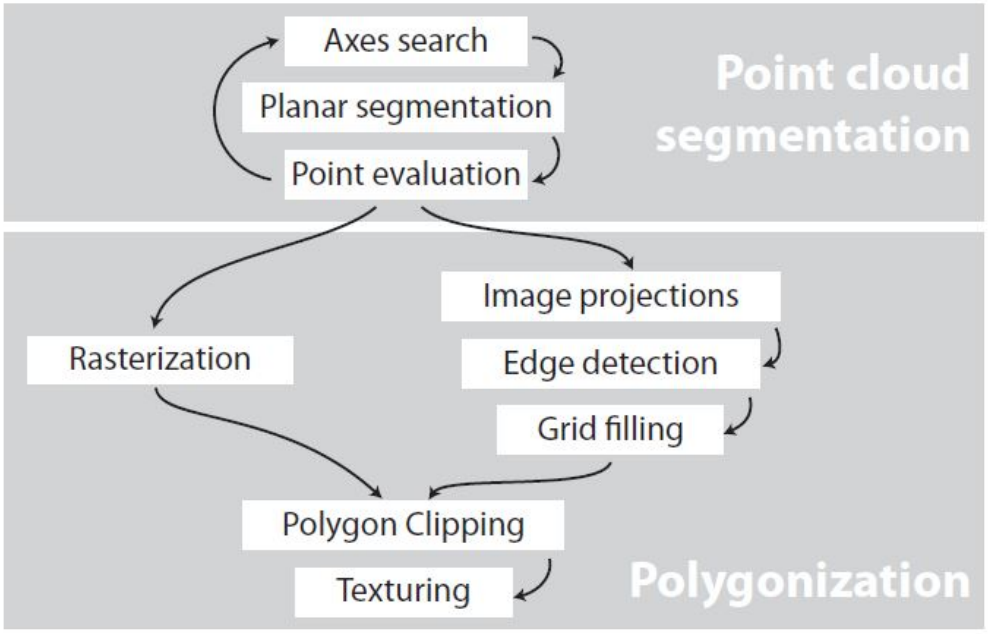


Figure 2.16: The reconstruction pipeline consisting of the segmentation and polygonization. There are 2 possible ways for the second step, depending on the availability of images in addition to the point set. Image taken from the dissertation of Reisner-Kollmann [RK13].

Edge detection is then performed, which projects all pictures onto their respective planes and creates gradient images. The positions of edges correspond to gradient changes of a certain extent. Extracted collinear edges are then merged to bigger segments in a grid filling process, illustrated in Figure 2.20. The grid is finalized by extending the merged line segments until they intersect with another line.

Now, the grid cells forming the model have to be determined. The point density of a cell is the ratio between the number of inlying points and its area, that has been normalized by all cell's median density. If this density is above a certain threshold, the cell is part of the model. In this case, the cell's boundaries are transformed back to 3D space to create the respective polygon.

The model's appearance can now be optimized by cutting intersecting polygons, which smoothens the surface. The final step is the texturing, which can be realized by taking the previously chosen projected images and stitch them together. The result can be seen in Figure 2.21.

This approach is best suited for scenes with buildings consisting of a large number of parallel faces. An interesting aspect is the usage of optional imagery for both optimization and texturing.

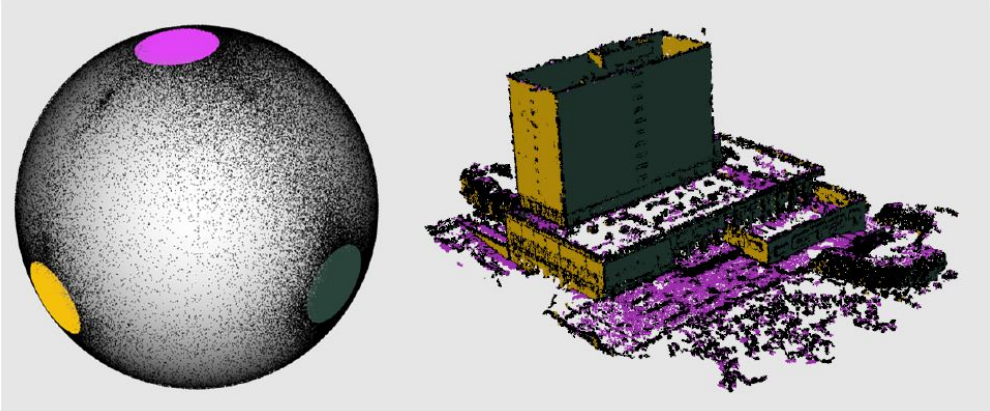


Figure 2.17: Segmentation of the point cloud. The sphere on the left illustrates the 3 sets of surface normals belonging to a specific cluster. The point cloud on the right is separated into its respective clusters. Image taken from the dissertation of Reisner-Kollmann [RK13].

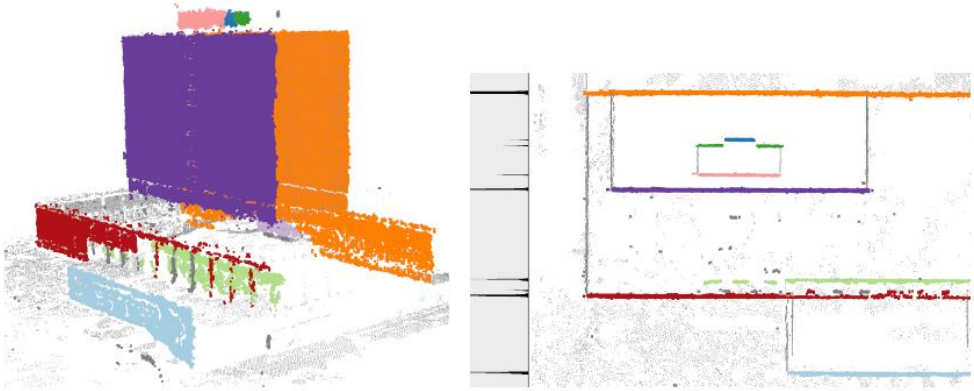


Figure 2.18: Plane detection for the clusters visible in (a). The 1D projection in (b) creates a histogram, whose peaks correspond to the positions of the planes. Image taken from the dissertation of Reisner-Kollmann [RK13].

2.3 Point Cloud Processing with Primitive Shapes

The shape of of man-made objects mainly consists of a combination of geometric primitives, which is especially the case for architectural scenarios. The use of primitive shapes to describe the appearance of a point cloud also forms the basis for the solution presented in this thesis. A comprehensive research on this approach is given in the dissertation of Schnabel [Sch10]. In contrast to the approach presented in the dissertation of Reisner-Kollmann [RK13], this solution takes only the point cloud as an input. The contribution of Schnabel's work is an algorithm for

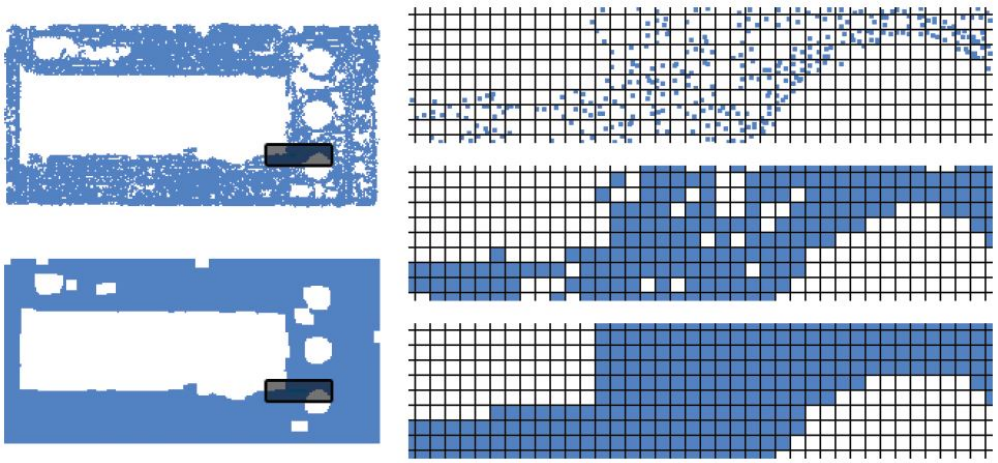


Figure 2.19: Cluster closing. The points are projected to their 2D fitting plane and rasterized. Small holes can be closed morphologically. Image taken from the dissertation of Reisner-Kollmann [RK13].

Figure 2.20: Merging of nearly collinear line pieces (left) to bigger segments (middle). The grid is constructed by extending each line until it intersects with another one. Image taken from the dissertation of Reisner-Kollmann [RK13].

detecting and inserting shape primitives within the point cloud and replacing the corresponding regions, thus providing a compression of the data. Additionally, it presents a means for completing insufficiently sampled parts by synthesizing them with shapes.

The dissertation consists of 4 main contributions for processing point clouds. First, an algorithm is presented, that extracts geometric primitives out of point clouds generated from real-world objects. The second one deals with an approach for point cloud compression by replacing parts of it with fitted primitive shapes. Additionally, a feature-detection solution is presented, that stores fitted primitives into a graph structure. This way, features can be described and and extracted from the dataset. Finally, a solution is presented, that uses the detected primitives to build a complete mesh with the capability of filling holes by extending individual primitives. The final output is a closed mesh, that approximates the point cloud as much as possible.

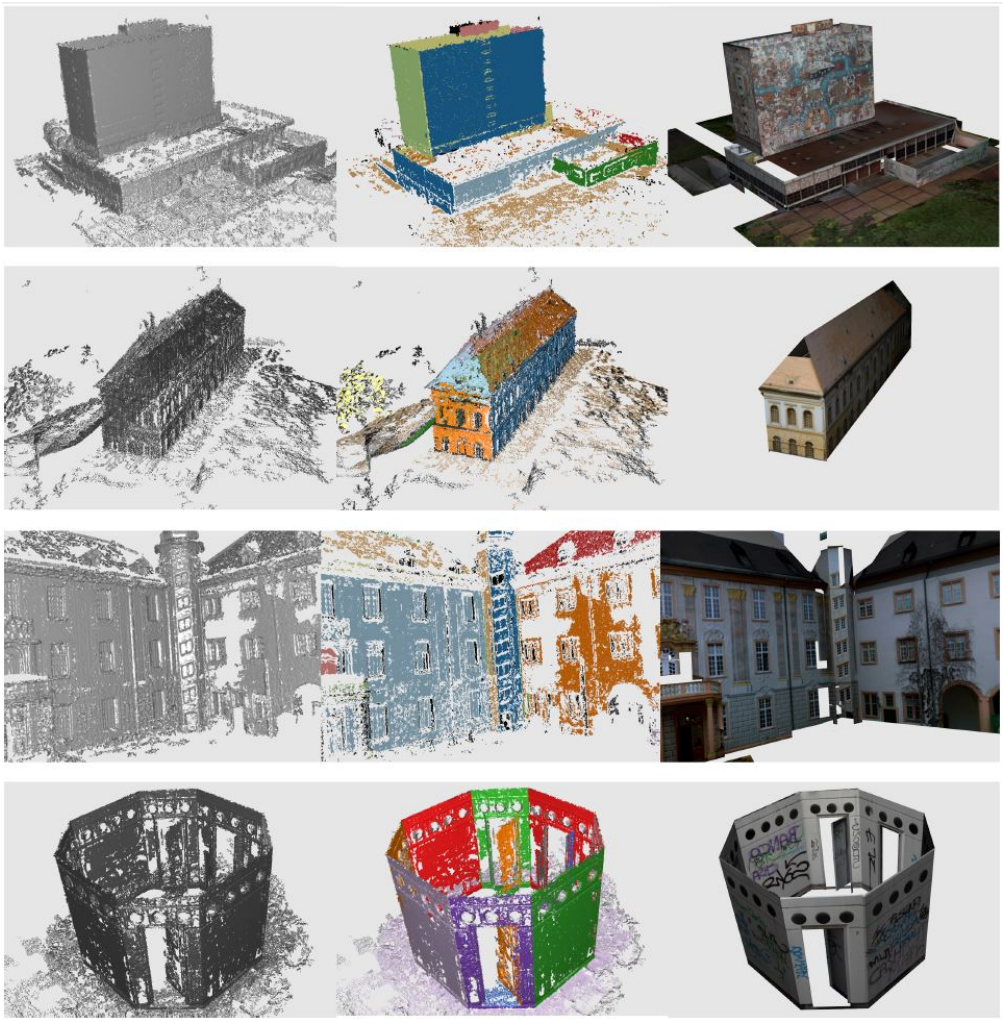


Figure 2.21: Results from a few datasets. The images on the left show the point sets, the clusterings are depicted in the middle, and the final textured images are shown on the right. Image taken from the dissertation of Reisner-Kollmann [RK13].

The summary of the whole work of Schnabel would exceed the limits of this thesis. Hence, only the first chapter will presented below, since it is most relevant for the scope of this work.

Real-world scanned data forms the basis of the work presented in the dissertation. The author points out, that there are 2 main challenges arising from this fact. First, datasets of this type are often of large scale. Second, the amount of noise in the point cloud, that comes from measuring errors, is a challenge for a possible solution. It demands the solution to be robust.

Schnabel's solution is able to detect planes, cylinders, spheres, cones and tori. The fitting algorithm uses the Random Sample Consesus (RANSAC) method to fit hypothetical primitive

models into the point cloud and test them for validity. Details on RANSAC are summarized in Section 3.3. The basic idea of this approach in the scope of primitive fitting is to find parameters for a hypothetical model, that fits into the point cloud in such a way, that the sum of distances from the samples to the surface of the primitive model is minimal. The algorithm generates and evaluates a predefined number of randomly generated parameterized shapes and selects the one with the best result.

The original object represented by a dataset usually consists of a combination of different shapes. Hence, the data for a single RANSAC run is reduced to a smaller size. It is therefore necessary to implement a preliminary phase, that selects suitable regions for further processing. Schnabel therefore introduces a 2-phase approach as a solution. The algorithm begins with the analysis of the dataset with hierarchically structured sampling for generating candidates for the primitives. The algorithm starts with small subsets and tries to fit primitives to it. The hypothetical model is then evaluated. If it has been proven to be valid, subsequent larger subsets are taken as an input for further fitting and evaluation.

The first step is the estimation of shapes. The input is a point cloud, that stores positions and normals of individual samples. Depending on the type of geometric primitive, a different number of samples is needed for a full definition. The following listing summarizes the parameters.

- Plane: 3 points p_1, p_2, p_3 fully define a surface.
- Sphere: 2 points p_1, p_2, that form the diameter. The sphere's center c is the halfway point on the connecting straight line.
- Cylinder: 2 points p_1, p_2 together with their normals n_1, n_2. The cylinder's axis is defined with the cross product $a = n_1 \times n_2$. The 2 lines spanned by the points and their respective normals are projected along a onto the plane $a \cdot x = 0$. The intersection of these projections is the cylinder's center c, the radius is the distance from c to the point p on the plane.
- Cone: 3 points p_1, p_2, p_3 are used. The apex c is the intersection of the 3 planes formed by the points and their respective normals. 3 new points q_1, q_2, q_3 are then calculated by starting at the apex c and adding the normalized direction vector from c to each of the p_i. These new points span a plane, whose normal is the cone's axis. The opening angle ω is finally defined as $\omega = \frac{\sum_i \arccos{(p_i - c) \cdot a}}{3}$.
- Torus: 4 points p_1, p_2, p_3, p_4 together with their normals n_1, n_2, n_3, n_4. Candidates for the rotational axis are the up to 2 lines, that intersect the 4 lines defined by the points and their respective normals. The line, that causes the lowest error in respect to the 4 points is taken as the axis. A plane is then constructed, that is rotated around the axis. The minor radius is defined by a circle, that is defined by 3 points out of the set collected by the rotating plane. The major radius is equal to the distance of the circle center to the torus' axis.

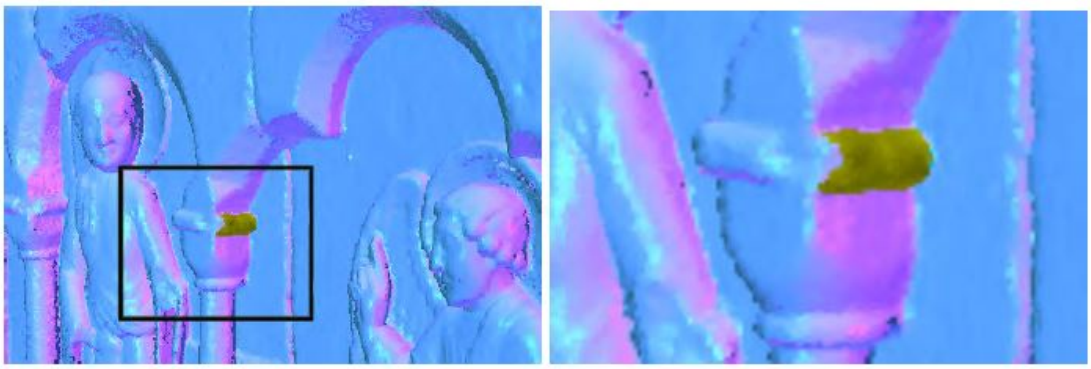


Figure 2.22: Example for a complex dataset with an integrated cylindrical shape. Due to efficient non-uniform sampling, the number of necessary candidates for shape validation is decreased by 3 orders of magnitude. Image taken from the dissertation of Schnabel [Sch10].

The complexity of the RANSAC procedure is dependant on both the number of minimal sets holding the points for the parameterized models, as well as the cost of validating the result. The probability calculations given in the dissertation come to the conclusion, that the success rate of finding suitable sample sets directly influences the runtime complexity.

A smaller distance between 2 points increases the chance of these points to be part of the same elementary primitive. It is therefore crucial to begin with a small search radius and find appropriate constraints for increasing it in the subsequent steps. Exploiting this fact, Schnabel proposes a non-uniform sampling strategy, that takes both outlier density and shape size into account for adjusting the size of the minimal sets. Spatial relationships between the samples are mirrored in an octree for increased performance.

The benefit of this approach is the fact, that far less shape candidates need to be created and validated in order to achieve the same probability. Schnabel gives an example situation presented in Figure 2.22. It also demonstrates, that his approach is also suitable for finely detailed data, which can be approximated by geometric primitives despite its complexity. In the Figure, a dataset of 2 million samples contains a cylindrical part. The corresponding region is part of a detailed facade-like structure. The non-uniform sampling approach decreases the number of necessary validation candidates by 3 orders of magnitude.

As the sampling algorithm progresses, the shape candidates' size is increased until a certain iteration limit is reached. The size of the largest feasible shape is not known a priori, so an alternative stopping condition has to be found. Schnabel proposes a solution, that analyzes a selection of additional candidates and picks the best one out. If the probability of finding another shape with the same size as the selected one is higher than the predefined threshold, the algorithm is terminated, because the chance of finding a better candidate with the same size is low.

The quality of a shape is reflected in a score, that is dependant on 3 factors. First, an ϵ-band is

formed around the shape. The set of samples lying within these borders is relevant. Second, the normals of the points are compared to the normals of the shape's curvature. The difference has to be within a certain range in order to accept the point as an inlier. Finally, if a point has passed the first 2 constraints, it is tested for connectivity. It is considered for the score only if it formes the shapes' largest connected component.

Schnabel continues to detail scoring contraints and detection and handling of special cases. Details can be found in his dissertation. The main idea of his work is the fitting of geometric primitives to a point cloud in order to approximate the dataset as good as possible. In contrast to the methodology presented in this thesis, the work of Schnabel relies solely on automatic processes without user interaction. The iterative fitting and validation of shapes with varying size leads to high complexity of the implementation, but produces good results, as can be seen in Figure 2.23. The image shows both the original dataset of a choir screen and the resulting approximation with primitive shapes. It proves, that even datasets with high details can be approximated by detecting and fitting primitives, which are small enough to capture the fine structures.

2.4 Optimization-Based Snapping for Modeling Architecture

Another means of interaction in an approximation process for point clouds is the usage of sketch-based techniques, as it is introduced in the work of Arikan et al. [ASF+13]. In contrast to the contribution in this paper, the so-called "O-Snap" approach creates individual faces instead of a readily fitted primitive.

Figure 2.24 illustrates the workflow of this approach. The basis for the reconstruction task is a point cloud, which may include incompletely sampled surfaces. The first step is the separation of the data into individual planar parts of the surface. For each subset the points are extracted, that define the approximative polygon. In order to create a seamless model, adjacent boundaries are found automatically and snapped together. If no neighborhood relation could be found, the user has the ability to sketch a polygon over an incompletely sampled surface area to finish the seamless model.

Depending on the quality of the source data the reconstruction process may oppose an increased effort to the user, even if parts of the final model can be created automatically. However, this approach has the benefit, that a broad range of 3D point clouds describing various shapes can be reconstructed in an intuitive way.

The proposed system consists of 2 main parts, that make up the complete reconstruction task. The first stage is comprised of automated processes. In this first phase the input 3D point cloud is separated into segments lying on the approximate same plane. These segments are then transformed into polygons corresponding to the plane's position, orientation and outline. The result is a combination of polygons that make up a model, which - depending on the quality of the source data - may have holes due to insufficiently sampled regions. Furthermore, the boundaries of adjacent polygons are not tied together, which leads to gaps in the model. The solution to the

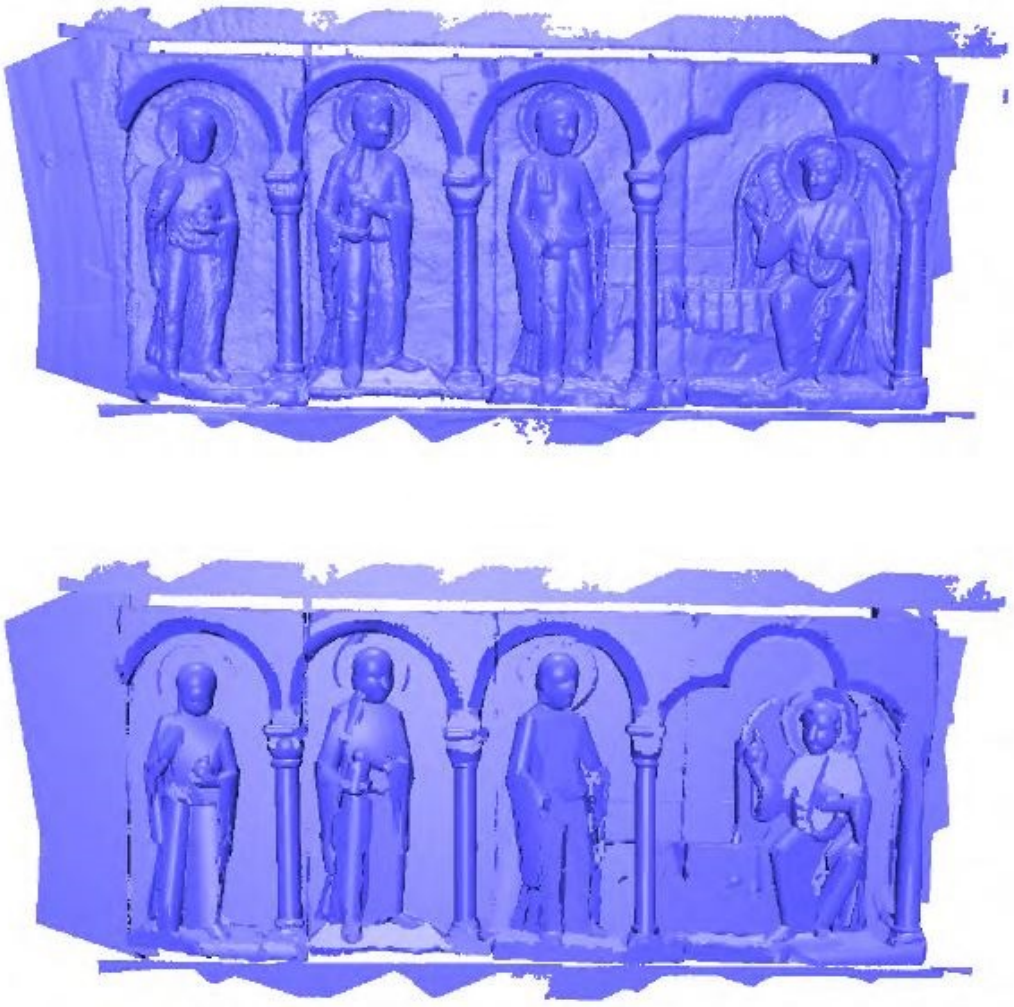


Figure 2.23: Results from the approximation algorithm. The original data of a choir screen consists of fine details (top). The approximation is able to cover the details of the figures and the columns with 372 shapes (bottom). Image taken from the dissertation of Schnabel [Sch10].

latter problem is the first main contribution of the authors. They introduce an optimization-based snapping algorithm, that searches for adjacent boundary regions and adjusts the positions of the respective corner points in order to close the gaps between the individual faces. The second main part of the reconstruction process is the treatment of larger holes in the dataset, which could not be handled in the automated stage. Arikan et al. introduce an interactive tool for sketching the outline of the incompletely sampled surface area directly onto the 3D view in order to complete

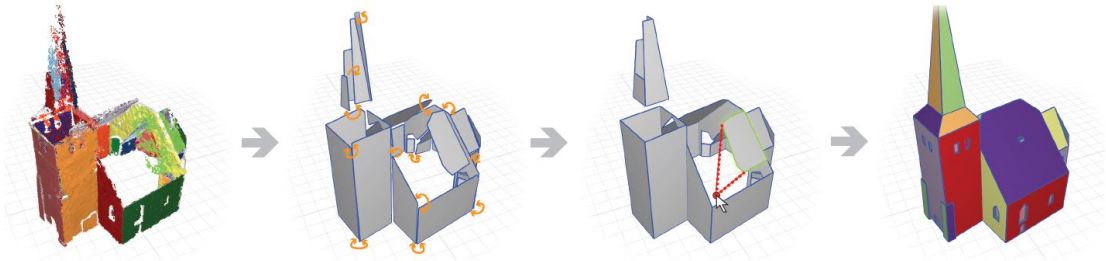


Figure 2.24: Overview of the O-Snap approach. A coarse and incomplete dataset (first image) is transformed into a combination of point cloud subsets, which represent an individual face. An automatic procedure fits polygons to the subsets in order to create a 3D face-based model (second image). Local adjacency conditions are automatically detected and polygon vertices are adjusted to connect the individual faces (third image). Incomplete polygons can be fitted to the rest of the model by using a sketch tool to draw the correct outlines of the respective face. The result is a polygonial model, that approximates the original dataset (fourth image). Image taken from the paper of Arikan et al. [ASF+13].

individual regions and snap them to neighboring parts of the reconstructed model.

Figure 2.25 illustrates the polygonalization process. A RANSAC-based procedure fits planes to the point cloud, where the original objects has a planar segment. Nearby parallel structures may be consolidated to one plane, but later treatment using manual input and k-means clustering allows the separation of segments to refine the final model.

In the first phase, local RANSAC-based approach is used to separate the initial point cloud into segments, which lie on the same approximate plane. Nearby parallel elements, such as balcony faces on a building's facade, may be consolidated into one single segment. Using a Least Median of Squares (LMS) estimator, the main structure - in the given example this would be the building's wall - can be extracted in order to fit the model onto it.

The next step is polygonalization, which converts the segments into connected components, eg. the set of front faces of similar balconies on a building. Figure 2.26 gives an overview of the conversion process. A segments consists of a set of points forming a hypothetical polygon. First, boundary points are extracted and stored in an ordered form. Using local Principal Component Analysis (PCA) the vertex normals are initialized and smoothed in accordance to neighboring vertices. Using PCA avoids the problem of having varying normals due to noisy sets. The resulting information can finally be used to create consistent edges of the segment and compute the final polygon by running a corner detection procedure.

The output of the polygonalization step is a set of unconnected polygons, a so-called polygon soup. Holes and gaps of varying size are still present in the model and need to be removed. The proposed solution is a snapping algorithm, that brings adjacent polygons together in both the automatic and the interactive stage.

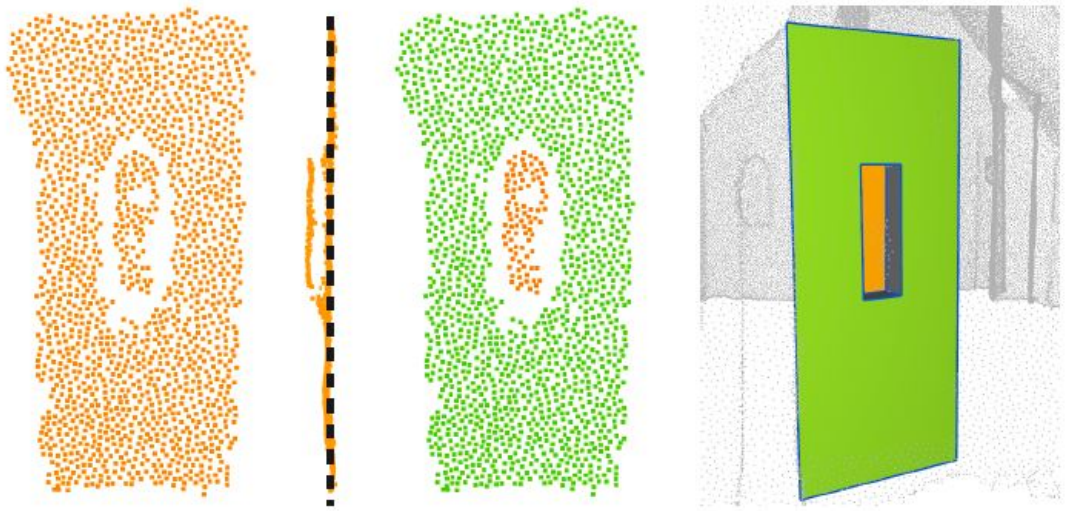


Figure 2.25: Overview of the initial reconstruction phase. The input is the point cloud , that is separated into-nearby parallel structures (left image). A plane is placed to the position and orientation of the dominant structure (second image) and transferred to a polygon (fourth image). In the interactive phase the user can do a detailed separation of nearby parallel data (third image) to refine the model. Image taken from the paper of Arikan et al. [ASF+13].

The first step of the snapping procedure is the search for adjacent polygons, performing vertex-vertex/edge/face and edge/edge matches. A valid match can be constrained by 5 mechanisms:

- Global parameter r_{max} defining the maximal gap size
- Intrinsic stability to prevent self-intersections, flip-overs or edge/diagonal collapses
- A set of matching candidates if the intrinsic stability restricts too much solutions
- Local pruning for fixing problems mostly caused by the set of matching candidates
- Global pruning to prevent the degeneration of thin polygons

For vertex-vertex matching a search radius for each model vertex is defined, that is equal to half the minimal distance between the vertex and the polygon's boundary. This prevents matching across polygon edges, causing self-intersections and flipovers, as well as polygon vertices, causing edge or diagonal collapses. If there is no vertex in the search radius of a given vertex, its closest point lying inside the distance r_{max} is added to the candidate set. A match between 2 vertices v_1 and v_2 has been found, if v_1 is included in the candidate set of v_2 and vice versa. In order for 2 points of adjacent polygons to collapse into one corner vertex, the 2 points have to be

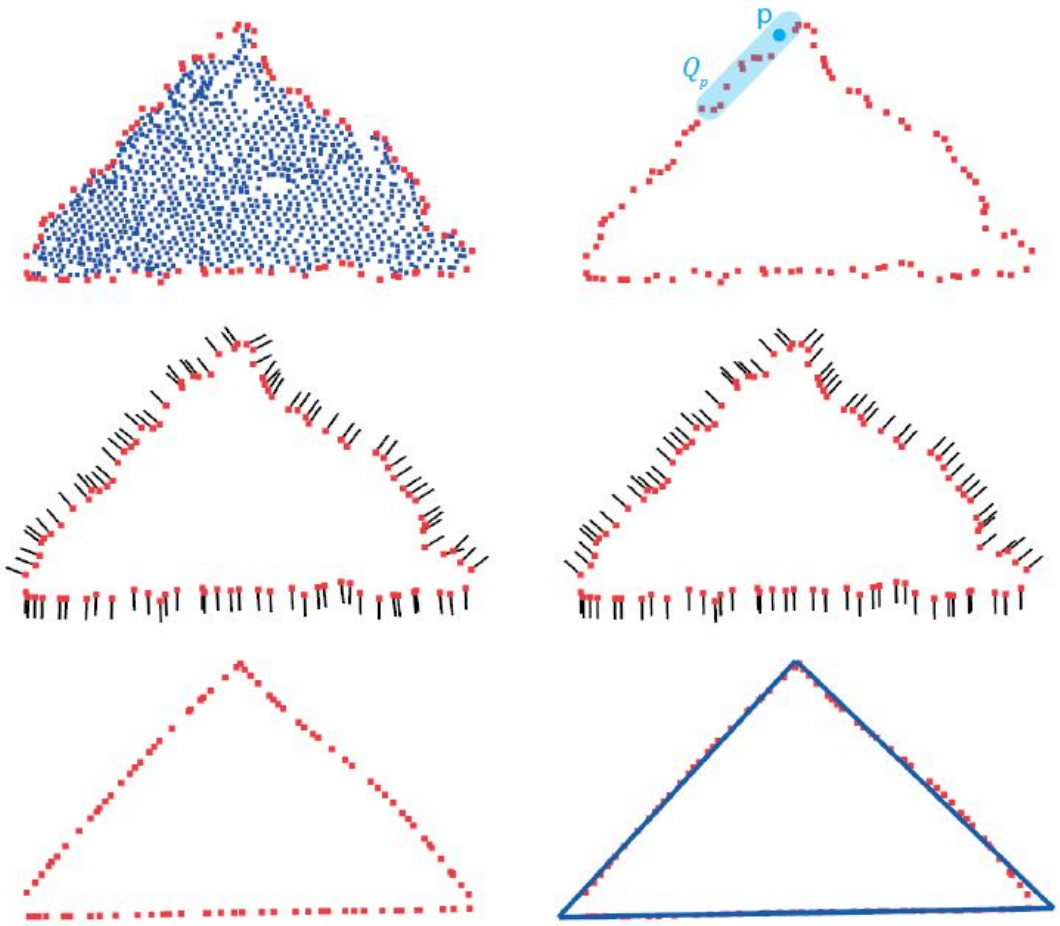


Figure 2.26: The polygonalization pipeline. Given a connected component found by the RANSAC-based procedure, an ordered set of boundary points is extracted (top left). Using PCA (top right), the points' vertex normals are initialized (middle left) and smoothed, too (middle right). This information is used to flatten the edges (bottom left) and extract a polygon by using a corner detection procedure (bottom right). Image taken from the paper of Arikan et al. [ASF+13].

within a predefined distance to the intersection line of the 2 polygons' corresponding supporting planes.

Matches between vertices and edges are found in a similar way. The search radius of an edge is defined at its 2 endpoints. A neighboring vertex is a match, if its orthogonal projection onto the extended edge lies within its radius. Analog to the vertex-vertex case, both vertex and edge have to be situated within a predefined distance to the intersection line of the supporting planes.

A vertex-face match is valid, if the vertex's orthogonal projection onto the face lies within its

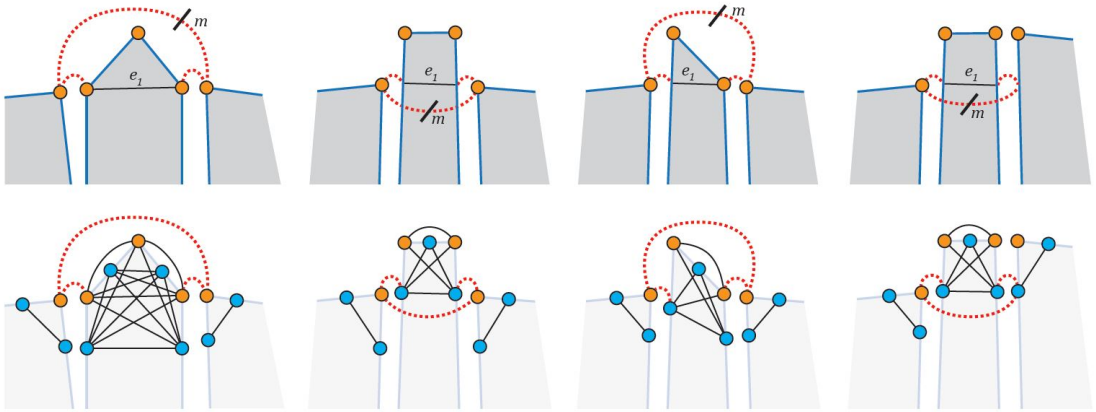


Figure 2.27: Examples (top row) and solutions (bottom row) for situation, where cyclic matches cause the central edges e_1 to collapse and therefore violate the stability constraints. Such cycles are detected and removed by the algorithm. Image taken from the paper of Arikan et al. [ASF+13].

boundaries, and if the distance to the vertex lies within the radius.

Edge-edge matches can be defined in 3 different ways. The first possibility is a vertex-vertex match between the edges' endpoints. Second, the combination of a vertex-vertex and a vertex-edge match is valid. Finally, 2 vertex-edge matches define an edge-edge match.

Local pruning follows the matching step and corrects eventually false vertex-vertex matches. If 2 or more vertices q_i from a polygon Q have the same matching point p from another polygon P, whis would violate the intrinsic stability constraint possibly leading to self-intersections, flip-overs or edge/diagonal collapses. To prevent this, local pruning removes all matches but the one with lowest distance. The same procedure is done with invalid vertex-edge or vertex-face matches.

In addition to corrections for local instabilities, global pruning needs to be performed to prevent false collapses on a model scale. Figure 2.27 illustrates some examples for such cases. If matches occur in a cyclic way, edges are contracted and lost leading to a violation of the intrinsic stability constraint. Global pruning finds those cycles and eliminates them. All matchings are stored in a graph structure, which allows the detection and removal of cyclic structures.

Following the finding of adjacencies, the optimization is performed, which closes the gaps using the information gathered in the matching procedure. A simple vertex merge is not possible, since this may violate the planarity of the individual polygons. Moreover, the movement of the boundary vertices causes the supporting planes to be changed, increasing the point cloud deviation. In the special case of urban reconstruction, orthogonality has to be considered. The proposed optimization procedure takes these contraints into account and tries to close the gaps as much as possible.

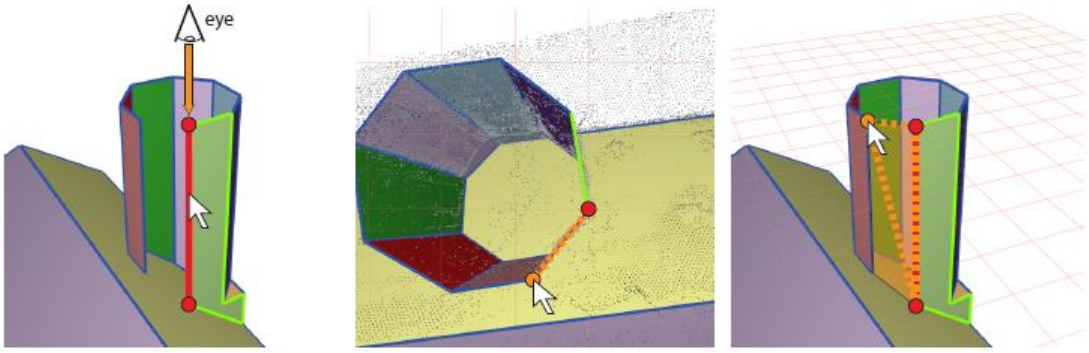


Figure 2.28: Creation of a polygon without using a previously generated supporting plane. The user selects an edge of an adjacent face (left). The camera aligns its viewing axis to the selected edge, which lets the user select a vertex from another adjacent face (middle). This information is sufficient for the tool to create a new polygon, that fills the hole (right). Image taken from the paper of Arikan et al. [ASF+13].

After polygonal faces have been extracted and aligned in the automatic phase, the user can now make additions and corrections to the reconstructed model. The workflow has been reduced to involve only 2D interaction. The user selects the incomplete segment and sketches an outine onto its supporting plane. There is no need to sketch exact polygon outlines. The snapping and optimization algorithm from the automatic phase is monitoring the changes and adjusts the sketched elements to the existing model. The camera automatically adjusts to an orthogonal top-down view onto the selected segment, if the user wishes so. In addition to the creation of entirely new faces, the user is able to manipulate previously sketched or automatically reconstructed elements. Several polygons, that share the same supporting plane, can be merged together by using a drag-and-drop gesture.

In addition to the creation and manipulation of polygons on automatically extracted supporting planes, the user has the ability to create entirely new faces. This is a useful option for scenarios, where missing or noisy data did not allow the algorithm to create a supporting plane. Figure 2.28 illustrates such a case.

In some cases the point cloud may have significant holes, which leads to incomplete or even entirely missing faces in the reconstructed model. Figure 2.29 illustrates an example, where a door parallel to a facade element is reconstructed, but not its surrounding wall elements. In this case, the user can define a hierarchy, that connects the door polygon to the main wall. The snapping and optimization algorithm automatically creates the missing pieces and snaps all elements together to form a complete surface.

Arikan et al. present some results, where they demonstrate the abilities of their solution. The creation of a model out of a point cloud generated by photogrammetric methods is illustrated in Figure 2.30. A building complex was captured and transformed into a sparse point cloud using photogrammetric methods. Using the proposed tool, it was possible to create a first simple model

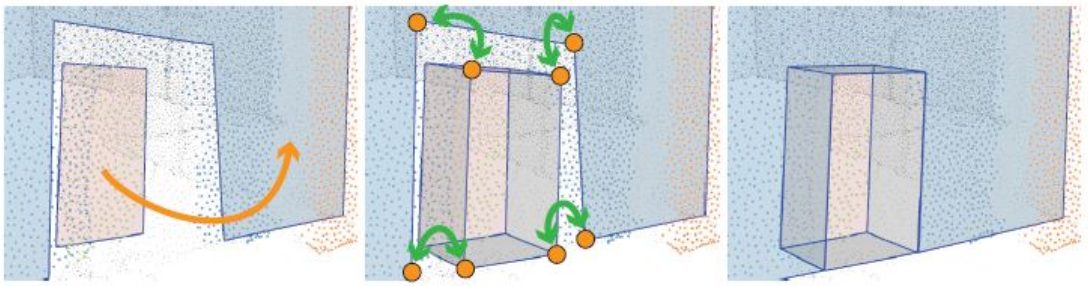


Figure 2.29: The interaction workflow for handling incompletely reconstructed model parts. A hierarchical relationship between a dominant plane (eg. a wall) and a smaller polygon can be created by the user with a mouse click (left). An automatic procedure adds extruded faces (middle), that allow the smaller polygon to be snapped to the dominant one (right). Image taken from the paper of Arikan et al. [ASF+13].

in a matter of few minutes. For refinement, images of the actual object can be projected onto the model to help the user identify and manipulate incomplete or incorrectly modelled regions. After 20 minutes it was possible to recreate the scene with higher detail.

A final evaluation showed that the proposed methods lead to sophisticated results in a short amount of time. Both the simplicity of the interaction technique, as well as its support from automatic optimization algorithms, provide a straightforward workflow and flexibility. However, the authors point out that their proposed approach is only applicable to objects consisting of planar surfaces. Man-made objects, especially urban areas, are very well suited, since they are composed of a finite set of polygons. On the other side, natural scenarios or curved elements cannot be reconstructed.

2.5 Image-based Modeling Techniques

In addition to approaches, that rely on a reconstructed point cloud, scientific research has been done on the reconstruction of objects and scenes directly out of a set of images without using intermediate 3D datasets. One example is the work on the PhotoBuilder tool of Cipolla et al. [CR99], which aims at the generation of 3D models directly out of uncalibrated 2D imagery. The approach takes advantage of orthogonality and parallelism usually found in architectural scenes. Camera information from uncalibrated images can be calculated in order to get viewpoints necessary for computing spatial positions. 3D Scenes usually incorporate vanishing points in their 2D representation, which is an important phenomenon exploited by the tool. The result is a VRML model, that can be used for further manipulation and rendering tasks.

Orthogonality in architectural scenes is also exploited in the work of Musialski et al. [MLS+10], presenting an interactive tool for editing façades. Using a set of perspective photographs from ordinary consumer cameras, the user is able to interactively create ortho-textures of façades, being supported by automatic adjustment and optimization processes.

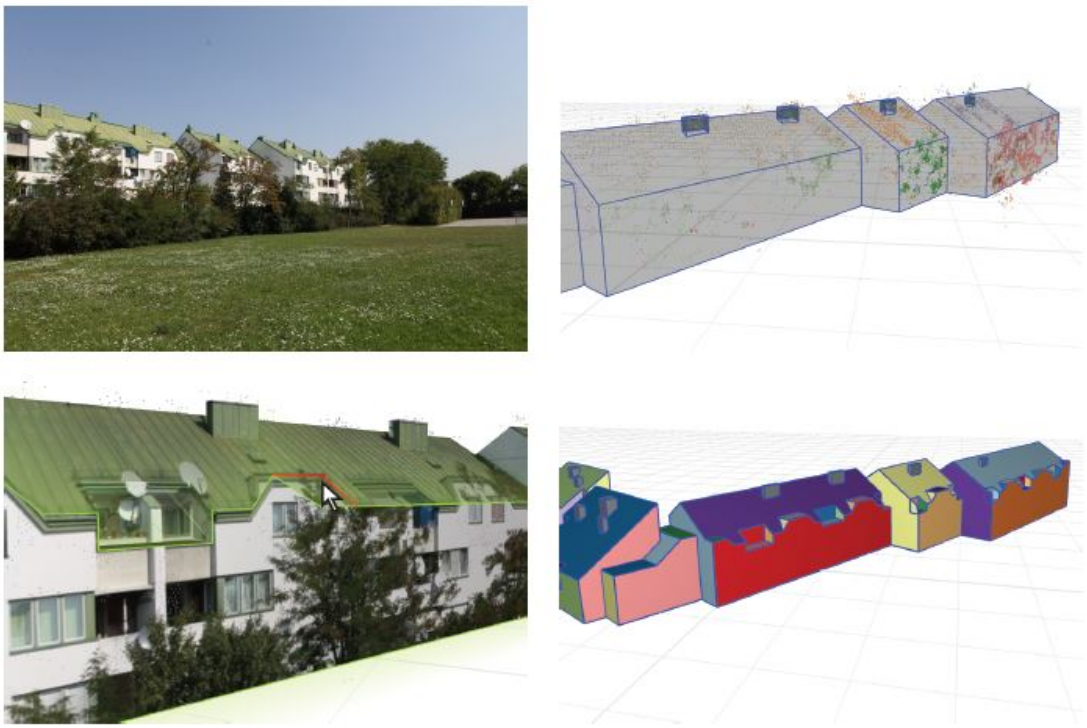


Figure 2.30: Example for an entirely reconstructed dataset of a building (photo on top left). Taking a sparse point cloud the reconstruction of a simple model was possible in 3 minutes (top right). Image reprojection onto the model helps the user to create and adjust fine details (bottom left) and create a more sophisticated model in 20 minutes (bottom right). Image taken from the paper of Arikan et al. [ASF+13].

Modeling façades is the goal of the research work of Xiao et al. [XFT+08], too. The input for this algorithm is a series of photographs taken along streets. In contrast to the previously mentioned work, this approach relies on an intermediate coarse point cloud for modeling the individual elements of façades, which share a common main plane representing the building walls.

One problem of modern architectural scenes is the lack of surface details necessary for detecting features, challenging automated reconstruction algorithms. In order to gather 3D data out of a set of images, it is necessary to determine features in the individual pictures, giving a clue on the spatial relationships. Modern architecture is comprised of mostly plain, uniformly colored surfaces. In this case we speak of so-called Manhattan World scenes. Furukawa et al. [FCSS09] present a solution for a Multi-View Stereo technique especially fitted to this kind of data. Exploiting planarity and orthogonality, which is a common feature in modern architecture, it is possible to extract spatial information for computing depth maps and subsequent 3D models.

Exploitation of the special features of Manhattan World scenes is also the basis for the work of

Vanegas et al. [VAB10], who present an approach for defining a grammar, which describes the previously mentioned characteristics of modern architectural scenes. Calibrated aerial imagery of urban areas form the input for their algorithm, which is able to transform the data into approximative 3D models. The authors' later work from 2012 uses Manhattan World grammar to extract building models out of 3D laser range scan data [VAB12].

2.6 Registration with Iterative Closest Point

The Iterative Closest Point (ICP) algorithm computes a transformation for aligning 2 datasets in such a way, that the sum of distances between each sample from one dataset to its nearest neighboring sample from the other dataset is minimal. Section 3.5 provides a description of ICP following the work of Horn [Hor87].

Besl et al. [BM92] use ICP for their algorithm, that provides a registration of 3D shapes in all 6 degrees of freedom. Given an initial translation and rotation definition, the algorithm converges to a a mean-square metric's local minimum. A comprehensive analysis on ICP and other registration algorithms is provided by Pottmann et al. [PHYH06]. Together with a different method using helical motion, ICP is examined in a technical report from Musialski [Mus09]. Another alternative to ICP is an approach presented by Pottmann et al. [PLH04]. It is based on instantaneous kinematics and analysis of a surface's squared distance function. ICP and similar alternatives to these solutions are evaluated and used in the work of Potmann et al. [PLH02] for registering multiple views of a 3D object simultaneously.

CHAPTER 3

Methodology

3.1 Data Source: Structure from Motion (SfM)

The data that forms the source for the methods presented in this thesis comes from Structure from Motion. This technique allows the extraction of 3D data out of a series of 2D images. The idea behind SfM is the detection of common features in a series of 2 or more images in order to calculate their spatial positions. The full explanation of the algorithm would exceed the scope of this thesis, but the process can be summarized as follows. Figure 3.1 illustrates the basic principle.

2 or more pictures of a scene share a common region, e.g. a window on a facade. The pictures

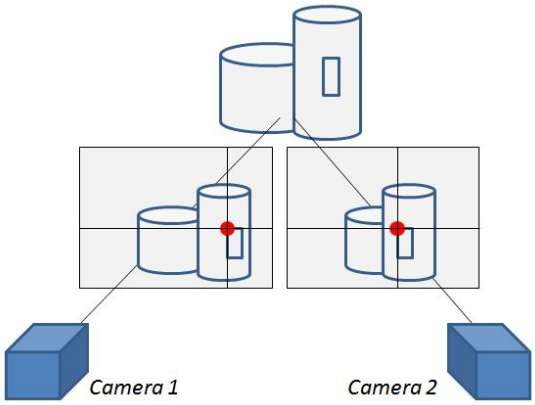


Figure 3.1: The principal of Structure from Motion (SfM). The different pixel location of common image features between 2 or more images give information on the spatial position of that point, e.g. the red marked corner of the window.

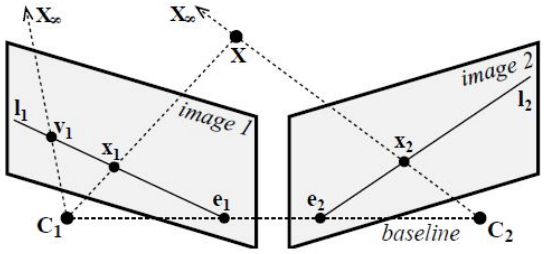


Figure 3.2: The principle of epipolar geometry. The point X is projected to the points x_1 and x_2 on the images. The epipolar line is indicated with l_1 and l_2, stretching out from the two epipoles e_1 and e_2. Together with the baseline and the projection line from X, they create the positions of the projections of C_1 and C_2. Image taken from the paper of Musialski et al..

have been taken from a different angle and distance, so the position of the window on the individual images is different. The algorithm looks for certain features, that can be described and compared, e.g. a specific corner of the window. The important thing to notice is, that this feature must be invariant to scale, position, rotation and brightness variations, since the camera distances and angles differ from picture to picture. A pixel-wise comparison is therefore not applicable. One technique, that is used to incorporate this fact, is Scale-Invariant Feature Transform (SIFT). This approach allows the definition of such features to be compared throughout the whole image set.

The SfM technique is a generalized version of the Stereo Vision approach. Having 2 different images of the same scene, a point in the space present in both images has a different position on each of the pictures. Their relation to each other follows epipolar geometry, whose principle is denoted in Figure 3.2. The point X is projected to the points x_1 and x_2 on the two images. Its projections lie on the epipolar lines l_1 and l_2, which also include the respective epipoles e_1 and e_2. Together with the baseline and the projection line from X, the epipoles create the positions of the projections of C_1 and C_2. The epipolar plane is spanned by the point X and the 2 camera centers C_1 and C_2.

The detection of common corresponding points for reconstruction is done using the previously mentioned SIFT operators. The accuracy and density of the reconstructed data depends on the number of images, on the overlapping amount and on the resolution, which is crucial for the finding of a large number of corresponding points.

If a feature has been detected and found on 2 or more images, the calculation of its spatial position can be performed. Every detected feature is transformed into a point in 3D space and added to the dataset. The number, and therefore the grade of detail, of the point cloud depends on how much features have been detected and reconstructed. Images with plane, uniformly colored surfaces are generally bad for reconstruction, since the finding of features is difficult or not possible. On the other side, very complex scenes with a lot of noise may cause the algorithm to find too much features and incorrect correspondences between the images.

In addition to the reconstructed point cloud, the intrinsic and extrinsic parameters of the pictures' cameras can be computed. They are also stored in the appropriate datafile.

3.2 Data Storage: N-View Match (NVM)

The NVM file format is part of the work of Changchang [Cha], who presents a visual structure from motion system, that includes the clustering algorithms of Furuwaka [Fur]. The tool stores information on both the reconstructed point cloud and the camera parameters, which were extracted from the input images. The basic structure consists of two lists holding the details of sample points and camera details. It starts with the description of the images and their respective extracted camera parameters. Each item holds the following information:

- File name
- Focal length
- Quaternion rotation
- Camera center
- Radial distortion

Each image, which has to be supplied in addition to the NVM file, is listed along with the position and orientation (extrinsic parameters) and the focal length and radial distortion (intrinsic parameters) of the camera that took it. In this implementation only the extrinsic parameters are used to adjust the view to the desired parameters, since this is sufficient for navigation in the point cloud. The second list stores the sample coordinates that make up the point cloud. Each item consists of the following parameters:

- Position
- Color
- Numer of measurements
- List of measurements
  - Image index
  - Feature index
  - Pixel position in image

3.3 Random Sample Consensus (RANSAC)

The RANSAC algorithm was first introduced by Fischler and Bolles in 1981. [FB81] It has a broad range of applications, where it is necessary to fit a model into a dataset. In the scope of this thesis, a 3D point cloud is the data input. RANSAC randomly takes samples and tries to fit a parameterized geometric primitive to the point cloud in such a way, that the maximum number of samples lies in proximity to the object's surface. The fitting of an object is done for a predefined number of times. The parameterized model, that fits to the point cloud in the best way is selected.

The input is a dataset, eg. a 3D point cloud, consisting of a number of samples. The first step is to take a random selection out of the set for setting up a hypothetical model. The cardinality of the set is dependant on the number of points necessary to define the model. For instance, a sphere may only need 2 points for its center and radius to be defined. The algorithm then sets up a model from the selected points. The next step is evaluation. All samples from the input data are tested against the hypothetical model. The distance from each sample to the surface of the parameterized model is computed. If it lies within a predefined threshold, the point is counted as a so-called inlier and contributes to the quality score of the model. If the distance lies beyond the threshold, the point is added to the set of so-called outliers, indicating a bad fit. Depending on the implementation the algorithm stores both inliers and outliers or just the set of inliers for the hypothetical model. The random selection, model creation and evaluation is performed for a predefined number of iterations. After this has been done, the model parameters leading to the largest set of inliers are selected as the best candidate to be used for further calculations. In some implementations, the consensus set of the winning candidate is taken as the input for the creation of a final model with regression techniques.

The RANSAC algorithm for geometric fitting can therefore be summarized as in the following listing.

1. Select random samples and store them in a set. Its cardinality is dependant on the number of necessary points to setup a hypothetical model.
2. Generate a model out of the samples in the random set.
3. Evaluate the model by calculating the distance from the object's surface to each sample from the data. Point with a distance lying within a predefined threshold are stored as inliers (consensus set). All others are counted as outliers.
4. Select the model with the highest number of inliers.
5. Optional step: take the winner consensus set and use it as an input for the final model.

Although RANSAC is a rather primitive approach due to its high dependancy on the random selection, it is an effective solution for data with unknown features.

3.4 Quaternion Rotation

The camera rotation data in the NVM dataset is defined by quaternions, whose definition and usage is summarized in this section. The work of Horn [Hor87] explains this topic.

A quaternion Q is a 4-dimensional mathematical structure described in equation 3.1.

$$Q = x + y + z + w \tag{3.1}$$

It can be described as a vector with 4 components or a complex number $\mathring{q}$ with 1 real and 3 imaginary parts, as given in Equation 3.2.

$$\begin{aligned} \mathring{q} = q_0 + iq_x + jq_y + kq_z \\ i^2 = -1 \\ j^2 = -1 \\ k^2 = -1 \end{aligned} \tag{3.2}$$

A quaternion is called purely imaginary, when only the imaginary parts are unequal to zero. A unit quaternion has a magnitude of 1, which can be calculated in the same way as with ordinary vectors, as it is shown in equation 3.3.

$$|Q| = x * x + y * y + z * z + w * w \tag{3.3}$$

Multiplication of two quaternions can be realized with matrices. The product of $\mathring{p}$ and $\mathring{q}$ in both ways is defined in Equation 3.4.

$$\begin{aligned} \mathring{p}\mathring{q} &= \begin{bmatrix} r_0 & -r_x & -r_y & -r_z \\ r_x & r_0 & -r_z & r_y \\ r_y & r_z & r_0 & -r_x \\ r_z & -r_y & r_x & r_0 \end{bmatrix} \mathring{q} = \mathbb{P}\mathring{q} \\ \mathring{q}\mathring{p} &= \begin{bmatrix} r_0 & -r_x & -r_y & -r_z \\ r_x & r_0 & r_z & -r_y \\ r_y & -r_z & r_0 & r_x \\ r_z & r_y & -r_x & r_0 \end{bmatrix} \mathring{q} = \bar{\mathbb{P}}\mathring{q} \end{aligned} \tag{3.4}$$

The bottom-right 3x3 submatrix in $\bar{\mathbb{P}}$ is the transpose of the one in $\bar{\mathbb{P}}$, so multiplication of two quaternions is non-commutative.

The conjugate $\mathring{q}*$ of a quaternion $\mathring{q}$ can be taken by negating the imaginary parts, as given in Equation 3.5. The respective matrix of the conjugate is equal to the transpose of the original one.

$$\mathring{q}^{\star} = q_0 - iq_x - jq_y - kq_z \tag{3.5}$$

The dot product of two quaternions $\mathring{p}$ and $\mathring{q}$ is defined as the sum of the individual components' products, as given in Equation 3.6.

$$\mathring{p} \cdot \mathring{q} = p_0 q_0 + p_x q_x + p_y q_y + p_z q_z \tag{3.6}$$

This observation leads to the existence of an inverse $\mathring{q}^{-1}$ of a quaternion $\mathring{q}$, as described in Equation 3.7.

$$\mathring{q}^{-1} = (\frac{1}{\mathring{q}} \cdot \mathring{q})\mathring{q}^{\star} \tag{3.7}$$

The inverse of a unit quaternion is equal to the conjugate.

An important property is the fact, that the product of a quaternion $\mathring{q}$ with its conjugate $\mathring{q}^{\star}$ is real. Given the quaternion's matrix Q, this means $QQ^T = \mathring{q} \cdot \mathring{q} I$, with I being the 4x4 identity matrix. It leads to the real product $\mathring{q}\mathring{q}^{\star}$, given in Equation 3.8.

$$\mathring{q}\mathring{q}^{\star} = (q_0^2 + q_x^2 + q_y^2 + q_z^2) = \mathring{q} \cdot \mathring{q} \tag{3.8}$$

The matrices of quaternions are orthogonal, so that dot products are preserved. Following the two definitions in Equation 3.9, one gets the conclusion given in Equation 3.10. The

$$\begin{aligned} (\mathring{q}\mathring{p}) \cdot (\mathring{q}\mathring{r}) = (Q\mathring{p}) \cdot (Q\mathring{r}) = (Q\mathring{p})^T (Q\mathring{r}) \\ (Q\mathring{p})^T (Q\mathring{r}) = \mathring{p}^T Q^T Q \mathring{r} = \mathring{p}^T (\mathring{q} \cdot \mathring{q}) I \mathring{r} \end{aligned} \tag{3.9}$$

$$(\mathring{q}\mathring{p}) \cdot (\mathring{q}\mathring{r}) = (\mathring{q} \cdot \mathring{q})(\mathring{p} \cdot \mathring{r}) \tag{3.10}$$

Due to their properties, quaternions are a useful tool for describing rotations in 3D space. A vector can be represented by a purely imaginary quaternion. Rotation does not change a vector's length or the angle between two vectors, so it does not change the dot product. The same counts for reflection: it does not change length or angle, but it changes the orientation of a cross product. The usage of quaternions for rotation demands the mapping of two purely imaginary

quaternions to be purely imaginary as well, since it shall represent a vector. As stated above in Equation 3.10, the multiplication of two quaternions $\mathring{p}$ and $\mathring{r}$ by a unit quaternion $\mathring{q}$ preserves dot products of $\mathring{p}$ and $\mathring{r}$. Simple multiplication cannot be used, because the product of a unit quaternion and a purely imaginary quaternion is not necessarily purely imaginary. So instead of a simple multiplication the composite product can be used, which is defined in Equation 3.11. The resulting $\mathring{r}'$ is purely imaginary.

$$\mathring{r}' = \mathring{q}\mathring{r}\mathring{q}^{\star} \tag{3.11}$$

Equation 3.12 explains the composite product in further detail, with Q and $\bar{Q}$ referring to the matrices of $\mathring{q}$.

$$\mathring{q}\mathring{r}\mathring{q}^{\star} = (Q\mathring{r})\mathring{q}^{\star} = \bar{Q}^T(Q\mathring{r}) = (\bar{Q}^T Q)\mathring{r} \tag{3.12}$$

Particular attention must be drawn to the product $\bar{Q}^T Q$, presented in Equation 3.13. Given that $\mathring{q}$ is a unit quaternion with $\mathring{q} \cdot \mathring{q} = 1$ and the matrices Q and $\bar{Q}$ being orthonormal, the bottom-right 3x3 submatrix of the product must also be orthonormal. More importantly, it is equivalent to the commonly used rotation matrix R for transforming a vector r into $r' = Rr$.

$$\bar{Q}^T Q = \begin{bmatrix} \mathring{q} \cdot \mathring{q} & 0 & 0 & 0 \\ 0 & (q_0^2 + q_x^2 - q_y^2 - q_z^2) & 2(q_x q_y - q_0 q_z) & 2(q_x q_z + q_0 q_y) \\ 0 & 2(q_y q_x + q_0 q_z) & (q_0^2 - q_x^2 + q_y^2 - q_z^2) & 2(q_y q_z - q_0 q_x) \\ 0 & 2(q_z q_x - q_0 q_y) & 2(q_z q_y + q_0 q_x) & (q_0^2 - q_x^2 - q_y^2 + q_z^2) \end{bmatrix} \tag{3.13}$$

3.5 Iterative Closest Point

In addition to the fundamentals of quaternions, the work of Horn [Hor87] describes the Iterative Closest Point (ICP) algorithm, that is used in the approximation procedure presented in this thesis. Given 2 point clouds P_1 and P_2, ICP tries to find the best fit of one dataset to the other with a minimum sum of squared distances. In other words, P_1 is translated and rotated in such a way, that the sum of distances from each of the samples in P_1 to the nearest point in P_2 is minimal.

The first step of ICP is the translation of both datasets to the origin, so that the orientation is absolute. For each point cloud the gravity point is calculated and subtracted from all samples in the respective dataset. For calculation of the rotation parameters one needs to find the unit quaternion that maximizes the Equation 3.14. Here, $r'_{l,i}$ denotes the rotated i-th sample from the

first (left) dataset with $r'_{l,i} = (z'_{l,i}, y'_{l,i}, z'_{l,i})^T$. The definition of $r'_{r,i}$ counts to the second dataset and is analog to the first one.

$$\sum_{i=1}^{n} (\mathring{q}\mathring{r}'_{l,i}\mathring{q}^{\star}) \cdot \mathring{r}'_{r,i} \tag{3.14}$$

As stated in the previous section on quaternions, this can be rewritten as given in Equation 3.15.

$$\sum_{i=1}^{n} (\mathring{q}\mathring{r}'_{l,i}) \cdot (\mathring{r}'_{r,i}\mathring{q}) \tag{3.15}$$

The parts in the first and second brackets respectively can be rewritten in matrix form as in Equation 3.16.

$$\begin{aligned} \mathring{q}\mathring{r}'_{l,i} &= \begin{bmatrix} 0 & -x'_{l,i} & -y'_{l,i} & -z'_{l,i} \\ x'_{l,i} & 0 & z'_{l,i} & -y'_{l,i} \\ y'_{l,i} & -z'_{l,i} & 0 & x'_{l,i} \\ z'_{l,i} & y'_{l,i} & -x'_{l,i} & 0 \end{bmatrix} \mathring{q} = \bar{\mathbb{R}}_{l,i}\mathring{q} \\ \mathring{r}'_{r,i}\mathring{q} &= \begin{bmatrix} 0 & -x'_{r,i} & -y'_{r,i} & -z'_{r,i} \\ x'_{r,i} & 0 & rz'_{r,i} & y'_{r,i} \\ y'_{r,i} & z'_{r,i} & 0 & -x'_{r,i} \\ z'_{r,i} & -y'_{r,i} & x'_{r,i} & 0 \end{bmatrix} \mathring{q} = \mathbb{R}_{r,i}\mathring{q} \end{aligned} \tag{3.16}$$

Both $\mathbb{R}_{r,i}$ and $\mathbb{R}_{l,i}$ are skew symmetric and orthogonal, so it is possible to rewrite the maximum equation as given in 3.17.

$$\sum_{i=1}^{n} (\bar{\mathbb{R}}_{l,i}\mathring{q}) \cdot (\mathbb{R}_{r,i}\mathring{q}) \tag{3.17}$$

This can be further rewriten as in Equation 3.18, which introduces symmetric matrices N_i that make up a matrix N defined as $N = \sum_{i=1}^{n} N_i$. The precise steps to get to this formula are explained in further detail in the work of Horn [Hor87].

The matrix N plays an important role for the least-squares problem, that describes the challenge of finding the absolute orientation. First, let M be a 3x3 matrix, that is defined as in Equation 3.18.

$$M = \sum_{i=1}^{n} (r'_{l,i} {r'_{r,i}}^{T}) = \begin{bmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{bmatrix} \tag{3.18}$$

The coefficients S_{ii} are defined as given in Equation 3.19. Depending on their parameters, they are formed by summing up the products of each sample's component from one dataset with the component of the nearest sample of the other one.

$$\begin{aligned} S_{xx} &= \sum_{i=1}^{n} x'_{l,i} x'_{r,i} \\ S_{xy} &= \sum_{i=1}^{n} x'_{l,i} y'_{r,i} \\ &\quad etc. \end{aligned} \tag{3.19}$$

With the coefficients S_{ii} being calculated, one can now setup the 4x3 matrix N, which is defined in Equation 3.20.

$$N = \begin{bmatrix} (S_{xx} + S_{yy} + S_{zz}) & S_{yz} - S_{zy} & S_{zx} - S_{xz} & S_{xy} - S_{yx} \\ S_{yz} - S_{zy} & (S_{xx} - S_{yy} - S_{zz}) & S_{xy} + S_{yx} & S_{zx} + S_{xz} \\ S_{zx} - S_{xz} & S_{xy} + S_{yx} & (-S_{xx} + S_{yy} - S_{zz}) & S_{yz} + S_{zy} \\ S_{xy} - S_{yx} & S_{zx} + S_{xz} & S_{yz} + S_{zy} & (-S_{xx} - S_{yy} + S_{zz}) \end{bmatrix} \tag{3.20}$$

The important fact is, that the unit quaternion $\mathring{q}$ maximizing $\mathring{q}^T N \mathring{q}$ is the eigenvector of N corresponding to the largest positive eigenvalue. The resulting $\mathring{q}$ is then used to rotate the first dataset towards the other one to minimize the sum of squared distances.

The ICP algorithm can be summarized as follows. The first step is to calculate the gravity points for each of the two datasets. They are then subtracted from all samples $r_{l,i}$ and $r_{r,i}$ respetively to get normalized data $r'_{l,i}$ and $r'_{r,i}$. The algorithm then sets up the matrix N by calculating all coefficients. There, sample from the first dataset $r'_{l,i}$ is paired with its nearest neighbor $r'_{r,i}$ from the second one. The algorihm now calculates eigenvectors and eigenvalues of N and takes the eigenvector corresponding to the largest positive eigenvalue as an input for the quaternion $\mathring{q}$. The latter is used to rotate the first dataset towards the second one. Since ICP is an iterative approach, the whole procedure is repeated. The second iteration takes the rotated data and finds nearest neighbors again to setup a new matrix N, and so on. Iterations are performed until a threshold or the maximum iteration count has been reached.

3.6 k-d Tree

The individual samples in the point cloud are stored in a list of 3D vectors. In each iteration of the ICP algorithm, the nearest neighbor in the point cloud has to be determined for every sample from the parameterized model. Running through the whole list every time is very inefficient and would dramatically increase the computation time. A k-d tree (short for k-dimensional tree) can

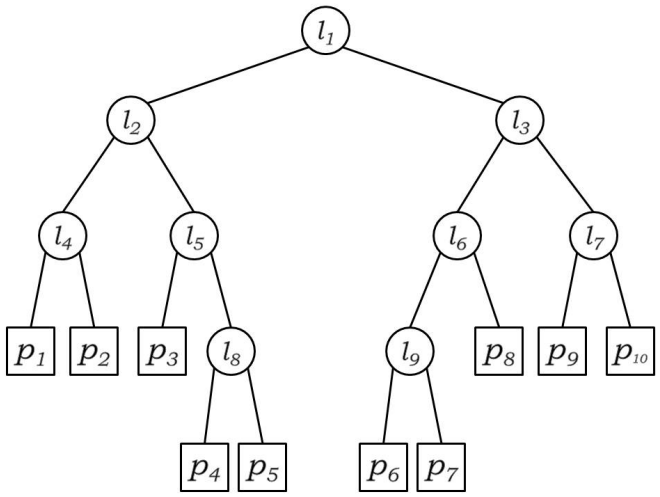


Figure 3.3: A sample k-d tree. Datapoints p_i are stored in the leafs. Internal nodes represent hyperplanes, that split the space in alternating coordinate axes.

be used to accelerate the task of finding the nearest neighbor by setting up a data structure that can be traversed very efficiently and fast. The book on geometric data structures for computer graphics by Langetepe et al. [LZ06] provides information on the construction and usage of this structure.

The k-d tree is a special form of a binary space-partition tree. In the scope of this thesis, following the use of 3D vectors in the dataset the dimension k of the tree is 3. Each internal node is assigned to one of the k dimensions and splits the space with a hyperplane perpendicular to the axis of k. Each leaf contains one or more data points, and each point is stored only once. Figure 3.3 illustrates a sample k-d tree.

The setup of a k-d tree can be done in many ways. One possibility is the so-called canonical construction with median values, which leads to a balanced k-d tree. For a dataset with 3 dimensions, e.g. vectors in space, the internal nodes form separating 1D hyperplanes that are organized in an alternating order. The first level node separates the first dimension; the second level separates the second dimension, and so on. The level beneath the third level starts with the first dimension again. The separation value is always the median of the previously ordered set of values. Figure 3.3 illustrates the structure of the tree. The root l_1 represents the first separation in the first dimension. The children l_2 and l_3 build hyperplanes that separate the data in the second dimension. The third-level nodes l_4, l_5, l_6 and l_7 do the same thing in the third dimension. Depending on the input values there could be four or more levels, where the nodes start to separate the first dimension again.

The k-d tree is an efficient data structure for fast traversal. It is especially useful in the case of nearest neighbor search and dramatically decreases the search time in contrast to a plain traversal of the point list.

3.7 Fitting Process

The fitting solution supports four types of geometric primitives: cones, cylinders, spheres and cuboids. The fitting procedure combines two approaches for fitting and optimization. First, a RANSAC-like procedure finds initial parameters for the selected geometric primitive. These values serve as an approximative solution that is refined in the second step, which consists of the iterative closest point method presented by Horn et al. [Hor87]. The initial solution is iteratively refined until the difference between the point cloud and the parameterized object surface is minimized to a certain extent. The initial fitting process for cones, cylinders and parallelepipeds follows the geometric construction steps presented in the master thesis of Garcia [Gar09]. The output of this first step is a parameterized primitive with correct scaling but not optimal positioning and rotation. With ICP an optimal rotation can be determined, whereas the positioning can be found by first translating the parametrized object to the coordinate system's origin and then back to the correct position in the dataset by using the gravity point of the point cloud.

First step: RANSAC

The first part of the fitting procedure uses a RANSAC-like algorithm to find initial parameters for an approximation later to be refined in the ICP step. A certain amount of randomly selected points is taken from the dataset to form the MSS (minimal sample set). Depending on the parameter set of the object the cardinality of the MSS differs. One can now use these points to setup a model for the primitive that is tested against all other points in the dataset. If the difference between the modeled object's surface and a certain point in the dataset is within a predefined threshold, this point is counted as an inlier. If the distance lies beyond that threshold, it is added to the set of outliers.

Second step: ICP

After the RANSAC algorithm has found parameters for an approximative solution the orientation of the object is refined until the sum of distances between the surface and the data points is reduced to a certain threshold. The algorithm iteratively rotates the object to an orientation where the sum of distances from the data points to the surface is lower than the previous state.

Before the iteration loop starts both the dataset and the model's parameter points are translated around the coordinate system's origin by subtracting the model's gravity point. This ensures that the rotation center of the object is equal to the barycenter. After the iteration loop the model is translated back to its final position by adding the object's gravity point to the its parameter values.

Research, implementation and testing indicated, that an inverse approach leads to better results after ICP. Instead of looking for nearest neighbors in the dataset, the algorithm works the other way round. For each sample from the point cloud, the nearest neighboring sample from the model surface is looked up. After the rotation has been computed and executed on the model,

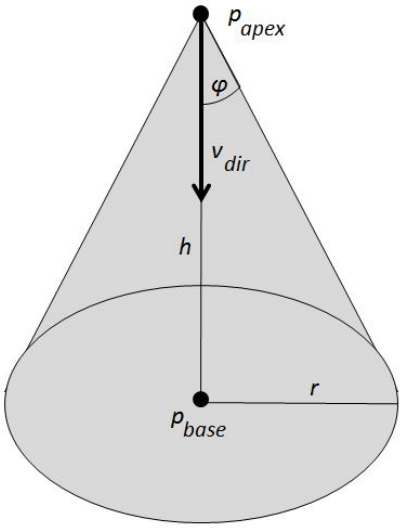


Figure 3.4: The parameters of a cone. Appearance and orientation in space is fully defined by an apex position with an aperture angle, as well as an orientation vector for the axis and the cone's height.

the k-d tree has to be recreated. This approach may lead to a slightly longer computation time, but produces more feasible results, especially when the dataset contains larger holes.

Cone

The construction of a conic surface is described in the master thesis of Garcia [Gar09]. For the reconstruction and definition of a cone one needs to determine its height and aperture angle. Additionally, for position and orientation of the object in 3D space the apex position and orientation of the cone's central axis has to be found. Figure 3.4 depicts the parameters of an oriented cone in 3D space.

Figure 3.5 illustrates the first step of the geometric construction of a cone. The MSS of each RANSAC iteration consists of 4 randomly selected points p_i. The first 3 are taken for the construction of the center point of a circle that goes through all 3 points. This can be achieved by setting up 3 planes, where the intersection point equals the center of the circle.

From the 3 points one can generate two directional vectors v_1 and v_2, whose construction follows equation 3.21.

$$\begin{aligned} \vec{v_1} &= \vec{P_1P_2} = P_2 - P_1 \\ \vec{v_2} &= \vec{P_1P_3} = P_3 - P_1 \end{aligned} \tag{3.21}$$

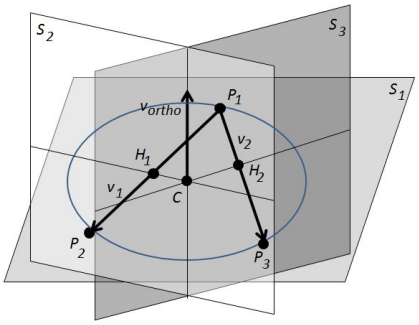


Figure 3.5: Construction of a circle in 3D space. 3 points form a plane. Second and third planes are constructed with halfway positions between the points and their directional vector. The circle containing all points has its center at the intersection of the 3 planes.

Together with the halfway positions H_1 between P_1 and P_2, and H_2 between P_1 and P_3 respectively, they form two planes. The cross product of v_1 and v_2 forms the surface normal v_{ortho} of a plane that contains all 3 points. Intersection of these 3 planes can be achieved by solving a linear equation system given in equation 3.23. The result is the center C of the circle with the coordinates (x, y, z). One can now easily determine the radius r by calculating the distance from the center to one of the 3 points, given in equation 3.22.

$$r = \frac{C\vec{P}_1}{\left|\left|C\vec{P}_1\right|\right|} \tag{3.22}$$

$$\begin{aligned} x_{v_1}x + y_{v_1}y + z_{v_1}z &= x_{v_1}x_{P_1} + y_{v_1}y_{P_1} + z_{v_1}z_{P_1} \\ x_{v_1}x + y_{v_1}y + z_{v_1}z &= x_{v_1}x_{P_1} + y_{v_1}y_{P_1} + z_{v_1}z_{P_1} \\ x_{v_1}x + y_{v_1}y + z_{v_1}z &= x_{v_1}x_{P_1} + y_{v_1}y_{P_1} + z_{v_1}z_{P_1} \end{aligned} \tag{3.23}$$

The remaining construction steps are presented in Figure 3.6. For the construction of the apex A two straight lines L_1 and L_2 have to be determined. The first one is equal to the height axis of the cone and is defined by the center C and the surface normal v_{ortho} of the circle plane. For the construction of L_2 the 4th point P_4 is taken from the MSS and orthogonally projected onto the circle plane, resulting in the projected point P_4'. Equation 3.24 describes the projection of a point P to a plane defined by a surface normal v and a point O lying on the plane.

$$P' = P - ((P - O) \cdot v)v \tag{3.24}$$

The straight line $C\vec{P}_4$ is then calculated and intersected with the circle to obtain an additional point S. This can be achieved by starting from the center C and going along the normalized vec-

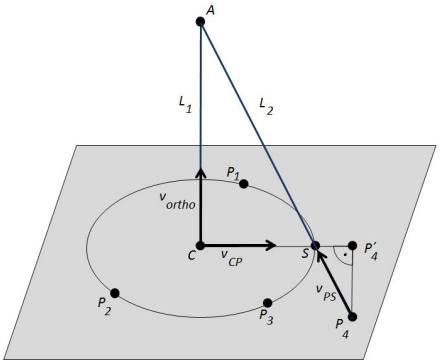


Figure 3.6: Construction of a cone. 3 points form a base surface with the normal being the orientation of the cone's axis. A fourth point is used to construct the apex.

tor $v\vec{C}P$ by the length of the previously calculated radius r. Equation 3.25 shows the calculation of the point S.

$$S = C + \frac{v\vec{C}P}{||v\vec{C}P||}r \tag{3.25}$$

Having computed S, one can construct the straight line L_2, which connects P_4 with S and intersects L_1 at the apex A of the cone. There are various methods for calculating the intersection point. One way to achieve this is to use trigonometry. Let $\frac{\phi}{2}$ be the half of the cone's aperture. It can be determined with the triangle spanned by S, P_4 and P_4', given in equation 3.26.

$$\phi = \arccos \frac{\left|\left|P_4\vec{P}_4'\right|\right|}{\left|\left|P_4\vec{S}\right|\right|} \tag{3.26}$$

Having the angle, one can then determine the height h of the cone with equation 3.27

$$h = \left|\left|C\vec{A}\right|\right| = \frac{\left|\left|C\vec{S}\right|\right|}{\arctan\phi} \tag{3.27}$$

The position of the apex A can finally computed by starting at the center C and taking the direction of the vector $P_4\vec{P}_4'$ by the length of the newly computed height h, as it is given in equation 3.28

$$A = C + \frac{P_4\vec{P}_4'}{\left|\left|P_4\vec{P}_4'\right|\right|}h \tag{3.28}$$

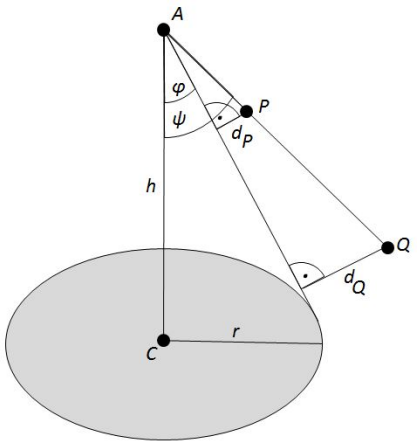


Figure 3.7: Orthogonal distance from two points P and Q to the surface of the cone. The angle ψ between the axis and the straight lines $\bar{AP}$ and $\bar{AQ}$ is the same, but the point-to-surface distances d_P and d_Q change with the distance from the respective point to the cone's apex.

Now the fully constructed cone model has to be tested against all points in the dataset. The distance from a point to the cone's surface can be determined by measuring the difference between the aperture and the angle between the straight line from the apex to the point and the cone's axis. This is illustrated in Figure 3.7. However, a point near the cone's base can have the same angular difference as another one closer to the apex, but they do not have the same distance to the surface, as it is depicted in the Figure.

To solve this problem, Garcia proposes to take the distance from the point to the apex into account, thus providing a distance measure that is qualified for arbitrary positions of the data points relative to the cone. The formula is provided in equation 3.29. Given the cone's aperture ϕ, the apex A and the base center C, one can compute a distance measure d from the cone's surface to a point P by combining the angular difference and point-to-apex distance.

$$\begin{aligned} \vec{u} &= \vec{AC} = C - A \\ \vec{v} &= \vec{AP} = P - A \\ \vec{w} &= \frac{\vec{v}}{||\vec{v}||} \\ \psi &= \arccos(u \cdot w) \\ d &= \sin\psi - \phi\,||\vec{v}|| \end{aligned} \tag{3.29}$$

The distance measure is calculated for every point in the dataset. If it lies within a predefined threshold, the point is added to the set of inliers, otherwise it is part of the outliers set. This step finalizes one iteration of the RANSAC algorithm. After a predefined number of iterations

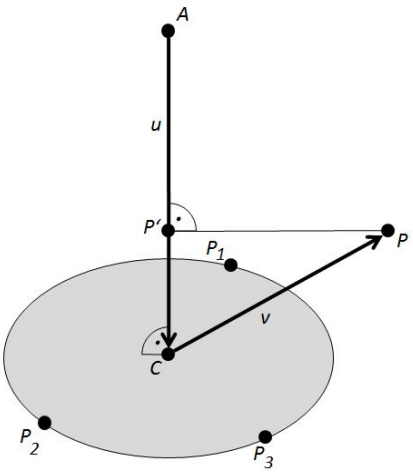


Figure 3.8: Orthogonal projection of a point P onto a straight line spanned by C and A. The resulting point P' is the position, where the point P has the shortest distance to the straight line.

the parameters of the model with the highest number of inliers is selected and forwarded to the second stage of the fitting procedure. Before the parameters are forwareded to the ICP algorithm, a final calculation has to be made. The previously calculated height h only defines the distance from the apex to the randomly constructed circle plane, so the model may not include all samples from the dataset. In order to achieve this, the base plane may be shifted towards or away from the apex. The cone's actual height and therefore the position of the base plane can be found by projecting the data points to the central axis and take the projected position, that has the greatest distance to the apex, as the new position of the base. The projection of a point to a straight line is depicted in equation 3.30 and Figure 3.8.

$$\begin{aligned} \vec{u} &= \vec{AC} \\ \vec{v} &= \vec{CP} \\ P' &= P - \vec{v} + \frac{\vec{v} \cdot \vec{u}}{||\vec{u}||^2}\vec{u} \end{aligned} \tag{3.30}$$

The projection of all points to the cone's axis also supports the creation of truncated cones. For this purpose not only the projected point with maximum distance from the apex is stored, but also the point with minimum distance. The radius at both positions can be calculated by trigonometry using the aperture and the respective distance from the apex.

The ICP algorithm now takes the conic surface and the dataset as an input and computes the distances between individual samples of the surface and their nearest neighbors in the dataset. With this information it is possible to extract parameters for rotating the cone to an orientation with lower sum of distances. With ICP the model is iteratively rotated to an optimal position in

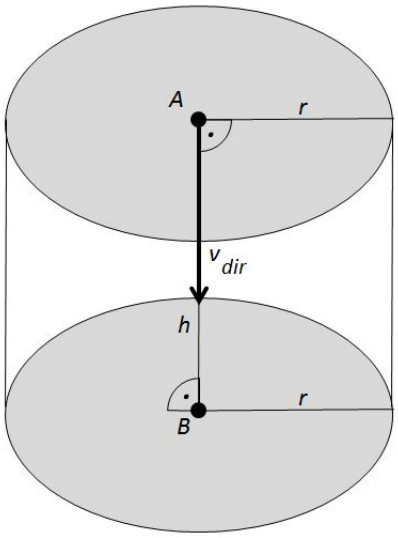


Figure 3.9: The parameters of a cylinder. Position, scale and orientation are fully defined by two points A, B and a radius r.

the point cloud. After a certain threshold or the maximum number of iterations has been reached the fitting process is over and the parameters for the cone are stored.

Cylinder

A cylinder is defined by height and radius. With position and orientation taken into account, the definition is complete with 2 points A and B for the bottom and top faces as well as the radius r. Again, the construction steps for a cylinder are described in the master thesis of Garcia [Gar09]. Figure 3.9 illustrates the parameters of a cylinder.

For setting up a model for a cylinder a MSS with 3 points is needed. Just like the algorithm for the cone, one takes 3 points and calculates the center of a circle. Combined with the normal of the surface containing the MSS set and the newly calculated center, the main axis of the cylinder is described. The construction process can be seen in Figure 3.5. The radius r is equal to the distance from one point in the MSS to the newly computed center C, which is equal to either A or B in Figure 3.9. This concludes the construction process; the cylinder's position and orientation is fully defined with a straight line and a radius. The scale along the cylinder's rotation axis will be defined later.

For counting inliers and outliers the shortest distance from each point in the dataset to the cylinder axis is compared with the previously computed radius. If the difference lies within a predefined threshold, the point is added to the set of inliers. Otherwise it is part of the outliers set. The calculation of the shortest distance d from a point P to a line defined by a point M and a

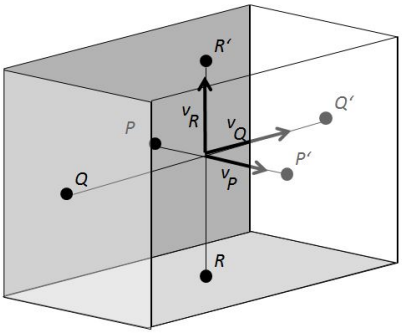


Figure 3.10: The parameters of a cuboid. Position, scale and orientation are fully defined by 3 pairs of points and 3 directional vectors for the faces' orientations.

directional vector v is given in equation 3.31.

$$\begin{aligned} \vec{u} &= C\vec{P} \\ \vec{v} &= P_1\vec{P}_2 \times P_!\vec{P}_3 \\ d &= \left|\left|\frac{\vec{v} \times \vec{u}}{||\vec{v}||}\right|\right| \end{aligned} \tag{3.31}$$

Having selected the parameters, which yield to the largest set of inliers, the endpoints of the cylinder axis have to be determined, which describe the positions of the object's top and bottom face. This is achieved by projecting the first point P_1 from the dataset to the axis as given in equation 3.24 and then projecting all other points P_i as well. The projections P_i' are then compared to the first projected point in both directions of the axis and the ones with largest distance in both directions describe the position of the top and bottom face with the axis as the surface normal. All necessary parameters have been found and the optimal rotation can now be determined by using the ICP algorithm. Before the first iteration samples of the cylinder surface are generated. For each sample the algorithm finds the nearest neighbor in the point cloud and the matrix N can be computed. Again, the eigenvector corresponding to the largest positive eigenvalue determines the quaternion for rotating the cylinder towards the optimal orientation.

Cuboid

The cuboid is of all presented primitives the most complex object to the fitted, since it is not a rotational solid. It consists of 3 pairs of parallel planes in a way that each pair is orthogonal to the others. Figure 3.10 illustrates the parameters of a cuboid. Position, orientation and scale are fully defined by 6 points lying on their respective planes and 3 surface normals for the pairs of parallel faces. Construction of a cuboid is also part of Garcia's master thesis [Gar09].

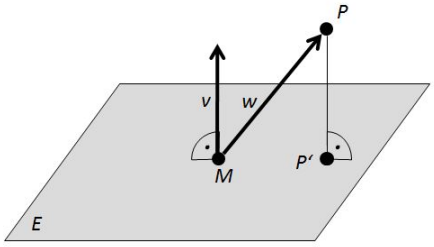


Figure 3.11: Orthogonal projection of a point P onto a plane defined by a point M and a surface normal $\vec{v}$

The RANSAC algorithm creates a random MSS of 5 points in the first step. Points P_1, P_2 and P_3 of that set are used to create the first plane E_1. Similar to Figure 3.5 the surface normal $\vec{v_P}$ is the cross product of two vectors spanned by the 3 points, shown in equation 3.32.

$$\begin{aligned} \vec{v_1} &= \vec{P_1 P_2} = P_2 - P_1 \\ \vec{v_2} &= \vec{P_1 P_3} = P_3 - P_1 \\ \vec{v_P} &= \vec{v_1} \times \vec{v_2} \end{aligned} \tag{3.32}$$

The first plane E_1 is now fully defined. Point P_4 is then orthogonally projected onto it. The orthogonal projection of a point P to a plane defined by a point M and a surface normal $\vec{v}$ is defined in equation 3.33 and Figure 3.11.

$$\begin{aligned} \vec{w} &= \vec{MP} = P - M \\ P' &= P - (\vec{w} \cdot v)\vec{v} \end{aligned} \tag{3.33}$$

The resulting point P_4', together with P_4 and the remaining P_5 from the MSS are used to compute the plane E_2, which is orthogonal to the first one. Analog to equation 3.32 the surface normal $\vec{v_Q}$ of the plane can be computed. The third plane E_3 is orthogonal to the first 2 ones, so its surface normal $\vec{v_R}$ is the cross product of the normals of plane E_1 and E_2. Figure 3.12 illustrates the construction process for the 3 planes.

Now that the surface normals of all 3 planes are known, one needs to find the position of the planes within the point cloud, which is equal to the task of finding the values D1, D2 and D3 that are part of the definition of the planes given in equation 3.34.

$$\begin{aligned} E_1 &: x_{v_P} x + y_{v_P} y + z_{v_P} z = D_1 \\ E_1 &: x_{v_Q} x + y_{v_Q} y + z_{v_Q} z = D_2 \\ E_1 &: x_{v_R} x + y_{v_R} y + z_{v_R} z = D_3 \end{aligned} \tag{3.34}$$

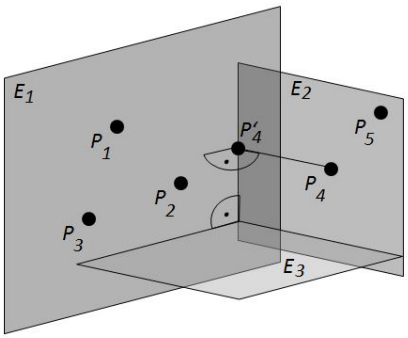


Figure 3.12: Construction of a cuboid's first 3 faces. The first 3 points P_1, P_2 and P_3 span the first plane E_1. The second plane E_2 is orthogonal to E_1 and is spanned by points P_5, P_4 and its orthogonal projection P_4' onto E_1. The cross product of the two surface normals is the normal of the third orthogonal plane E_3.

The values D_i are initially set to zero, meaning that all 3 planes go through the origin. For each plane the orthogonal distance to each point in the dataset is computed and stored in an array. The arrays' most occurring values are then selected as the new values D_i for the planes. Equation 3.35 explains the calculation of the distance from a point P to a plane through the origin with a surface normal $\vec{v}$.

$$d = P \cdot \vec{v} \tag{3.35}$$

The next step is the computation of inlier and outlier sets. A point from the dataset is an inlier, when its distance to the nearest plane is below a predefined threshold. The algorithm computes the distance to every plane and takes the lowest one for testing against the threshold. Equation 3.35 has to be extended to a general case, where planes do not necessarily go through the origin, but are defined by a point M and the surface normal $\vec{v}$. The new method for calculating the distance to the datapoints is defined in equation 3.36, which is based on the definition of a plane with a normal vector $\vec{v}$ and a value D indicating the orthogonal distance to the origin.

$$d = x_{\vec{v}} x_P + y_{\vec{v}} y_P + z_{\vec{v}} z_P + D \tag{3.36}$$

Again, the distance d has to be lower than a predefined threshold for the point to be considered as an inlier. This time not only the number of inliers is interesting, because the outliers are relevant as an input for the computation of the 3 missing parallel planes of the cuboid. The procedure of finding the values D_4, D_5 and D_6 for these planes is similar to the computation of the first ones, except here only the previously created set of outliers is considered as a data input for the calculation of the D_is. Again, the distance occurring most is taken for the individual plane's value D_i.

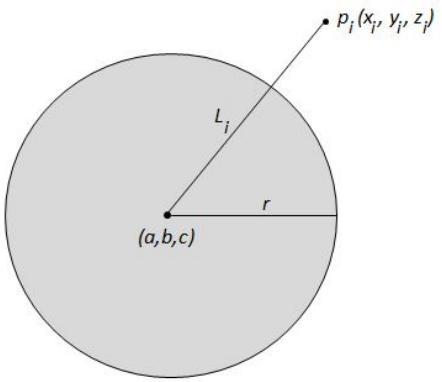


Figure 3.13: The parameters of a sphere

After selecting the set of parameters leading to the most inliers, the six planes are intersected with each other to get the 8 corner points of the cuboid. This can be achieved by solving 8 linear equation systems, each with the 3 plane equations to be intersected. The computed corner points can be used to create samples on each face, which are forwarded to the first ICP iteration. Again, the object is iteratively rotated to a position with minimal sum of distances from the surface to the individual dataset points.

Sphere

In terms of fitting complexity, the sphere is the simplest object to be fitted to the point cloud. It is only necessary to compute 2 parameters, which is position of the center and the radius. If the sample points are equally distributed, the center would be equal to the point cloud's gravity point and the radius described by the average distance from the center to the sample points. However, one cannot assume that the point cloud is equally distributed, so it is necessary to introduce a more advanced algorithm that can cope with such situations. Using RANSAC and ICP is not applicable in the same way as with the other primitives mentioned before. ICP finds a rotation for a best fit of a model, but rotating a sphere is obviously useless. RANSAC alone may not lead to satisfying results.

Figure 3.13 illustrates the parameters of a sphere. It is defined by a center with coordinates (a, b, c) and the radius r. The distance from the center to any point p_i in the dataset is defined by L_i. Eberly [Ebe08] introduces an iterative method for fitting a sphere to a non-uniformly distributed point cloud. The surface of a sphere is defined as $(x-a)^2+(y-b)^2+(z-c)^2 = r^2$, with (a,b,c) being the center. The goal of Eberly's approach is the minimization of the energy function

$$E(a, b, c, r) = \sum_{i=1}^{m} (L_i - r)^2 \tag{3.37}$$

with $Li = sqrt((xi-a)^2 + (yi-b)^2 + (zi-c)^2)$. Performing a partial derivation of the energy function with respect to r, a, b and c leads two the four equations given in (Eq)

$$\frac{\partial E}{\partial r} = -2\sum_{i=1}^{m}(L_i - r) \tag{3.38}$$

$$\frac{\partial E}{\partial a} = -2\sum_{i=1}^{m}(L_i - r)\frac{\partial L_i}{\partial a} = 2\sum_{i=1}^{m}((x_i - a) + r\frac{\partial L_i}{\partial a}) \tag{3.39}$$

$$\frac{\partial E}{\partial b} = -2\sum_{i=1}^{m}(L_i - r)\frac{\partial L_i}{\partial b} = 2\sum_{i=1}^{m}((y_i - b) + r\frac{\partial L_i}{\partial b}) \tag{3.40}$$

$$\frac{\partial E}{\partial c} = -2\sum_{i=1}^{m}(L_i - r)\frac{\partial L_i}{\partial c} = 2\sum_{i=1}^{m}((z_i - c) + r\frac{\partial L_i}{\partial c}) \tag{3.41}$$

These partial derivatives are now set to zero to obtain the equations given in (Eq)

$$r = \frac{1}{m}\sum_{i=1}^{m} L_i \tag{3.42}$$

$$a = \frac{1}{m}\sum_{i=1}^{m} x_i + r\frac{1}{m}\sum_{i=1}^{m}\frac{\partial L_i}{\partial a} \tag{3.43}$$

$$b = \frac{1}{m}\sum_{i=1}^{m} y_i + r\frac{1}{m}\sum_{i=1}^{m}\frac{\partial L_i}{\partial b} \tag{3.44}$$

$$c = \frac{1}{m}\sum_{i=1}^{m} z_i + r\frac{1}{m}\sum_{i=1}^{m}\frac{\partial L_i}{\partial c} \tag{3.45}$$

Taking the latter 3 equations, replacing r with the substitution from Eq1 and using $\partial L_i/\partial a = (a - x_i)/L_i$, $\partial L_i/\partial b = (b - y_i)/L_i$ and $\partial L_i/\partial c = (c - z_i)/L_i$ one gets the 3 equations

$$a = \bar{x} + LL_a = F(a, b, c) \tag{3.46}$$

$$b = \bar{y} + LL_b = G(a, b, c) \tag{3.47}$$

$$c = \bar{z} + LL_c = H(a, b, c) \tag{3.48}$$

The variables $\bar{x}$, $\bar{y}$ and $\bar{z}$ denote the average coordinate from the data points and $\bar{L}$ is the mean value of all L_i. Together with

$$\bar{L_a} = \frac{1}{m} \sum_{i=1}^{m} \frac{a - x_i}{L_i} \tag{3.49}$$

$$\bar{L_b} = \frac{1}{m} \sum_{i=1}^{m} \frac{b - y_i}{L_i} \tag{3.50}$$

$$\bar{L_c} = \frac{1}{m} \sum_{i=1}^{m} \frac{c - z_i}{L_i} \tag{3.51}$$

one can setup an iterative approach with $a_0 = \bar{x}$, $b_0 = \bar{y}$ and $c_0 = \bar{z}$ as initial values. This way the center of the hypothetic sphere is iteratively translated towards the optimal position. The final step is the computation of the radius r, which is equivalent to the mean distance from the newly computed center (a_f, b_f, c_f) to all samples in the point cloud.

$$r = \frac{1}{m} \sum_{i=1}^{m} L_i \tag{3.52}$$

The sphere is now fully defined with a given center point C and a radius r.

Comparison

The fitting parameters for the 4 individual shapes can now be summarized as presented in Table 3.1.

	Sphere	**Cylinder**	**Cone**	**Cuboid**
Points needed for RANSAC	-	3	4	5
RANSAC parameters	center c, radius r	center 1 c_1, center 2 c_2, radius r	center c, apex a, aperture ϕ	corner points $n_1 ... n_8$

Table 3.1: Parameter summary for the individual primitive types.

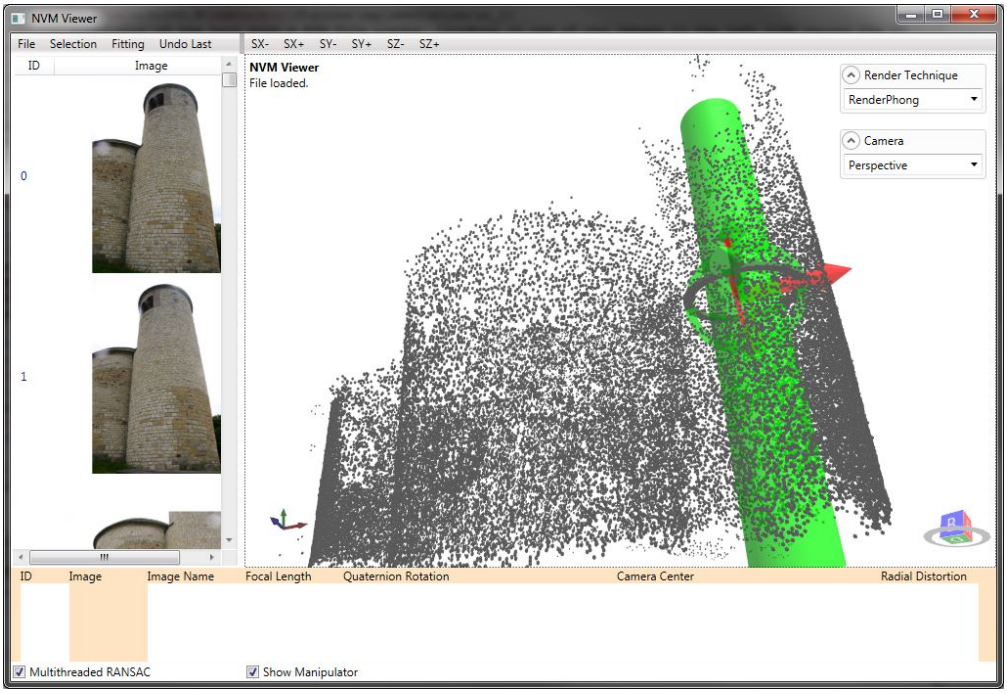


Figure 3.14: The selection tool. The primitive is dragged to the desired region with the help of the image list as a visual guidance and orientation help.

3.8 Interactive Selection and Scene Reconstruction

Datasets from reconstructed scenes usually consist of a combination of multiple primitives. The relationships between the individual elements can be diverse. All sorts of scenarios have to be taken into account, whether the scene is made up of separate objects or a combination of overlapping primitives. It is therefore necessary to implement a tool for selecting regions of the point cloud to be transferred into a geometric primitive. For this purpose an interactive selection feature has been implemented, which is illustrated in Figure 3.14.

For each desired region to be transferred to a primitive the user has the ability to select the appropriate object type, which is added to the scene. A manipulator is rendered onto the shape, so that the user can translate, rotate and scale it roughly to the desired region. The selection of the primitive type is dependant on the desired region of the point cloud to be simplified. If the user wants to transform a box-shaped point set, he chooses a cuboid primitive and drags it to the desired region. When the fitting process is started, all points, that lie inside the selection object, are taken as the input data for the RANSAC and ICP algorithms.

When the fitting process has finished, the newly created primitive is added to the reconstructed scene and the selection shape is removed from the view.

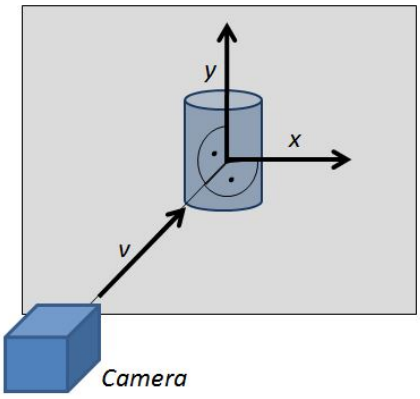


Figure 3.15: The geometry for object translation with the mouse. The 2D movement of the mouse is transformed to a 3D translation in space. The x and y axes of the movement are orthogonal to each other and the viewing axis.

3.9 Navigation

Displaying a scene as a point cloud has some drawbacks when it comes to perception. Because of the lack of faces in the scene, occlusion cannot be conveyed to the viewer. The appearance of an object is only rendered by the means of individual points on its surface, so occluded parts of the scene are visible through the object in the foreground, making it difficult to distinguish between individual parts and connected elements. As already mentioned, the NVM file format does not only store the point cloud, it also contains the camera parameters of each image taken. As a consequence, the fitting tool does not only render the data as a point cloud, but also provides a list of images with corresponding camera data. The user can select the desired viewpoint from the image list and is presented with the corresponding 3D view of the point cloud. This image-based technique provides a convenient way of orientation. For instance, in an architectural scene there may be a series of photos of facades with windows and doors. The location of such details can be hard to find in the point cloud, because of the lack of texture. Figure 3.16 illustrates such a scenario. The location of the church's door is hard to do on the point cloud, but can be easily seen on the photography in the list. The 2D visual representation of the scene therefore supports the user and accelerates the task of navigation.

3.10 Data Output

After the scene has been reconstructed to a certain amount of desired detail, the geometry can be stored for usage in other editors or viewing tools. Wafefront's OBJ file format has been chosen for data storage, since it is well known and commonly used throughout the set of available 3D editing and viewing applications. Listing 3.1 shows an example for a file describing a cuboid. For each reconstructed primitive, 3 elements are stored in the file. First, the object is given a name to be identified within the file. The editor names objects after their type followed by an

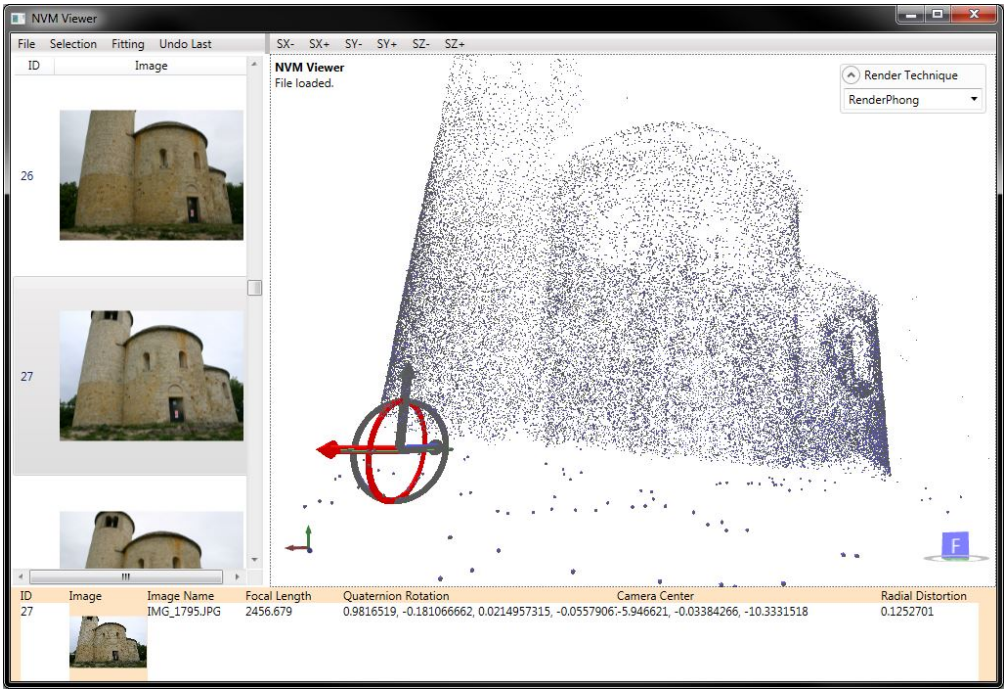


Figure 3.16: Example view of the St. George dataset. The round building part on the right has a window on the front, as well as on the back. The back window is reconstructed in a higher detail and "shines through" to the foreground. This is confusing to the viewer, who might take the samples of the back window as part of the front window. The image of the selected camera view on the left gives a clue of the true position of the window.

ascending number indicating the number of objects of this type. For example, if the scene has two spheres and one cylinder, the objects are named $sphere1$, $sphere2$ and $cylinder1$. After the name definition the object's vertices are listed, one vertex per line. The letter v is followed by the definition of the position consisting of 3 coordinates. Faces are defined in the third part. The letter f is followed by a series of numbers corresponding to the indices of the vertices in the vertex list. This is repeated for all objects in the scene.

Listing 3.1: Sample OBJ ASCII file with a description of a cuboid.

```
o Cuboid  #Name of object

#vertex definitions
v 1.0 -1.0 -1.0
v 1.0 -1.0 1.0
v -1.0 -1.0 1.0
v -1.0 -1.0 -1.0
v 1.0 1.0 -1.0
```

```
v 1.0 1.0 1.0
v -1.0 1.0 1.0
v -1.0 1.0 -1.0

#face definitions
f 1 2 3 4
f 5 8 7 6
f 1 5 6 2
f 2 6 7 3
f 3 7 8 4
f 5 1 4 8
```

OBJ files are stored in the ASCII format, so their content can be opened and edited with standard text processing tools. Subsequent editing, e.g. vertex-level manipulations and texturing, can be done in 3D modeling applications.

CHAPTER 4

Implementation

4.1 Technology

For the implementation of the modeling tool Microsoft C# was used, combined with the Windows Presentation Framework (WPF) for the graphical user interface. The rendering output was developed with the Helix Toolkit [HEL], which is a toolset consisting of various functions for modeling and rendering. It also provides sophisticated features for implementing a 3D view with manipulator objects that were used to realize the modeling tasks. For setting up a kd-tree and solving systems of linear equations the ALGLIB library [ALG] is used.

4.2 Multithreaded RANSAC

The RANSAC phase incorporates the creation and validation of a large number of models, which constitutes high computation costs. In order to accelerate this process, one can harness the multicore architecture of current generation CPUs. The user can choose between two modes for the fitting procedure. The first one executes all RANSAC iterations one after another in a single thread. The second one splits the work into 4 threads and executes them in parallel, taking advantage of the machine's multiple cores. Each thread picks and stores the parameters, as well as the number of inliers of the best fitting model. After all 4 threads have finished their work, the parameters of the model with the highest number of inliers is selected among the 4 candidates. The benefits of this method are presented in the Results section 5.

4.3 User Interface

The graphical user interface mainly consists of a large viewing area for displaying the dataset along with the simplified primitives. The user first loads the desired NVM data via the file menu, which is then loaded into the program. The point cloud is created from the data, as well

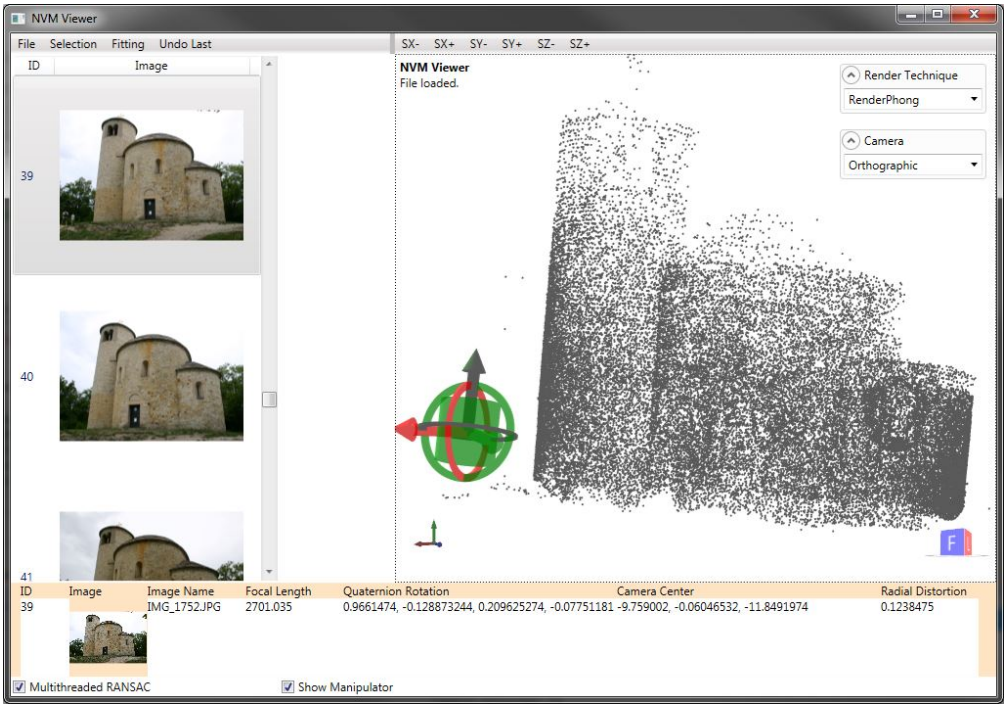


Figure 4.1: Screenshot of the tool with a NVM file loaded. Clicking on one of the images in the left list causes the 3D view to change the camera to the same viewpoint as the image has been shot from.

as the images along with their camera parameters. The view can be translated and rotated with the mouse or via the camera list. Selecting an item from the list causes the camera view to orient itself to the camera parameters given to the chosen image. This is especially useful for orientation within the point cloud, because the user is presented with both the image of the real scene as well as the reconstructed scene from the same point of view. The camera parameters are displayed on the detail section on the bottom of the window along with a thumbnail of the selected image.

Manual navigation through the point cloud can be done with the mouse. Holding the $Ctrl$ key and dragging the mouse rotates the view, whereas the same action with the $Shift$ key performs a view translation. The user can zoom the view by using the mouse wheel or the middle mouse button. The usage of the $Ctrl$ and $Shift$ key instead of different mouse buttons enables the use of the tool on notebooks, whose trackpads may not be equipped with the full set of mouse buttons. Figure 4.1 provides a sample screenshot of the tool with a NVM file loaded.

After adding the desired shape to be fitted to a region in the point cloud, it is visualized together with the manipulator object. This item is used to translate and rotate the selection object to the desired location in the dataset. Dragging the object by clicking on itself causes the mouse movements to translate the selection shape on the plane that is perpendicular to the viewing

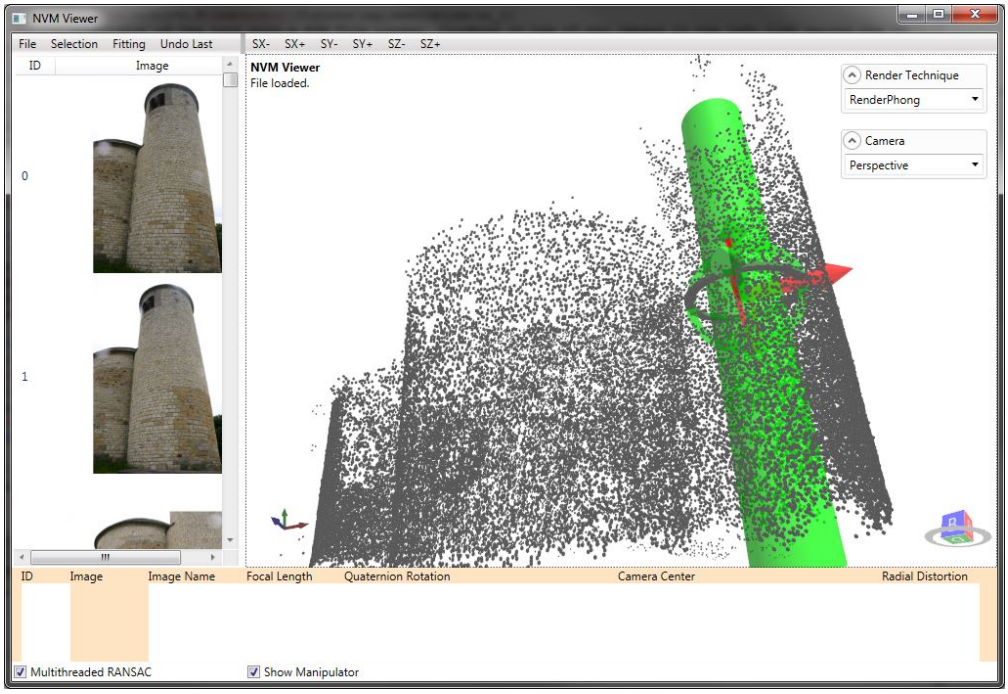


Figure 4.2: Screenshot of the tool with a selection primitive already put into place. The height is already scaled to the height of the tower. Selection objects are rendered semi-transparent to prevent the point cloud to be occluded.

axis. Using the arrows of the manipulator, the user can translate the object on one specific axis only. The circles can be dragged to rotate the shape around its respective axis. The size of the manipulator object is automatically adjusted to the outer dimensions of the selection model. Additionally, it can be hidden from the 3D view to improve clarity. Since the framework does not support scaling with the manipulator tool, this function can be called via the buttons $SX-$, $SX+$, $SY-$, $SY+$, $SZ-$ and $SZ+$ in the menu bar. Depending on the type of object, the scaling buttons are active or not. A sphere has only one degree of scale - the radius, hence only the X buttons are active. The cylinder can be scaled in 2 dimensions (X for the radius and Y for the height), whereas the cone's dimensions can be altered in 3 axes. ((X for the bottom radius, Y for the height and Z for the top radius) The cuboid has also 3 button pairs active, so the user can change width, height and depth. Figure 4.2 depicts a screenshot with a cylindrical selection object already translated to the desired point in the dataset. Height and rotation is already adjusted, only the radius needs to be adapted to the tower's width.

After some fitting is done, the scene gets more and more filled with reconstructed primitives, making it hard to perceive the underlying point cloud. The drop-down menu on the top right offers a series of rendering options for this case, e.g. a wireframe presentation of the entire scene. Selection models and the reconstructed primitives are rendered semi-transparent, so that the surrounding point cloud is always visible without having any occlusions. Placing the selection

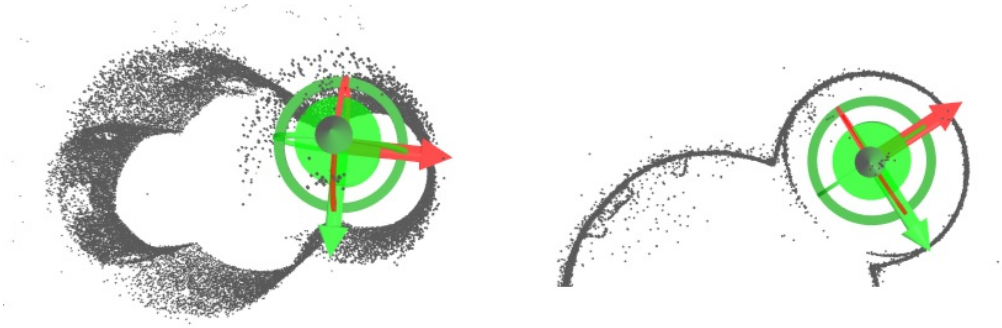


Figure 4.3: A perspective view conveys a correct spatial view of the scene to the user, but placement of selection models is difficult in some situations (left). The outline of shapes in the point cloud can be better perceived with an orthographic camera, allowing the user to correctly place the selectiob object (right).

object in the point cloud can be a complex task, depending on the density of the data. The outline of regions, that resemble a specific shape, can be better perceived in an orthographic view. Figure 4.3 illustrates an example scenario. Placing a cylinder in the center of a ring-shaped outline of a building can be better realized using an orthographic projection, which omits the distortions occuring in perspective views.

After the shape has been roughly translated, rotated and scaled to the desired region, all points lying within the object are gathered to serve as the input for the fitting process. The user can choose the desired primitive type via the Fitting menu. Usually, he selects the same shape as he chose the selection object type.

If the point cloud is very sparse or very dense, it may become difficult to perceive shapes. By using the image list for navigation, the user can easily determine outlines in the point cloud, such as cylindrical towers or box-shaped buildings. This simplifies the modeling process to a great extent and can also be used for a final check of the extracted primitive set.

The reconstructed scene can be stored in an OBJ file via the file menu. This widely supported format can be opened in a variety of modeling and viewing tools for later usage.

CHAPTER 5

Results

The solution was tested in both performance and visual quality of results. The machine used for development and testing is an Apple Macbook Pro with a third-generation mobile Intel Core i5 running at 2.5GHz with 2 cores and 4 GB of RAM. The operating system is Microsoft Windows 7 x64 running Visual Studio 2012.

5.1 Individual Shape Fitting

To test the quality of the presented fitting algorithm, artificially created datasets for the individual primitive types have been created in 3 different versions. Each shape has been generated with an uniformly sampled surface with and without the addition of noise. An additional noisy version with reduced sample count tests the algorithm for sparse point cloud fitting.

Figure 5.1 illustrates the outcome of the sphere fitting procedure. The iterative least squares fitting algorithm produces good results for both regular and noisy datasets. In addition, it is capable of fitting spheres into incomplete datasets, as can be observed in the bottom row.

Figure 5.2 shows the results for the cylindrical shape. It shows the dataset, the RANSAC output and the final output after ICP for all 3 dataset creations. It can be observed, that the cylinder fitting algorithm produces good results for both dense and sparse datasets, as well as noisy versions.

The results for cone fitting can be seen in Figure 5.3. The fitting output is quite similar to the cylindrical version. More noise leads to greater variability in the RANSAC output, which therefore needs to be refined with ICP. A reduction of the sample count increases the chance of finding a well-fitted shape with RANSAC, since the iteration count is not changed.

Cuboids fitting also proved to be stable even with sparse point cloud inputs, as can be observed in Figure 5.4. Again, the introduction of noise leads to not optimally oriented models, which

have to be rotated in the ICP phase. As for cones and cylinders, sparse datasets can also be transformed well.

5.2 Practical Results

This section presents some results for datasets consisting of a combination of primitive shapes, in order to demonstrate the capabilities and performance of the presented approach.

Dataset with Miscellaneous Objects and Comparison

The first real-world dataset has been generated out of a set of 200 photos of a set of objects on a table. All 4 primitive types are covered in this scenario, with the box shape being present 2 times. The dataset together with intermediate steps and the final result is shown in Figure 5.5.

For comparison purposes, the dataset has been artificially recreated. The point cloud and the respective results are depicted in Figure 5.6. One can observe the similarity to the results from the real-world dataset. This proves the robustness of the algorithm against noise and irregularly sampled surfaces.

In addition to visual evaluation, the computing time for the individual fitting procedures has been measured. All processes have been performed in single-threaded, as well as multi-threaded mode. Each individual task has been executed 10 times. The average computation time for both modes can be seen in Tables 5.1 and 5.2.

Shape	Real Scene: Points	Real Scene: Time	Artificial Scene: Points	Art. Sc.: Time
Cuboid	8,039	16.322	5,819	11.393
Cone	4,312	7.119	18,720	25.361
Cylinder	4,320	3.500	9,720	7.222
Sphere	472	0.015	5,184	0.064

Table 5.1: Computation time (in seconds) for single-threaded fitting of individual shapes, comparing real-world and artificial dataset.

Shape	Real Scene: Points	Real Scene: Time	Artificial Scene: Points	Art. Sc.: Time
Cuboid	8,105	8.007	5,819	5.808
Cone	4,320	3.232	18,720	15.152
Cylinder	4,320	2.500	9,720	3.114

Table 5.2: Computation time (in seconds) for multi-threaded fitting of individual shapes, comparing real-world and artificial dataset. Since the fitting process for sphere does not contain RANSAC, there are only results for single-threaded computation.

Several things can be observed. Obviously, the computation time increases with the number of samples in the point cloud. Since some procedures require the traversal of the samples more

than once, e.g. the cuboid fitting algorithm, the calculation time does not rise in an exactly linear way with the number of points, but it is an approximative assumption for the computation time. Multi-threaded RANSAC fitting takes about half the time needed for single-threaded execution. The performance gain is dependant on the capabilities of the CPU.

The sphere is fitted in the shortest amount of time. This is due to the fact, that it uses a different, less complex algorithm for reconstruction. Cuboid, cone and cylinder incorporate a RANSAC procedure in the first half of their algorithm, but the sphere reconstruction uses a single iterative approach without the generation of complex models in each step.

Second best is the cylinder. This is due to the fact, that the creation of a hypothetical model is straightforward. The circle plane, center and surface normal are easily computed, and the check for inliers tests all samples with the same radius. In contrast to the cylinder, the cone's model setup is more complex. In addition to the base plane the apex needs to be computed. This process needs an additional point, which has to be projected onto the plane, and several equations need to be created. The checking procedure involves trigonometric calculations, which need more time to finish.

Lastly, the fitting of cuboids takes the longest time. This is due to the fact, that the computation for this type of shape is the most complex one. It has to construct 3 planes for finding the initial parameters for a cuboid. After that, the 3 parallel planes have to be determined.

The results of the cuboid, cone, and cylinder are highly dependent on the random selection for the MSS. The iteration count for RANSAC has been set to 8,000 in order to get a result with high sufficient quality. However, in some cases RANSAC does not find a proper fitting. In this case the whole fitting process is repeated until a result is found, which increases the computation time. The ICP step presents only a minor part of the overall computation time, since it does not involve model creation, which is a resource-hungry task.

In terms of fitting accuracy, an important property of RANSAC has to be discussed. Due to the fact, that points for hypothetical models are selected in a random manner, there is no guarantee, that the algorithm finds a feasible result. To avoid or minimize the risk of creating poor results, one can increase the number of model generations. As stated above, the iteration count in the presented tool has been set to 8,000, which is a compromise between a high probability of a qualified result and computation time. To decrease the latter, the RANSAC procedure gets the selection model's main axes, which are compared to the orientation of each created hypothetical model. If the difference is too high, all further calculations will be skipped, and the next random selection is done. This accelerates the RANSAC procedure and therefore leads to shorter fitting times. A possible further improvement for the future would be a an additional preprocessing step, that takes sample points from certain positions (e.g. points with opposite locations in the point cloud) instead of a purely random selection.

Real-World Dataset

For testing the image-based approach, the St. George image dataset [STG] has been taken and transformed into a NVM file containing both pictures and the reconstructed point cloud. It

consists of a church, that is comprised of 3 cylinders with varying dimensions and half-domes as roof elements. Doors and windows are integrated into the walls. Figure 5.7 illustrates the transformation from the raw point cloud into an approximative set of primitives. The regions of the individual building parts are selected piece by piece and transformed into a polyhedral primitive - in this case a cylinder.

It can be observed, that the fitting process produces feasible results and is robust against noise, that is unavoidable in datasets coming from image-based reconstruction techniques. Even special features of the building, like doors and windows, have no influence, that could lead to a bad fitting of the main cylindrical structure. The dataset consists of 50,757 samples. The central cylinder is made up of about 20,000 points and takes about 6 seconds to be transformed into a polyhedral primitive. The building's highest tower is made up of about 16,000 points, which takes approximately 4 seconds for fitting a cylinder to it. Leaving out the samples from the surrounding area, the 13,800 points from the smallest building part take about 3.5 seconds to be transformed to a primitive.

Artificial Dataset

To test the algorithm with scenarios of more complexity, an additional artificial dataset has been created, that incorporates all primitive types in several positions, dimensions and orientations. Figure 5.8 shows the artificially created point cloud, intermediate modeling steps and the result viewed in both the modeling tool as well as Blender. [BLE] It took about 10 minutes to model the entire scene.

The orthographic view is a useful tool for aligning the selection models to the desired region in the point cloud. As can be seen in the resulting model, all parts of the buidling could be well transformed into polyhedral primitives. However, since all objects are processed individually, special constraints are not considered in the fitting process. This may lead to situations similar to the one with the cylindrical tower, that has a spherical roof. Since both the cylinder and sphere have been fitted individually, the sphere does not follow the cylinder's axis, resulting in a shifted rooftop. This could be avoided by implementing an axis-alignment feature.

However, the overall result is promising and proves, that the approach is well applicable for datasets such as the the presented ones.

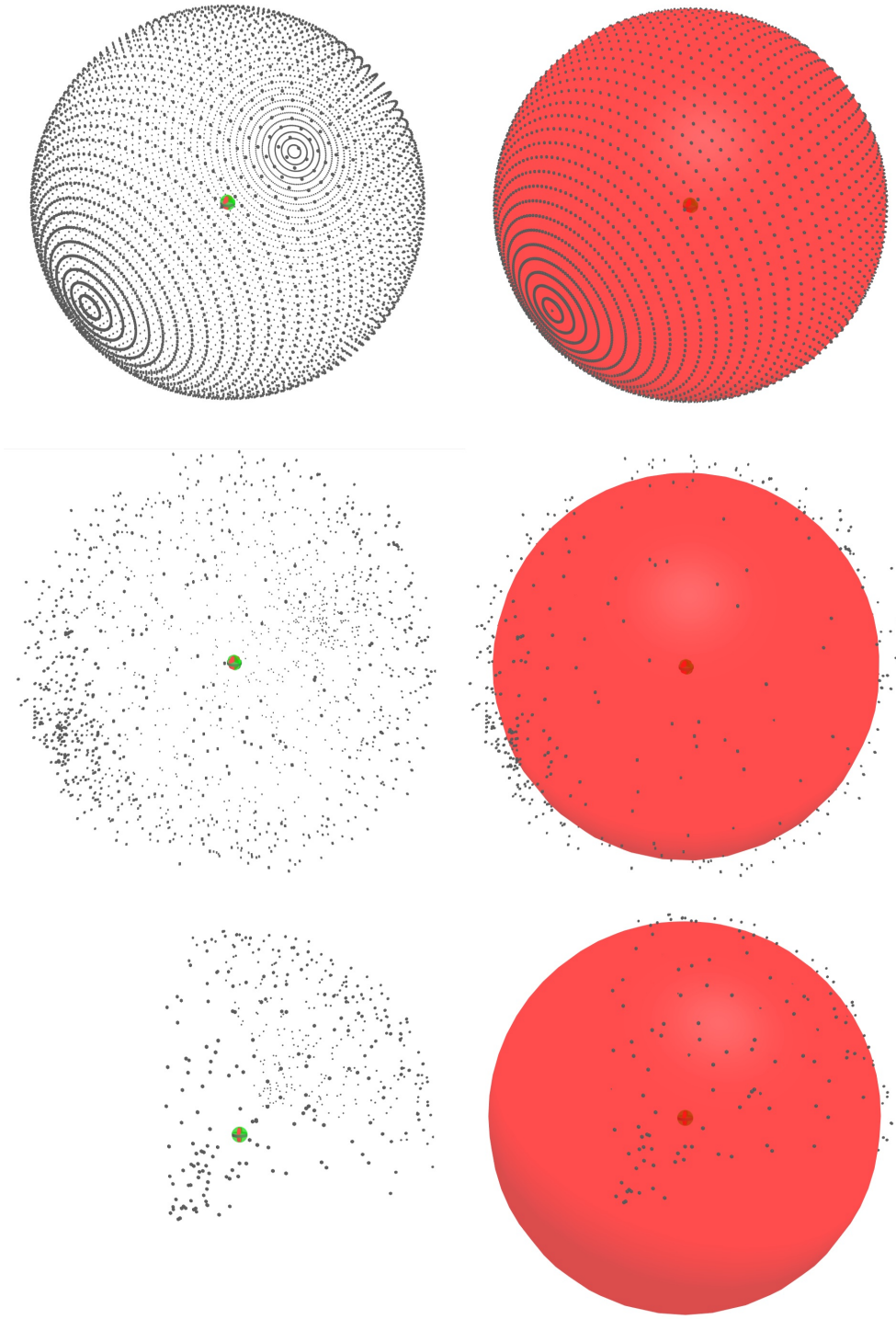


Figure 5.1: Artificial datasets (left column) and results (right column) for sphere fitting. Both regular (top row) and noisy (middle row) datasets can be transformed well. The algorithm is also capable of fitting spheres to incomplete datasets (bottom row).

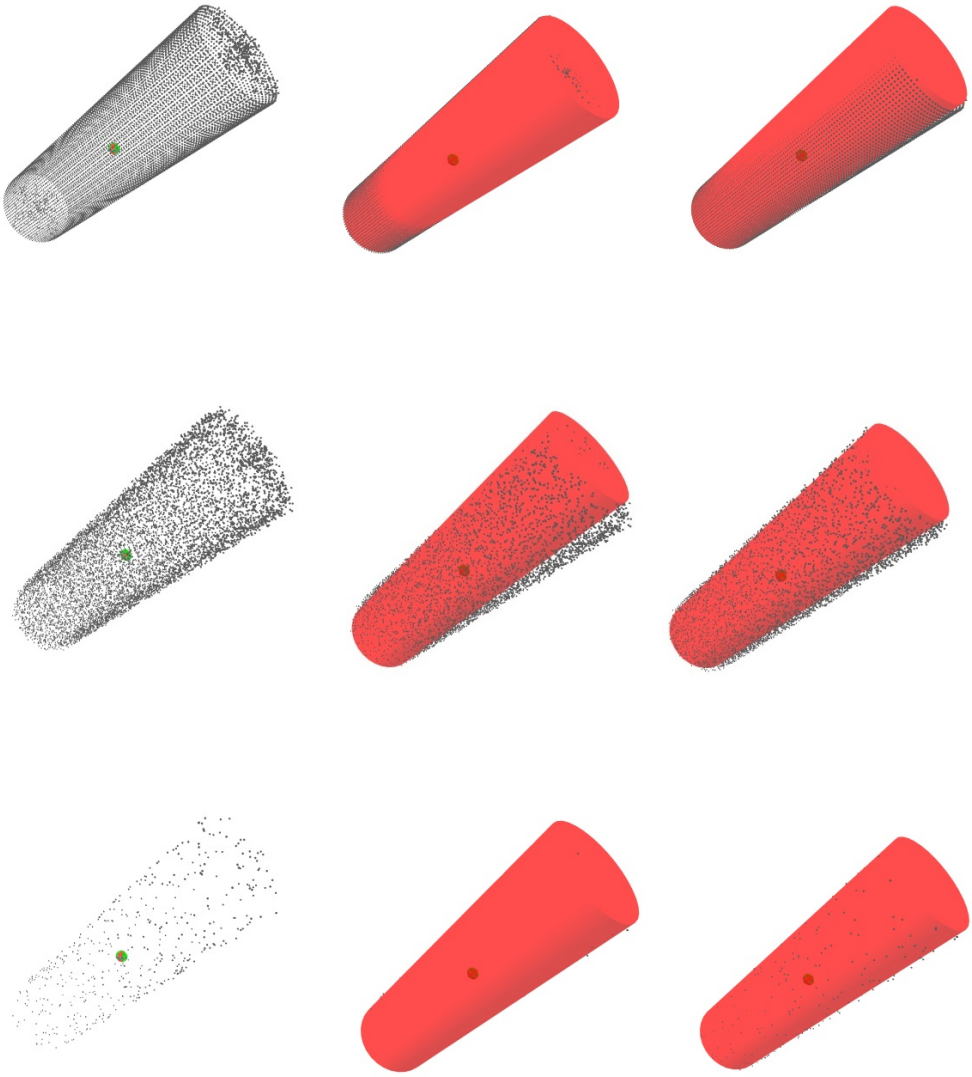


Figure 5.2: Artificial datasets (left column), RANSAC output (middle column) and results after ICP (right column) for cylinder fitting. Both regular (top row) and noisy (middle row) datasets can be transformed well, although noisy datasets require a refined orientation with ICP in most cases. Sparse point clouds can also be fitted well by the algorithm (bottom row).

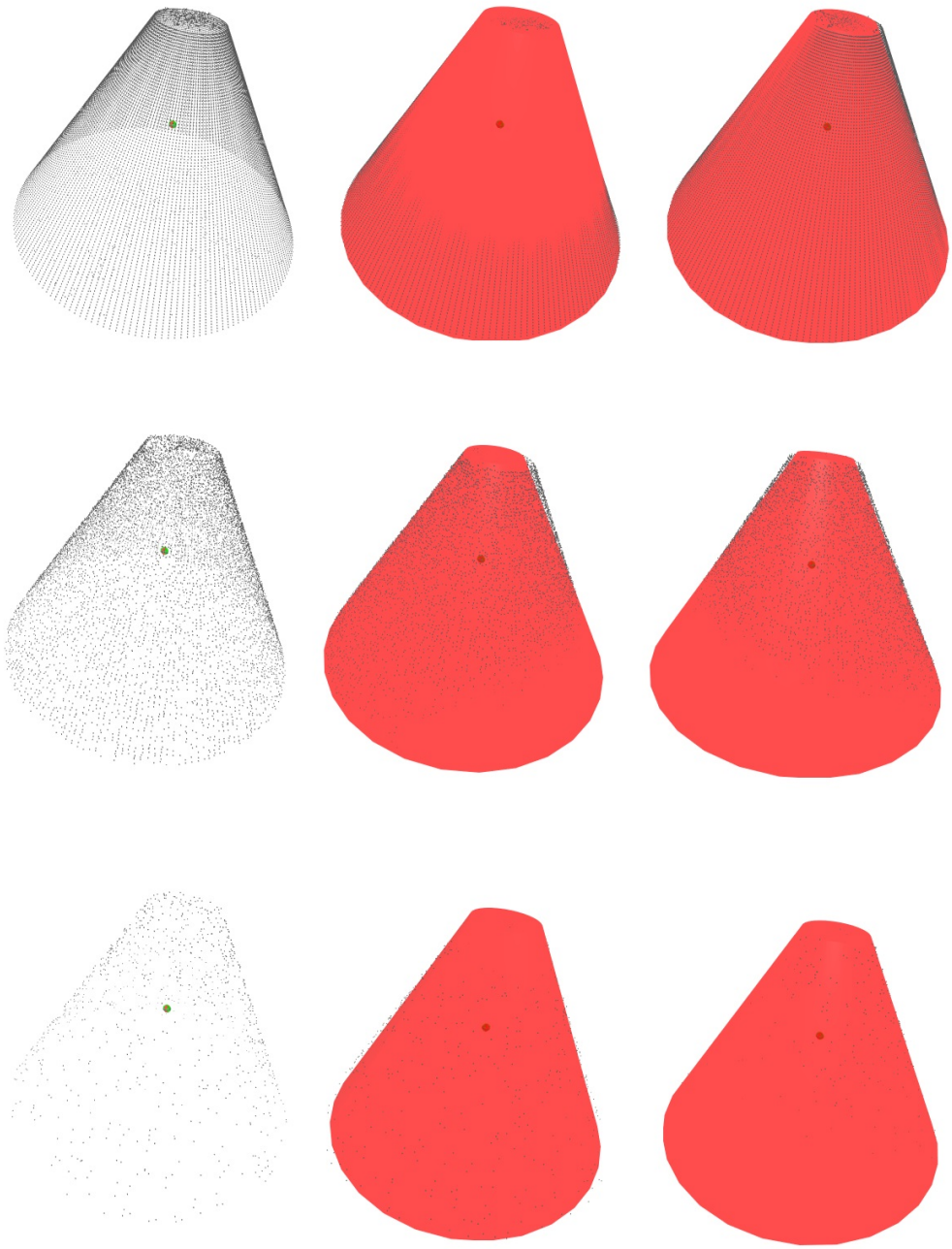


Figure 5.3: Artificial datasets (left column), RANSAC output (middle column) and results after ICP (right column) for cone fitting. Both regular (top row) and noisy (middle row) datasets can be transformed well. Similar to cylinder fitting, noisy datasets require a refined orientation with ICP in most cases. Sparse point clouds increase the chance of finding well-fitted initial model with unchanged RANSAC iteration count (bottom row).

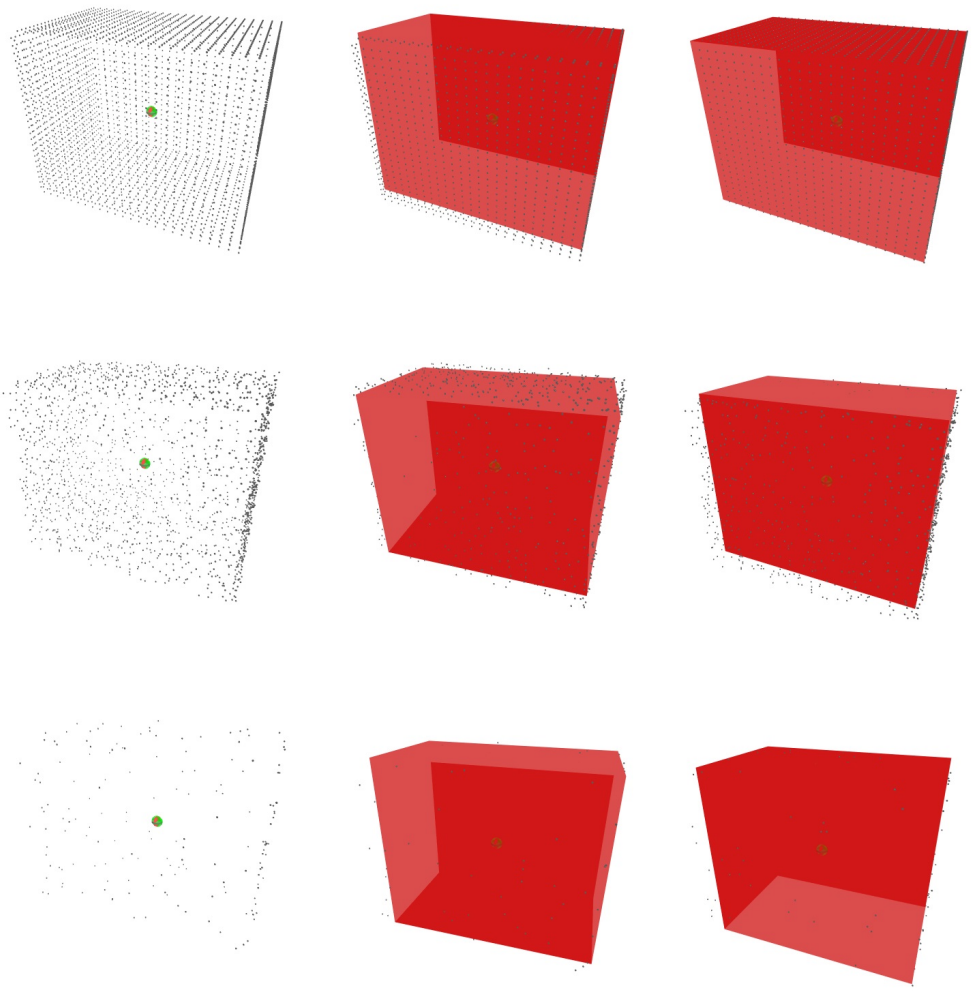


Figure 5.4: Artificial datasets (left column), RANSAC output (middle column) and results after ICP (right column) for cuboid fitting. The outputs for regular (top row), noisy (middle row) and sparse datasets (bottom row) can be interpreted the same way as with cylinders and cones.

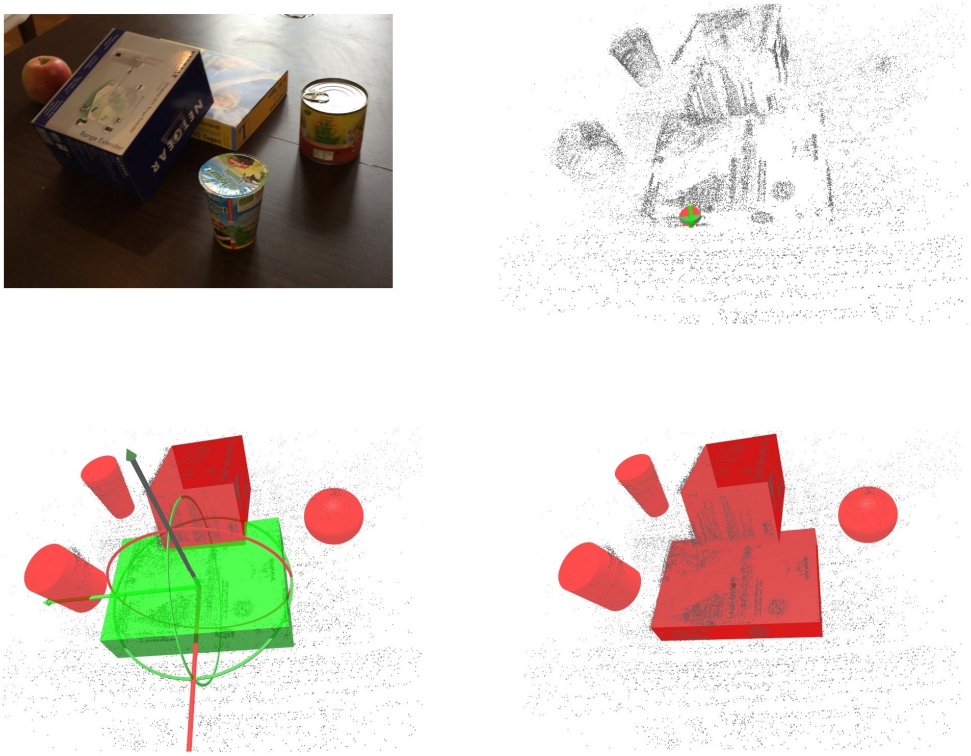


Figure 5.5: Example fitting results for a real-world dataset containing miscellaneous objects on a table (top left). The point cloud contains a significant amount of noise and irregularly sampled surfaces (top right). Primitives are fitted piece by piece using the 4 available selection models (bottom left) until the whole scene has been approximated with geometrical shapes (bottom right).

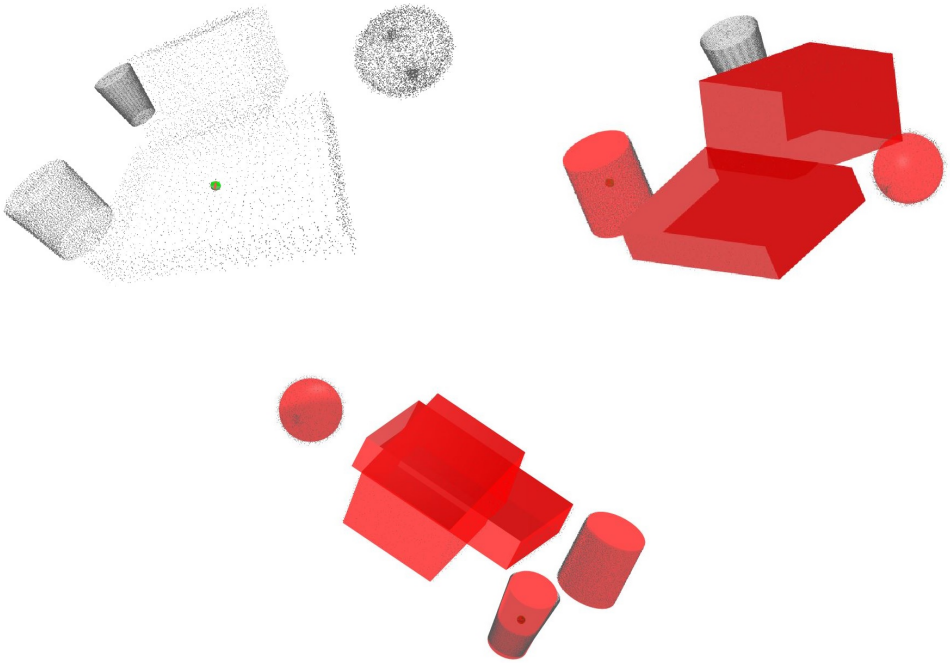


Figure 5.6: Fitting results for the artificially recreated dataset. The lack of greater noise in the dataset (top left) can be observed. The result (top right, bottom) is similar to the real-world counterpart, proving the robustness of the algorithm against noise and irregularity.

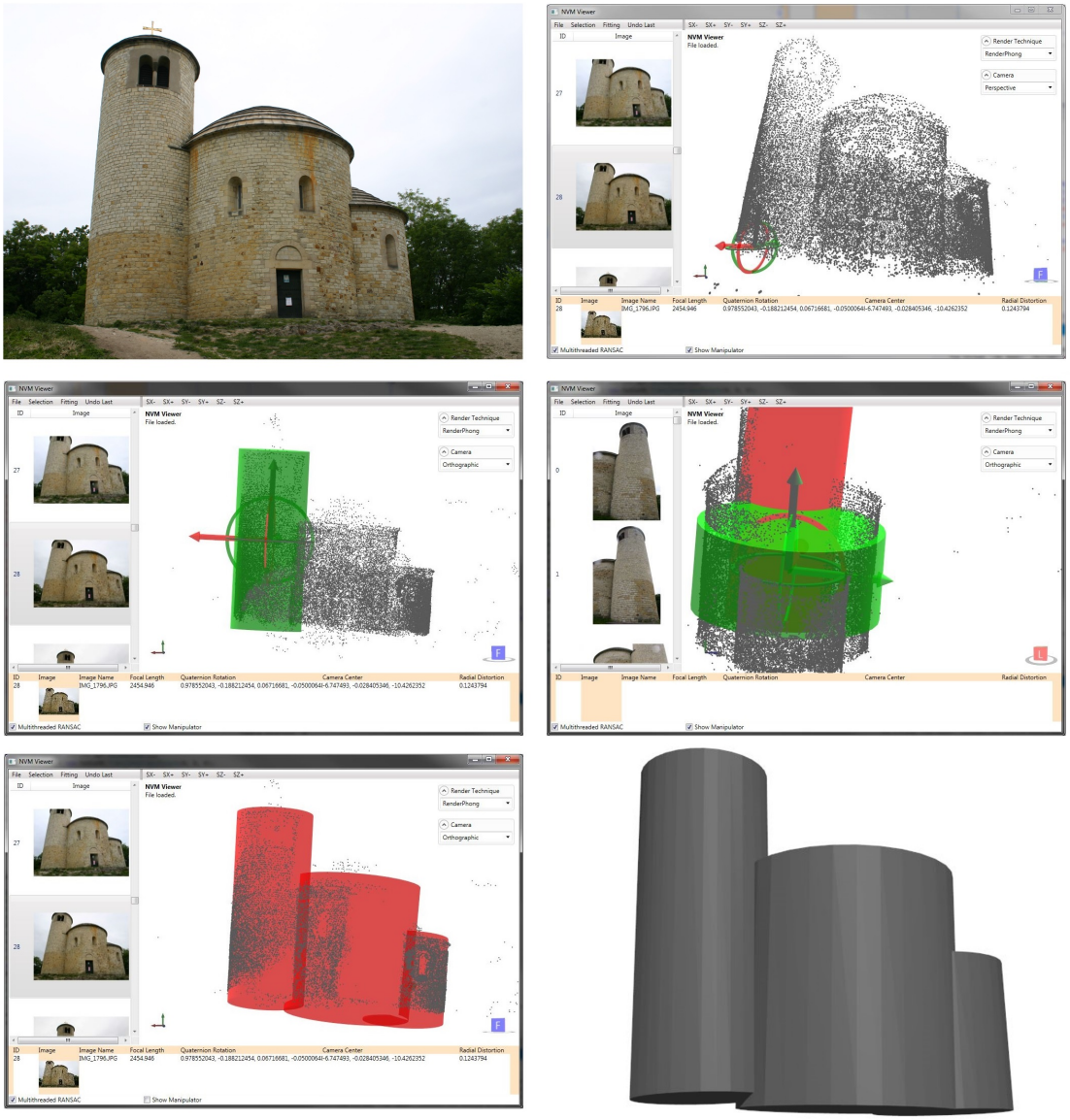


Figure 5.7: The St.George NVM file transformed into an approximative set of polyhedral primitives. The real object (top left) mainly consists of 3 cylinders. First, the data is loaded into the tool (top right). A cylindrical selection object is placed and scaled to the desired dimensions within the point cloud (middle left). After the first object has been fitted (shown in red), the second part of the building is prepared to be converted (middle right). In the end all 3 cylindrical parts have be transformed to an approximative primitive (bottom left). The scene can be stored as an OBJ file and viewed and further processed with other tools (bottom right).

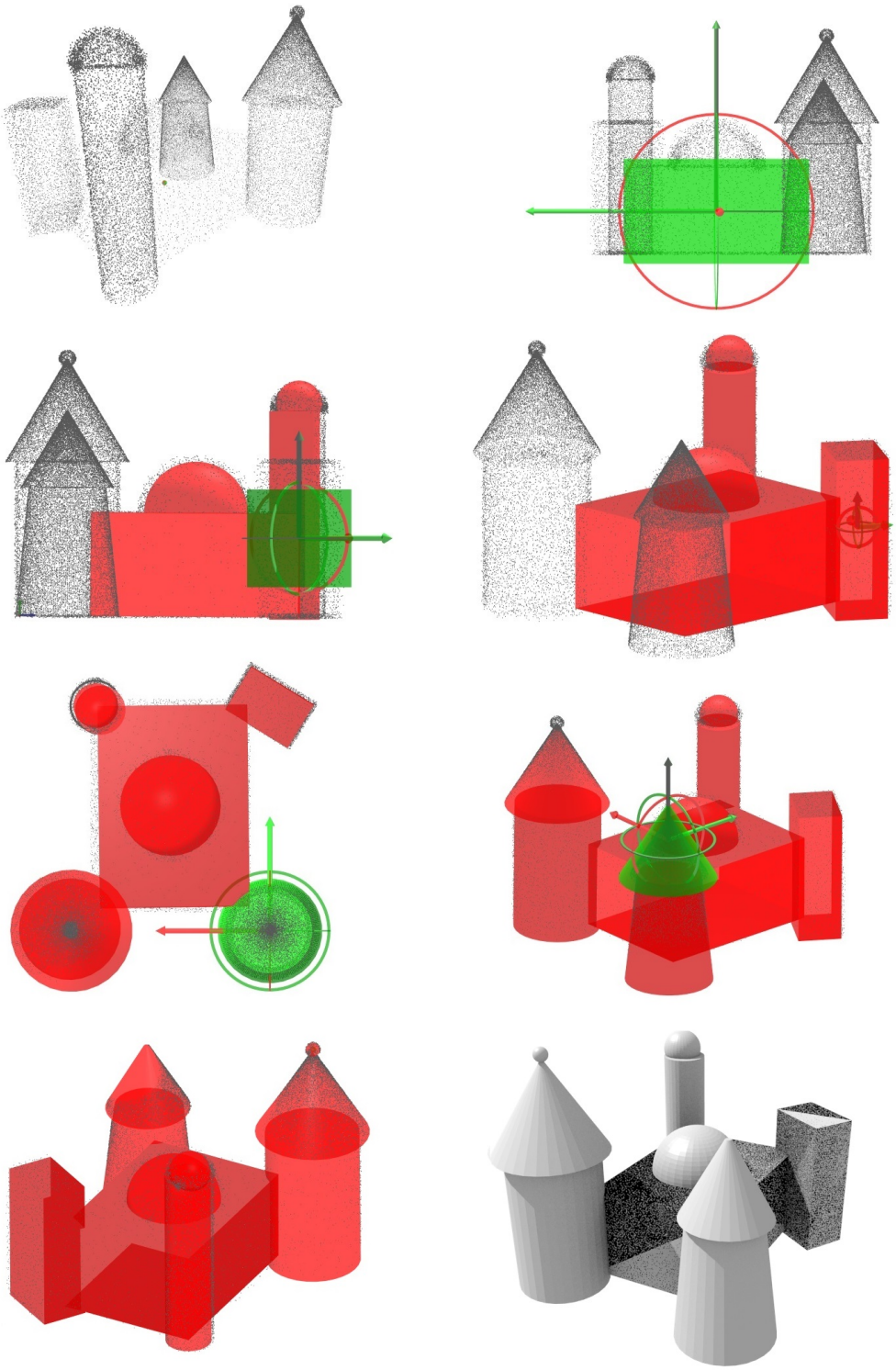


Figure 5.8: Artificial dataset of a castle (top left) with intermediate modeling steps (top right and middle 2 rows) and resulting approximative scene viewed in the modelerer (bottom left) and Blender [BLE] (bottom right).

CHAPTER 6

Conclusion

6.1 Synopsis

This thesis presented a new approach for converting a 3-dimensional unstructured point cloud into simplified geometry consisting of polyhedral primitives. The user can interactively choose the desired shape and region, in which the corresponding primitive is fitted by an automated process. The whole reconstructed scene can be stored as an OBJ file for later usage in other modeling applications. A possible application field for the presented solution is the transformation of complex data into simplified geometry, that can be used as proxy geometry in numerous application.

Point cloud representations raise a number of problems and challenges to the user. Navigation and orientation gets more difficult and time-consuming as the number of samples increases. Occlusion is not supported, since the objects are not made out of solid meshes. This makes it difficult to distinguish between objects in the foreground and other elements in the background. Shades and textuers are also lost due to the lack of solid geometry to be shaded.

Scenes of man-made objects (e.g. urban areas) usually can be approximated quite well by a set of geometric primitives. The simplicity of such data is lost, when it is stored in a point cloud containing a large number of samples following the objects' surfaces. In order to transform such data into an easily perceivable collection of geometric primitives, the user needs a conversion tool.

The proposed solution for this problem works as follows: The input for the presented approach is a NVM file generated out of a set of uncalibrated images. The file contains both the reconstructed 3D point cloud, as well as camera parameters of the individual source pictures. The presented algorithm fits polyhedral primitives to manually selected regions in the point cloud using RANSAC. The orientation of the newly established object is then refined with ICP by

minimizing the sum of squared distances from the primitive's surface to the surrounding point cloud region.

First, the user has to choose the region, which has to be transformed into simplified geometry. He can do this by selecting the appropriate primitive shape and placing it at the desired location in the point cloud. Switching the perspective camera to an orthographic representation helps to better detect outlines of individual regions, such as the diameter of a cylindric part. The user can apply rotation and scaling until the selection matches the region to be transformed. The next step is the automatic fitting process, that is a combination of RANSAC and ICP. In the first phase a predefined number of hypothetical models is generated and tested for feasbility. The parameters of the model, that has the lowest sum of squared distances to the point cloud, is selected. Depending on the outcome of RANSAC, the result may not have an optimal rotation, which is corrected in the ICP phase. In this last step, the object is rotated to an orientation that minimizes the error against the point cloud even further. The newly fitted primitive is added to the set of already created others which can be stored into an OBJ file. This widely used filetype can be viewed and manipulated in a broad range of modeling software for further usage.

In order to gain feasible results in the automatic fitting procedure, the number of RANSAC iterations has to be set to an appropriate value. 8,000 iterations have been chosen to gain sophisticated results and to minimize possibly weak fittings. The multi-core architecture of current-generation processors can be exploited by running RANSAC in 4 different threads, each running through 2,000 iterations to find the best fitting model. Depending on the underlying capabilities of the CPU, the number of threads can be easily adjusted in the implementation to get an even higher acceleration. After all threads have been executed, the model with the most inliers is selected among the 4 candidates for further ICP processing. Since the user roughly places selection models over the desired region, it is possible to use this information for further improvements. To reduce the computation time, RANSAC omits the evaluation of randomly created models, whose axes differ significantly from the orientation of the selection model aligned by the user. The calculations for about 10 - 20% of the randomly created models can be skipped this way.

Using the point cloud's source images as an additional means for navigation provides a useful tool during the modelin process. Images do not only convey occlusion and spatial properties of the scenery to the user, but also surface properties, e.g. shade and texture, which may convey some additional helpful visual impression. Having loaded the camera parameters assigned to the selected image, the view automatically changes to the viewpoint, where the image has been taken from. This provides an intuitive way of navigation and is a useful tool for the final inspection of the approximative primitive set. To extend this idea, a possible future addition would be the rendering of the image in the 3D view's background with the point cloud being rendered in front of it.

The presented fitting approach is highly expandable and fits well into application areas, where simple and effective representations of complex 3D datasets are desired. An example scenario is the creation of simple geometry out of a reconstructed urban scene for viewing applications on mobile devices with restricted graphics capabilities.

6.2 Conclusion

The solution presented in this thesis is aimed at scenarios, where point clouds reconstructed from images of man-made objects need to be transformed into simple proxy geometry consisting of a set of polyhedral primitives. Several observations were made during the research work.

The first one concerns the reliability of RANSAC. Since this algorithm relies on the random selection of sample points, there is no absolute guarantee, that a feasible result is produced. One way to overcome this, is the increase of the iteration count. In the presented solution, 8,000 RANSAC iterations are executed to find a feasible model. This decreases the chance of a resulting bad fit to nearly zero. However, in some seldom cases, the fitting has to be started again to find a better model.

Another observation is the huge benefit of using multiple threads for RANSAC iterations. This cuts the computation time for the whole fitting process to about half of the time consumed by a single-threaded run. One can harness the multi-core structure of current-generation CPUs to dramatically increase the performance of such algorithms.

A big challenge during the implementation was the definition of the threshold parameters, that define inliers and outliers of the individual hypothetical models in the RANSAC stage. Depending on the primitive type, different values had to be defined. Setting the values too high may lead to unfeasible results. A threshold being too low may not generate inliers at all, so that no model can be found. The defined parameters assured feasible results for the tested datasets. However, although the distance measures for the inlier computations have been designed with various point cloud scales in mind, datasets with greater variety in scale and noise may need adapted threshold parameters. Future work on this approach may include a resizing algorithm to bring the loaded dataset to an equal scale, allowing the threshold parameters to be fixed.

The amount of noise does not influence feasible results to a certain extent. A possibly suboptimal orientation of a RANSAC model is correctly oriented in the subsequent ICP stage. The presented approach is therefore robust against noise and irregularly sampled surfaces, which often occur in image-based reconstructions.

6.3 Future Work

Future work on this approach can be done in 3 main aspects. Experiments with test data have shown, that RANSAC sometimes has issues with finding a suitable model. The randomness of the procedure bears the risk, that feasible parameters may not be found at all. Increasing the iteration count may decrease the chance of a bad result, but naturally leads to longer execution times. Future research could be done on the development and testing of efficient evaluation algorithms for hypothetical models.

The set of avaible polyhedral primitives could be extended as well. This can be done by not only adding entirely new types (e.g. ellipsoids), but also implementing truncated primitives, like half-domes or cuts of cylinders and cones.

Lastly, the interaction techniques of the tool could be improved for faster interaction and more complex modeling tasks. This may include a means for manipulating position, scale and rotation of fitted primitives or improving the image-based navigation by rendering the selected image to the background of the 3D view according to the camera parameters. Another possiblity could be the use of dragged rectangles for a 2D selection of point cloud samples. This would reduce the amount of time needed for definition and alignment of selection models.

Bibliography

[ALG] Alglib. http://www.alglib.net/. Accessed: 2013-09-28.

[ASF+13] Murat Arikan, Michael Schwärzler, Simon Flöry, Michael Wimmer, and Stefan Maierhofer. O-Snap: Optimization-Based Snapping for Modeling Architecture. *ACM Transactions on Graphics*, 32(1):6:1–6:15, January 2013.

[BLE] Blender. http://www.blender.org. Accessed: 2013-11-28.

[BM92] Paul J. Besl and Niel D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[Cha] Changchang, Wu. VisualSFM : A Visual Structure from Motion System. http://ccwu.me/vsfm/. Accessed: 2013-09-28.

[CR99] Roberto Cipolla and Duncan Robertson. 3D models of architectural scenes from uncalibrated images and vanishing points. In *Proceedings 10th International Conference on Image Analysis and Processing*, volume 0, pages 824–829, Venice, 1999. IEEE Comput. Soc.

[DTC04] Anthony R. Dick, Philip H. S. Torr, and Roberto Cipolla. Modelling and interpretation of architecture from several images. *Int. J. Comput. Vision*, 60(2):111–134, November 2004.

[DTM96] Paul Debevec, Camillo Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 11–20, New York, NY, USA, 1996. ACM.

[Ebe08] Eberly, David. Least Squares Fitting of Data. http://www.geometrictools.com, 2008. Accessed: 2013-09-28.

[FB81] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.

[FCSS09] Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. Manhattan-world stereo. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1422–1429, Miami, FL, June 2009. IEEE.

[Fur] Furukawa, Yasutaka. Clustering Views for Multi-view Stereo (CMVS). `http://www.di.ens.fr/cmvs/`. Accessed: 2013-09-28.

[Gar09] Sergio Garcia. Fitting Primitive Shapes to Point Clouds for Robotic Grasping. Master's thesis, Royal Institute of Technology, 2009.

[HEL] Helix 3D Toolkit. `http://helixtoolkit.codeplex.com`. Accessed: 2013-09-28.

[Hor87] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.

[LZ06] E. Langetepe and G. Zachmann. *Geometric data structures for computer graphics*. Ak Peters Series. A K Peters, 2006.

[MLS+10] Przemyslaw Musialski, Christian Luksch, Michael Schwärzler, Matthias Buchetics, Stefan Maierhofer, and Werner Purgathofer. Interactive Multi-View Façade Image Editing. In *Vision, Modeling, and Visualization (VMV)*, pages 131–138, Siegen, Germany, 2010. Eurographics Association.

[Mus09] Przemyslaw Musialski. Point Cloud to Model Registration. Technical report, VRVis Research Center, Vienna, 2009.

[MWA+12] Przemyslaw Musialski, Peter Wonka, Daniel G. Aliaga, Michael Wimmer, Luc van Gool, and Werner Purgathofer. A Survey of Urban Reconstruction. In *EUROGRAPHICS 2012 State of the Art Reports*, EG STARs, pages 1–28. Eurographics Association, May 2012.

[NSZ+10] Liangliang Nan, Andrei Sharf, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Smartboxes for interactive urban reconstruction. *ACM Trans. Graph.*, 29(4):93:1–93:10, July 2010.

[PHYH06] Helmut Pottmann, Qi-Xing Huang, Yong-Liang Yang, and Shi-Min Hu. Geometry and Convergence Analysis of Algorithms for Registration of 3D Shapes. *International Journal of Computer Vision*, 67(3):277–296, March 2006.

[PLH02] Helmut Pottmann, Stefan Leopoldseder, and Michael Hofer. Simultaneous Registration of Multiple Views of a 3D Object. *Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Vol. XXXIV, Part 3A, Commission III*, pages 265–270, 2002.

[PLH04] Helmut Pottmann, Stefan Leopoldseder, and Michael Hofer. Registration without ICP. *Computer Vision and Image Understanding*, 95(1):54–71, July 2004.

[PY11] Charalambos Poullis and Suya You. 3D Reconstruction of Urban Areas. In *2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*, pages 33–40, Hangzhou, May 2011. IEEE.

[RK13] Irene Reisner-Kollmann. *Reconstruction of 3D Models from Images and Point Clouds with Shape Primitives*. PhD thesis, Vienna University of Technology, 2013.

[Sch10] Ruwen Schnabel. *Efficient Point-Cloud Processing with Primitive Shapes*. Dissertation, Universität Bonn, December 2010.

[SSS+08] Sudipta N. Sinha, Drew Steedly, Richard Szeliski, Maneesh Agrawala, and Marc Pollefeys. Interactive 3d architectural modeling from unordered photo collections. *ACM Trans. Graph.*, 27(5):159:1–159:10, December 2008.

[STG] St.George Dataset. `http://cmp.felk.cvut.cz/projects/is3d/Data.html`. Accessed: 2013-05-04.

[VAB10] Carlos A. Vanegas, Daniel G. Aliaga, and Bedrich Benes. Building reconstruction using manhattan-world grammars. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 358–365, San Francisco, CA, USA, June 2010. IEEE.

[VAB12] Carlos A. Vanegas, Daniel G. Aliaga, and Bedrich Benes. Automatic extraction of Manhattan-World building masses from 3D laser range scans. *IEEE Transactions on Visualization and Computer Graphics graphics*, 18(10):1627–1637, October 2012.

[vdHDT+07] Anton van den Hengel, Anthony Dick, Thorsten Thormählen, Benjamin Ward, and Philip H. S. Torr. A shape hierarchy for 3d modelling from video. In *Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, GRAPHITE '07, pages 63–70, New York, NY, USA, 2007. ACM.

[VKH06] Vivek Verma, Rakesh Kumar, and Stephen Hsu. 3d building detection and modeling from aerial lidar data. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2213–2220, 2006.

[XFT+08] Jianxiong Xiao, Tian Fang, Ping Tan, Peng Zhao, Eyal Ofek, and Long Quan. Image-based façade modeling. *ACM Transactions on Graphics*, 27(5):161:1–161:10, December 2008.

[ZN10] Qian-Yi Zhou and Ulrich Neumann. 2.5d dual contouring: a robust approach to creating building models from aerial lidar point clouds. In *Proceedings of the 11th European conference on computer vision conference on Computer vision: Part III*, ECCV'10, pages 115–128, Berlin, Heidelberg, 2010. Springer-Verlag.