

Object Modelling Framework for RGBD data

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Alexander Svejda

Matrikelnummer 0625934

an der

Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze

Mitwirkung: Dr. Aitor Aldoma

Wien, 17. April 2015

(Unterschrift Verfasserin)

(Unterschrift Betreuung)

Object Modelling Framework for RGBD data

MASTER'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering & Internet Computing

by

Alexander Svejda

Registration Number 0625934

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze
Assistance: Dr. Aitor Aldoma

Vienna, 17. April 2015

(Signature of Author)

(Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Alexander Svejda
Mellergasse 18, 1230 Wien

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 Verfasserin)

Acknowledgements

I want to thank all people supporting me throughout the process of writing this thesis.

First, I want to thank my advisor Markus Vincze for making it possible to write my thesis at the Automation and Control Institute. My sincere gratitude goes to Aitor Aldoma for his great support, motivation and constructive feedback. I want to thank him for his great guidance and the effort he put into supporting me throughout the course of this thesis. I also want to thank Johann Prankl and Thomas Mörwald for their assistance and help.

Besides that, I want to thank all the people supporting Open Source Software used in this project. Special thanks go to the community developing the Point Cloud Library, as it greatly simplified development of the framework.

Finally, I want to thank my parents, Helmut and Susanne, for their immense support throughout my whole studies.

Abstract

With the fast development and improvement of range cameras in recent years, also the number of applications using these sensors increased dramatically. However, besides some simple analysis of range data, there is also a need for detailed and complete reconstruction of 3D scenes and objects in order to use the data provided by the camera in a simple way. The 3D objects can be directly used in various applications, e.g. in the field of robotics for object recognition and tracking or as input for 3D printers.

The purpose of this thesis is to create a framework for reconstruction of 3D objects using range and image data provided by RGBD cameras. This thesis consists of three parts:

First, different algorithms are reviewed and implemented for reconstructing and texturing 3D objects from range images. Because reconstruction is quite a complex task, for achieving best results it is crucial to implement and evaluate as many algorithms as possible. We examine algorithms for different parts in the process of reconstruction, starting with registration of point clouds from different views, optimizing the registered point clouds, generating a mesh from the aligned point clouds and finally texturing the resulting mesh using image data provided by the camera. Finally, we also investigate reconstruction of objects using multiple sequences from different object poses.

The algorithms examined in the first part are combined into a single, highly configurable and extensible framework. It provides a simple and extensible solution for a great range of different reconstruction scenarios. A simple-to-use graphical user interface has been developed for configuring and running the framework on both recorded scenes and live camera streams. Based on the framework a command line application has been developed allowing unattended analysis and batch processing of multiple objects.

The last part of the thesis analyses the reconstruction results, the algorithms and their configurations. Because the reconstruction process consists of several steps, where each step uses the calculation results of the previous step, also the interaction between the algorithms and the impact of changing one algorithm in the pipeline is evaluated. The results are compared to other popular object reconstruction frameworks, highlighting the strengths and weaknesses of each framework.

Kurzfassung

Durch die schnelle Weiterentwicklung von Tiefenbildkameras in den letzten Jahren sind auch die Anwendungen, welche diese Sensoren nutzen, gestiegen. Abgesehen von einfachen Anwendungen gibt es jedoch das Bedürfnis, exakte und vollständige 3D Szenen und Modelle zu rekonstruieren. Solche 3D Modelle finden vor allem im Bereich der Robotik für Objektwiedererkennung und im Bereich der 3D Drucker Anwendung.

Das Ziel dieser Arbeit ist die Erstellung eines Software-Frameworks zur Rekonstruktion von 3D Objekten unter Verwendung von Farb- und Tiefeninformation von RGBD Kameras. Die Arbeit besteht aus drei Teilen:

Zuerst werden unterschiedliche Algorithmen für die Rekonstruktion und Texturierung von 3D Modellen untersucht und umgesetzt. Da der Rekonstruktionsprozess relativ komplex ist, ist es wichtig, möglichst viele und unterschiedliche Algorithmen zu analysieren und umzusetzen. Wir untersuchen verschiedene Algorithmen, beginnend bei der Registrierung von Punktwolken aus unterschiedlichen Blickwinkeln, Optimierung der registrierten Punktwolken, Erzeugung eines Polygonmodells und schließlich der Texturierung der Modells unter Verwendung der Farbbilder, welche von der Kamera aufgenommen werden. Außerdem behandeln wir auch die Rekonstruktion von Objekten, welche in unterschiedlichen Stellungen aufgenommen wurden.

Die untersuchten Algorithmen werden anschließend in ein einfach konfigurierbares und erweiterbares Framework eingefügt. Ziel ist eine einfache und erweiterbare Lösung für eine große Bandbreite an Rekonstruktionsszenarien. Eine einfach zu nutzende grafische Benutzeroberfläche wurde entwickelt, welche die Benutzung des Frameworks sowohl auf aufgenommenen Sequenzen als auch auf Live-Streams ermöglicht. Basierend auf dem Framework wurde auch eine Konsolenapplikation entwickelt, welche unbeaufsichtigte Analysen und Verarbeitung von mehreren Objekten gleichzeitig ermöglicht.

Der letzte Teil der Arbeit beschäftigt sich mit der Analyse der Ergebnisse der Rekonstruktion, der Algorithmen und deren Konfigurationen. Da der Rekonstruktionsprozess aus mehreren Schritten besteht, die aufeinander aufbauen, werden auch Abhängigkeiten zwischen diesen Schritten untersucht. Die Ergebnisse werden mit anderen aktuellen Applikationen verglichen, um Stärken und Schwächen der verschiedenen Lösungen aufzuzeigen.

Contents

1	Introduction	1
2	State of the art	5
2.1	Data acquisition	5
2.2	Object reconstruction	6
2.3	Modelling frameworks	8
3	Object reconstruction	9
3.1	Basics	9
3.2	Pairwise registration	12
3.3	Segmentation	15
3.4	Multiview refinement	15
3.5	Alignment of multiple sequences	17
3.6	Noise modelling and post processing	19
3.7	Surface reconstruction	20
3.8	Texturing	21
4	Framework & User Interface	25
4.1	Object Modeller Library	25
4.2	Graphical User Interface	35
4.3	Command Line Application	37
5	Evaluation	39
5.1	Evaluation methods	39
5.2	Results for different setups	40
5.3	Comparison to other frameworks	50
6	Conclusion	53
	Bibliography	55

Introduction

This thesis describes the development of a highly configurable and extensible framework targeted at 3D object reconstruction using RGBD input data for a broad range of reconstruction scenarios.

There has been much research on range cameras in the last years. Starting with Microsoft's Kinect ¹ these kind of cameras have become affordable for the broad mass. This resulted in a general availability of these sensors. Also Google announced recently that they are working on an embedded range camera for tablets and smartphones ².

Range cameras are a special type of camera where a second sensor is used to provide range information for each pixel, besides the standard color information. There are different techniques for determining this value, where the Time-Of-Flight approach is the most common today. Besides that, different approaches for acquiring 3D data existed for longer time. Laser scanners provide exact data of the analysed object, though they lack to provide color information of the object. The technology of different data acquisition devices will be discussed in detail in Chapter 2.

The general availability of this technology resulted in a wide range of applications using this additional information. Because the first sensor was shipped with Microsoft's Xbox 360, most of the applications were targeted at the gaming market in the beginning. Microsoft used a simple analysis of depth information to detect persons in front of the camera, resulting in new interaction possibilities for users of their console. However, there is still a demand for an exact reconstruction of 3D scenes and objects from range images in order to create more complex applications using range images. In the field of robotics, the generated models can be used for object recognition or object tracking [1] [2]. Another application is to use the reconstructed models as input for 3D printers [3]. Besides that the models can be used in any graphical 3D application, which makes our framework interesting for game designers, film studios or architects. Although the quality of the reconstructed models might not be sufficient in these cases, it can be used as basis for further improvement or for fast prototype modelling.

¹<http://www.microsoft.com/en-us/kinectforwindows/>

²<https://www.google.com/atap/projecttango/>

The main focus of this thesis lies on developing an easily extendable framework and user interface for object reconstruction using a set of standard, robust modelling algorithms.

We propose to design the reconstruction process as pipeline consisting of several components. Each component solves one particular problem of the reconstruction process. This structure has various advantages. First, splitting up the problem simplifies the overall organization and allows to solve each step individually. Second, this approach allows to easily exchange the algorithm used in one component without affecting other parts of the pipeline. Figure 1.1 shows the structure of the pipeline.

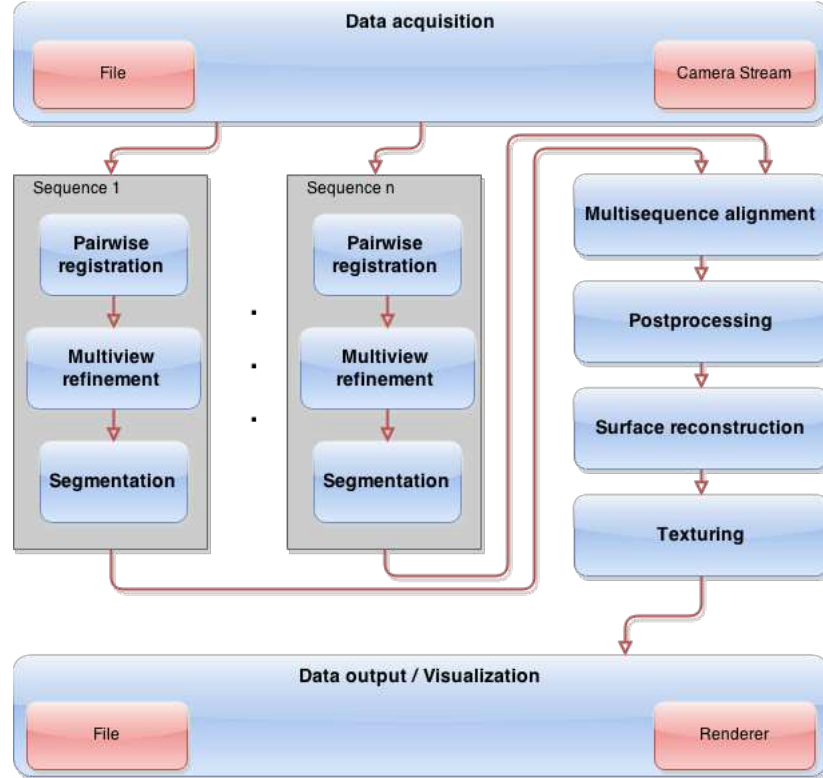


Figure 1.1: Structure of the reconstruction pipeline

First, the source data will be acquired. Both recorded sequences from files and live streams from cameras can be analysed. Usually, the data obtained is represented as a set of 3D points with associated color information. We will refer to this structure as point cloud in the remainder of this thesis.

The purpose of the pairwise registration is to find a transformation matrix between adjacent source point clouds. This allows to transform all input images into a single coordinate system by applying the transformation matrices subsequently in the right order.

The (optional) multiview refinement step is responsible for minimizing the registration error of the transformation matrices from the first step. As the registration in the first step is calculated



Figure 1.2: Virtual scene containing reconstructed objects

between adjacent input point clouds, usually an error is accumulating resulting in a drift between the first and the last source point cloud. By analysing the transformation on a global level, this error can be reduced, resulting in a better overall registration.

As the purpose of this thesis is to reconstruct only a single object and not the complete scene, the next step consists of separating the object from the background. As a side effect, this step will also increase performance of the following components, as the data size is decreased dramatically.

Multisequence alignment is used only for scenarios where the object has been recorded in different object configurations. This allows to reconstruct the whole object, as in one sequence, some parts of the object are not visible and always occluded, i.e. the bottom of the object.

In the postprocessing step, the model is optimized for further processing. Optimization involves merging all source point clouds into a single point cloud. It consists of removal of duplicate points and optimizing data of the final point cloud by smoothing noisy data. This step is crucial in order to generate smooth models and exact texture maps in the next steps.

Next, the point cloud is converted into a 3D polygon mesh, as this is the primarily used format when dealing with 3D object data. Finally, the RGB data from the source images is used to create a single texture map, which can be applied to the mesh.

Different algorithms for each of these components are reviewed and evaluated. Finally, the most interesting algorithms are implemented in our framework. Figure 1.2 shows a virtual scene containing some of the objects that have been reconstructed using our framework. Figure 1.3 shows the visualization of different reconstruction steps in our pipeline.

The remainder of this thesis is organized as follows. Chapter 2 describes the state of the art of object reconstruction. Algorithms that solve parts of the object reconstruction problems as well as complete frameworks for object reconstruction are presented. Chapter 3 describes the

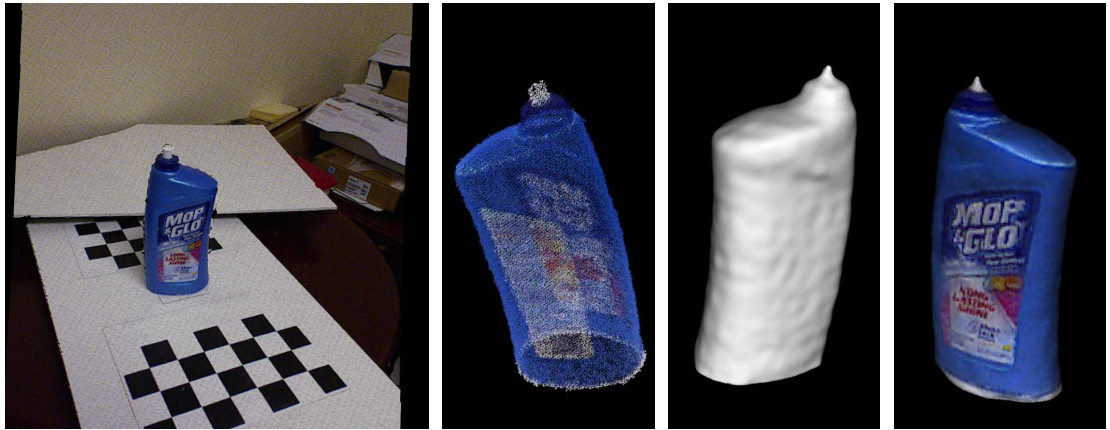


Figure 1.3: Reconstructed floor cleaner bottle

algorithms used in the framework in detail. The different steps involved in object reconstruction are described. For each step, at least one method will be presented. Chapter 4 describes the structure of the implemented framework, as well as the implementation details for each method used. The graphical user interface is presented, as well as a command line application suitable for unattended analysis and batch processing. Chapter 5 evaluates the results obtained using the reconstruction framework. The different methods and parametrization of algorithms implemented are compared against each other. The results are also compared to other reconstruction frameworks. The thesis closes with a conclusion, interpreting the results obtained. Also ideas for future work are presented.

State of the art

This chapter gives an overview of state of the art reconstruction algorithms and frameworks. Also related technologies for data acquisition are presented.

2.1 Data acquisition

Basically, the presented framework and algorithms can deal with any kind of point cloud data. There are different technologies for acquiring such 3D voxel data. Some of the algorithms presented in chapter 3 originated from analysing and visualizing voxel data obtained by CT devices. Another common approach for acquiring 3D point data are laser scanners. A laser point is emitted onto the object point by point. The distance to the camera is then calculated by triangulating the laser point at the object with the position of the laser and the camera. Although this approach delivers high quality results, scanning progress can be slow compared to other techniques. In addition, this approach does not provide color information of the scanned object. Also, such scanning devices are usually quite expensive.

A different approach especially used in recent years are RGB-D cameras. This is a special kind of camera, that not only captures RGB images, but also the depth information for each pixel. The most common technology used for this purpose are Time-Of-Flight cameras. The distance for each pixel is calculated by measuring the time the light needs to be reflected back to the camera. Alternatively, also the phase shift of the light can be used to measure or improve measurement of the distance. An advantage of this approach is that the scene is recorded all at once, compared to the laser scanner, which records point by point. This makes it suitable for scenes with moving object and also for real time applications, as these cameras usually deliver many frames per second.

Another approach for obtaining depth data is to project an infrared light pattern and determine the distance by measuring the distortion of the pattern. Although this technique existed for a longer time, it has become interesting in recent years, mainly due to the fact that Microsoft released an RGB-D camera packaged with their gaming console Xbox 360. This step made such cameras affordable for the broad mass.

Another approach is to use a stereo camera setup. It uses two fixed cameras, which are extrinsically and intrinsically calibrated. By analysing the resulting images from each camera, usually using some kind of edge detection and correlation algorithm, the distance of each point to the camera can be approximated by finding corresponding points in each image. However, the quality of results strongly depend on the scene, e.g. the distance can not be estimated if there is not enough color information in the images. Furthermore, this approach introduces additional complexity, resulting in an additional source of errors.

2.2 Object reconstruction

In this chapter, common methods for each of the components in our proposed pipeline will be presented. Also, recent approaches and current research in this area are mentioned.

There has been intensive research in the field of object reconstruction over the last years, especially pushed through the broad availability of RGBD cameras in recent time. There are many different algorithms solving parts of the object reconstruction process, which will be presented in this chapter.

The pairwise registration is the most crucial step in our object reconstruction pipeline, as all other components rely on accurate transformation matrices obtained in this step. Various methods exist for finding a rigid transformation between two point clouds with unknown initial poses for different scenarios, using either 2D image data, 3D point data or both.

One of the most common method is the Iterative Closest Point algorithm (ICP), which works on 3D point data. An overview of different variants of the ICP-algorithm can be found in [4]. It uses 3D point cloud data to find the transformation matrices. Basically, corresponding points in two point clouds are analysed to estimate a transformation matrix between them. Although the results of this algorithm are usually very good, it relies on an initial alignment of the pointclouds, as the search for corresponding points is done using a nearest neighbour search from the source point.

Another approach for aligning two point clouds involves analysis of 2D image data. Different techniques exist for estimating transformations using this information. A relatively simple class of methods are marker based approaches. A special marker is added to the registration setup, which can be tracked in an easy way. The transformation can then be calculated by using the position of the marker obtained in the previous step. For example, checkerboards can be easily and accurately detected. This however requires to have control over the registration setup, which might not be the case in all scenarios. An advantage of this approach is that it can be computed in a very fast and efficient way.

A more complex approach uses a general purpose camera tracker. It calculates features from the source images, which can then be tracked between two frames. A broad range of feature detection algorithm exist: Harris [5], FAST [6], SIFT [7] or SURF [8]. A feature detection approach specifically targeted for point cloud analysis is ImGD [9]. For transformation estimation, using SVD [10] is a commonly used approach. Another technique specifically developed for point cloud data is RGB-D Slam System [11]. The camera tracking approach is generally applicable for most registration setups, though it also introduces some constraints. First, the modelled object has to provide enough visual structure in order to find enough features for tracking. Sec-

ond, the transformation between the two considered frames must not be too large, as this might be problematic for tracking the camera position.

To refine the alignment and reduce errors, a technique called multiview refinement [12] can be used. Assuming that during the pairwise registration, an error accumulates over time, this method aims at reducing this error by refining the transformations on a global level.

For conversion of the pointcloud to a 3D polygon mesh, two commonly used approaches exist. Marching cubes [13], initially intended for visualizing CT scans, divides the model into cubes with a certain resolution, generating polygons for each cube depending on how it intersects with the object surface. Different variants of this algorithm exist [14] [15].

A more complex approach is Poisson reconstruction [16]. The goal of this method is to calculate an indicator function, which defines whether a point lies inside the object or not. The authors of the paper describe a technique to approximate the indicator function by reducing the problem to a standard poisson equation.

A different approach targeted especially at noisy source data is Surfel based geometry reconstruction [17]. It generates small planar patches at points in the point cloud, called surfels. Markov Random fields are used to optimize the resulting mesh considering two terms: the parallelism between adjacent surfels and the amount of overlap. The orientation of neighbouring patches and their overlap influence the resulting polygon mesh.

A common approach for segmenting the modelled object from the background is to split segmentation into two steps. First, the plane, where the object resides, is calculated. This can be done by fitting a plane into the point cloud using RANSAC [18]. Another method would be to use a checkerboard to detect the plane, where the object stays on. This technique is especially interesting for scenarios, where the checkerboard is used for pairwise registration anyway. The second step is responsible for separating the object from the plane and the background. Euclidean cluster extraction [19] can be used to solve this problem. It uses a method similar to a flood fill algorithm to obtain objects above the plane. A nearest neighbour search is performed iteratively, adding all neighbours within a specified range to the cluster. As a result, this method not only separates objects from the background, but also creates individual clusters for each object.

Various methods exist for estimating a rigid transformation between multiple sequences of a modelled object recorded from different object poses. Considering the methods presented so far, ICP could be used to align two sequences. This however, requires an initial alignment estimation, as ICP only works, if the source point clouds are close to each other. Usually, this process is split up in four steps. First, keypoints and features are calculated in either the 2D images or from the 3D data. Next, corresponding points in source and destination point cloud are matched, where bad correspondences are rejected. Third, a transformation using the correspondence information is estimated. Finally, these steps are repeated until the alignment error is below a certain threshold.

Although texture map generation in multiview reconstruction scenarios is a relatively new research area, there are a handful of interesting methods suitable for our framework. [20] describes a multiband blending approach for generating high quality texture maps. Visual seams at texture patch boundaries are removed by blending textures from different input images. [21] extend the method by improving the mapping algorithm, also taking visibility and occlusion into

account. Furthermore, they propose a method for removing specular highlights from the modelled object. Another interesting approach is described by [22]. The authors observed, that in multiview scenarios, texture information is redundant across different input images. Using this fact, they propose a method capable of generating texture maps with higher resolution than the original source images. Also [23] describes an interesting approach, segmenting the mesh into associated clusters before applying their texturing algorithm.

2.3 Modelling frameworks

As described in the last section, there are many algorithms partially solving the problem of object reconstruction. There are however only a few complete frameworks covering the whole reconstruction process of a 3D model.

One of the most widely known frameworks is Kinect Fusion [24] [25] ¹. This is not a complete object modelling framework, as it consists only of tracking the camera position and registering each frame in a global model. No texturing is involved, and it is not capable of modelling a single object either.

Two applications for complete object reconstruction will be evaluated in this thesis. ReconstructMe ² is a (commercial) real time 3D object reconstruction application. By analysing a live camera stream, it delivers a reconstructed and textured 3D polygon mesh in real time. For segmentation, it uses a quite simple approach. The bounding box around the object has to be defined either by the user in their application, or by placing a marker into the registration setup before starting the recording process.

Another tool for object modelling is the In-Hand Scanner ³ included in PCL (Point Clouds Library). It uses only pairwise registration (ICP) for aligning point cloud data. Segmentation is done based on a bounding box. As this scanner is intended to be used with objects held in the hand of the user, the hand is segmented using color information. Texturing is also not considered in this application.

¹<http://msdn.microsoft.com/en-us/library/dn188670.aspx>

²<http://reconstructme.net/>

³http://pointclouds.org/documentation/tutorials/in_hand_scanner.php

Object reconstruction

This chapter presents algorithms for the different steps involved during object reconstruction. First, some basic algorithms will be presented, which are often used or needed for object reconstruction, or which are crucial for other algorithms presented to work. Then, methods will be described to align multiple point clouds into a single point cloud. Techniques will be presented to reduce the alignment error and improve quality of the merged point cloud. Afterwards, the conversion of the point based representation of the model into a 3D polygon mesh will be discussed. Finally, a method for texturing the resulting mesh from the input RGB data will be explained.

3.1 Basics

This subsection describes some important basic data structures and algorithms involved when analysing 3D point cloud data.

Point cloud

When working with 3D point data, it is usually represented using a point cloud. This is a simple set of points in 3D space. Each point has 3D coordinates and color values assigned initially after recording. One can distinguish between organized and non organized point clouds. Organized means, that point data is arranged in an image like structure, where each point can be accessed through 2D coordinates. This is usually the case for point cloud data acquired by RGBD cameras. This property is important for some algorithms presented in this thesis, as some operations are much faster when working on organized point clouds. When applying certain operations on the point cloud, e.g. merging two point clouds, the organization of points is lost. Besides the position and color of the point, also other data types can be associated with a point. The most important to mention here is the normal of the point, which is used by many algorithms presented.

Normal estimation

Two methods for estimating normals in a point cloud will be presented in this chapter. For organized point clouds, the integral image normal estimation can be used to efficiently estimate the normals for each point in the point cloud. If point cloud data is not organized, or the quality of the integral image normal estimation is not sufficient, a different approach will be presented. It is slower regarding computational performance, but yields better results and does not rely on the point cloud being organized.

Integral Image Normal Estimation is a fast and efficient way to estimate normals in a point cloud, however quality of the normals is worse compared to other techniques. The technique presented is based on the paper [26]. This method first calculates an integral image using the depth values of each point in the input image, thus it relies on the input data being organized. An integral image is created by summing the value of all pixels between the current pixel and the origin. It allows to efficiently calculate sums of rectangles in the image. The image is then smoothed to reduce noise from the input image. The smoothing area is chosen based on the depth values of each point using two criteria: First, the size of the smoothing area is dependent on the distance of the point to the camera, as usually points that are far away from the camera have a worse signal-to-noise ratio. Second, areas with large depth changes are removed from the smoothing area, which prevents smoothing over object borders. After smoothing the image, the normals for each point can be calculated. For each point, one vector between the left and the right neighbour and another vector between the upper and lower neighbour are calculated. The normal of the point is obtained by calculating the cross product of these two vectors.

A different approach gives better results compared to the Integral Image Normal Estimation and also works on unorganized point clouds. It is based on a paper by [27]. The idea is to find a plane tangential to the surface at the position of the point. This can be formulated as a least-squares plane fitting estimation problem using the neighbourhood of the point. It can be solved by analysing the eigenvectors and eigenvalues of the covariance matrix of the neighbouring points using Principal Component Analysis as described in [26]. The problem of this approach is, that the direction of the normal can not be determined using this approach. However, when the camera view is known, calculation of the direction is simple, as the normals always point towards the camera.

RANSAC

RANSAC [18] stands for Random Sample Consensus. It is a method for fitting a mathematical model (e.g. a plane) into a set of data points. This set consists of inliers and outliers. It is an iterative approach and works as follows: First, a random subset is taken from the set of input points, which represent hypothetical inliers. Next, the model is fitted into these hypothetical inliers. Using the aligned model, the other data points are now tested against the model to check for two cases: If a data point is close enough to the model according to some confidence parameter, it is also considered as inlier and added to the set of hypothetical inliers, otherwise it is defined as outlier. Afterwards, an error metric is evaluated, measuring the quality of the model, depending on the number of inliers compared to the size of the data set. If the quality of

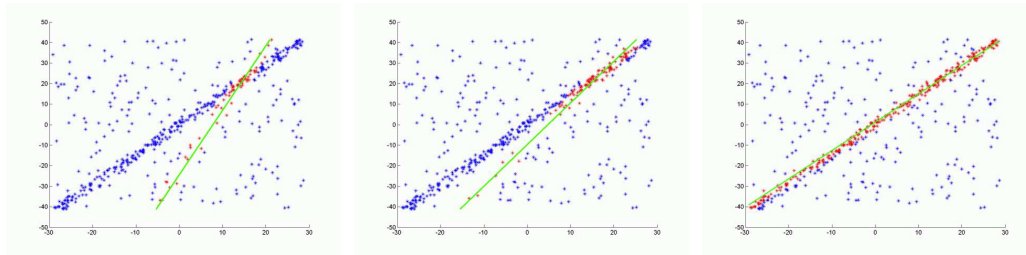


Figure 3.1: Different iterations of RANSAC¹

the model is good enough, the model is reestimated using all hypothetical inliers, and the error metric is evaluated again.

This procedure is repeated either a fixed number of times or until some confidence threshold has been reached. Each iteration either rejects the model or estimates a refined model with using a certain error metric. The best model found is kept and represents the result of the algorithm.

This method provides accurate and robust estimation of the model parameters, however the quality of results strongly depends on the number of iterations, thus also affecting computational performance of the algorithm. Figure 3.1 shows an example for 2D space, where a line is fitted to a data point set.

Efficient point cloud representation

Most algorithms presented need to find certain points in the point clouds in an efficient way. Especially neighbourhood searches are performed by several methods. Therefore it is important to manage point clouds in an efficient way. Two data structures will be presented: Octrees and KD-Trees.

An octree can be seen as extension of a binary tree for 3D space. The root node is represented by a cube containing all points in the point cloud. It is subdivided into eight child nodes each also represented by a cube, where each child node contains only points that lie within this cube. Usually, octrees are initialized with a certain depth depending on the point cloud data size. Figure 3.2 illustrates the organization of an octree.

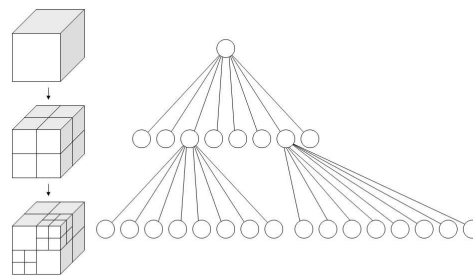


Figure 3.2: Illustration of an octree²

¹<http://de.wikipedia.org/wiki/RANSAC-Algorithmus>

A k-d tree is similar to an octree, with the difference that space is not partitioned uniformly and uses a binary tree for organization. Thus it is better suited for data sets with varying density. At each level, the data set is split along a specific dimension. The point used for splitting is the mean point in the dataset, Subsequent levels are splitted along the other dimensions. Figure 3.3 illustrates a k-d tree for 2D space.

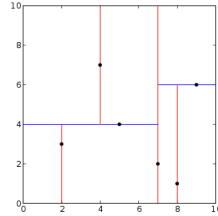


Figure 3.3: Illustration of k-d-tree in 2D space ³

3.2 Pairwise registration

The purpose of the pairwise registration is to find an estimation of the rigid transformation between two pointclouds. Two methods will be presented in this section. ICP [28] [29], which uses 3D point data for alignment, and a feature based camera tracker, which uses 2D features for estimating a transformation.

Iterative closest point

Iterative closest point algorithm is a well known and widely used method for aligning point clouds. It is suitable for generic input data, though registration results may vary depending on the input dataset. Furthermore, this algorithm needs an initial guess of the transformation that is close to the real transformation in order to work. There are many variations of this algorithm as described in [4]. This section gives an overview of the basic idea of iterative closest point algorithms.

Most of the ICP variants can be divided into six steps:

- Select a subset of the points from the source clouds.
- For each selected point in one point cloud find a corresponding point in the other point cloud.
- Assign a quality value to each correspondence according to a certain weighting scheme.
- Reject correspondences with low quality.

²<http://de.wikipedia.org/wiki/Octree>

³http://en.wikipedia.org/wiki/K-d_tree

- Find a transformation reducing the error for all correspondences based on some error metric.
- Repeat all steps until a termination criteria is reached.

Each of these steps can be implemented using different strategies.

The first step is the selection of points. There are different strategies for selecting points, where the most common ones are: Random selection of points, uniform selection, using all points or selecting points with high intensity gradient. All of these approaches can be used to select points only in the source point cloud or in both source and destination point cloud. Choosing the appropriate algorithm strongly affects the quality of the results and also the computational performance of the algorithm.

The next step is to find corresponding points in each point cloud. A simple approach is to search for the closest point in the destination point cloud. Other approaches are to search for the closest point at either the intersection of the normal of the source point with the surface of the destination point cloud or to project the source point to the surface of the destination point cloud using the vector of the camera view, from which the point cloud has been recorded. Each of these techniques may restrict the matching of points to some condition, e.g. the color of the points or the direction of the normals. This step makes clear, why an initial alignment for the two point clouds is needed for ICP. If the transformation of the two point clouds is too far apart, neighbourhood search will not provide appropriate results.

Next, each point pair is weighted using one of the following methods to measure the quality of the correspondence: A constant weight, weighting based on the distance between points, based on the angle between normals or based on the color difference of corresponding points. Also a combination of these methods is possible.

Using this information, correspondences with low quality can be removed and ignored in further steps. Rejection of points can be done in different ways:

- Reject points that are too far apart
- Reject a fixed percentage of points based on some metric, e.g. usually the distance
- Reject points outside the standard deviation of the average distance between points
- Reject points that are inconsistent with their neighboring points.
- Reject correspondences at object boundaries

Using the remaining correspondences, the transformation can be estimated using RANSAC [18]. The final step is to find an error metric describing the quality of the transformation. It can be the sum of squared distances between points, distance and color difference, or the distance of the source point to the plane, where the destination point is located using the normal of the destination point. All of these steps are iteratively repeated until either the error metric is below a specified threshold or a maximum number of iterations has been reached.

Camera tracking

A different approach for aligning point clouds is to track the camera position using 2D information from the RGB images. A simple technique is a marker based method, where a special marker (e.g. a checkerboard) present in source and destination image is tracked. However, this approach requires to have control over the registration scenario.

A more general approach is to detect arbitrary features for tracking the camera position. This technique can be used in a broad range of registration scenarios only limited by two constraints: First, the image have to provide enough visual information for feature detection. Second, the transformation between the two images must not be too large.

The camera tracking approach described is based on [9]. It is capable of tracking a live camera stream with standard framerate in real time. The camera tracking process can be split into three steps: Calculation keypoints and feature descriptors, matching of corresponding keypoints and estimation of transformation between two frames.

First, for each new frame, keypoints and feature descriptors are calculated. For keypoint detection, the FAST detector [6] is used. FAST keypoints focus at high performance detection for real time applications. For each pixel in the source image, a circular neighbourhood of 16 pixels is examined. The location is considered as keypoint, if there exists a contiguous number of n pixels, which are all either brighter or darker than the center pixel. Performance of this technique can be increased by using a machine learning approach to train a feature classifier in advance. Then, for feature detection, a decision tree is used to achieve high performance detection. Another problem of this approach is the detection of multiple interest points in adjacent locations. To overcome this issue, a score for each feature is calculated by summing the difference between the center pixel and the surrounding pixels. Now, for adjacent keypoints, only the one with the highest score is kept. Based on the keypoints obtained in the last step, feature descriptions are calculated using ImGD [9]. This feature descriptor is similar to the SIFT feature descriptor. For each keypoint, a neighbourhood of 16×16 pixels is considered. This area is then split up into 4×4 subregions. For each subregion, an orientation histogram of 8 values is calculated representing the average gradient magnitude in each direction.

Using the feature descriptions obtained in the last step, corresponding keypoints can be detected between two frames. A simple bruteforce approach is used to find for each feature from the source image the best feature in the destination image. To improve the quality of the corresponding keypoints, RANSAC [18] is used. First, the keypoint locations of the two images are mapped to 3D point coordinates. Then, RANSAC is used to estimated a transformation between the source and destination keypoints. Finally, the quality for each keypoint is calculated using the squared distance between source and target point after applying the estimated transformation. Only keypoints with good quality are considered in the next step. Finally, using the remaining keypoints and the estimated transformation from RANSAC algorithm, a more exact transformation is estimated using SVD [10].

The general workflow of the camera tracker is as follows: For each incoming image, keypoints and feature descriptions are calculated. If the investigated frame is the very first frame, it is stored and marked as keyframe. Subsequent frames that are analysed by the camera tracker are always compared against the keyframe stored in the camera tracker using the matching algorithm described above. If the estimated transformation exceeds a certain threshold, the new

frame is considered a keyframe and stored in the camera tracker. All subsequent source images are then compared against this new keyframe.

3.3 Segmentation

This section describes a method for separating the target object from the surrounding environment. This is done for two reasons. First, the goal of our reconstruction framework is to generate a 3D model of the object only. Second, performance of the further reconstruction process can be increased dramatically when calculated on the model only. The segmentation consists of two main steps. First, the plane where the object resides is detected. Afterwards, the object and the background are separated.

The goal of the first step is to find the plane, where the object is positioned on. RANSAC [18] is used to fit the plane into the point cloud, which returns the estimated coefficients of the table plane.

The second step is to find the object on the plane and extract it. Euclidean cluster extraction [19] is used to achieve this goal. First of all, only points above the plane are considered. Then, one of the remaining points is randomly chosen and added to a queue. Starting at this point, all neighbouring points within a certain distance are also added to the queue. This step is then applied to the points that have been added, until all points in the queue have been processed. These points now represent one object. If there are remaining points in the input data set, that have not been added to the queue, the algorithm can be repeated for these points. This should be only the case when multiple objects are placed on the table. This method can be efficiently implemented using an octree for representing the input point cloud.

3.4 Multiview refinement

When using the pairwise registration, the transformation between point clouds is usually calculated between two consecutive images of the object. The overall pose is the obtained by concatenating the poses from the pairwise registration. This results in an error that is accumulating based on the number of input clouds. Thus, the transformation between the first and the last point cloud might not match exactly. To overcome this issue, multiview refinement [12] can be used. The idea is to optimize the transformation on a global level on all point clouds and based on this data to distribute the error over all transformations in order to refine the alignment and find a more exact solution. An adapted version of LM-ICP [30] [31], which is capable of handling more than two frames is used to solve this problem.

The idea of LM-ICP is to align a source point clouds S and a target point clouds T by estimating a rigid transformation T , defined by a set of parameters a , by minimizing the error function:

$$E(T_a) = \sum_{i=1}^{N_S} e_i(T_a)^2$$

with

$$e_i(T_a) = \min_j \| \mathcal{T}_j - T_a(S_i) \|$$

being the distance of a transformed point $T_a(S_i)$ from the source point cloud to the closest point in the target point cloud \mathcal{T} . In case of a rigid transformation, a is a set of 7 parameters (3 for translation, 4 for rotation). Each iteration of LM aims at reducing the error function by updating the parameters of a . To achieve this, the Jacobian matrix of the error function is needed. The derivatives can be pre computed by the distance transform D of the target point cloud:

$$D(x) = \min_j \| \mathcal{T}_j - x \|$$

where $x \in X$, X being a discrete volume enclosing the target point cloud. Combining the above equations, we obtain $e_i(T_a) = D(T_a(S_i))$, which allows to define the derivatives as

$$\frac{\partial e_i}{\partial a} = \nabla_x D(T_a(S_i)) \nabla_{T_a}^\top T_a(S_i)$$

$\nabla_x D$ can be computed in advance by using finite differencing, whereas ∇_{T_a} has to be calculated analytically.

Until now, we have examined the LM-ICP algorithm for two point clouds. Next, we will investigate adaptations for multiple point clouds. Given a set of point clouds V^1, \dots, V^n which are initially aligned, the goal is to refine the absolute orientation. Using the prealigned point clouds, an overlap matrix can be computed, such that $A(h, k) = 1$ if an overlap of at least 30% is present between point cloud h and k . The registration error between two views V^h and V^k using a transformation a_h and a_k is defined as:

$$E(a_h, a_k) = \sum_{i=1}^{N_V^h} A(h, k) (D_\epsilon^k(T_{a_h a_k^{-1}}(V_i^h)))^2$$

where D_ϵ^k is the euclidean distance transform of V^k and $T_{a_h a_k^{-1}}$ aligns V^h to V^k . The overall registration error can be written as the sum of every pair in the set S of overlapping point clouds:

$$E(a_1, \dots, a_n) = \sum_{(h,k) \in S} \sum_{i=1}^{N_V^h} (D_\epsilon^k(T_{a_h a_k^{-1}}(V_i^h)))^2$$

$$E(a_1, \dots, a_n) = \sum_{(h,k) \in S} \sum_{i=1}^{N_V^h} (e_{h,k,i}(a_1, \dots, a_n))^2$$

The derivatives can be computed as:

$$\frac{\partial e_{h,k,i}}{\partial (a_1, \dots, a_n)} = \nabla_x D_\epsilon^k(T_{a_h a_k^{-1}}(V_i^h)) \nabla_{T_{a_h a_k^{-1}}}^\top (T_{a_h a_k^{-1}}(V_i^h))$$

The resulting Jacobian matrix is a sparse matrix, allowing to compute a solution using sparse solvers.

3.5 Alignment of multiple sequences

So far, all algorithms mentioned aimed at reconstructing a static object recorded from different angles. However, for a complete reconstruction of the object it is necessary to record multiple sequences of the object in different configurations, i.e. the object has to be turned around at least once to be able to reconstruct also the bottom of the object. The goal of this step is to align two partly reconstructed objects, where the relative rigid transformation between them is completely unknown. The alignment process can be divided into four steps: Search for keypoints and features in both point clouds, find correspondences, reject bad correspondences and estimate a transformation.

Keypoint & feature estimation

First, keypoints and features are extracted from each of the source images. A keypoint is an interesting point in the image. The most important property of a keypoint is that it is stable, which means that if a keypoint is identified in one image, the corresponding point in another image should be also detected as keypoint. Furthermore, the neighbourhood of the keypoint should have a unique appearance, allowing to describe it using a feature descriptor, which can be used for matching keypoints. A feature describes the area around a keypoint in a way, such that corresponding keypoints in different images have a similar feature description. 2D SIFT features [32] [7] have been used in this step. SIFT stands for Scale Invariant Feature Transform, and as the name suggests, the features are invariant to scale and rotation in different images. The features are obtained in the following way.

First, the keypoints have to be detected in one input image. In order to find keypoints invariant to scale, first a gaussian pyramid is built. Each level of the pyramid is convoluted by a gaussian kernel and then resampled by a factor of 1.5. After that, the Laplacian pyramid is approximated by calculating the difference between adjacent levels of the Gaussian pyramid. Next, for each level in the Laplacian pyramid, local minima and maxima are detected by comparing the pixel value to the value of the 8 neighbouring pixels. Each of these represent a keypoint.

After that a filter is applied to remove keypoints with low contrast and keypoints, that lie on edges, to improve the overall quality of the set of keypoints.

The last step handles the calculation of the feature description. For each keypoint, a 16x16 neighbourhood is considered, which is split into 4x4 subregions. A histogram is calculated for each subregion consisting of 8 values, each representing the average gradient magnitudes in that direction.

Correspondence estimation

Using the features obtained in the last step, the goal of correspondence estimation is to find corresponding keypoints using the information from the feature descriptors. For each feature descriptor in the source image, a corresponding feature descriptor in the target image is determined. This is done using a Kd-Tree search for a matching feature descriptor within a certain maximum distance. To improve the quality of the correspondences, a reciprocal search is per-

formed. Only correspondences which are found from source to target image and vice versa are considered.

Correspondence rejection

As not all found correspondences might be of good quality, bad correspondences should be rejected to improve registration results. This is done by using RANSAC [18] to align corresponding points from source and target image. Outliers are removed from the set of corresponding points, and only inliers are considered in the next step. As a side effect, this method also calculates a rough estimation of the transformation between the two point clouds, which can be used as initial estimation of transformation in the next step.

Transformation estimation

Finally, using the remaining correspondences and the initial transformation guess, the final transformation between the two point clouds can be calculated. For this purpose, SVD [10] is used. The transformation calculation is split in two steps: rotation and translation. The calculation of the transformation is simple. It is assumed that the centroid of the source and target point cloud is the same.

The goal of this step is to find a translation T and a rotation R , that aligns source (key)points s to corresponding target keypoints t . The relation can be defined as

$$t_i = Rs_i + T + N_i$$

where N_i is a noise vector. Finding the optimal transformation requires to minimize the least squares error:

$$\epsilon^2 = \sum_{i=1}^N \| s_i - Rt_i - T \|^2$$

It is assumed that the two point clouds have the same centroid. Using this information, the equation can be rewritten as

$$\begin{aligned} \epsilon^2 &= \sum_{i=1}^N \| s_{c_i} - Rt_{c_i} \|^2 \\ &= \sum_{i=1}^N \| s_{c_i}^T s_{c_i} + t_{c_i}^T t_{c_i} - 2s_{c_i}^T R t_{c_i} \|^2 \end{aligned}$$

where s_c and t_c represent the point sets translated in a way, such that the centroid lies in the origin. This formula is minimized if the last term is maximized, which can be reformulated as maximizing $\text{trace}(RH)$, where H is defined as

$$H = \sum_{i=1}^N t_{c_i} s_{c_i}^T$$

If the singular value decomposition of H is $H = U A V^T$, then the optimal rotation can be defined as $R = V U^T$.

Refinement

The results of the alignment can be refined by running ICP [28] on the point clouds, using the above obtained transformation estimation as initial guess for the ICP algorithm. The estimated transformation is exact enough for ICP to work and further refines alignment of the two partial objects.

3.6 Noise modelling and post processing

This section describes a method to merge all point clouds into a single point cloud and to smooth noisy data. It consists of two parts. First, a weight is calculated for each point in the point clouds representing the amount of noise present. Second, all point clouds are merged into a single, fixed resolution point cloud.

[33] observed that usually the noise in images acquired by RGBD cameras depends on two factors: The distance to the camera and the angle of the object surface relative to the camera. As the distance can usually be controlled in an object modelling environment, the method described focusses only on the viewing angle. Two key observations have been made. First, the amount of noise depends on the viewing angle. Especially high viewing angles (above 60) lead to immense noise. Second, noise is especially problematic at object boundaries, as the points in these regions jump between foreground and background. This also leads to inappropriate mapped color information, where the background color might appear on a point in the foreground and vice versa. Using these observations, a weighting function w_i for point i can be defined as

$$w_i = \left(1 - \frac{\theta - \theta_{max}}{90 - \theta_{max}}\right) \cdot \left(1 - \frac{1}{2} \cdot e^{\frac{d_i^2}{\sigma_L^2}}\right)$$

θ_i is the angle between the normal of point i and the viewing angle and θ_{max} is 60. d_i is defined as $d_i = \|d_i - d_j\|$, where d_j is the nearest point located at a depth discontinuity. σ_L is 0.0002 and represents the lateral noise sigma. As a result of this function points located near depth discontinuities or points with high viewing angle are assigned a lower weight. Using this value, points with weights below a certain threshold can be removed from the point cloud.

The final postprocessing step consists of merging all point clouds into a single one. In this step, the fact that the source point clouds are overlapping in some regions can be used to increase accuracy and further reduce noise. First, double walls are removed, which originate from noisy data or small registration errors. This is done by applying Moving Least Squares [34] with a small radius (1 to 2 mm). Point normals are used in this step to prevent that thin surfaces are

collapsed. After that, the point clouds are added to an octree with a certain leaf size. Points falling into the same leaf are averaged using the weight calculated in the last step. The outcome of this step is a single point cloud with a fixed resolution equal to the leaf size of the octree.

3.7 Surface reconstruction

This section describes the poisson reconstruction algorithm for surface reconstruction. The goal of this step is to create a 3D polygon mesh from the point cloud obtained in the last step.

Poisson reconstruction

The idea of Poisson reconstruction [16] is to find an indicator function, which approximates the real surface of the object. It returns 1 for points inside the object and 0 otherwise. The authors of the paper observed that there is a relationship between the points in the dataset and the indicator function: The gradient of the indicator function is 0 everywhere except near the surface of the object, where it is equal to the inverse normal of the point. Each point of the point cloud can be viewed as sample of the gradient of the indicator function. The problem thus can be reduced to finding a function, whose gradient approximates the points in the data set.

The surface of an object M can be represented by an indicator function χ , where points inside the model have a value of one and points outside the model have a value of zero. The gradient of the indicator function is a vector field that is zero everywhere except near the surface, where it is equal to the inward facing normal of the surface. Therefore, the points of the point cloud and their normals can be viewed as samples of the gradient of the indicator function.

$$\nabla \chi_M(\vec{s}) = -\vec{n}_s$$

For reconstruction of the surface, the idea is to integrate the normal field and interpolate between the given points of the point cloud. Because the indicator function is piecewise constant, $\nabla \chi_M(s)$ is not defined at the surface of the model. To solve this issue, a smoothing filter is applied to the indicator function.

As the surface of the object is not known, the integral of the surface can not be calculated directly. However, using the point samples of the point cloud, the vector field can be approximated. The surface is partitioned into distinct patches, one patch for each sample of the point cloud, which is scaled based on the resolution of the point cloud. The problem of surface reconstruction can now be defined as:

$$\nabla \chi_M = \vec{V}$$

However, as V is generally not integrable, no exact solution exists. By applying the divergence operator again, the equation can be rewritten as follows to form a poisson equation, which can be solved numerically using a least squares approximation

$$\Delta \chi_M = \nabla \cdot \vec{V}$$

The algorithm is implemented using an adaptive octree. Each sample is added to the octree in such a way that each leaf node contains exactly one sample. For each node o a function F_0 is assigned. The summation of all F_0 constructs a function space, whose linear span contains the indicator function χ_M . Each F_0 is defined as translation and scaling of a basis function F_B to the center and size of the leaf node o :

$$F_0(\vec{q}) = F_B\left(\frac{\vec{q} - o.c}{o.w}\right) \frac{1}{o.w^3}$$

where $o.c$ denotes the center of the node and $o.w$ the width of the node. The basis function is defined as the n -th convolution of a box filter with itself resulting in an approximation of a gaussian curve.

$$F_B(x, y, z) = (B(x)B(y)B(z))^{*n} \quad \text{with } B(t) = \begin{cases} 1 & \text{if } |t| < 0.5 \\ 0 & \text{otherwise} \end{cases}$$

The vector field can now be calculated as linear sum of all F_0 using a trilinear interpolation between neighbouring nodes.

$$\vec{V}(\vec{q}) = \sum_{s \in S} \sum_{o \in Nbr_D(s)} \alpha_{o,s} F_0(\vec{q}) \vec{n}_s$$

where $Nbr_D(s)$ are the eight nearest neighbour samples at depth D and $\alpha_{o,s}$ is the trilinear interpolation weight. The poisson equation can now be approximated, i.e. by using a conjugate gradient solver. This gives the indicator function that defines the surface of the object.

The final step for surface reconstruction is extraction of an iso surface from the computed indicator function. An adapted version of the marching cubes [13] [35] algorithm is used to reconstruct a polygon mesh. Basically, the marching cubes algorithm divides the space into uniform cubes with a certain resolution. Each cube contributes a polygon to the final mesh, depending on the type of intersection with the object surface.

3.8 Texturing

The last step in our reconstruction pipeline focusses on the generation of a texture map. The goal of this component is to generate a single, high quality texture map using the 3D mesh reconstructed so far and the source images with known transformation matrices relative to the model. This section describes a multiband texture mapping approach generating high quality texture maps [21].

The first part of this method is the mapping of each face of the 3D mesh to one of the provided source images, based primarily on the quality of the viewing angle. Using this information, the algorithm tries to find an optimal texture patch layout by preferring large texture patches. This means that textures with suboptimal viewing angle might be preferred by the algorithm to reduce the number of patch boundaries and thus also to minimize visible seams. As the final texture is merged from different source images, edges between two different source images will contain visual artifacts due to camera inaccuracies and registration errors. A method is presented to reduce seams and artifacts at texture patch boundaries by using a multiband blending

technique. It decomposes the source images into different frequency bands, which are blended at texture patch boundaries. This method is able to remove visual seams without introducing ghosting artifacts, usually appearing in blending methods. Furthermore the resulting texture quality is improved by removing specular light reflections and highlights. Finally a method is presented to pack multiple texture patches into a single texture map, thus making the algorithm suitable for generating texture maps that can be exported to standard 3D model formats. The next sections describe the technical details of the algorithm.

Texture patch layout

The most simple and straightforward approach for mapping polygons to input images would be to select the input image with the best viewing angle relative to the normal of the polygon. This would however result in highly fragmented texture maps, which on the one hand would waste much space in the final texture and on the other hand would increase number of texture patch boundaries, resulting in visible seams at polygon edges. To overcome this issue, a method is presented to prefer larger texture patches with close to optimal viewing angles. The idea is to select a minimal subset of the input images, where the angle between the normals of each polygons mapped to one input image and the viewing angle is below a specified threshold. This problem can be defined as minimal hitting set problem, which is NP hard. However, the authors of the paper proposed an efficient method, still finding an exact solution. First, all polygons are processed which are only visible in one input image, which should reduce the number of remaining images. Next, the remaining images are processed using an exhaustive search to find an exact solution. For complex models or when the number of input images is large, the second step can be replaced by an approximation algorithm, finding a close-to-optimal solution. The authors of the paper proposed to use a k-approximation of k-hitting problem to efficiently approximate the solution.

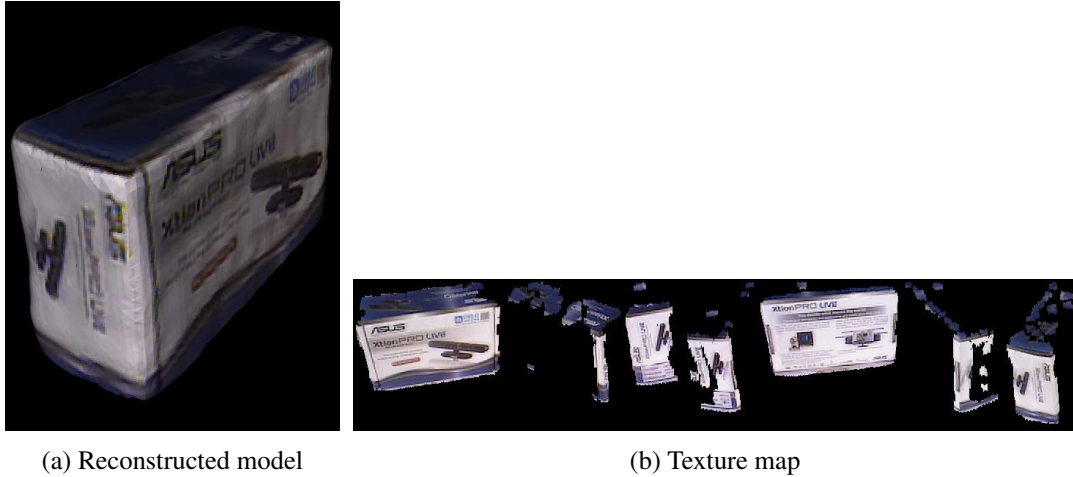


Figure 3.4: Texture patch layout

To reduce number of mapping errors, the authors of the paper provide a technique to take

self occlusion into account and thus minimize errors based on inappropriate mapped textures. They simply transform all triangles into their corresponding viewing image, saving the nearest triangle for each pixel in the viewing image. This assures that only triangles get mapped that are actually visible from the viewing image. Figure 3.4 shows an example of a texture map produced by this algorithm.

Multiband Blending

As already discussed, merging the final texture from multiple input images results in visible artifacts and color discontinuities at texture patch boundaries. This problem occurs due to camera inaccuracies and small errors in the registration process of point clouds. To minimize visible seams, a multiband blending method is introduced.

The authors of the paper observed, that for a simple blending approach it is hard to find an appropriate width of the transition zone. If it is too small compared to the average feature size, seams will still be visible. On the other hand, if the width is too large, features will be visible as ghosting artifacts. Especially for images with both large and small features, it would be impossible to choose an appropriate blending width. To solve this issue, a multiband blending technique can be used. As an image can be seen as composition of different frequencies, the image can be decomposed into multiple components. The transition width can then be chosen independently for each image component. The remainder of this section describes the method in detail.

First, a *Gaussian Pyramid* of the original images is built, where the first level is the original image. The next levels are created by applying a kernel-constant Gaussian function $g(\sigma)$. The level b of the pyramid is denoted by $G_b(I)$, where I represents the input image and B is the total number of levels in the pyramid.

$$G_b(I) = G_{b-1}(I) * g(\sigma)$$

After that, a *Laplacian Pyramid* is approximated by subtracting adjacent levels of the gaussian pyramid. Each level of this pyramid represents a component of the original image limited to a certain frequency band.

$$L_b(I) = G_b(I) - G_{b+1}(I)$$

Figure 3.5 shows the construction process of Gaussian and Laplacian pyramids. In the next step, textures are blended for each triangle using the following formula,

$$T(x) = \sum_{b=1}^B \sum_{i=1}^{|I|} \frac{W_{b,i} * L_b(l_i)(\Pi_i(x))}{\sum_{i=1}^{|I|} W_{b,i}(x)}$$

where I is the set of input images, x is a mesh triangle, Π_i is the projection from the 3D model to the viewing image I_i and W is the weighting function described next.

The weighting function is calculated in the following way:

$$W_{b,i}(x) = \begin{cases} (\alpha_i(x) < \frac{\pi}{2} * \frac{b-1}{B-1}) ? (\frac{\pi}{2} - \alpha_i(x)) : 0, & 2 \leq b \leq B \\ (\alpha_i(x) == \max\{\alpha_j(x) | 1 \leq j \leq |I|\}) ? (\frac{\pi}{2} - \alpha_i(x)) : 0, & b = 1 \end{cases}$$

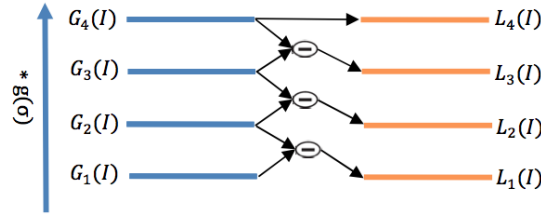


Figure 3.5: Generation of Gaussian and Laplacian pyramids [21]

This approach infers that higher frequency parts of the image are used from a small viewing angle, whereas lower frequency parts are blended from multiple viewing angles. Thus, the blending width is small for small features, whereas it is large for large features.

Highlight removal

To further improve quality of the resulting texture map, it is necessary to remove influences by specular light reflections from the surrounding area. The authors of the paper observed, that specular light reflection are usually visible in only one input image, whereas other input images show the underlying texture as it is. Using this fact, highlights can easily be detected by calculating the average color for each polygon and detecting color deviations in different input images. The highlight can then be removed by blending the area from other input images, where the highlight is not visible. Figure 3.6a shows the original textured model. Figure 3.6b shows the model after applying the multiband blending technique. Figure 3.6c shows the result after applying the highlight removal method.



Figure 3.6: Different steps in highlight removal algorithm [21]

Framework & User Interface

As already stated earlier, the main goal of this thesis is to implement an extensible framework for object reconstruction. This chapter describes the structure of the framework and the user interface. The software package consist of three components. The object modeller library contains all analysis algorithms. Furthermore it offers methods to build a reconstruction pipeline using these algorithms. It is intended for direct programming access and integration in other frameworks or tools. The Object Modeller Command Line Application is intended for unattended analysis and automated batch reconstruction. The Object Modeller User Interface is targeted at users without programming skills. It offers a configuration interface for setting up the reconstruction pipeline and changing parameters of the algorithms. The output of each analysis step can be visualized separately.

4.1 Object Modeller Library

The Object Modeller Library is the core component that has been developed. The following points were the most important during the development of the framework:

- Easily extendable, especially regarding the implementation of new analysis algorithms
- Easy to use
- Usable as library, command line tool or graphical user interface
- Suitable for a broad range of reconstruction scenarios

To achieve all of the points mentioned, the library was built in a modularizable way. It can be divided into two main parts. The `Core` package contains all relevant classes necessary for setting up and running a reconstruction pipeline. The `Analysis` package contains the implementations of the algorithms. Figure 4.1 shows the dependencies of all components of the framework, which will be discussed in the following subsections in detail.

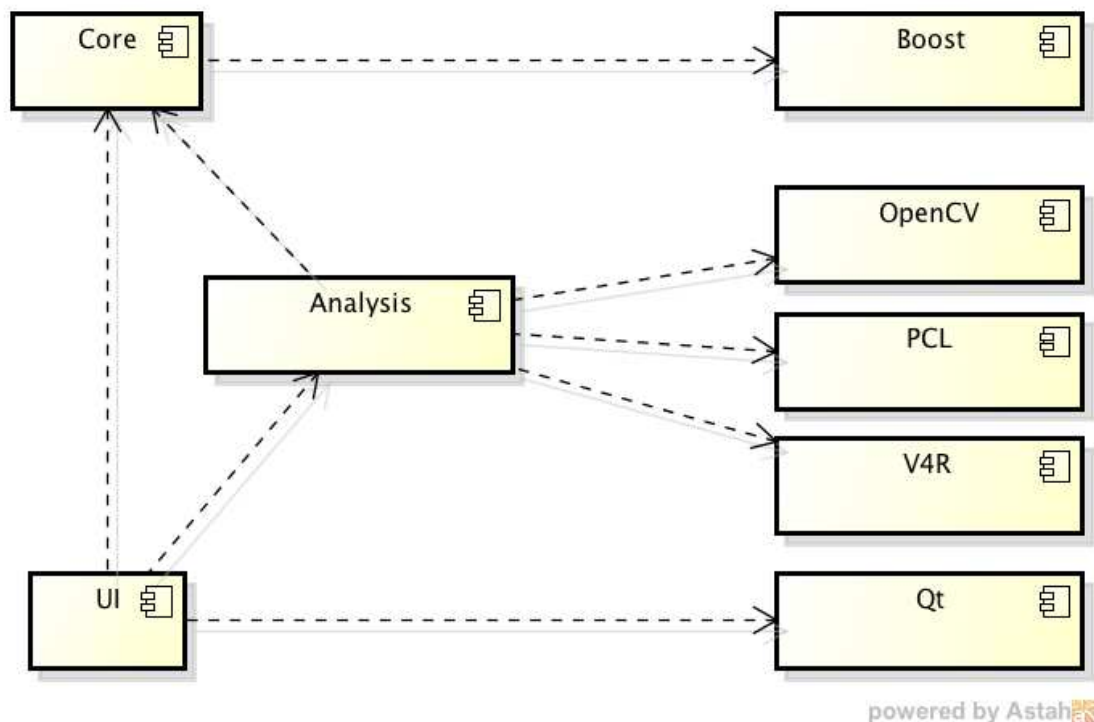


Figure 4.1: Component dependencies

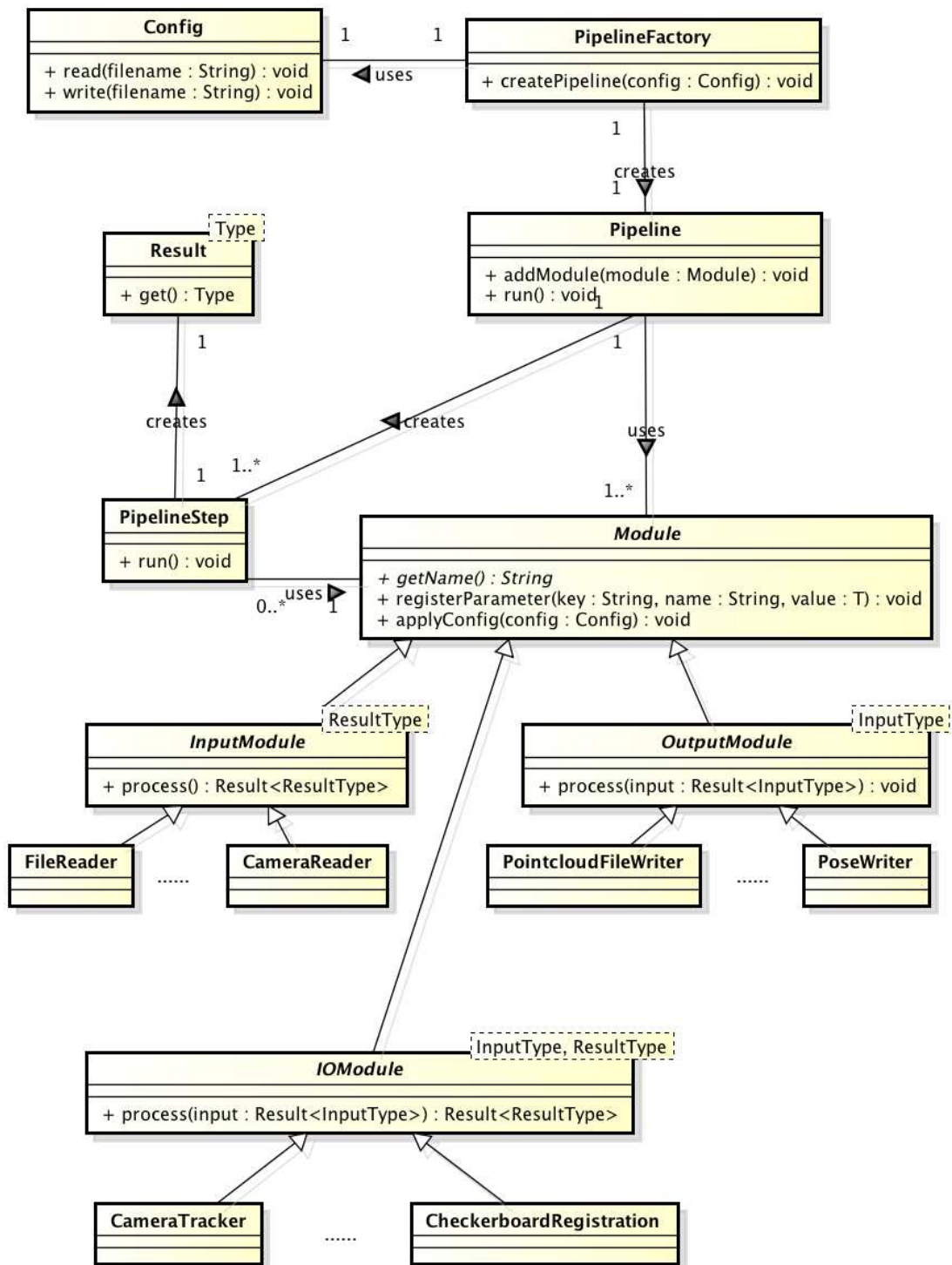
Core package

This package contains the complete logic for handling the reconstruction process and provides a base implementation for analysis algorithms. Figure 4.2 shows the class diagram of the library. The two most notable classes are `Module` and `Pipeline`.

The `Module` class is the base class for all analysis algorithms. It has three subclasses, each for a different type of algorithm: `InputModule`, `OutputModule` and `IOModule`, depending on the number of input and output parameters the algorithm has. These base classes only provide two abstract methods, that have to be implemented by an algorithm subclass:

- `T process(U input)` This method contains the analysis logic of the algorithm
- `String getName()` Returns the human readable name of the algorithm

The parameters and return value of the `process` method depend on the template parameter used in the class definition of the `Module` subclass. Furthermore there is a third (protected) method `registerParameter`. It can be used in the constructor of the class to register configuration parameters for the algorithm. This is important for further usage of the algorithm in the pipeline, as these parameters can then be automatically read from a configuration file, allowing configuration of the algorithms without recompiling the library.



powered by Astah

Figure 4.2: Class diagram of the library

The structure of the `Module` class shows the simplicity and extensibility of the library. For adding a new algorithm it is only necessary to create a `Module` subclass and implement two methods (where one method just returns a string) and to optionally register configuration parameters in the constructor. Listing 4.1 shows an example implementation of an analysis algorithm.

```

1
2 class AnalysisAlgorithm :
3     public IOModule< OutputType , boost::tuple<InputType1 , InputType2> >
4 {
5     private:
6         int parameter1;
7         float parameter2;
8
9     public:
10
11     AnalysisAlgorithm() : IOModule("analysisAlgorithmName")
12     {
13         registerParameter("parameter1", &parameter1, 0);
14         registerParameter("parameter2", &parameter2, 1.5f);
15     }
16
17     std::string getName()
18     {
19         return "Sample Algorithm";
20     }
21
22     OutputType process(boost::tuple<InputType1 , InputType2>)
23     {
24         OutputType result;
25
26         // analysis is done here
27
28         return result;
29     }
30 }

```

Listing 4.1: Example for an implementation of an algorithm

The other important part of the library is the `Pipeline` class. This is the main class used by users of the framework. It is responsible for managing modules and handling the reconstruction process, and offers two methods.

The `addModule` function allows to add a module to the pipeline. It takes an instance of a `Module` subclass and the necessary input parameters for it, and returns a `Result` instance, typed with a template parameter depending on the return type of the `Module` class. This object can be used in further calls to `addModule` as input parameter. The second method is called `run` and starts the reconstruction process with the modules added so far to the pipeline. Listing 4.2 shows an example for building a reconstruction pipeline.

```

1
2 int main( int argc , const char* argv[] )
3 {

```

```

4 // load configuration file
5 Config config("config.txt");
6
7 // initialize modules
8 FileReader fileReader;
9 CheckerboardRegistration registration;
10 FileWriter fileWriter;
11
12 Pipeline pipeline(config);
13
14 // add modules to pipeline
15 Result files = pipeline.addModule(fileReader);
16 Result registered = pipeline.addModule(registration, files);
17 pipeline.addModule(fileWriter, registered);
18
19 // start reconstruction
20 pipeline.run();
21 }

```

Listing 4.2: Example usage of the pipeline

All calls to `addModule` and all `Result` return values are statically typed through template parameters. This assures that it is not possible to pass invalid input parameters to modules.

When adding a module to the pipeline, the pipeline internally creates an instance of the `PipelineStep` class, which is responsible for managing the reconstruction state for the module, when the pipeline is run. It extracts the input parameters and invokes the `process` method of the module. After completion of the algorithm, the result of it is stored in the associated `Result` object.

As illustrated in Figure 4.1 the core package only depends on the Boost library. Primarily the `Tuple` implementation has been used to simplify passing multiple parameters to modules. Furthermore, static type checking patterns have been used to assure, that input parameters passed to modules match the signature of the processing function.

Advanced use cases

The pipeline is also capable of handling more complex reconstruction scenarios. This section lists some examples of the possibilities using the reconstruction pipeline.

An instance of the `Config` class can be passed upon construction of the pipeline in order to provide configuration parameters for the analysis algorithms. The configuration can be read from a file. Each parameter of an algorithm is uniquely identified by the name of the algorithm and the name of the parameter. Type conversion is done automatically for common types depending on the declaration of the parameter. The config file simply consists of a list of key value pairs.

The pipeline class is also capable of handling reconstruction scenarios, when multiple sequences of the object are involved. Each `InputModule` subclass has a method `getNrInputSequences()`, which can be overridden by subclasses to return the number of sequences produced by the module. For example, the `FileReader` offers a configuration parameter for reading input files from multiple folders, each containing a different sequence of the object. The further handling of multiple sequences is then done automatically by the

pipeline: Each step in the pipeline is executed separately on each sequence. In addition, there is also a method `getNrOutputSequences` for all subclasses of `OutputModule`, which can be overridden in cases, where the algorithm merges multiple sequences into one sequence. Currently, this is only used by the `SIFTFeatureMatcher`, which takes all input sequences and merges them into one sequence.

The `Renderer` class is interesting for applications with graphical user interfaces. A subclass of it can be registered at construction of the pipeline to allow graphical output of the processed results of the pipeline. The pipeline is able to automatically detect, whether a result can be displayed by the renderer or not, i.e. if the result is a point cloud or mesh, then it is passed to the renderer, while for example a transformation matrix is not.

Usually, when a module is added to the pipeline, an instance of a `PipelineStep` class is created, which keeps track of the reconstruction state and handles input and output parameters. There are also two special subclasses, `PipelineStepChoice` and `PipelineStepOptional`, which usage is especially interesting when the type of steps in the analysis pipeline should be configurable through the configuration parameters. It allows to enable a step in the reconstruction pipeline through the configuration file, or to choose between different algorithms for one processing step.

The `PipelineFactory` class is intended to simplify the initialization for a reconstruction pipeline. Currently it offers a method to create a standard reconstruction pipeline suitable for most reconstruction scenarios. It features a selection of different registration algorithms and also an optional step for multiview refinement through configuration parameters. In the future, this class can be extended to support the complete construction of a pipeline through configuration files, eliminating the need to recompile the library or application, if a different pipeline is needed, or it could offer at least a set of common reconstruction pipelines for different reconstruction scenarios. Listing 4.3 shows, how the code for creating a reconstruction pipeline can be simplified when using the factory.

```

1
2 int main( int argc , const char* argv[] )
3 {
4     // load configuration file
5     Config config("config.txt");
6
7     // create pipeline
8     Pipeline pipeline = PipelineFactory::createStandardPipeline(config);
9
10    // start reconstruction
11    pipeline.run();
12 }

```

Listing 4.3: Example usage of the pipeline factory

The design of the pipeline allows to handle most standard reconstruction scenarios solely through configuration files, without the need to change the code of the application. There are however edge cases, that can currently not be handled by the pipeline. One example for such a case is running the library on embedded systems with limited memory. The pipeline is designed to run the reconstruction step by step, keeping all input data in memory at once. For systems

with limited memory, this might not be possible. For these cases, the design of the library allows a simple workaround. Because all analysis algorithms provide a clean interface and have no dependencies to other components of the framework, it is simply possible to use instances of `Module` subclasses directly without using the pipeline. Listing 4.4 shows how to run the analysis algorithms without using the reconstruction pipeline.

```
1
2 int main( int argc , const char* argv[] )
3 {
4     // load configuration file
5     Config config("config.txt");
6
7     // initialize modules and apply configuration
8     FileReader fileReader;
9     fileReader.applyConfig(config);
10
11     CheckerboardRegistration registration;
12     registration.applyConfig(config);
13
14     FileWriter fileWriter;
15     v.applyConfig(config);
16
17     // run the modules without pipeline
18
19     std::list<pcl::PointCloud> clouds = fileReader.process();
20     std::list<pcl::PointCloud> alignedClouds = registration.process(clouds);
21     fileWriter.process(alignedClouds);
22 }
```

Listing 4.4: Running modules directly without pipeline

Analysis package

This package contains all implementations of analysis algorithms. Each algorithm is represented by one `Module` subclass. The package is organized in subpackages based on the components of our proposed pipeline. Figure 4.3 shows the structure of the package in detail. Most of the algorithms listed here have already been discussed in chapter 3. This section gives a short description of the modules, that have not been presented so far and about the implementation details of the algorithms presented in the last chapter.

Reader

This package provides modules for loading data into the pipeline. Various reader modules have been implemented to read data from one or multiple files (e.g. point cloud data, extracted indices, transformations). Configuration also allows to read files from multiple subfolders, allowing to structure them for multisequence reconstruction scenarios.

Besides that a camera reader has been implemented, which reads input data directly from a connected RGBD camera. It provides a live stream of the RGB images, which is passed to the `Renderer` instance. The capturing of a frame for further analysis has to be done manually

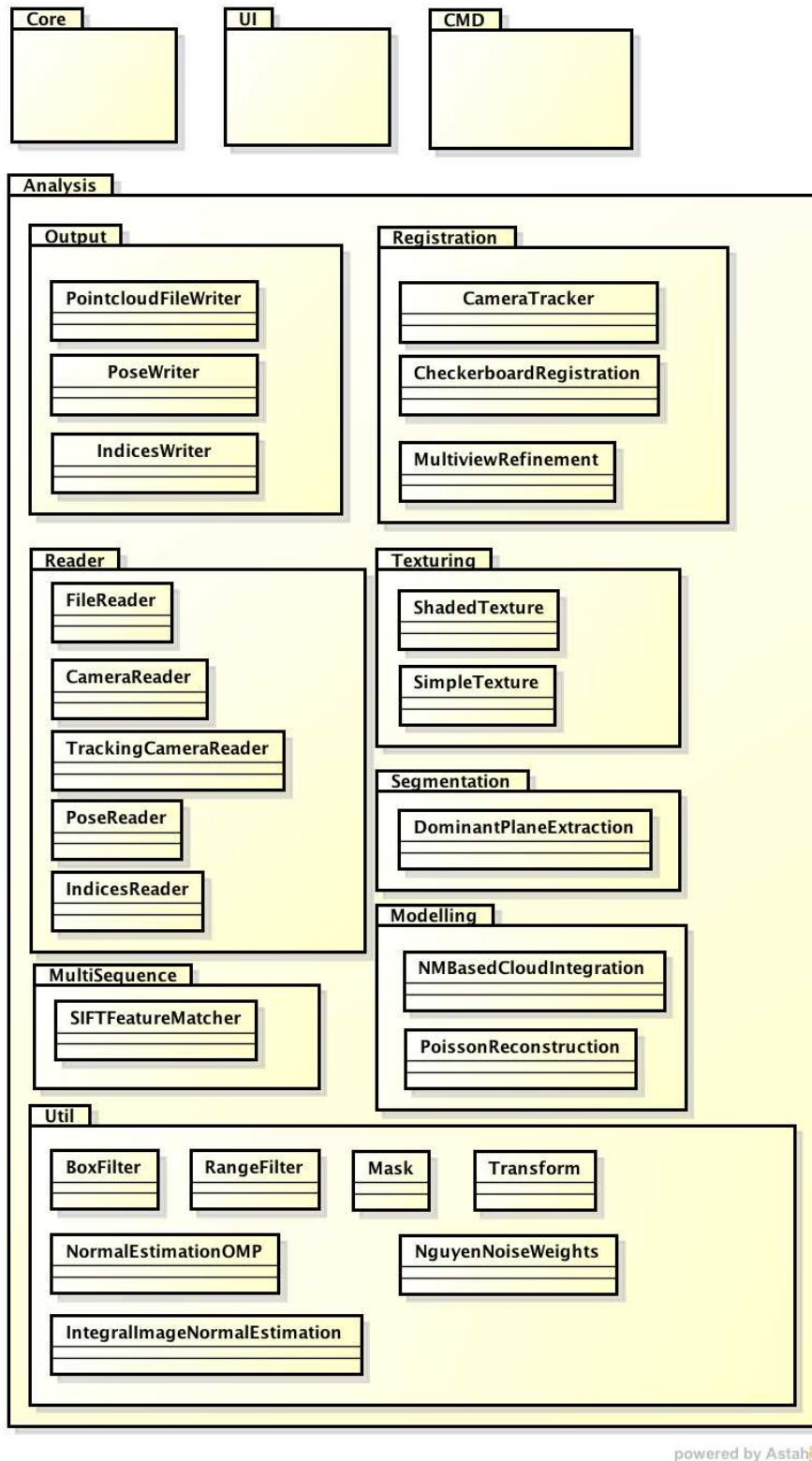


Figure 4.3: Package structure of all components

through the call of a callback function. Usually, applications using this reader capture a frame, when a button is pressed by the user. The reader continues to run until it is cancelled by the user. It also offers methods to record multiple sequences.

Another reader implemented is the `TrackingCameraReader`, which is a mixture between the `CameraReader` and the `CameraTracker`. The live stream of the `CameraReader` is fed directly to the `CameraTracker` and analysed on the fly. If the movement of the camera exceeds a specified threshold, the frame is captured and the transformation is stored. This eliminates the need for the user to manually trigger frame captures. In addition, the camera tracker assures that an ideal number of frames are captured needed for an optimal object reconstruction.

Pairwise registration

Two methods for pairwise registration have been implemented. First, the feature based camera tracker as described in chapter 3 has been implemented. Configuration parameters have been added to allow changing the number of features and the number of tiles used. Second, a module for tracking checkerboards has been added to the framework. OpenCV was used for tracking the checkerboard and estimating the transformation.

Segmentation

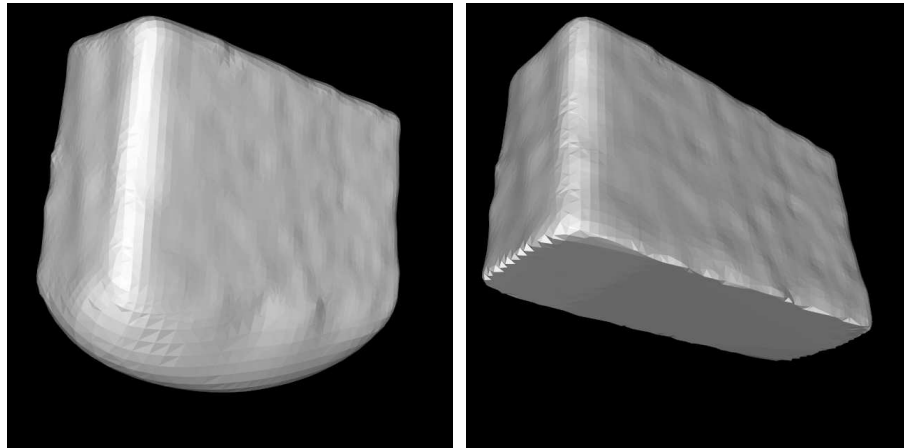
The segmentation algorithm is based on the method described in chapter 3.

Multisequence alignment

The multisequence alignment has been implemented using the approach described in chapter 3.

Modelling

The point cloud optimization and merging has been implemented as described in chapter 3. For surface reconstruction, the Poisson reconstruction has been implemented as presented. Configuration parameters have been added to allow control over the depth of the reconstruction and the number of samples per node. One problem of the Poisson reconstruction, which is also mentioned by the authors of the original paper, is the reconstruction of objects with holes. For sequences recorded from one object pose, there is always a hole at the bottom of the object. The surface reconstruction using Poisson results in an extension to the model, which can be seen in figure 4.4a. To circumvent this problem, the following adaption has been made. First, the convex hull of the original point cloud model is calculated. Then, all points of the final mesh outside the convex hull are determined. Finally, all outliers are projected on the convex hull using the center of the convex hull as direction for projection. The result after applying this method can be seen in figure 4.4b.



(a) Poisson reconstruction adds an extension at the bottom hole (b) After applying the cropping algorithm, the surface at the bottom is flat

Figure 4.4: Results for Poisson reconstruction

Texturing

For texturing, two methods have been implemented. The first one is a very simple approach, which is not even a texturing algorithm in the common sense. Instead of creating a texture map, this module simply uses the recorded point colors for shading the generated mesh. The quality of the output using this method strongly depends on the resolution used for constructing the mesh with poisson reconstruction.

Furthermore, the texturing algorithm as described in chapter 3 was implemented. For simplicity, the highlight removal was not considered.

Utility

Various utility modules have been implemented needed for the framework.

The range filter is used to filter points, that are far away from the camera. The box filter is used to filter all points that are not within a specified box. The configuration parameters allow to translate, rotate and scale the box arbitrarily in 3D space. The mask filter is used to filter points from a point cloud based on a list of indices provided to the module. The transform module is used to transform point clouds using the given transformation matrices. Also the two normal estimation methods described in chapter 3 have been implemented.

Output

The file writer can be used to write point cloud data to one or multiple files. The pose writer can be used to write transformation matrices for point clouds to one or multiple files. The indices writer can be used to write a list of indices for point clouds to one or multiple files. All of these output modules offer configuration parameters to define a file pattern, also for multiple sequences.

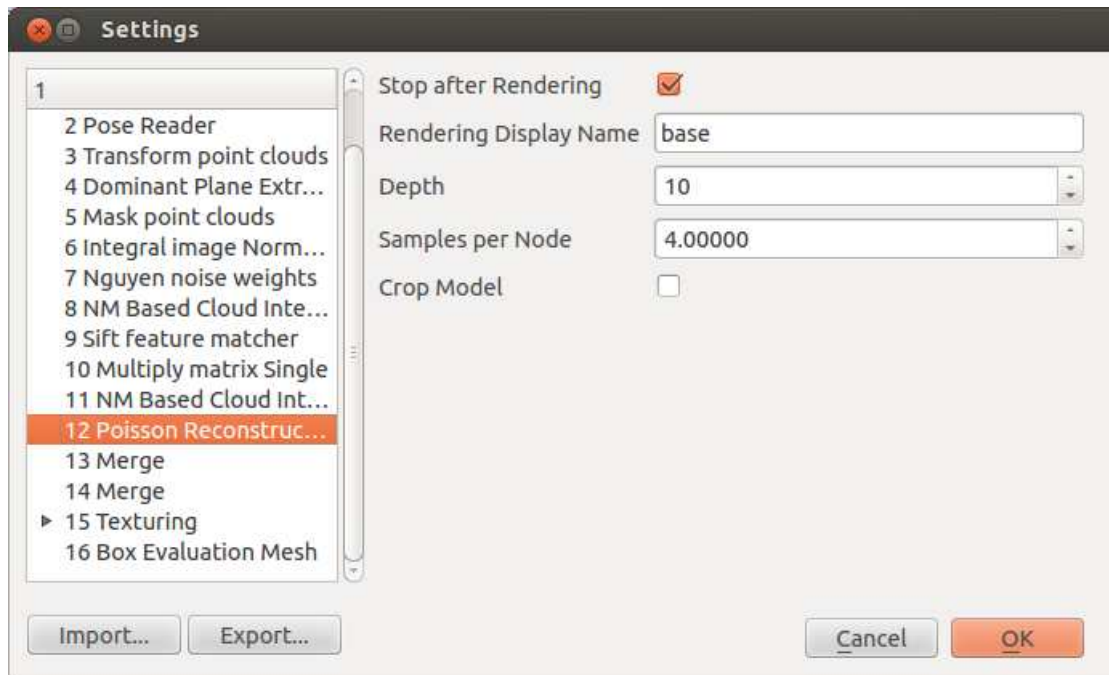


Figure 4.5: Configuration window

External dependencies

As illustrated in Figure 4.1, this component has some dependencies to other libraries. Most notably, PCL is heavily used, as it provides many algorithms regarding analysis of point cloud data. Furthermore, the OpenCV library is used for analysing 2D image data. It is primarily used for detecting 2D features (checkerboard, camera tracker and SIFT features) and for generating the mesh texture. Finally, this package depends on V4R, a library developed at the Vienna University of Technology, providing higher level analysis algorithms for point cloud data.

4.2 Graphical User Interface

The graphical user interface is primarily targeted at users with no programming skills. It consists of two views. The configuration window shows the structure of the pipeline. The configuration parameters for each algorithm can be changed here. Additionally, for some steps in the reconstruction pipeline, it offers a selection from multiple available algorithms. Also, some steps can be optionally activated, depending on the analysis scenario. Figure 4.5 shows the configuration window.

The main window shows the output of the reconstruction pipeline and offers interaction possibilities. The steps for reconstruction can either be run all at once or step by step. Each step is graphically visualized. The reconstruction process can be paused at any time, allowing to further investigate the output of each step. The output window allows to rotate and scale

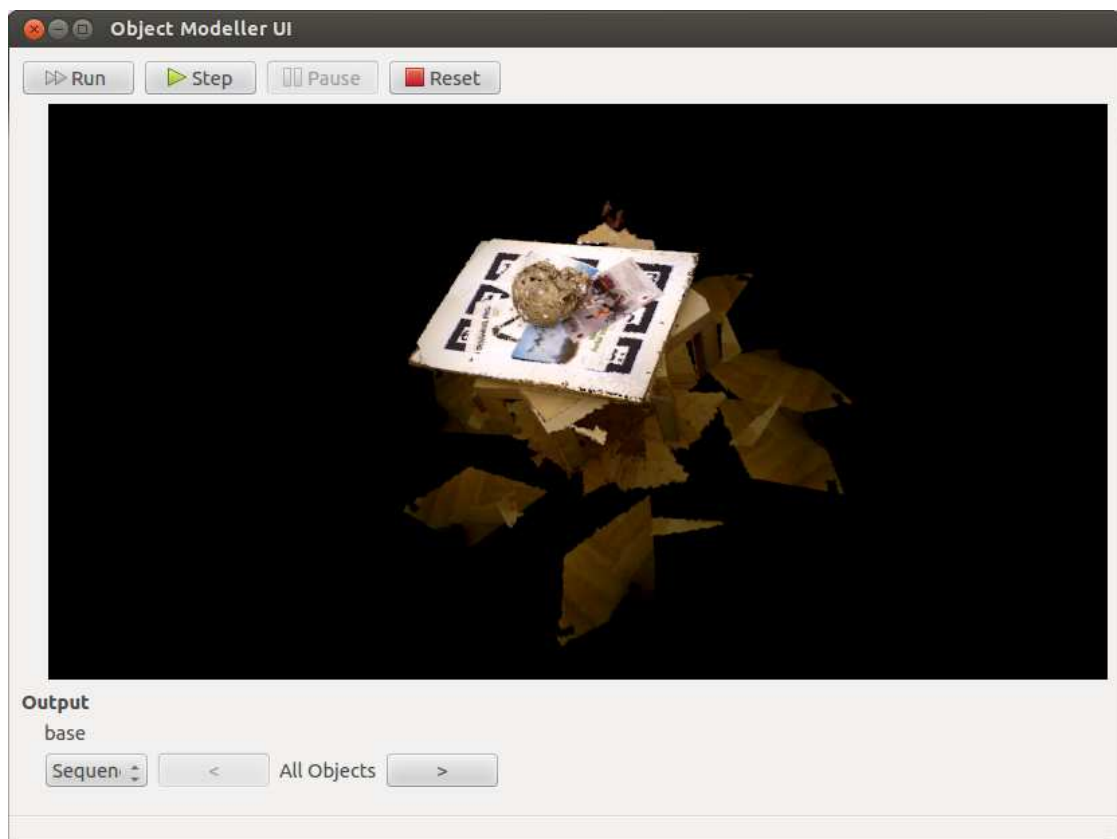


Figure 4.6: Main window

the object. Furthermore, the user can select to display each point cloud individually or all data merged into a single point cloud. Figure 4.6 shows the main window.

Advanced use cases

Some modules provide advanced interaction possibilities through the user interface. When the `CameraReader` module is run, buttons appear allowing the user to take a snapshot, start a new sequence or to finish the recording step. Furthermore, the visual output changes, showing also the live stream of the camera besides the recorded point clouds.

Another notable interaction case appears when using the `BoxFilter`. Before the filter is run, execution is paused and the box used for filtering the point cloud is visually displayed in the source point cloud. The user has then the possibility to change the size and transformation of the box, adapting it to the input data. Optionally, the modified transformation of the box can be saved to the configuration file. Figure 4.7 shows the box filter in action.

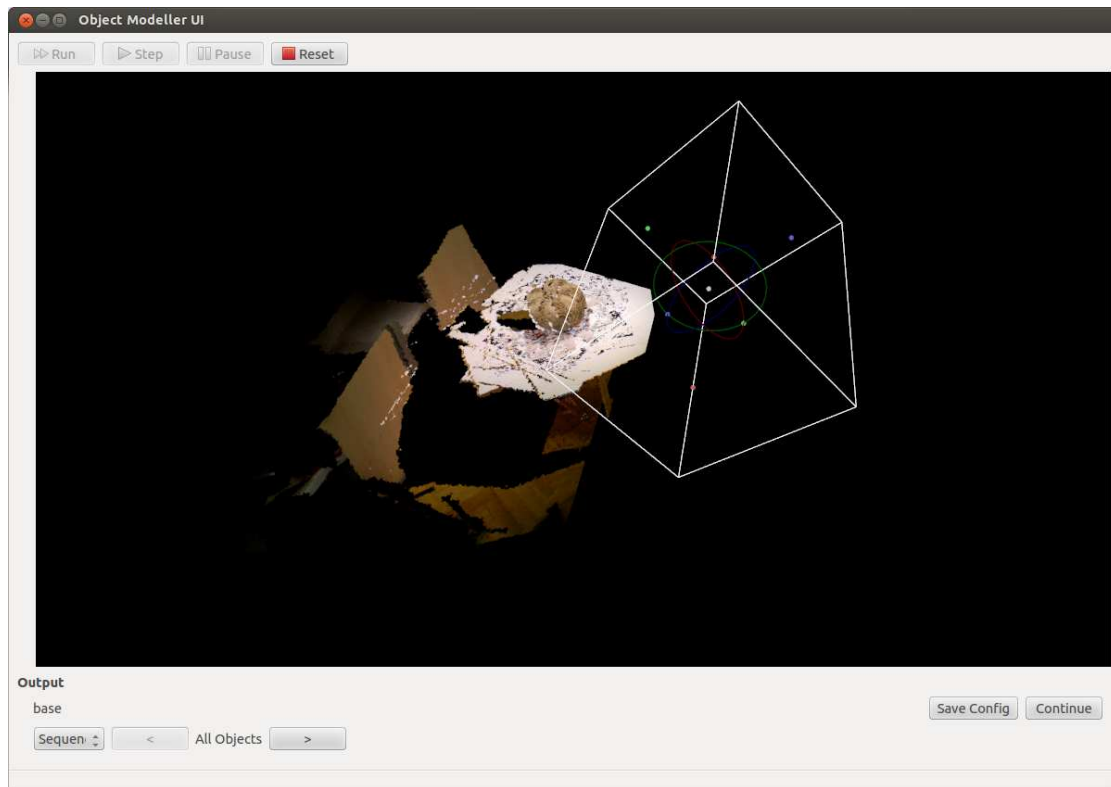
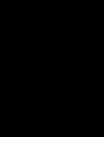


Figure 4.7: Box filter interaction

4.3 Command Line Application

The command line application is primarily intended for batch processing of multiple recorded sequences, or reconstructing a single sequence using different configuration parameters. It has a single required parameter, the path to a configuration file. Furthermore, all parameters present in the configuration file can also be specified directly on the command line, allowing to override these parameters. This makes it easier to run a reconstruction process using different parameters.

Although the command line application is intended to be used for unattended runs and batch analysis, it also offers a graphical output window using PCL's renderer component. Most of the interaction possibilities offered by the graphical user interface discussed in the last section are also accessible in the command line application through keyboard shortcuts.



Evaluation

In this chapter we evaluate the results obtained when reconstructing objects. First, the methods used to measure the quality of reconstruction are described. Next, we investigate reconstruction results for different reconstruction setups and configurations. Finally, the results of our framework are compared to other reconstruction frameworks.

5.1 Evaluation methods

We recorded over 20 different objects using various recording setups for the purpose of evaluation. They are evaluated using different setups and configurations of our reconstruction pipeline. Analysis of the results is done in two ways. First, we compare the results visually against each other. Although this approach is subjective, it is sufficient to check the difference between two reconstruction setups. This comparison shows, whether any errors occurred during the reconstruction process and if the resulting reconstructed object is visual appealing. Second, we use a simple cubic object with known size to achieve a measurable value of the quality of the reconstructed object. Points or vertices in the object are projected onto the surface of the real object to measure the average deviation of the reconstructed points compared to the real object. Furthermore, this cube has been recorded in 10 degree steps, resulting in a sequence of 36 frames. This allows to measure 2 additional terms for algorithms which output transformation matrices (registration and multiview refinement): First, the average registration error is measured, which should be 10 degree for this sequence. Second, the overall accumulated error is evaluated, since this value should sum up to 360 degree. In addition, also a sequence has been recorded using the tracking camera reader.

Besides that, we also compare computational efficiency of each configuration, although this was clearly not the focus when developing the reconstruction framework. Nevertheless, this might give interesting data for optimizing future implementations.

5.2 Results for different setups

In this section, the different components of our reconstruction pipeline are evaluated. Also different configurations of each algorithm are considered if applicable.

Pairwise registration

This section focusses on comparing pairwise registration using the feature based camera with different configuration parameters and the checkerboard registration. Table 5.1 shows the average error for the algorithms.

Method	# frames	Avg. point error	Avg. angle error	Accumulated angle error
Checkerboard	36	9.19mm	0.32°	1.11°
Checkerboard	18	7.65mm	0.38°	0.26°
Camera Tracker	36	4.60 mm	0.31°	4.41°
Camera Tracker	18	5.09 mm	0.60°	8.10°
Tracking Reader	30	4.02mm	-	2.60°

Table 5.1: Results of pairwise registration

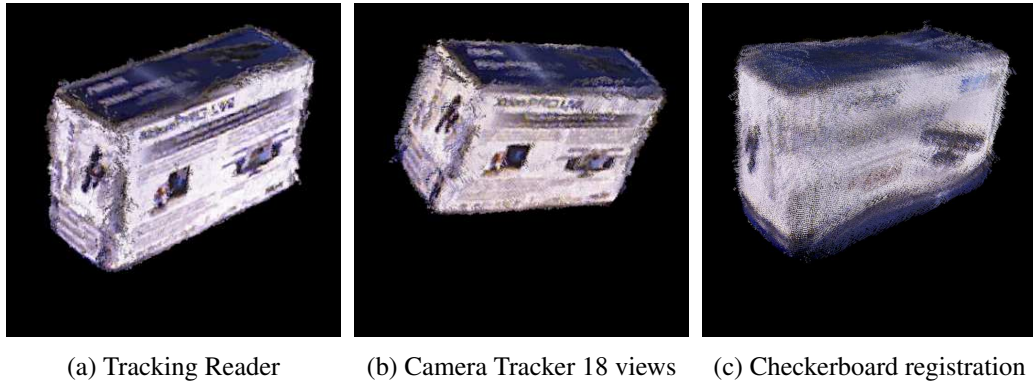
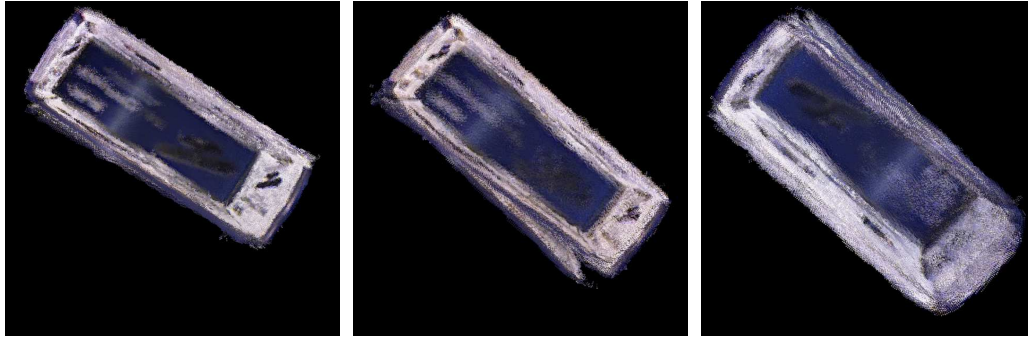


Figure 5.1: Results for pairwise registration viewed from top

The best results have been obtained by using the tracking camera reader. Using the camera tracker on the prerecorded sequences gave good results for sequences with high number of frames, but quality decreased dramatically when used only on half of the frames. The checkerboard registration gave the best results for the angular alignment, although the average deviation of points is worse than using the camera tracker. This shows, that angular alignment of the checkerboard registration is good, but the positioning is inexact. Figure 5.1 shows the visual results from the registered point cloud viewed from the top, Figure 5.2 shows the view from bottom. It can be clearly seen, that using the camera tracker results in an angular error between the first and last frame, whereas the error for the checkerboard registration is smaller, but the model looks bloated due to inexact positioning.



(a) Tracking Reader (b) Camera Tracker 18 views (c) Checkerboard registration

Figure 5.2: Results for pairwise registration viewed from bottom

Multiview reconstruction

This section evaluates results when using multiview reconstruction. Table 5.2 shows the measured results. The values in parenthesis show the results before applying the multiview reconstruction from the last section.

Method	# frames	Avg. point error	Avg. angle error	Accumulated angle error
Checkerboard	36	8.20mm (9.19mm)	0.33°(0.32°)	0.07°(1.11°)
Checkerboard	18	7.17mm (7.65mm)	0.38°(0.38°)	0.02°(0.26°)
Camera Tracker	36	4.47mm (4.60 mm)	0.35°(0.31°)	0.14°(4.41°)
Camera Tracker	18	4.06mm (5.09 mm)	0.46°(0.60°)	2.66°(8.10°)
Tracking Reader	30	3.53mm (4.02mm)	-	0.06°(2.60°)

Table 5.2: Results of pairwise registration



(a) Tracking Reader (b) Camera Tracker 18 views (c) Checkerboard registration

Figure 5.3: Results for multiview refinement viewed from top

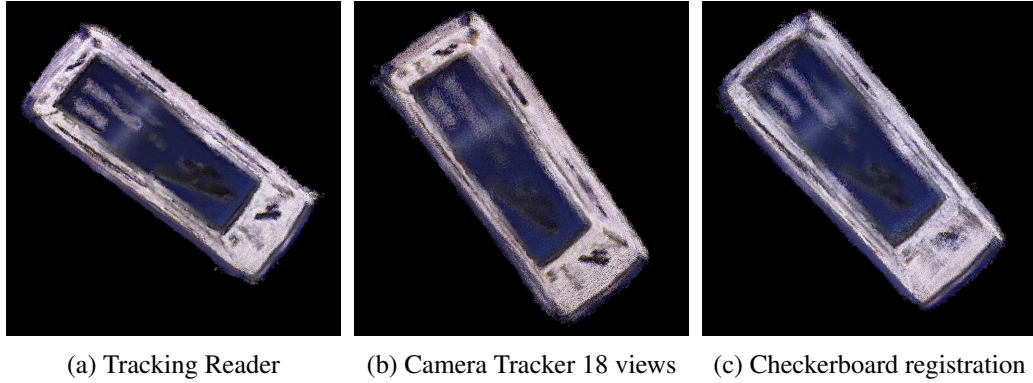


Figure 5.4: Results for multiview refinement viewed from bottom

It can be seen, that for all cases, the multiview reconstruction algorithm was able to improve the quality of the model. Only for the sequence using the camera tracker on half of the frames, the accumulated error was a little bit larger. The average point deviation for the checkerboard is still high, although the visual result looks quite good. This can be explained by investigating our measurement method: The resulting model is compared against a box model of fixed size. Because of the inexact positioning of the checkerboard algorithm, the resulting point cloud is smooth after multiview refinement, but also the overall size of the model is growing, which explains the higher values. Figure 5.3 shows the results after applying multiview refinement viewed from the top, Figure 5.4 shows the view from bottom.

Postprocessing

This section evaluates the effects of the postprocessing techniques described earlier.

Method	Number of Frames	No Multiview	Multiview
Checkerboard	36	10.92mm	8.20mm
Checkerboard	18	8.69mm	8.23mm
Camera Tracker	36	4.98mm	4.89mm
Camera Tracker	18	5.44mm	4.08mm
Tracking Reader	30	4.15mm	3.91mm

Table 5.3: Results of postprocessing algorithm

Results in table 5.3 show, that the average deviation of points is getting worse after postprocessing. However, the quality of the model is clearly improved as seen in figure 5.5 and 5.6. This again can be explained by the way we measure: By applying the postprocessing algorithm, the model slightly grows, which explains the higher values.



(a) Tracking Reader (b) Camera Tracker 18 views (c) Checkerboard registration

Figure 5.5: Results for postprocessing viewed from top

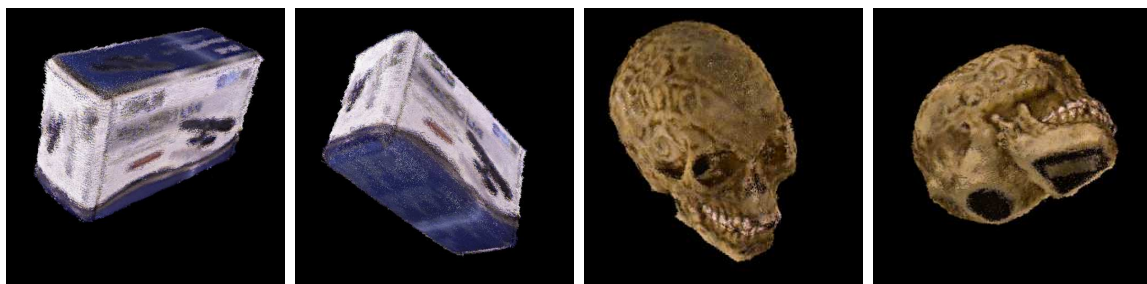


(a) Tracking Reader (b) Camera Tracker 18 views (c) Checkerboard registration

Figure 5.6: Results for postprocessing viewed from bottom

Multiple sequences

The alignment of two sequences has been also evaluated on the recordings of the box. During recording, the box has been flipped around 180 degree in order to allow measurement of the resulting output.



(a) Box from top (b) Box from bottom (c) Skull from top (d) Skull from bottom

Figure 5.7: Results for multisequence alignment

After applying the alignment algorithm, we obtained an almost perfect rotation of 179.87° . The algorithm has also been evaluated on other sequences, giving an exact alignment for all of them. Figure 5.7 shows the results for the multisequence alignment.

Surface reconstruction

For evaluation of surface reconstruction, a more complex sequence of a skull was analysed. The camera tracker has been and multiview refinement has been used for registration. Then, different configurations of the poisson reconstruction algorithm have been examined. Also the effect of the resolution in the postprocessing step has been taken into account.

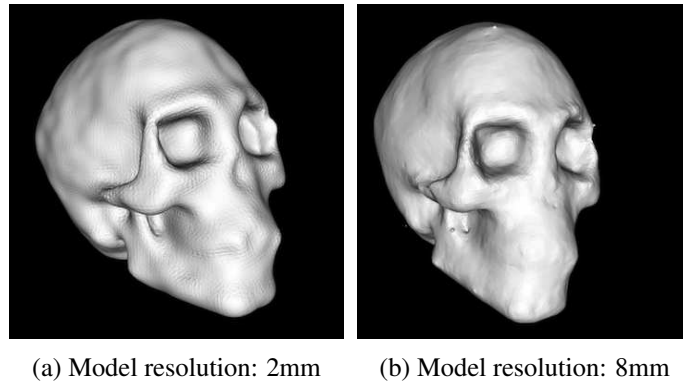


Figure 5.8: Results for Poisson reconstruction using different model resolutions

Figure 5.8 shows the results for different model resolutions. Higher resolution result in higher detail of the reconstructed surface, but also has a negative effect on runtime of the algorithm.

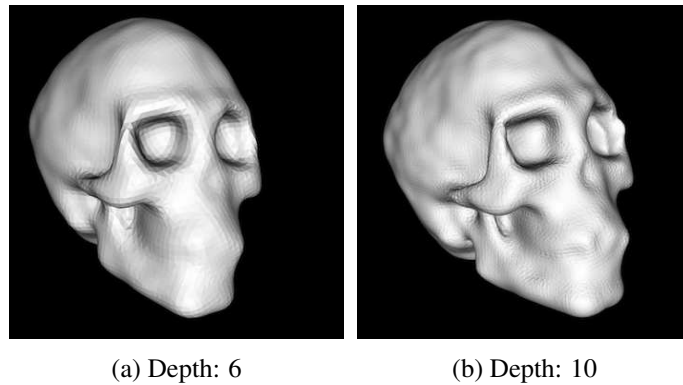


Figure 5.9: Results for Poisson reconstruction using different depth values

Figure 5.9 shows the results for two different depth parameters. This parameter controls the depth of the octree used in poisson reconstruction and thus affects the detail of the reconstructed

object. Higher values of this parameter allow a more detailed reconstruction of the object, but also decreases performance of the algorithm.

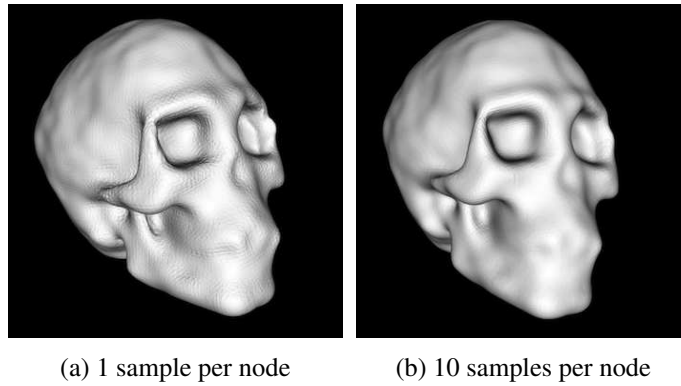


Figure 5.10: Results for Poisson reconstruction using different number of samples per node

Finally, the number of samples per node controls the smoothing of the resulting surface. Higher values smooth the object surface, but also result in some small missing details, which can be seen in Figure 5.10. In order to obtain detailed and smoothed objects, a high resolution of the input model should be used to maintain detail and the number of samples per node should be greater than one to get a smoothed surface.

Texturing

The following section shows the results of the texturing algorithm. It has been evaluated on two sequences. Figure 5.11 shows the results for the box sequence without multiband blending, Figure 5.12 shows the results after applying the multiband blending algorithm. As it can be seen, due to the lack of multiband blending, artifacts are visible especially at the edges of the object. Figure 5.13 shows the results on the skull sequence. It shows, that texture mapping is inaccurate especially in complex regions like the eye socket. Also, reflected highlights are visible in the second picture.



(a) No multiband blending (b) No multiband blending

Figure 5.11: Results for texturing algorithm on box sequence without multiband blending



(a) With multiband blending (b) With multiband blending

Figure 5.12: Results for texturing algorithm on box sequence with multiband blending

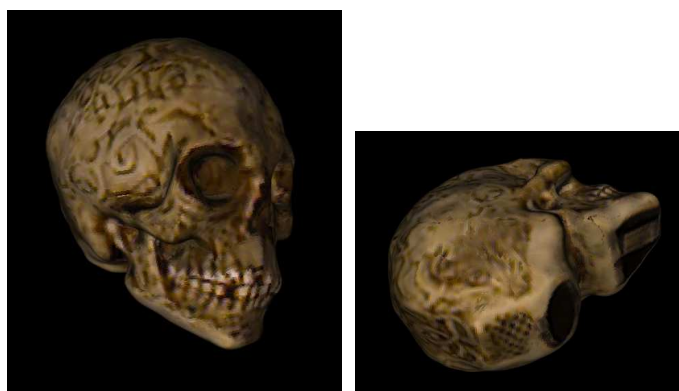


Figure 5.13: Results for texturing algorithm on skull sequence

General results

In this section, some of the reconstructed objects are shown. Figures 5.14 to 5.21 show the reconstruction steps involved. For each figure, the first picture shows one of the original input images. The second image shows the reconstructed and smoothed point cloud. The third picture shows the reconstructed polygon mesh. The fourth picture shows the final reconstruction result after applying the texturing algorithm.

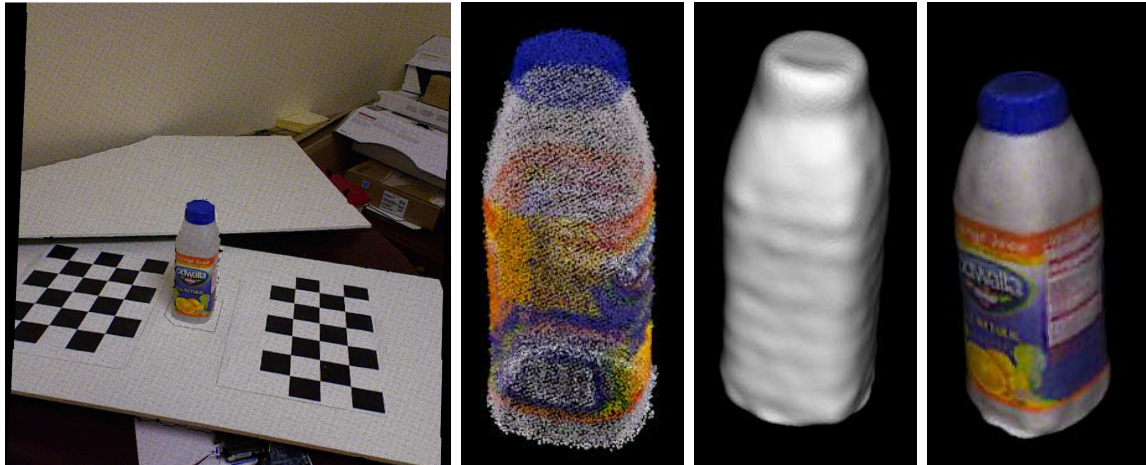


Figure 5.14: Reconstructed bottle

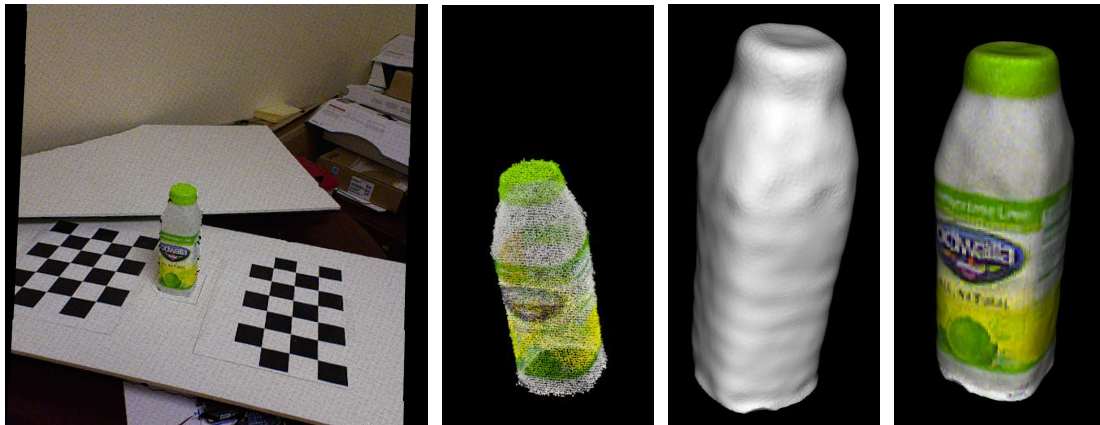


Figure 5.15: Reconstructed bottle

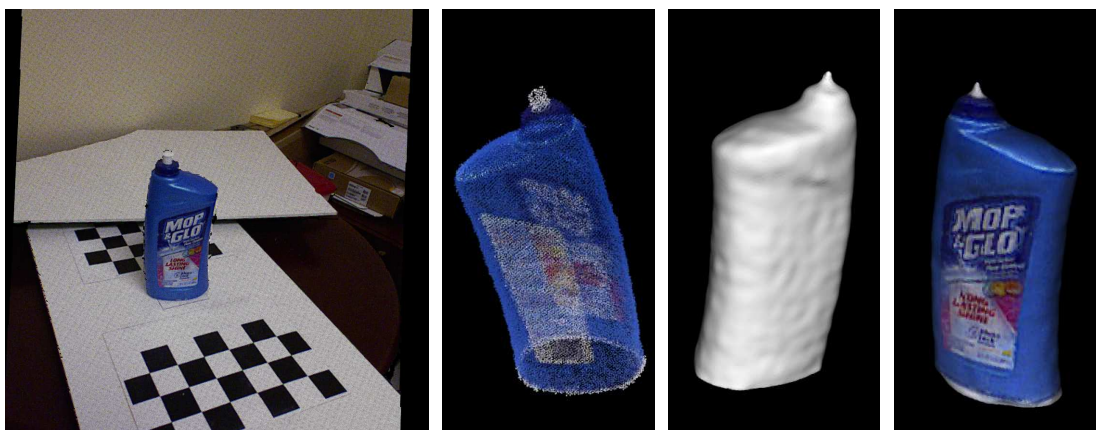


Figure 5.16: Reconstructed floor cleaner tank

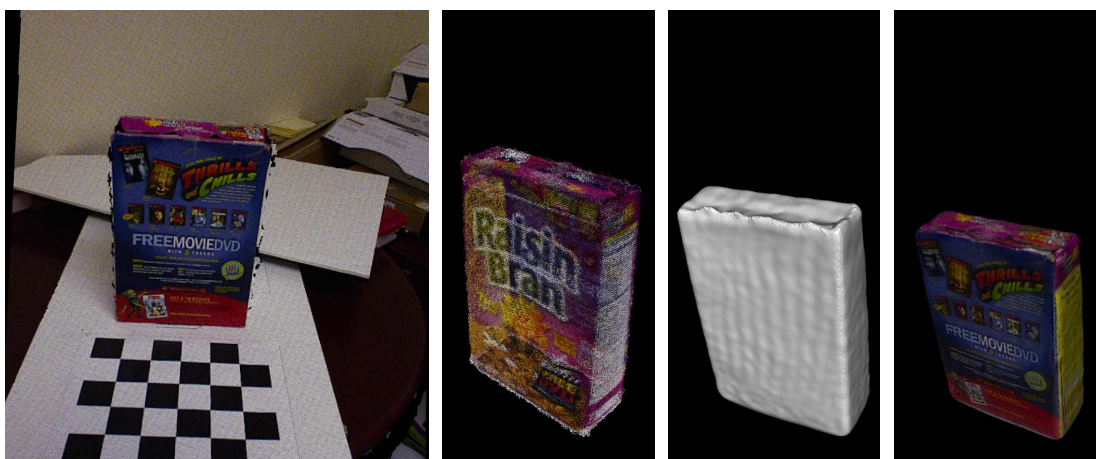


Figure 5.17: Reconstructed cornflakes package

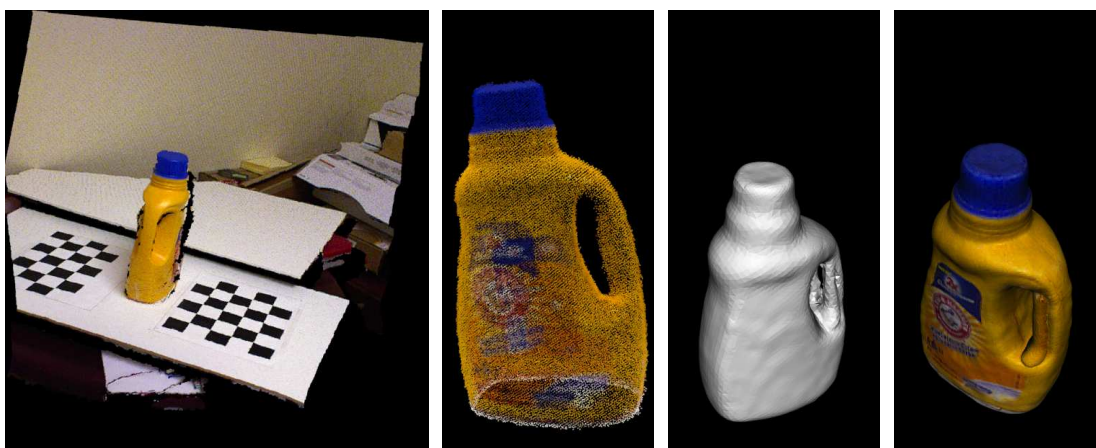


Figure 5.18: Reconstructed detergent bottle

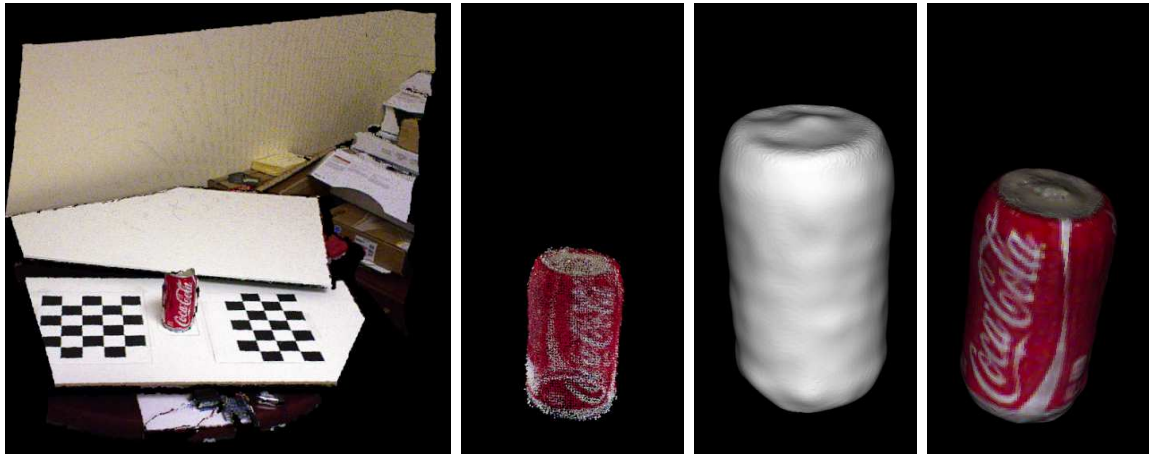


Figure 5.19: Reconstructed coke can

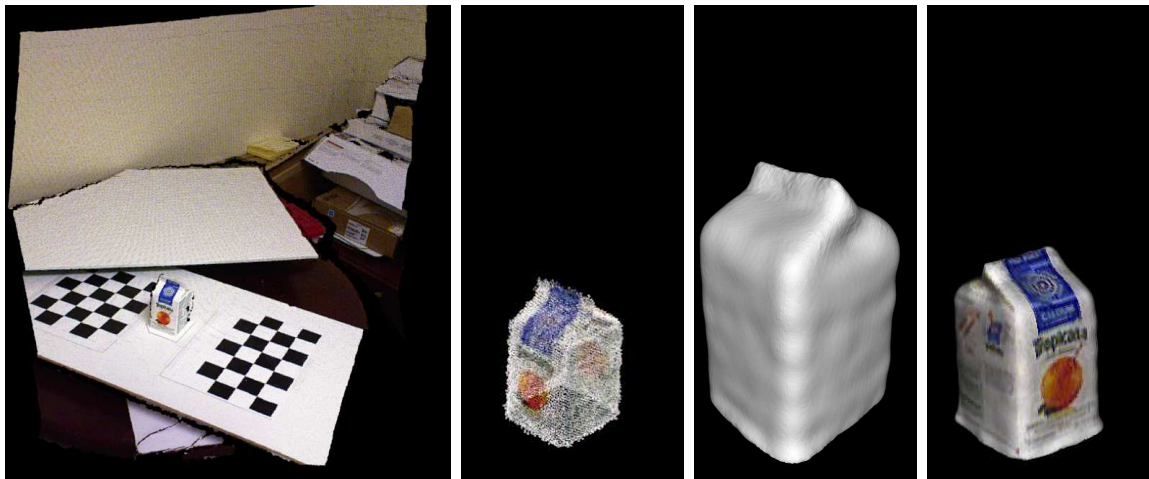


Figure 5.20: Reconstructed milk carton

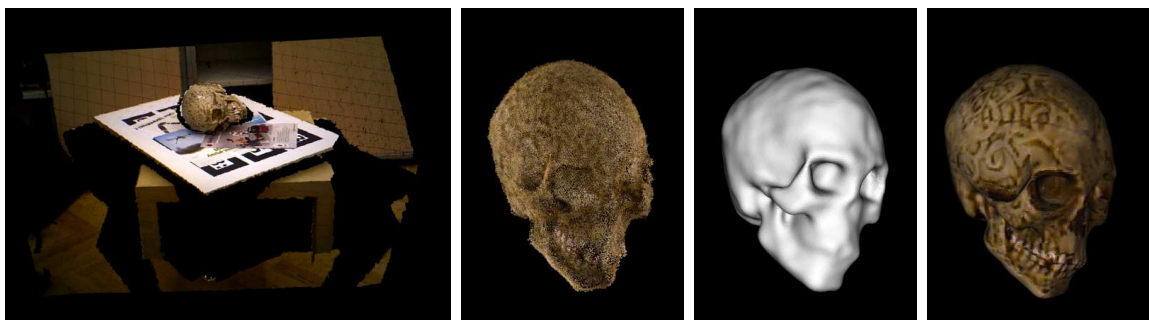


Figure 5.21: Reconstructed skull

5.3 Comparison to other frameworks

We compared the results of our reconstruction framework to two other object modelling tools: Reconstruct Me and PCL In Hand Scanner. In this section we show the results for each framework on a reconstruction of a toy truck. Figure 5.22 shows a picture of the object.



Figure 5.22: Truck source image

First, we reconstructed the object using Reconstruct Me. Figure 5.23 shows the reconstructed mesh. As it can be seen, the resulting polygon mesh is quite noisy. Also there are some major registration errors on the front of the car.

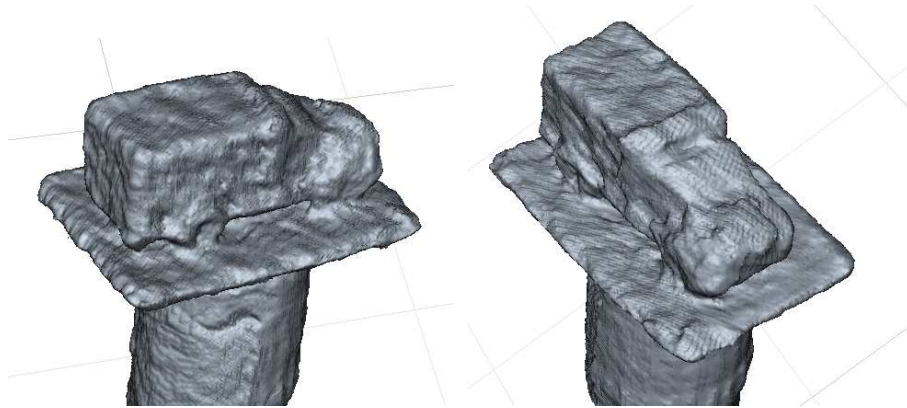


Figure 5.23: Reconstructed polygon mesh using ReconstructMe

Figure 5.24 shows the model after applying the texture. While the underlying mesh is the same, by applying the texture the resulting model looks nicer, and noisy data is not that obvious for the human eye. The texture itself looks quite blurry, and colors seem to be brighter than they should.



Figure 5.24: Textured polygon mesh using ReconstructMe

Next, we evaluated the same object using PCL's In Hand Scanner. Figure 5.25 shows the reconstructed point cloud. It looks quite smooth and accurate. However, when looking at the reconstructed mesh in Figure 5.26a and the textured mesh in Figure 5.26b, quality of the reconstruction is not satisfying. The algorithm used by this tool produces many holes in regions of the object, where reconstruction could not be calculated with enough confidence regarding the accuracy of the model. Also the texture is very blurry and inaccurate regarding color fidelity.

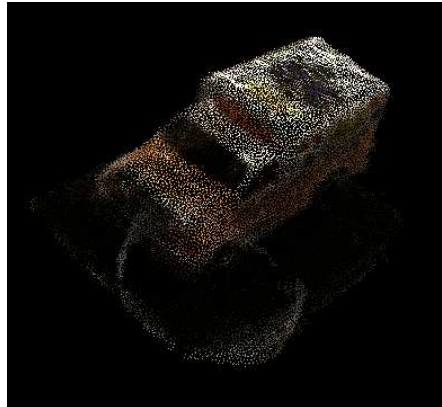


Figure 5.25: Reconstructed point cloud using In Hand Scanner

Finally, we reconstructed the model using our framework. Figure 5.27 shows the reconstructed mesh. Although the surface is still noisy, it is smoother than the model obtained using ReconstructMe and also does not contain any holes like the one modelled using the In Hand Scanner. Figure 5.28 shows the textured mesh using the multiband texture blending approach presented. It can be seen that the resulting texture is much sharper than compared to the other two reconstructed models. Furthermore, the colors are closer to the original model. The level of detail in the texture is much better compared to the other two approaches, which can be seen especially at the blue sign on the roof of the car and the radiator grill.

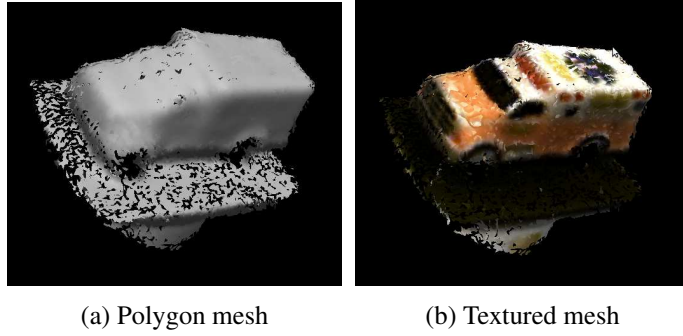


Figure 5.26: Reconstructed mesh using In Hand Scanner

However, the biggest downside of our approach is that computation of the model takes much more time compared to the other tools. Both ReconstructMe and the In Hand Scanner provide reconstruction in real time, whereas object reconstruction using our framework takes much longer in the magnitude of a few minutes.

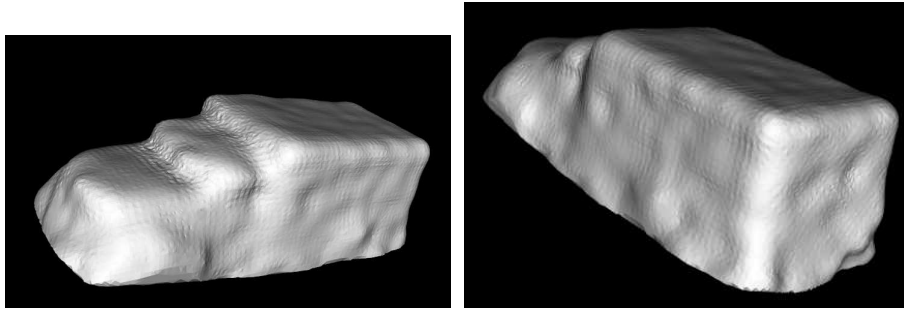


Figure 5.27: Reconstructed polygon mesh using our framework



Figure 5.28: Textured polygon mesh using our framework

Conclusion

The problem of modelling an object from multiple range images acquired by depth sensors has been investigated throughout this thesis. We proposed a pipeline for reconstructing textured objects from multiple recorded poses. First, the pairwise registration was presented, aligning the different input images into the model coordinate system. An approach was introduced to reduce the registration error on a global level. Next, a method was introduced to merge the different sequences and smooth the resulting model. Finally, the mesh reconstruction and texturing algorithm was described.

After evaluation of these algorithms, a highly configurable and extendable framework has been developed. We showed the simplicity of adding new algorithms to the framework. Furthermore, it has been shown, that the framework is capable of handling various reconstruction scenarios. A command line application as well as a graphical user interface have been developed to allow easy use of the framework even for users without programming knowledge.

The resulting framework has been tested on over 20 different sequences. Different registrations setups and configuration parameters have been evaluated. The best reconstruction results have been achieved using the tracking camera reader. The multiview refinement algorithm was crucial in order to improve registration accuracy. Using the poisson reconstruction and the described texturing algorithm, we were able to reconstruct good looking 3D objects.

Finally, the results have also been compared to other modelling frameworks. Although our framework can not compete with the other frameworks in terms of computational performance, we achieved better results in the reconstruction process. Registration of the models is more exact and the reconstructed surface looks smoother. Also the texturing gives more realistic looking results.

Although the reconstructed objects using our framework look quite good, there are a few issues that could be addressed in future work. First, more algorithms for each step in the reconstruction pipeline could be implemented. This allows on the one hand to cover even more reconstruction scenarios. On the other hand, it would allow to have more specific reconstruction setups depending on the scene. Second, the computational performance of the reconstruction is

very poor at the moment. Improving the performance by optimizing the algorithms or implementing new algorithms might make it possible to execute reconstruction in real time.

Bibliography

- [1] Aitor Aldoma, Zoltan-Csaba Marton, Federico Tombari, Walter Wohlking, Christian Potthast, Bernhard Zeisl, Radu Bogdan Rusu, Suat Gedikli, and Markus Vincze. Point cloud library. *IEEE Robotics & Automation Magazine*, 1070(9932/12), 2012.
- [2] Aitor Aldoma, Federico Tombari, Johann Prankl, Andreas Richtsfeld, Luigi Di Stefano, and Markus Vincze. Multimodal cue integration through hypotheses verification for rgb-d object recognition and 6dof pose estimation. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2104–2111. IEEE, 2013.
- [3] Jürgen Sturm, Erik Bylow, Fredrik Kahl, and Daniel Cremers. Copyme3d: Scanning and printing persons in 3d. In *Pattern Recognition*, pages 405–414. Springer, 2013.
- [4] S. Rusinkiewicz and M. Levoy. Efficient variants of the icp algorithm. In *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, pages 145–152, 2001.
- [5] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.
- [6] Edward Rosten, Reid Porter, and Tom Drummond. Faster and better: A machine learning approach to corner detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(1):105–119, 2010.
- [7] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.
- [8] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision—ECCV 2006*, pages 404–417. Springer, 2006.
- [9] Johann Prankl, Thomas Mörwald, Michael Zillich, and Markus Vincze. Probabilistic cue integration for real-time object pose tracking. In *Computer Vision Systems*, pages 254–263. Springer, 2013.
- [10] K Somani Arun, Thomas S Huang, and Steven D Blostein. Least-squares fitting of two 3-d point sets. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (5):698–700, 1987.

- [11] Felix Endres, Jürgen Hess, Nikolas Engelhard, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the rgb-d slam system. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1691–1696. IEEE, 2012.
- [12] Simone Fantoni, Umberto Castellani, and Andrea Fusiello. Accurate and automatic alignment of range surfaces. In *3DIMPVT*, pages 73–80. Citeseer, 2012.
- [13] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *ACM Siggraph Computer Graphics*, volume 21, pages 163–169. ACM, 1987.
- [14] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. *Surface reconstruction from unorganized points*, volume 26. ACM, 1992.
- [15] Jonathan C Carr, Richard K Beatson, Jon B Cherrie, Tim J Mitchell, W Richard Fright, Bruce C McCallum, and Tim R Evans. Reconstruction and representation of 3d objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76. ACM, 2001.
- [16] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, 2006.
- [17] Vedrana Andersen, Henrik Aanæs, and Jakob Andreas Bærentzen. Surfel based geometry reconstruction. In *Theory and Practice of Computer Graphics*, pages 39–44, 2010.
- [18] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.
- [19] Radu Bogdan Rusu. Semantic 3d object maps for everyday manipulation in human living environments. *KI-Künstliche Intelligenz*, 24(4):345–348, 2010.
- [20] Cédric Allene, J-P Pons, and Renaud Keriven. Seamless image-based texture atlases using multi-band blending. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008.
- [21] Zhaolin Chen, Jun Zhou, Yisong Chen, and Guoping Wang. 3d texture mapping in multi-view reconstruction. In *Advances in Visual Computing*, pages 359–371. Springer, 2012.
- [22] Bastian Goldluecke and Daniel Cremers. Superresolution texture maps for multiview reconstruction. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1677–1684. IEEE, 2009.
- [23] Minh Hoang Nguyen, Burkhard Wünsche, Patrice Delmas, Christof Lutteroth, Wannes van der Mark, and Eugene Zhang. High-definition texture reconstruction for 3d image-based modelling. In *WSCG*, pages 39–48, 2013.

- [24] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.
- [25] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, 2011.
- [26] Stefan Holzer, Radu Bogdan Rusu, M Dixon, Suat Gedikli, and Nassir Navab. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2684–2689. IEEE, 2012.
- [27] Jens Berkmann and Terry Caelli. Computation of surface geometry and segmentation using covariance techniques. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(11):1114–1116, 1994.
- [28] Yang Chen and Gérard Medioni. Object modeling by registration of multiple range images. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 2724–2729. IEEE, 1991.
- [29] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics, 1992.
- [30] Simone Fantoni, Umberto Castellani, and Andrea Fusiello. Automatic multi-view surface matching. In *Eurographics (Short Papers)*, pages 25–28, 2012.
- [31] Andrew W Fitzgibbon. Robust registration of 2d and 3d point sets. *Image and Vision Computing*, 21(13):1145–1153, 2003.
- [32] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.
- [33] Chuong V Nguyen, Shahram Izadi, and David Lovell. Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*, pages 524–530. IEEE, 2012.
- [34] Peter Lancaster and Kes Salkauskas. Surfaces generated by moving least squares methods. *Mathematics of computation*, 37(155):141–158, 1981.
- [35] Michael Kazhdan, Allison Klein, Ketan Dalal, and Hugues Hoppe. Unconstrained iso-surface extraction on arbitrary octrees. In *Symposium on Geometry Processing*, pages 125–133, 2007.