Incremental 3D Line Segments Extraction for Surface Reconstruction from Semi-dense SLAM

by

Shida He

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Semi-dense SLAM systems have become popular in the last few years. They can produce much denser point clouds than sparse SLAM while being computationally efficient (using only CPU). In previous works, the surface of the viewed scene was reconstructed in real-time by combining sparse SLAM system and incremental surface reconstruction method. However, it is challenging to utilize the large scale point clouds of semi-dense SLAM for real-time surface reconstruction. In this thesis, in order to obtain meaningful surfaces and reduce the number of points used in surface reconstruction, we propose to simplify the point clouds generated by semi-dense SLAM using 3D line segments. Specifically, we present a novel incremental approach for real-time 3D line segments extraction. This approach reduces a 3D line segment fitting problem into two 2D line segment fitting problems, which take advantage of both image edge segments and depth maps. We first detect edge segments from keyframes. Then we search 3D line segments along the detected edge pixel chains by minimizing the fitting error on both image plane and depth plane. By incrementally clustering the detected line segments, the resulting 3D representation for the scene achieves a good balance between compactness and completeness. Our experimental results show that the 3D line segments generated by our method are highly accurate compared to other methods. With the reconstructed surfaces, we demonstrate that using the extracted 3D line segments greatly improves the quality of 3D surface compared to using the 3D points directly from SLAM systems.

Acknowledgements

I would like to thank my supervisor, Dr. Martin Jagersand. This work could not be done without his guidance. His suggestions are always helpful and constructive. I would also like to thank Dr. Dana Cobzas for the help. She is always so nice to students. Many thanks to my lab mates and friends: Xuebin, Vincent, Min, Masood, Jun, Menna, Rong, also to those who graduated: Camilo, Oscar, Xi and Sepehr. The time with you is my priceless treasure and thank you very much for that. Finally, I would like to thank my parents and also especially my love, Shuang. Your support and caring would be the cornerstone of any achievement I can make, in the past or in the future. Your love and well-being are my only wishes to the stars at night.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Reconstructing surface from a stream of images is an important yet challenging topic in computer vision. The 3D surface of the scene is essential for a variety of applications, such as photogrammetry, robot navigation, augmented reality, etc. With the development of Structure from Motion (SFM) and Multi-View Stereo (MVS), one can easily reconstruct a 3D point cloud from a set of images or a video stream. Alternatively, a 3D point cloud of a scene can also be obtained in real-time using a monocular Simultaneous Localization and Mapping (SLAM) system.

Given the expanding point cloud from a SLAM system, incremental surface reconstruction algorithms can be used to reconstruct a 3D surface representing the scene [23], [34]. With the recent development in SLAM systems, some monocular SLAM systems can produce dense or semi-dense point cloud which represents the scene more completely than the sparse point cloud produced by traditional SLAMs. Although the surface reconstruction algorithms can run in real-time on sparse point clouds, the statement does not hold true when the number of points is increased significantly. In this thesis, we propose to represent the scene using line segments in order to simplify the result of semi-dense SLAM and achieve real-time performance in the task of surface reconstruction.

Although point clouds are extensively used in 3D computer vision, there are several limitations in representing a scene using 3D point clouds. First, points in a point cloud are usually stored independently and present no structural

relationships. Second, point clouds require large storage space and they are inefficient in terms of representing the geometric structure of a scene. For example, it only requires two 3D points to define a linear structure but there are usually hundreds or thousands of 3D points along the structure in a semi-dense or dense point cloud. These limitations severely reduce the efficiency of post-processing and analysis.

To overcome these limitations, geometric and semantic primitives can be used to simplify and obtain better understanding of the scene. There are works that tries to use geometric primitives (*e.g.* planes, cuboids, spheres) to help the reconstruction and completion of objects and environments [28] [9] [62]. In a relatively predictable and restricted setting like the indoor environment, even semantics primitives (*e.g.* walls and ceilings) can be used to further help the process of reconstruction and enhance our understanding of the reconstructed surface [26]. Simple primitives like planes can be utilized in real-time to help with, not only the robustness, but also the density of points produced by SLAM systems [63] [51].

Line segment can be seen as the simplest geometric primitive. Although it is not the most efficient primitive, line segments still efficiently preserve much more structural information of a scene compared to point clouds. Many line segment based 3D reconstruction algorithms have been proposed [4] [27] [65] [22] [48] [36] [53] [40] [25]. Most of these methods rely on efficient line segment detection and inter-frame matching, which can be difficult for complex scenes without long line segments. However, with the dense or semi-dense point clouds available, one can estimate 3D line segments without explicit matching and triangulation.

In this thesis, we present a novel method to incrementally extract 3D line segments from the result of semi-dense SLAM. Those detected line segments present structural relations among points and represent the 3D scene efficiently. A novel edge aided 3D line segment fitting algorithm is proposed. We first detect edge segments from keyframes using Edge Drawing [59]. Then 3D line segments are incrementally fitted along edge segments by minimizing the fitting error on both image and depth related planes. Through experiments, we

show that our method produces accurate 3D line segments with few outliers. Compared with the semi-dense point cloud, our method greatly reduces the number of elements in the representation while keeping the major structures in the scene. We apply the 3D line segments extracted by our method to incremental surface reconstruction and improve the quality of reconstructed surface compared to using the 3D points directly from SLAM.

The rest of this thesis is organized as follows. In Chapter 2, we review works related to the methods presented in this thesis. In Chapter 3, we describe our proposed algorithm in details. We present our experimental results in Chapter 4 and conclude in Chapter 5.

Chapter 2

Related Works

In this chapter, we survey the related works to this work in the literature. The method proposed in this thesis is based on monocular semi-dense SLAM system. We first introduce the monocular SLAM systems and for completeness we also discuss sparse and dense SLAM.

In this thesis, we focus on 3D line segment detection from semi-dense SLAM and recognize the proposed method as the major contribution of this thesis. Comparable 3D line segment detection methods in the literature will be discussed after SLAM.

A targeted application of the proposed method is real-time surface reconstruction. We discuss different approaches for surface reconstruction from point clouds. Our survey focuses on methods that target fast and light-weight incremental surface reconstruction.

2.1 Monocular SLAM

Monocular SLAM systems take a stream of images (a video) from a single moving camera as input and output the camera poses of each image and a reconstructed map, which is usually represented as a point cloud [8]. Visual Odometry (VO) systems also compute the camera motion and reconstruct map. However, the major difference between VO ans SLAM is that VO systems do not perform global optimization [57]. On the contrary to SLAM system where the information about somewhere visited long time ago can be reused, VO systems only consider the current image and several recent images.

However, since VO and SLAM have basically the same input and output and they also have many common components, we do not list VO systems separately.

Generally, in the literature [57], SLAM system is often categorized into direct or indirect based on whether it extracts features out of the images or not. However, instead of the processing of input images, we focus more on the output of SLAM systems. Hence here we present another criterion to categorize them. Based on the characteristics of their output map, SLAM systems can be categorized into three categories: Sparse, Semi-dense and Dense.

2.1.1 Sparse monocular SLAM

Traditionally, most monocular SLAM systems are sparse because they operate on feature points detected from images. Since the number of reliable detected feature points is limited and usually at most a few hundreds of the best feature points are used in each frame due to performance consideration, the resulting map is often quite sparse in terms of the density of points.

MonoSLAM is considered to be the first real-time capable monocular SLAM system [11]. Based on Extended Kalman Filter (EKF), it estimates the camera motion and maintains the position and uncertainty of up to 100 points in the scene. Each point is defined using a small image patch which is being tracked in the following frames. The uncertainty bound of points is used to restrict the search area. The uncertainty of tracked points will gradually decrease as the number of observation increase. As the camera moves, new points will be detected and added to the map whose positions will be initialized by triangulation. As the first real-time monocular SLAM system, it shows that it is possible to estimate camera pose and reconstruct sparse points in the scene in real-time using a single camera. However, because the number of points is limited due to computational cost of the system, the working space of MonoSLAM is limited to a small area.

PTAM is another successful feature-based monocular SLAM system [30]. Figure 2.1 is a sample image of PTAM's interface. By utilizing two threads running in parallel, it is able to integrate Bundle Adjustment (BA) to real-time

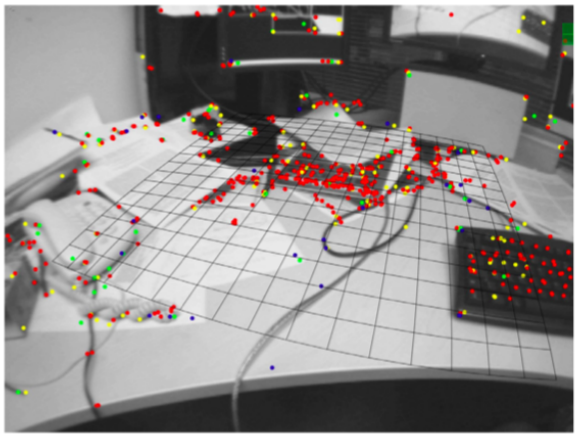


Figure 2.1: Sample image of PTAM system interface [30]

SLAM and maintain thousands of points without worrying about its performance. In each frame, feature points are detected and matched with points in the map. The camera motion is then estimated with these feature correspondences. As a keyframe-based SLAM system, PTAM will create a keyframe if it is far enough to other keyframe in terms of distance and time. Map points will be created by triangulate feature points detected in keyframes in the background mapping thread. Also in the background mapping thread, PTAM will perform local BA to optimize the camera poses and point positions for recent keyframes. Global BA operating on all keyframes will also be performed but it is less frequent than the local BA. By incorporating BA and separating tracking and mapping into two threads, PTAM achieves accurate result while maintaining real-time performance.

ORB-SLAM is a recent monocular SLAM system which uses ORB [50] feature descriptor to match feature points [37]. The system overview of ORB-SLAM is shown in Figure 2.2. Following and extending the idea of PTAM, it has three threads running in parallel: tracking, local mapping and loop closing. By detecting loops using a visual bag-of-words based method, ORB-SLAM can close loops and optimize camera poses using pose graph optimization. Instead of optimizing on the camera poses with 6 Degrees of Freedom (DoF) for translation and rotation, optimizing on the pose graph with one additional DoF for

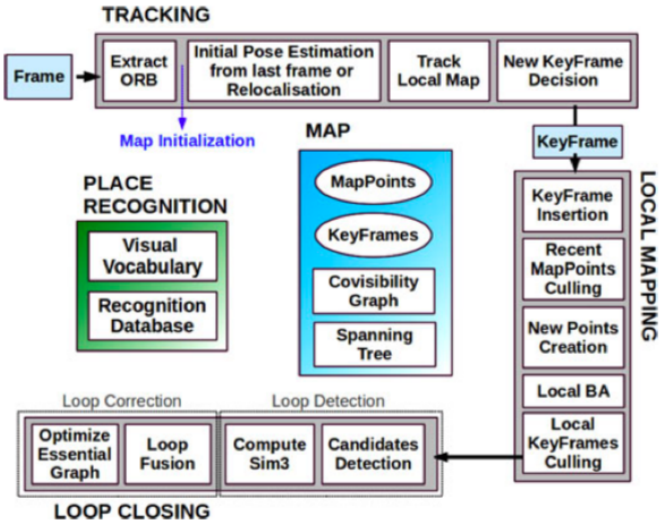


Figure 2.2: System overview of ORB-SLAM [37]

scale based on similarity transform helps on correcting the scale drift accumulated in the system. By optimizing on a pruned version of the pose graph instead of the full graph and running all the heavy optimization in a separate thread, ORB-SLAM can close loops without compromising the real-time performance. In the tracking thread, similar to PTAM, features are detected and matched against all the potentially visible points to find correspondences. After estimating the camera pose with these correspondences, a fast motion-only BA is performed to optimize the camera pose. For mapping, ORB-SLAM adopts a so-called "survival of the fittest" strategy for both keyframe and feature point selection, where it is generous in the creation of keyframes and feature points, but very restrictive in the culling of them. In this way, only the best keyframes and points will survive after a period of time. This approach improves the robustness and accuracy of the system since there are more keyframes and points available for calculation, while limits the growth of computation because redundant information is quickly discarded. By exploiting the covisibility between keyframes, the tracking and mapping are focused on a local covisible area which helps on operating in large environment. Together with automatic initialization and relocalization, robust and accurate

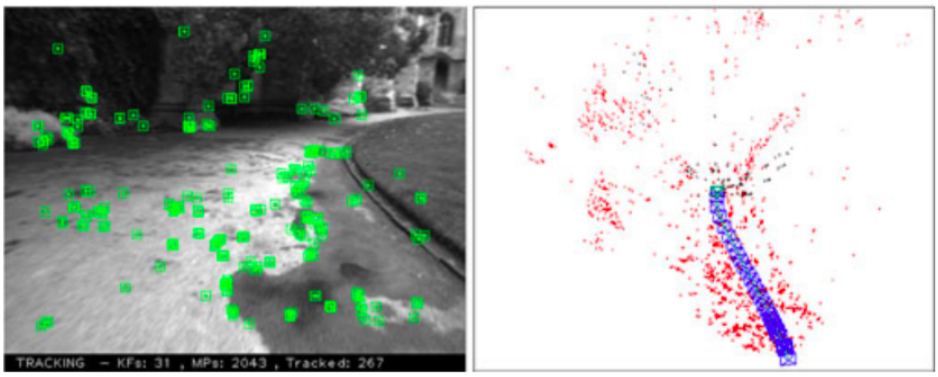


Figure 2.3: Sample result of ORB-SLAM [37]

performance makes ORB-SLAM a complete package for monocular SLAM. Sample result of ORB-SLAM is shown in Figure 2.3.

SVO is a fast semi-direct VO system [16]. It is a semi-direct method because it does not extract features out of every image frame. Instead, feature points are only extracted and matched in keyframes to initialize 3D points. A grid is used to divide the image into small areas, and a fixed number of feature points are extracted in each grid to make sure the points are spread across the whole image. Without features detected and matched in every frame, the small image patches at the location of the features are used to estimate the camera motion. Given 3D points in the map, by projecting the 3D points onto the image, the motion can be estimated by minimizing the photometric error between all the corresponded small image patches. In the next step, they further optimize the location of each individual patch in the current image by minimizing the photometric error. Finally, the motion estimation is refined by a motion-only BA and a structure-only BA on the current frame and recent keyframes. For the mapping, SVO maintains the uncertainties for the depth of each point which is updated by each observation of the same point. In this way, only the points with accurate and certain 3D location will be used in the motion estimation step. SVO keeps a fixed number of keyframe in its map and only optimizes point locations and camera poses in a local window, therefore, it is not a full SLAM system but a VO system.

DSO is a recently proposed fully direct sparse VO system [13]. Without

Figure 2.4: Depth map optimization of DTAM [43]

extracting features at all, it tries to reconstruct depth of sparse pixels with high gradient in the images. By dividing the image into smaller blocks, the points can be evenly distributed in the image. It performs optimization on the intensity of all points and the camera poses within a sliding window all together. The windowed optimization and marginalization for keyframe and points help DSO achieve a fast speed, since only a relatively consistent number of parameters are used to estimate the motion for a frame. Furthermore, DSO considers not only geometric calibration of the camera, but also the photometric calibration. It shows that the photometric calibration can help direct methods achieve accurate result because they are trying to minimize the photometric error between images. Because DSO does not rely on any feature detector, it has the advantage that it can sample point across any region as long as there is intensity gradient, which helps the system to estimate the camera motion more robustly in areas without rich texture.

2.1.2 Dense monocular SLAM

In contrast to only reconstructing feature points in 3D, a dense SLAM system can calculate the 3D positions for most if not all the pixels in images while localize the camera at the same time.

Typical example of this kind of dense method is DTAM [43]. By using a dense and sub-pixel accurate MVS reconstruction method, depth maps can be created, where the depth of every pixel is computed by stereo matching using hundreds of images. After the initial estimation, the depth maps are further optimized globally for smoothness. Shown in Figure 2.4, from left to right, the depth map is incrementally optimized and regularized. On the right of Figure

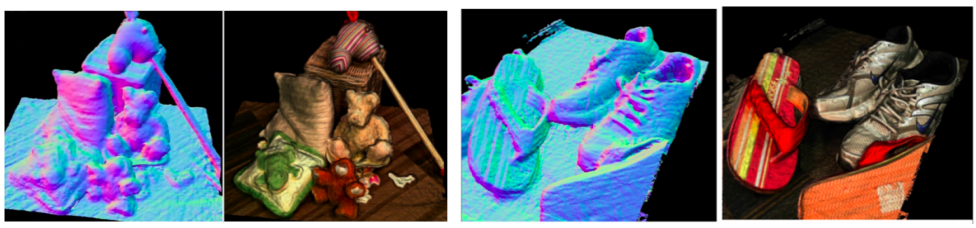


Figure 2.5: Sample result of MonoFusion [47]

2.4, result of PTAM is shown for comparison. The depth map reconstruction process is highly parallelisable and is sped up using a GPU. The reconstructed dense 3D model is used to track the camera location by projecting the entire model to a virtual camera to synthesize a novel view of the model. Then a 6 DoF direct whole image alignment is performed to refine the estimation of camera pose and minimize the photometric error between the synthesis and current frame. This tracking process can be implemented efficiently using a GPU and runs at frame-rate. One of the advantages of tracking the camera against a dense model is the capability to predict surface which means that it is easier to handle occlusion and moving parts in the scene.

Methods like DTAM need to produce a dense map in order to estimate the camera motion because images are matched against the reconstructed model. However, other SLAM/VO can also be combined with MVS methods to produce dense reconstruction of the environment. Essentially, given camera poses and corresponding images, MVS methods can be used to reconstruct dense depth maps by considering multiple images at a time. Before DTAM, Newcombe and Davison proposed a real-time dense reconstruction system built upon PTAM [41]. They compute depth maps of selected frames using the dense correspondence measurement in a bundle of nearby frames. The depth maps are then integrated into a global model. In [47], Pradeep *et al.* present MonoFusion, a dense reconstruction system based on volumetric fusion, where the depth map of keyframes are computed by stereo matching. They also use a method similar to PTAM to track the camera motion. Sample result of MonoFusion is shown in Figure 2.5.

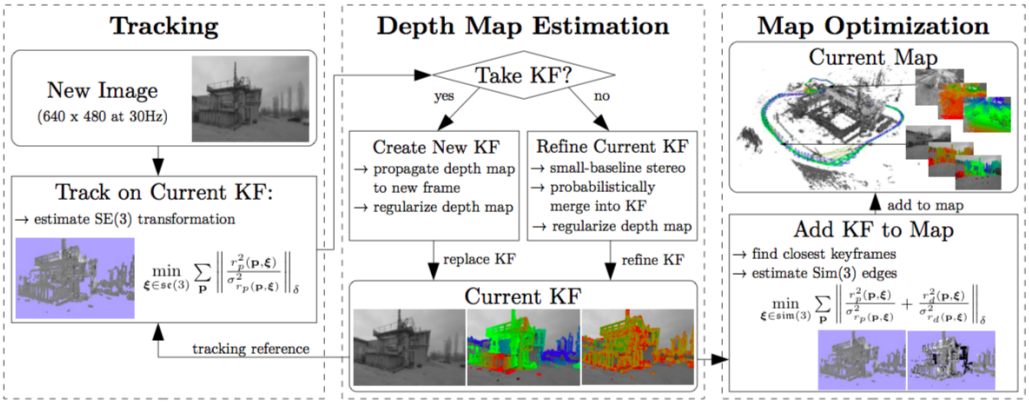


Figure 2.6: System overview of LSD-SLAM [14]

2.1.3 Semi-dense monocular SLAM

Between sparse and dense SLAM systems, semi-dense methods can reconstruct the depth of part of the image. Typically, semi-dense methods can reconstruct depth of pixels with high enough intensity gradient.

An example of semi-dense SLAM is LSD-SLAM [14]. The system overview of LSD-SLAM is shown in Figure 2.6. LSD-SLAM estimates the depth for all the pixels with non-negligible image gradient. By representing the probability of the inverse depth of each pixel as a Gaussian distribution, it can be propagated and updated over time through multiple image frames. The estimated pixel depth is then used to track the camera motion by direct image alignment. Essentially, the process of direct alignment with depth in consideration is estimating the 6 DoF camera motion by aligning current image to the synthesis of the reconstructed depth map of a keyframe. By only trying to estimate and use the depth of pixel with large image gradient, LSD-SLAM achieves real-time performance with only a CPU. Furthermore, a scale-drift aware formulation of direct image alignment method for estimating the 7 DoF similarity transform is proposed to construct pose graph and detect loops. Therefore, loop closure can be performed by pose graph optimization, which gives LSD-SLAM the ability to perform well on large scale environment. Sample result of LSD-SLAM is shown in Figure 2.7. In LSD-SLAM, the smoothness of pixel depth is considered during estimation, which means pixels whose depth is estimated

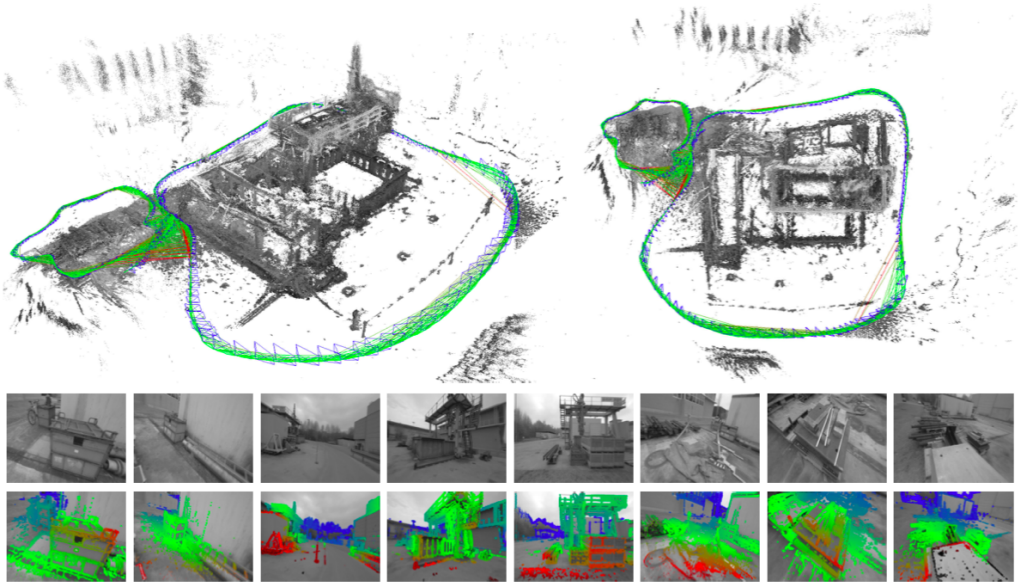


Figure 2.7: Sample result of LSD-SLAM [14]

need to have neighboring pixels to support it. The reconstructed pixels are mostly connected with some other pixels. Therefore, LSD-SLAM is classified as semi-dense whereas DSO is classified as sparse although it can have similar point density with LSD-SLAM.

Similar with dense method, the reconstructed semi-dense map does not have to be used in tracking. Semi-dense map can also be reconstructed by MVS methods which focus on high gradient areas on the image. Mur-Artal and Tardós present a probabilistic semi-dense mapping method for ORB-SLAM [38]. As shown in Figure 2.8, they build upon ORB-SLAM and developed a separate module for creating semi-dense point cloud. In this work, they estimate the depth of high gradient pixels in keyframes. Multiple hypothesis of the depth are established by searching across the epipolar line in other frames. These hypothesis are integrated by using a probabilistic formulation. The depth of pixels have to be supported by both the pixels in the same frame and the pixels projected to the same location from other frames. After removing uncertain pixels and pixels that are not supported, accurate semi-dense depth map can be obtained. Sample results of the reconstructed semi-dense point cloud is shown in Figure 2.9. Since they only consider pixels with large gradient

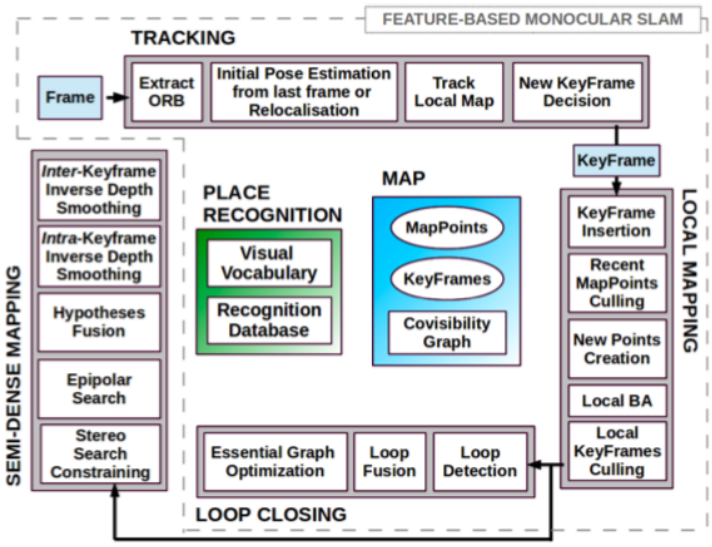


Figure 2.8: Overview of the semi-dense reconstruction system in [38]

and utilize a fast multi-view depth estimation method, their method can run in nearly real-time on a multi-core CPU.

2.2 3D Line segment detection

Although 3D points are the most commonly used 3D representation in SLAM methods, there are several limitations in representing scenes using point clouds. First, points in a point cloud do not store any structural information which is essential for applications like surface reconstruction. The lack of structural information makes point clouds easy to generate, but also makes them hard to infer any geometric relations among the points. Second, point clouds are not efficient in representing the geometric structure of a scene. A simple 3D line segment can be easily described by a line segment, but it needs a large number of points to represent its continuity.

Line segments representation preserves structural information of the scene much better than points. In fact, using line segment in computer vision is not a new idea. There is a long history of using line segments in areas like image registration and 3D reconstruction [3] [52]. However, most of the work focusing on 3D reconstruction are based on feature points and their powerful

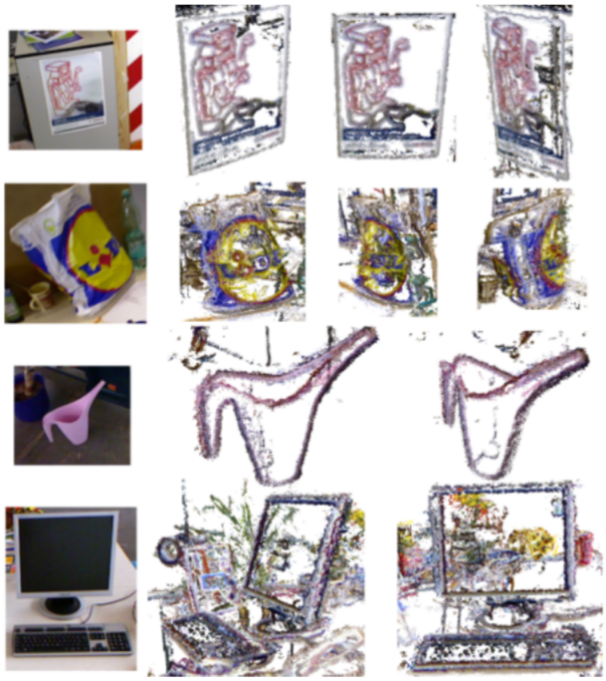


Figure 2.9: Sample result of semi-dense mapping in [38]

point descriptors like SIFT [35] and SURF [5]. Recently, invariant descriptors for line segments are also proposed and used to match line segments in different images [60] [66] [64]. Line segments have been used in applications like line segment based SFM methods [4] [65] [36] and line segment based MVS methods [27] [22]. Line segment based SFM methods try to estimate both the camera pose and 3D line segments at the same time, while line segment based MVS methods try to reconstruct the 3D line segments assuming the camera poses are known.

In order to use the line segments in images, one has to detect those line segments first. 2D line segment detection has been studied for a long time. Many algorithms have been proposed to detect the line segments in an image. Methods like Hough transform [12], LSD [18] and EDLines [1] have been popular for use in various applications. However, these methods are hard to extend to 3D space directly. To detect line segments in 3D, typical method either detect 2D line segments first and match line segments across frames to reconstruct 3D line segments, or detect 3D line segments from point cloud by

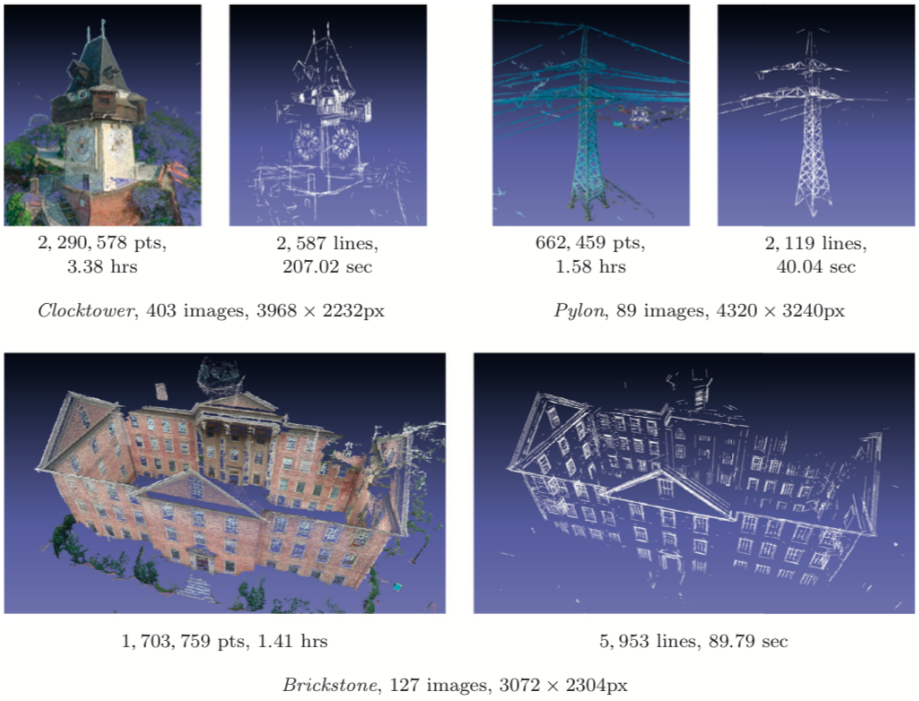


Figure 2.10: Sample result of Line3D++ [22] compared with result of PMVS [17]

3D line fitting.

The methods to detect 3D line segments can be categorized into three major classes: 3D line reconstruction from matching 2D lines, 3D line fitting directly from 3D points and 3D line extraction aided by 2D cues.

2.2.1 3D line reconstruction from matching 2D lines

Bartoli and Sturm proposed a line segment based SFM pipeline [4]. In their work, 3D line segments can be reconstructed efficiently, but the 2D line segments and their correspondences are manually labeled. Micusik and Wilder used relaxed endpoint constraints for line matching and developed a SLAM-like line segment based SFM system [36]. Hofer *et al*. matched 2D line segments detected from images of different views by pure geometric constraints [22]. All the candidates of possible 3D line segments are filtered and then clustered based on mutual support by solving a graph-clustering problem. Their 3D line segment reconstruction system is called Line3D++. Sample result of this work

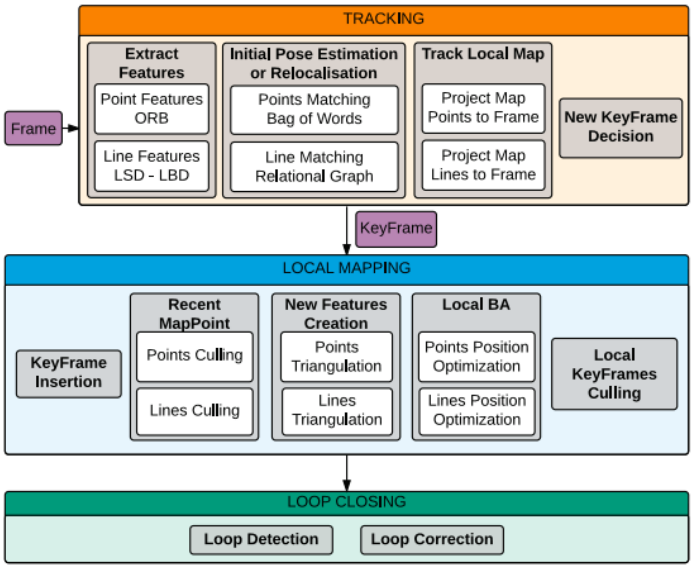


Figure 2.11: System overview of PLSLAM [48]

is shown in Figure 2.10. In [48], Pumarola *et al.* proposed PLSLAM, which detect line segments using LSD [18] and match them based on LBD descriptor [64]. As shown in Figure 2.11, they utilize the correspondences of both points and line segments to improve the robustness and accuracy of feature based SLAM system.

2.2.2 3D line fitting directly from 3D points

Roberts proposed a new representation for a line in [49]. Using this representation, Snow and Schaffrin developed an algorithm for solving the Total Least-Squares problem of 3D line fitting [53]. A directly fitted 3D line result is shown in Figure 2.12. However, this kind of methods are sensitive to noise and outliers. Random Sample Consensus (RANSAC) is relatively robust to small number of outliers [15]. However, RANSAC is time consuming in 3D space and the optimal line fitting is not guaranteed in the presence of a large amount of outliers.

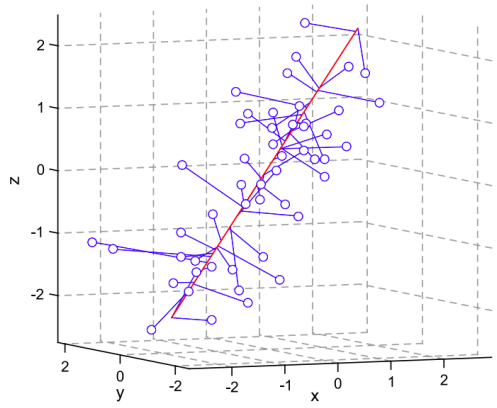


Figure 2.12: Directly fitted 3D line in [53]

2.2.3 3D line extraction aided by 2D cues

Woo *et al.* [61] detected 2D line segments from 2D aerial images first, and then used their corresponding 3D points on buildings' Digital Elevation Model (DEM) to fit 3D lines. Given RGB-D sensor, Nakayama *et al.* [40] transformed 2D points on detected 2D line segments directly to 3D using corresponding depth image. Then, 3D line segments are fitted by RANSAC in 3D space. Sample result of their method is shown in Figure 2.13.

Since most of the existing SFM or SLAM systems output point clouds as their mapping results, we prefer to make full use of these results and extract 3D line segments using both image and point cloud information. Our method belongs to the third class because we extract line segments from the semi-dense point cloud with the help of detected 2D edges. Specifically, instead of detecting 2D line segments and finding their corresponding 3D points for fitting 3D line segments afterwards, we fit the line segment in 3D by iteratively fitting its projections in two different planes. In other words, we directly detect 3D line segments by taking both 2D locations and corresponding depth of detected edge pixels into consideration in the line segment detection procedure.

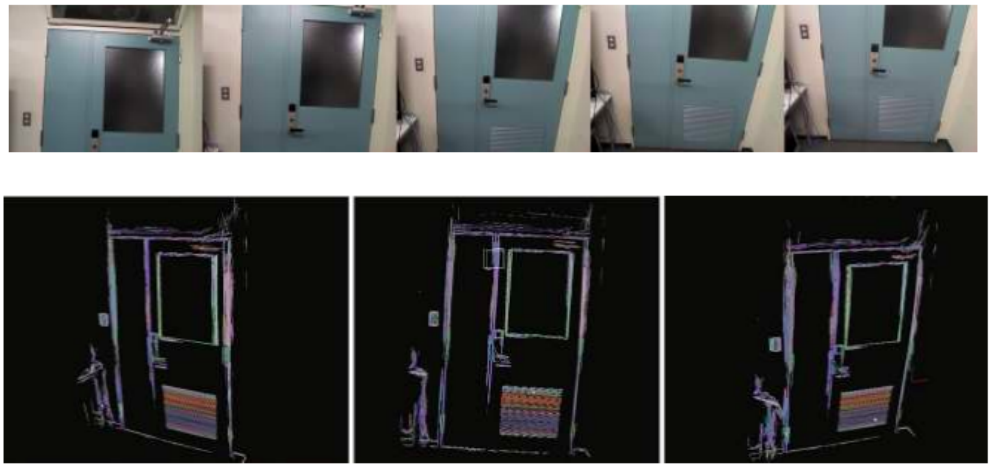


Figure 2.13: Sample of reconstructed 3D line segments in [40]

2.3 Incremental surface reconstruction

Surface reconstruction from point cloud has been studied for decades and is a very important aspect for computer vision and other fields, especially computer graphics [6]. The map reconstructed by a SLAM system can be converted to a point cloud. On one hand, for feature based SLAM system, the points are the 3D feature points detected and tracked across frames. The points are usually sparse, and duplicate points, which have the same or very close 3D location, seldom occur. On the other hand, for direct method, the point cloud are usually reconstructed by simply projecting the depth map into a single 3D world coordinate. In this case, the number of points in the point cloud is often much larger than those from feature based SLAM, and a large portion of the points can be duplicate points in 3D space. As the SLAM exploring more area in the environment, the map is expanding and new points are inserted to the point cloud. To reconstruct a surface that is constantly expanding, the surface reconstruction method needs to work in an incremental fashion.

Surface reconstruction from point cloud methods can be roughly categorized into two categories [29]: Explicit and Implicit.

2.3.1 Explicit surface reconstruction

Explicit surface reconstruction method works directly on the points and triangulate them to construct the mesh of the surface. In this section, we introduce a line of works that reconstructs the surface by discretizing the space into tetrahedra. In the context of surface reconstruction from result of incremental SFM or SLAM, methods like [34] [24] [32] use 3D Delaunay Triangulation to discretize the space into tetrahedra so that each point in the point cloud is a vertex for the tetrahedra. A surface in this discretized space then consists of connected faces of tetrahedra. By classifying all tetrahedra into "free-space" or "occupied", the surface can be approximately represented by the faces that divide the "free-space" tetrahedra and "occupied" tetrahedra. Naturally, the triangle faces forms a mesh that explicitly represents the surface.

The Delaunay Triangulation of 3D points is a valid choice for discretizing the space because the property that it has. As mentioned in [31] [2], under reasonable assumptions, the Delaunay Triangulation contains a good approximation of the surface if the point cloud is dense enough. By carefully and correctly selecting a subset of the facets in the triangulation, an approximation of the surface can be obtained.

Labatut *et al.* proposed a efficient method for reconstructing surface from point clouds [31]. They use 3D Delaunay Triangulation to discretize the space. Photometic consistency, visibility information and surface smoothness are used to determine whether a certain tetrahedron is "free-space" or not. They formulated the labeling problem as a graph-cut problem operating on the 3D dual graph, where each tetrahedron is a node in the graph and each edge represent a face connecting two tetrahedra. In this way, different regularization criteria can be considered together by converting them into weights of the edges in the graph. After solving this graph-cut problem and obtain the minimum cut, the surface is then represented by the edges that are cut. Although this method is efficient compared to other MVS methods at the time, it is still not real-time and can not be applied to incremental systems like SLAM.

Lovi *et al.* applied the idea of using 3D Delaunay Triangulation in surface

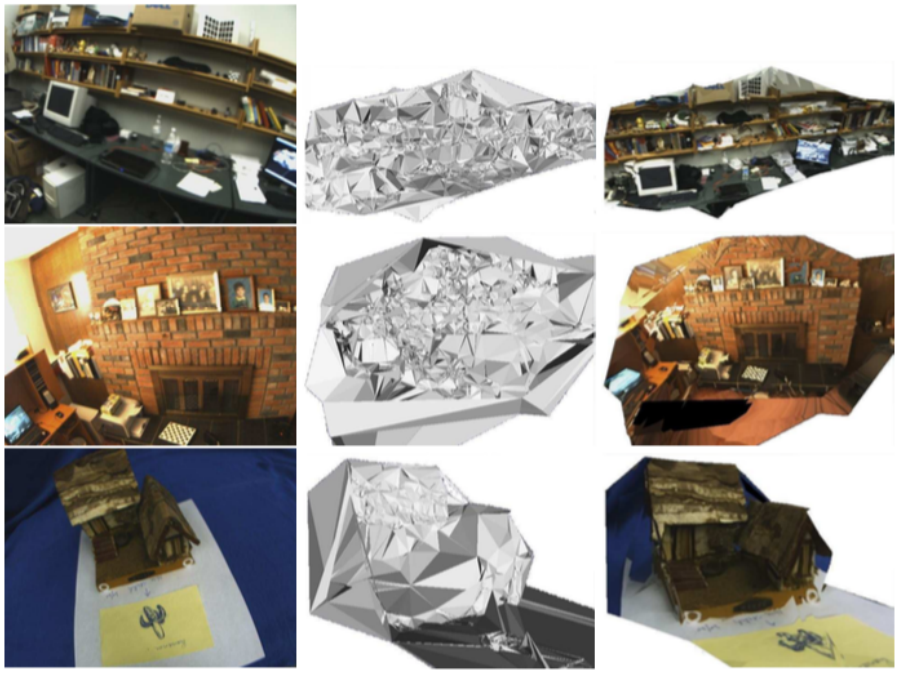


Figure 2.14: Sample result of surface reconstruction system in [34]

reconstruction from the point cloud of sparse SLAM system [34]. Some result of the surface reconstruction system is shown in Figure 2.14. After discretizing the space by triangulate the 3D points, they carved out all the tetrahedra intersecting with any viewing ray that connects a 3D point with a camera position. By using this space carving method that only considers visibility information, the computational cost for the surface reconstruction is greatly reduced. They also incrementally handles all the different events that affect the point cloud in the SLAM system including point addition, point deletion and point moving. When a point is added or deleted, the structure of triangulation is changed only in a local region. Thus, real-time performance can be achieved when operating on the point cloud from a sparse SLAM system. They applied the method to PTAM, where the sparse 3D points are used to reconstruct the surface. Each viewing ray is constructed by connecting the camera location of a keyframe and a 3D point. Within the implementation, in order to correctly handle all the events from PTAM, information of the viewing rays that intersect with a certain tetrahedron are all kept in the tetrahedron. They

proposed a forgetting heuristics that only keeps a single viewing ray for each tetrahedron. With the help of this forgetting heuristics, the computational cost is reduced and real-time performance is achieved.

Hoppe *et al*. proposed an incremental surface reconstruction method for online SFM methods and SLAM systems [24]. In this work, similar to [31], they discretize the space into tetrahedra by Delaunay Triangulation and perform graph-cut on the dual graph. They proposed a new formulation for the weight of the edges. In contrast to the formulation in [31], they only consider the visibility information and surface smoothness in assigning the weight. Also, only local information is used, which makes the method efficient when applied to large point clouds. By incrementally adding points into the 3D Delaunay Triangulation structure, their method can run in real-time on the point cloud from online SFM. However, the deletion and moving of points are not handled in this work. To obtain the minimum cut in real-time, dynamic graph-cut is used to efficiently solve the problem with increasing amount of points. They showed that by using dynamic graph-cut, the computational cost is relatively consistent and independent to the increasing number of points.

2.3.2 Implicit surface reconstruction

On the contrary to explicitly triangulate the points to obtain the mesh, implicit methods use intermediate representation to store and process the information of points and then extract the surface mesh from the implicit representation as a separate step. Essentially, the difference between implicit and explicit methods is the way they discretize the space. In the context of real-time incremental surface reconstruction, the space is usually discretized into voxels, which are regular cubes in the space, all aligned to certain axes. In this section, we introduce a line of work that try to integrate the dense depth maps produced by the SLAM systems into a consistent volumetric representation using Truncated Signed Distance Function (TSDF).

A TSDF is the truncated version of a signed distance function. The value of a signed distance function in a voxel represents its distance to the closest zero crossing, which corresponds to the surface. The value is positive on the

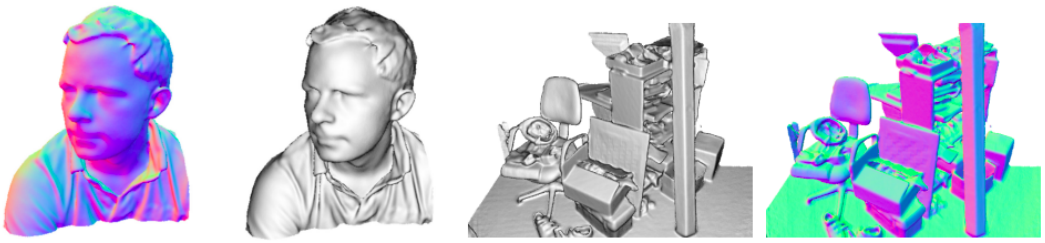


Figure 2.15: Sample result of KinectFusion [34]

free-space side of the surface and negative on the non-visible side. Since the information far away from the surface is not necessary for reconstructing the surface, the signed distance can be truncated at a certain value to reduce computation needs. Under assumptions, this truncation does not affect the final result. Multiple surface measurements can be integrated into a single global representation using TSDF. To extract a mesh of the surface out of the voxels, methods like Marching Cubes [33] can be used.

A very influential work that utilized the voxel volumetric representation is KinectFusion [42]. In this work, Newcombe *et al.* developed a SLAM system for Kinect, an inexpensive RGB-D camera that, along with the RGB images it captures, can produce depth maps in real-time using its infrared sensor. Given the dense depth maps, they adopted the method proposed by Curless and Levoy [10], which integrates all the depth measurements into a cumulative weighted signed distance field. The method is robust to noise and works incrementally, which are essential properties for a method to be used for a RGB-D SLAM system. By adapting this TSDF based method to run in parralel on a GPU, Newcombe *et al.* achieved real-time performance for integrating depth maps and reconstructing the surface. The camera poses of the depth maps are key components for correctly integrating the depth maps. They are estimated by performing a multi-scale ICP alignment between current sensor measurement and the predicted surface raycasted from the global TSDF. Being fully parallelsible, the camera tracking algorithm can be also implemented for a GPU. The full system can run in real-time to reconstruct the surface of an indoor area with the help of a GPU. Sample result of KinectFusion is shown

in Figure 2.15. Although they use a RGB-D camera in this work, their surface reconstruction method can also be used for other methods that generate dense depth maps. For instance, similar surface reconstruction methods are used in MonoFusion [47] and MobileFusion [46] to reconstruct the surface using a monocular RGB camera.

Although the TSDF based surface reconstruction method in KinectFusion [42] is impressive, it has its drawbacks. First of all, since each voxel needed to be allocated with a block of memory in the GPU, the reconstructed surface area is limited by the available GPU memory. In later works, several extensions are proposed to overcome this limitation. Steinbrücker *et al*. used a octree-based voxel grid to extend the working area [54]. Nießner *et al*. proposed a method called Voxel Hashing to dynamically allocate memory to the voxels on demand [44]. Another drawback of the TSDF based surface reconstruction methods is that they are computationally expensive. Although the voxels can be processed in parallel, a GPU is usually required to achieve real-time performance. There are works focusing on speeding up the reconstruction and run it on a CPU [54] [45]. Although these methods runs in real-time on the CPU, they focus on using the dense depth maps from RGB-D cameras, and are not well suited for our semi-dense results from SLAM.

2.3.3 Surface reconstruction with lines

Besides using only the points, some works also use lines in surface reconstruction. Hofer *et al*. mentioned using line segment to improve quality of reconstructed surface as an application for their line segment reconstruction method in [21] [20]. Sample result of method in [20] is shown in Figure 2.16. From left to right, sample input image, sparse point cloud, points combined with lines, surface reconstructed using points and surface reconstructed using points and lines are shown respectively. Although in these works, they did not describe the method they use to incorporate the information provided by the line segments, they showed the quality and completeness of the surface can be greatly improved by providing line segments in addition to the sparse point cloud.

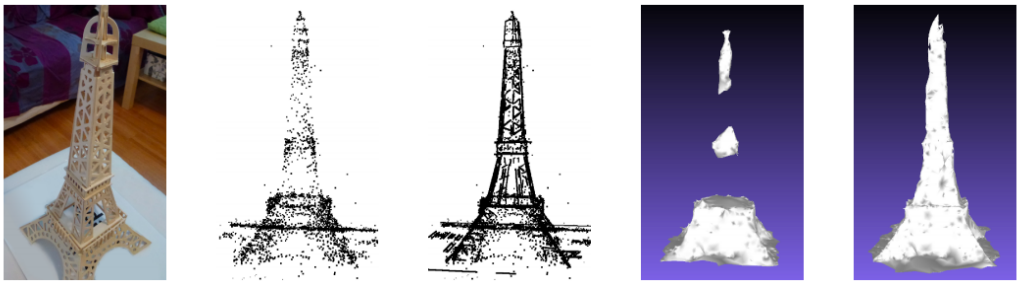


Figure 2.16: Sample result of surface reconstructed using points and lines in [20]

Sugiura *et al*. proposed a method to incorporate line segments in reconstructing the surface after running SFM on a set of images [56]. An overview of their system is shown in Figure 2.17. In this work, they adopted a similar strategy with [31] to reconstruct a surface using the points from SFM. The difference is that here they only consider the visibility information and surface smoothness to extract the surface. An initial surface is extracted to help the reconstruction of 3D line segments. To reconstruct the 3D line segments, they first detect the 2D line segments in the images using LSD [18] and then match them in different images. In order to match the 2D line segments, the pixels on the 2D line segments are matched across frames on the epipolar lines by comparing their DAISY descriptors [58]. The depth of a pixel on the estimated initial surface is used as the start point for epipolar search. To match a certain line segment, a 2D line segment will be fitted to the matched pixels on another image. If there exists a detected 2D line segment that is close to the fitted line segment, then it will be taken as the matched line segment. After obtaining the 2D line segment correspondences, 3D line segment are reconstructed by triangulation. Given the 3D line segments, they updated the Delaunay Triangulation structure by regularly sampling points on the 3D line segments and inserting them into the point cloud. To handle the visibility information of the line segments, similarly to carving out using rays from camera to 3D points, they carved out tetrahedra that intersect with the triangle formed by the camera and two end points of the 3D line segment. As a result, the quality of the surface is greatly improved with all the additional points and

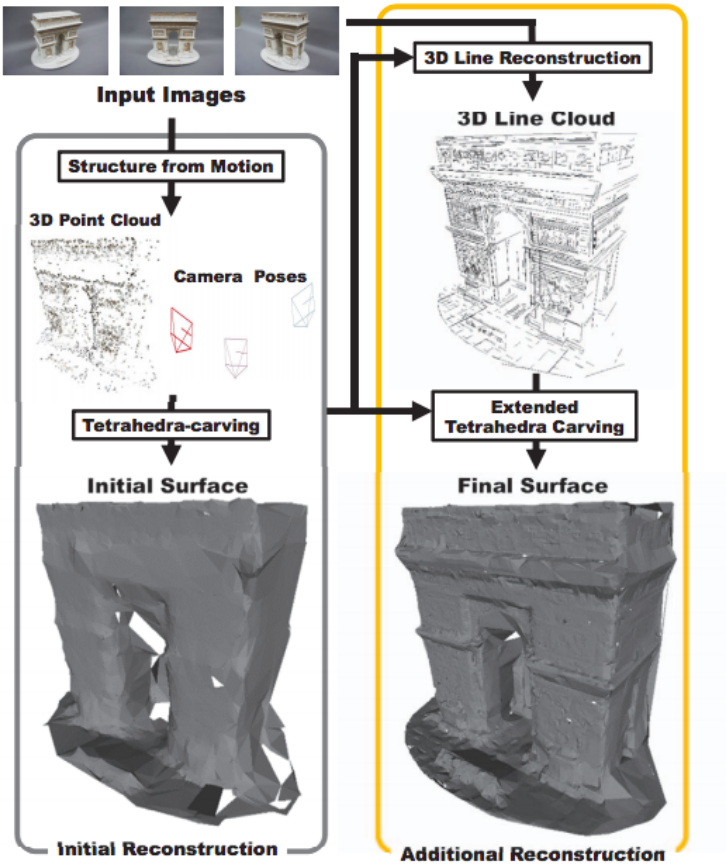


Figure 2.17: Overview of surface reconstruction system with points and lines in [56]

visibility information. Their approach of extracting the 3D line segments is actually very similar to our method in that it also compute pixel matching before reconstructing 3D line segments. In our method, we fit 3D line segments directly to the computed depth of pixels. The difference in this work with our method is that they only use the depth of the pixels as a criterion to match line segments in different images.

Chapter 3

Method

In this chapter, we present the detail of our method for 3D line segment extraction and incremental surface reconstruction. The overall workflow of our system is shown in Figure 3.1.

3.1 Incremental 3D line segment extraction

We introduce our incremental 3D line segment extraction method in this section. The input of our method is a video (image sequence) which is captured by a moving camera with known intrinsic parameters. The output is a line segment based 3D model of the scene. In order to extract 3D line segments from semi-dense point cloud, our method performs the following steps on each new keyframe (shown in Figure 3.1):

1. Compute semi-dense depth map
2. Edge aided 3D line segment fitting
3. 3D line clustering and filtering

3.1.1 Keyframes and depth maps generation

Our method operates on images and semi-dense depth maps of keyframes in SLAM systems. We implemented our method with two base SLAM systems: ORB-SLAM [37] and LSD-SLAM [14].

ORB-SLAM [37] is a feature based SLAM system which takes the image sequence from a moving camera and computes the camera poses in real time.

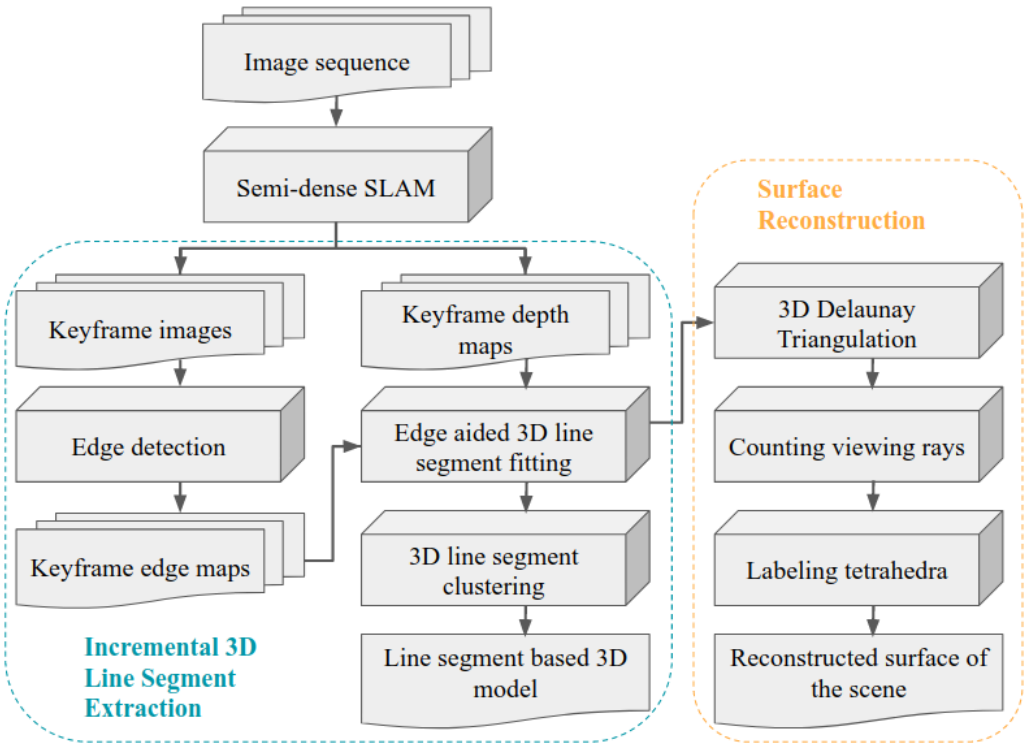


Figure 3.1: Workflow of our method

By applying advanced keyframe management, powerful feature descriptor, local and global bundle adjustment and loop closure detection, it can robustly track the camera pose and map the environment. Mur-Artal and Tardós [38] present a semi-dense module which is able to compute a semi-dense depth map for each keyframe. Although ORB-SLAM is originally a feature-based sparse SLAM system and the computation of depth map does introduce some computation overhead, the final point cloud is clean and accurate, which enables us to extract 3D line segments out of the point cloud with few outliers. As a side effect of using multiple keyframes as references to compute the depth map, the depth map computation is delayed by a few keyframes, i.e. the depth map is not constructed as soon as the keyframe is created. The delay can be beneficial as it allows the local bundle adjustment to optimize the camera pose, but it is a major limitation for extracting 3D line segments in real time.

LSD-SLAM [14] is a direct semi-dense SLAM system which computes the camera poses by direct alignment. Compared to ORB-SLAM with semi-dense

module, depth maps of LSD-SLAM are denser and noisier [38]. However, it can generate semi-dense depth maps for all its keyframes without delay, which enables our line segment extraction method to perform in real-time.

In Section 4, we demonstrate results of our method using the two SLAM systems mentioned above. In principle, other keyframe based dense or semi-dense SLAM system could be used to generate the semi-dense depth maps as well.

3.1.2 Edge aided 3D line segment fitting

Fitting 3D line segments directly from point clouds is difficult and time consuming. It is challenging to develop criteria for separating points into different groups in order to fit line segments. In this thesis, we propose to use 2D image information on keyframes to help 3D line segment fitting from semi-dense point clouds.

We first extract edge segments from keyframes using Edge Drawing [59]. Edge Drawing is a linear time edge detector which can produce a set of accurate, contiguous and clean edge segments represented by one-pixel-wide pixel chains. Now the detected edge segments of the a keyframe can be expressed as $ES = \{es_1, es_2, ..., es_n\}$ where $es_i = \{p_1, p_2, ..., p_m\}$ is an edge segment formulated as a pixel chain. p_i represents a pixel which is a vector of (x, y, Z) where x and y are the image coordinates and Z is its corresponding depth in the semi-dense depth map. The number of edge segments in a keyframe and number of pixels in an edge segment are denoted by n and m respectively. It is worth noting that image pixels with high intensity gradient are more likely to be selected for computing depth value in the semi-dense SLAM system. Edge segments are detected based on pixel intensity gradients as well. Thus, the detected edge pixels are very likely to have depth values. The pixels which have no depth values will be considered as outliers in the line fitting process.

3D line segments of a keyframe are extracted from those detected image edge segments by Algorithm 1. The main idea of this algorithm is to reduce a 3D line fitting problem to two 2D line fitting problems. The coordinate frames are defined as shown in Figure 3.2. $\mathbf{C - XYZ}$ is the camera coordinates. The

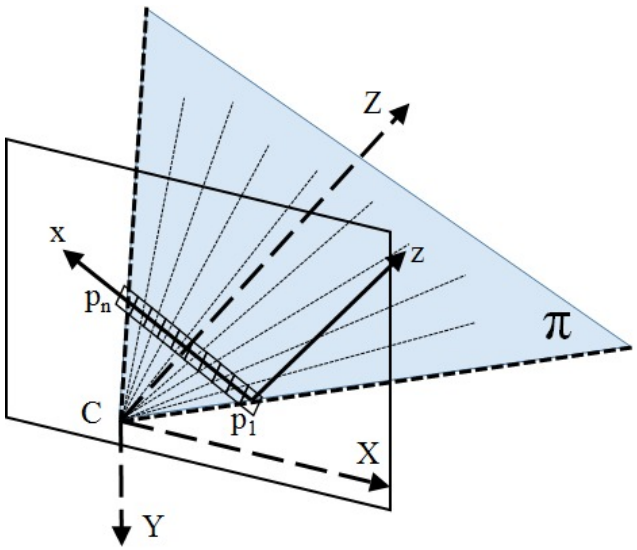


Figure 3.2: Line segment fitting coordinates

X-axis and Y-axis are parallel to the image coordinates. The Z-axis is the depth direction. $\{p_1...p_n\}$ represent a detected 2D image line segment. Line segment pixels $\{p_1...p_n\}$ and their corresponding real-world points are all located on the same plane π. The x-axis is defined by vector $\overrightarrow{p_1p_n}$ while z-axis is parallel to the Z-axis. For each edge segment, the algorithm initially takes its first L pixels to fit two 2D lines (l_{im} and l_{depth}) in the image coordinate frame and the $\mathbf{p_1}$-$\mathbf{xz}$ coordinate frame using total least square method. The line l_{im} is fitted based on the pixels' (x, y) values while l_{depth} is fitted based on (D, Z). Z is the pixel's depth and D is the distance from p_1 to the pixel's projection on the x-axis. Total least square 2D line fitting is performed by solving Singular Value Decomposition (SVD) [19]. It is worth noting that the plane $\mathbf{p_1}$-$\mathbf{xz}$ is not always the same with plane π, which is determined by C, p_1 and p_n. The plane $\mathbf{p_1}$-$\mathbf{xz}$ is orthogonal to the image plane where l_{im} is fitted. The two planes, $\mathbf{p_1}$-$\mathbf{xz}$ and π, are the same plane only if the image center lies on the line segment p_1p_n. Although the 3D points may not be located on the plane $\mathbf{p_1}$-$\mathbf{xz}$, it is much easier to perform line fitting on $\mathbf{p_1}$-$\mathbf{xz}$ rather than π. Given the image plane coordinate of the points, actual 3D positions can be easily recovered using the depth of points in $\mathbf{p_1}$-$\mathbf{xz}$ plane.

Algorithm 1 Edge aided 3D line segment fitting

Input: An edge segment which is a list of pixels: $es = \{p_1, p_2, ..., p_m\}$, where p_i denotes the i-th pixel on the edge segment

Output: A set of fitted 3D line segments: LS

Initialize two empty set of pixels: $pixels$, $outliers$
for each $p_i \in es$ **do**
    if $pixels = \varnothing$ **then**
        Move first L pixels in es to $pixels$
        Fit two 2D lines l_{im} and l_{depth} to $pixels$
    end if
    Compute distance d_{im} from $(p_i.x, p_i.y)$ to l_{im}
    Compute distance d_{depth} from $(p_i.D, p_i.Z)$ to l_{depth}
    if $d_{im} < e_1$ & $d_{depth} < e_2$ **then**
        Move p_i to $pixels$
    else
        Move p_i to $outliers$
    end if
    if $es = \varnothing$ || $|outliers| > L$ **then**
        if $|pixels| > L$ **then**
            Fit l_{im} and l_{depth} to $pixels$
            Compute the 3D line segment and add to LS
        end if
        Empty $pixles$ and $outliers$
    end if
end for

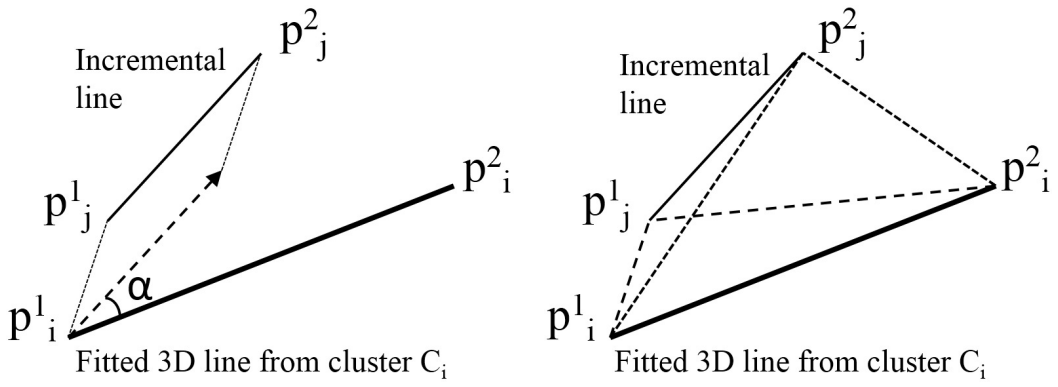
After obtain an initial line segment, we compute the distances of the next pixel in pixel chain to l_{im} and l_{depth} in their corresponding coordinate frames. Note that D and Z have different units. To have the same unit as D, Z is multiplied by the focal length f before distance computation. If both distances are smaller than certain thresholds, e_1 and e_2 respectively, we will add the pixel to the fitted pixel set to extend the line. Otherwise the pixel will be considered as an outlier. If L consecutive pixels are outliers, we stop the current line segment search and start a new line segment search. Another pair of total least square fittings on the two planes are performed to obtain the final 3D line for each line segment. The 3D line segments are represented by their endpoints, which are estimated by projecting the points corresponding to its first and last pixel on to the final 3D line. After traversing all of the edge segments of the keyframes, we can aggregate one 3D line segment set LS_k for each keyframe.

3.1.3 3D line segment clustering and filtering

To obtain a consistent reconstruction of the environment, 3D line segments $LS_{all} = \{LS_1, LS_2, ..., LS_n\}$ extracted from different keyframes are first registered to the same world coordinate system. The registered 3D line segments are denoted as $ls_{all} = \{ls_1, ls_2, ..., ls_w\}$. Here w denotes the total number of 3D line segments from all keyframes. Directly registering all 3D line segments will produce redundant and slightly misaligned result. We address this problem by proposing a simple incremental merging method.

The main idea of our merging method is clustering closely located 3D line segments and fitting those cluster sets with new 3D line segments. As illustrated in Figure 3.3, the angle and distance measures are used for clustering. The angle measure α is defined as:

$$\alpha = acos(\frac{\overrightarrow{p_j^1 p_j^2} \cdot \overrightarrow{p_i^1 p_i^2}}{|\overrightarrow{p_j^1 p_j^2}||\overrightarrow{p_i^1 p_i^2}|}) \tag{3.1}$$



(a) Angle measure (b) Distance measure

Figure 3.3: Clustering by angle and distance

The distance measure d is computed as:

$$d = min(d_1, d_2) \tag{3.2}$$

$$d_1 = |\overrightarrow{p^1_j p^1_i}| + |\overrightarrow{p^1_j p^2_i}| - |\overrightarrow{p^1_i p^2_i}| \tag{3.3}$$

$$d_2 = |\overrightarrow{p^2_j p^1_i}| + |\overrightarrow{p^2_j p^2_i}| - |\overrightarrow{p^1_i p^2_i}| \tag{3.4}$$

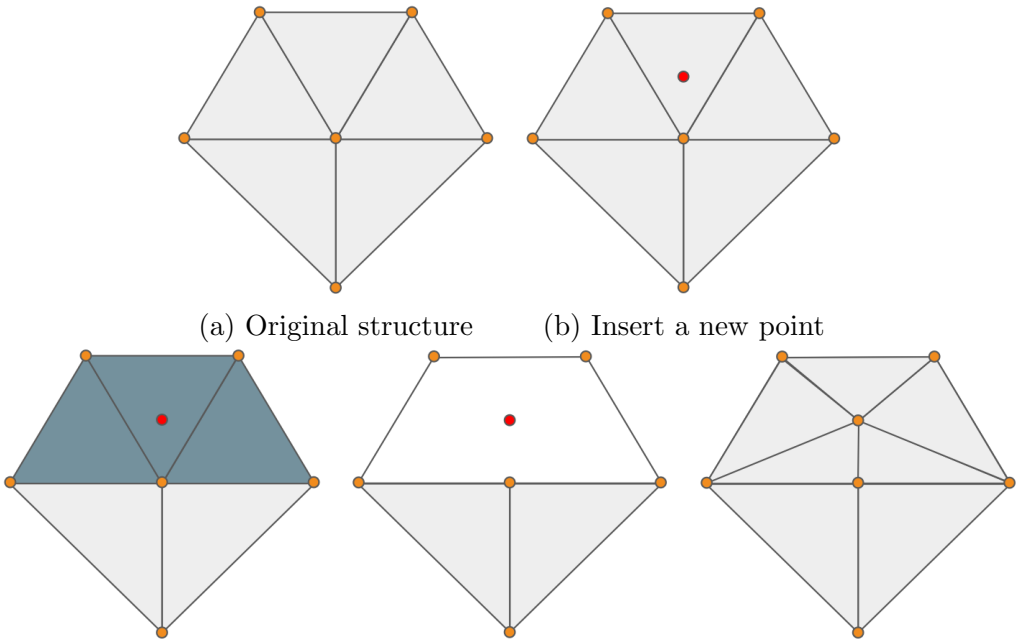
Specifically, we take the first 3D line segment ls_1 as the initial cluster C_1. Then, we compute the angle and distance measure between the initial cluster (single line segment) and the next 3D line segment ls_2. If the angle α and distance d are smaller than certain thresholds (λ_α and λ_d respectively), we add ls_2 to the cluster C_1. Otherwise, we create a new cluster C_2. For each cluster, if it contains more than one 3D line segments, we will fit a new 3D line segment to represent the cluster. The direction of the new line segment is determined by performing SVD on the matrix consisting of all points in P_{ep}, where P_{ep} denotes the set containing all the endpoints of line segments in this cluster. A new 3D infinite line is then determined by the direction together with the centroid of P_{ep}. In order to obtain a 3D line segment from this infinite line, we project endpoints in P_{ep} onto the generated infinite 3D line and compute the furthest projections with respect to the centroid in both directions. The 3D line segment between these two furthest projection points is taken as the fitted line segment of the cluster. This process is repeated until all the line segments in ls are clustered. Clusters with small size ($|C_i| < \lambda_C$)

are filtered out in the end. In this way, we can merge a large number of line segments into fewer clusters and generate new 3D line segments with higher quality.

3.2 Surface reconstruction

Given the extracted 3D line segments, we can apply incremental surface reconstruction methods [23], [34] to obtain a mesh representing the 3D surface. Our surface reconstruction operates on the endpoints of extracted line segments, and is based on the method presented in [34]. Our surface reconstruction pipeline is different with [34] mainly in how the tetrahedra is labeled. Although the space carving method used in [34] can accurately reconstruct the surface with correct viewing rays, it does not handle the outliers very well. If an outlier viewing ray is inserted into the structure, it may carve out tetrahedra incorrectly which can have a large impact on quality of the surface. To solve the problem, we applied a graph cut based approach to categorize the tetrahedra, which takes surface smoothness into consideration to alleviate the damage caused by outliers.

Although it is possible to use clustered line segments as the input to our surface reconstruction method, it would affect the speed of our system significantly. Reconstructing surface using line segment can involve sampling multiple points on the line segments [56]. Since a clustered line segment would update its final position after a new line segment is inserted to this cluster, we need to update all the points sampled on the line segment, which can be expensive to perform. In order to obtain real-time performance of the final system, we use the unclustered line segments for surface reconstruction. We can observe that the unclustered line segment are generally short line segments. Because the clustered line segments are computed from these short line segments, using endpoints of them naturally resembles the sampling procedure. Through experiments, we observe that using unclustered line segment can still produce high quality surface while being much more efficient than using clustered line segments.



(a) Original structure (b) Insert a new point

(c) Conflicting cells (d) Remove cells (e) Local triangulation

Figure 3.4: Inserting a point into an existing Delaunay Triangulation structure

Here we present the detail of our surface reconstruction method. The surface reconstruction method can be divided into the following three steps:

1. Discretize the space into tetrahedra by 3D Delaunay Triangulation
2. Assign weights to tetrahedra according to the viewing rays
3. Divide all tetrahedra into two-categories: free space and occupied space

3.2.1 3D Delaunay Triangulation

Given an initial point cloud, we can apply 3D Delaunay Triangulation to point cloud to obtain the tetrahedra. As a keyframe is passed to the surface reconstruction pipeline, all the new points from it are inserted into the structure. To insert the points, instead of performing 3D Delaunay Triangulation on all the points all together, we can perform incremental and local update to the structure. Since the number of points inserted at a time tends to be much smaller than the total amount of points, incremental insertion can be much

more efficient. Inserting a point into the structure of 3D Delaunay Triangulation involves three steps. First, we need to find all the conflicting cells, i.e., tetrahedra that will lose the Delaunay property if the point is inserted. Then all these cells and involved conflicting points are deleted. Finally, the new point and conflicting points are inserted into the structure to perform a local 3D Delaunay Triangulation. In Figure 3.4, we present the procedure of inserting a point into an existing Delaunay Triangulation structure. Although we use 2D triangulation in the figure for illustration, the procedure can easily generalize to 3D space. During the insertion of a point, only tetrahedra from a limited local region are involved, which ensures the efficiency of the process.

Deleting and moving points are also possible and can be achieved with similar procedure as presented in [34]. In our system, points are only inserted into the structure without deleting and moving, thus we omit the detailed description related to these procedures.

3.2.2 Counting viewing rays

After inserting all the new points from a keyframe, we process the viewing rays from the keyframe. We create a viewing ray counter for each tetrahedron, which records how many viewing rays are intersecting with the tetrahedron. Each viewing ray connects the camera position of the keyframe and a 3D point in the 3D Delaunay Triangulation structure. Here we can treat the viewing ray as a line segment in 3D space. We traverse all the intersecting tetrahedra along the line segment and increase their viewing ray counter. The viewing ray counters of tetrahedra are later used to separate free space and occupied cells.

An illustration of the process is shown in Figure 3.5, where a line segment p_1p_2 is visible to camera c. By traversing along the two viewing rays, cp_1 and cp_2, we increment the viewing ray counters in t_1, t_2 and t_3. Similar to the previous section, although we show a 2D example in the figure, the process can easily generalize to work in 3D space.

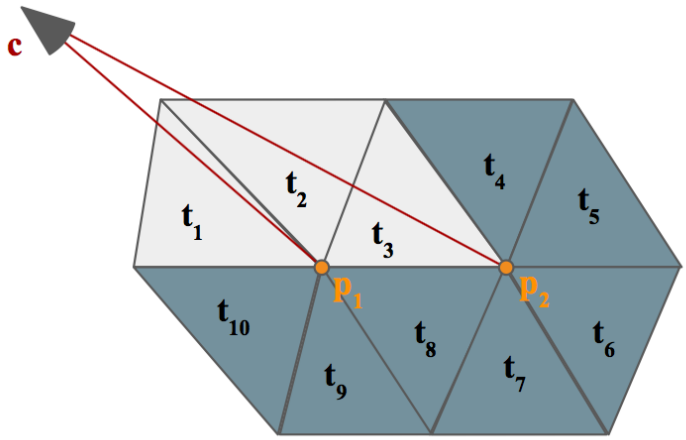


Figure 3.5: Processing viewing rays. p_1 and p_2 are end points of a line segments seen by camera c. Triangles $t_i, i \in [1, 10]$, are cells of the Delaunay Triangulation structure.

3.2.3 Labeling tetrahedra

We formulate the problem as a simple minimum cut problem in order to label all the tetrahedra. We create a graph G in which the vertices represent the triangulated tetrahedra. There are two extra vertices in G, source s and sink t, which represent the free space and occupied space labels respectively. Edge vertex is connected to s and t by two edges e_s and e_t. If e_s is cut then it means that the tetrahedron represented by the vertex does not belong to free space and vice versa. E_s represents a set consisting of all the e_s, while E_t consists of all the e_t. We collect the e_s and e_t for all vertices and name the set of edges E_{vis} since they represent the visibility of tetrahedra. Vertices representing adjacent tetrahedra are connected by edges E_{smooth}. Each edge $e_{smooth} \in E_{smooth}$ represents a triangle shared between the two neighbouring tetrahedra. An example of G is illustrated in Figure 3.6, which contains four vertices: v_1, v_2, v_3 and v_4. In this example, $e_{1,2}, e_{2,3}, e_{3,4} \in E_{smooth}$, while edges connected to s or t belongs to E_{vis}.

In order to label the tetrahedra, we assign different weights to all the edges

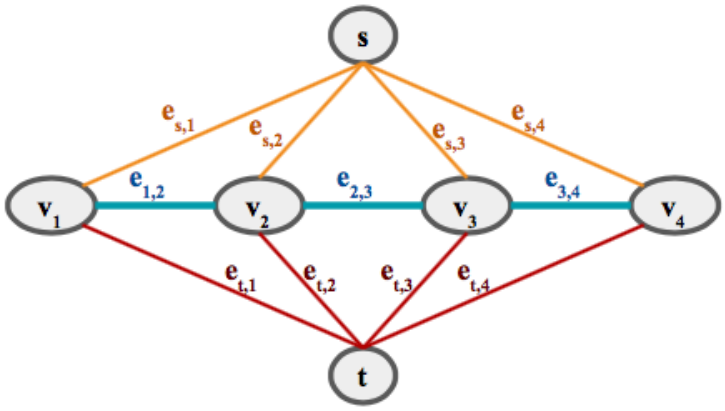


Figure 3.6: Example graph for labeling tetrahedra. s and t represent the source and sink respectively. v_1, v_2, v_3 and v_4 are vertices representing tetrahedra.

in G, and solve it by graph cut. We try to minimize the total cut weight W_{total}:

$$W_{total} = \sum_{E_{vis}} w_{vis} + \lambda_{smooth} * \sum_{E_{smooth}} w_{smooth} \tag{3.5}$$

In the above equation, w_{vis} represents the weight for visibility, which is contributed by the visibility rays. In order to take smoothness into account, we add w_{smooth} to the formula, through which we aim to reduce the number of neighbouring tetrahedra separated by the cut. λ_{smooth} is a user-defined parameter that balances the two weights.

For each $e_{vis} \in E_{vis}$, the w_{vis} of it is calculated as:

$$w_{vis} = \begin{cases} volume_v & \text{if } e_{vis} \in E_s \text{ and } count_v > 0 \\ 0 & \text{if } e_{vis} \in E_s \text{ and } count_v = 0 \\ 0 & \text{if } e_{vis} \in E_t \text{ and } count_v > 0 \\ volume_v & \text{if } e_{vis} \in E_t \text{ and } count_v = 0, \end{cases} \tag{3.6}$$

where $volume_v$ is the volume of the tetrahedron connected by e_{vis} and $count_c$ is the number of intersecting viewing ray of it. Essentially, we assign positive weight to an edge connecting a tetrahedron with a label vertex (s or t), if the viewing ray count agrees with the label. In this way, we encourage cutting the edge that connects to the opposite label. By using the volume of tetrahedra as weights, we aim to prevent smoothing out large tetrahedra in the labeling process, which helps to preserve major structures of the reconstructed scene.

Each e_{smooth} connects two adjacent tetrahedra and w_{smooth} in Equation 3.5 is calculated for each $e_{smooth} \in E_{smooth}$ as $area_{smooth}$, which is the area of the shared triangle of the two neighbouring tetrahedra. By using the area of the triangle as weights, we aim to prevent cutting through large facets. In other words, we aim to reduce the area of the reconstructed surface, which in effect smooths the surface.

After solving the graph cut problem, each cut e_{vis} represents the label of a tetrahedron. We can obtain all the cut e_{smooth}, which represents all the triangles that divided free space and occupied space tetrahedra. By connecting all the triangles, the mesh of the surface is reconstructed.

Chapter 4

Experiments

In this chapter, we present the results of our 3D line segment extraction and surface reconstruction method on image sequences from the TUM RGB-D dataset [55] and the EuRoC MAV dataset [7].

4.1 Implementation

The experiments in this chapter are performed on a desktop computer with a quad-core Intel i7-6700k CPU. We use two semi-dense SLAM as our base systems: ORB-SLAM [37] and LSD-SLAM [14]. LSD-SLAM is an open source direct semi-dense SLAM system [14], to which we can apply our method directly. For ORB-SLAM, we use the open source ORB-SLAM2 [39] package and implemented the semi-dense module in C++ as described in [38]. Parameters in Algorithm 1 and incremental line segment clustering are set as follows in all our experiments: $L = 0.02 * min(w, h)$, $e_1 = 0.002 * min(w, h)$, $e_2 = 0.003 * min(w, h)$, $\lambda_\alpha = 10$, $\lambda_d = 0.02$, $\lambda_C = 3$, where w and h denote the width and height of images respectively. In our surface reconstruction pipeline, we set $\lambda_{smooth} = 0.01$ for best results.

4.2 Qualitative Comparison

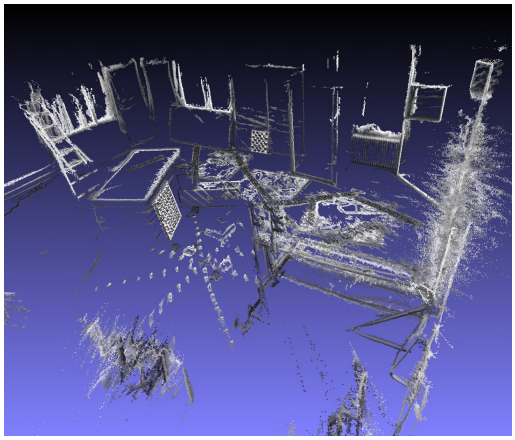
The results of our 3D line segment extraction method running on selected test sequences using ORB-SLAM are illustrated in Figure 4.1, Figure 4.3, Figure 4.5 and Figure 4.7. In Figure 4.2, Figure 4.4, Figure 4.6 and Figure 4.8, we

present the results of our method using LSD-SLAM. The results of our edge-aided fitting method accurately fit the semi-dense point clouds as shown in the (a) and (e) sub-figures of the above mentioned figures. They still capture the major structures of the scene while reducing the number of 3D elements greatly.

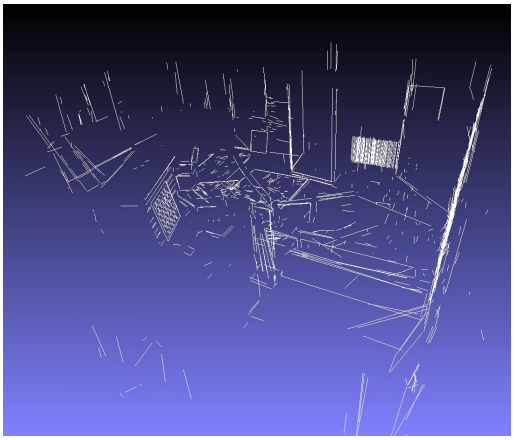
We first compare our results with those from Line3D++ [22]. The results of Line3D++ on the test sequences is shown in the (c) sub-figures of figures from Figure 4.1 to Figure 4.8. In our experiments, Line3D++ uses line segments detected by EDLines together with the keyframe images and camera poses output by ORB-SLAM to construct 3D line segments. However, since it relies only on the geometry constraints of line segments, it is fragile in some cases, such as complex indoor environment or areas without long, straight line segments. Therefore, Line3D++ tends to produce outliers due to the ambiguity of geometric line matching in such cases. In Figure 4.1c and Figure 4.2c, a large number of outlier line segments can be seen in area near the wall at the right side of image. On the contrary, our method utilizes the accurate semi-dense depth maps. Since the depth maps are checked multiple times and filtered to produce confident points, the results of our method have fewer outliers as shown in Figure 4.1e, Figure 4.1f, Figure 4.2e and Figure 4.2f .

In contrast to Line3D++, semi-dense points can cover regions with large image gradient, such as boundaries and contours, where straight lines may be absent. Since our method takes both intensity and depth information into consideration, it is robust to outliers caused by intensity noise so that it can extract shorter yet still accurate line segments than EDLines. Thus, our results fit curves better and captures finer details than Line3D++. As shown in Figure 4.3, the long curves on the right side of the image are not well captured by Line3D++ in Figure 4.3c, but can be clearly seen in the result of our method in Figure 4.3e and Figure 4.3f. Similarly, in Figure 4.4, we can observe that our method can fit line segments to the point cloud from areas where Line3D++ failed to extract line segments. Also in Figure 4.5, Figure 4.6, Figure 4.7 and Figure 4.8, it can be seen clearly that our method captures much more structural information compared to Line3D++.

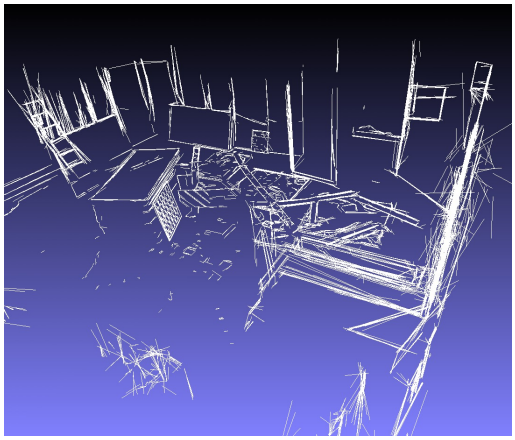
(a) Sample original image



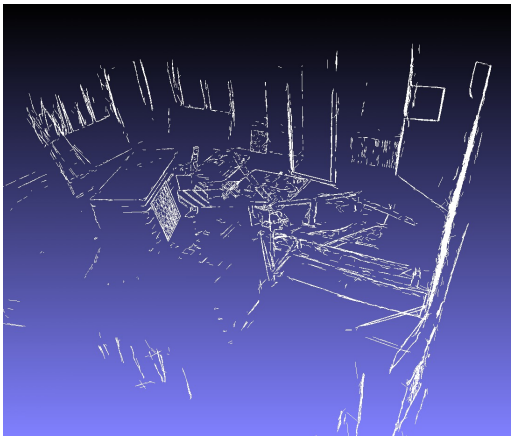
(b) Semi-dense point cloud



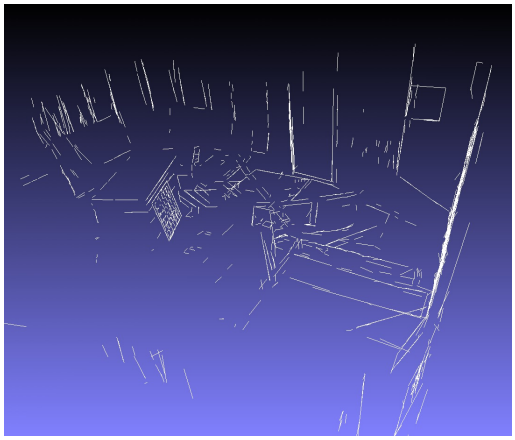
(c) Line3d++



(d) Decoupled fitting using EDLines



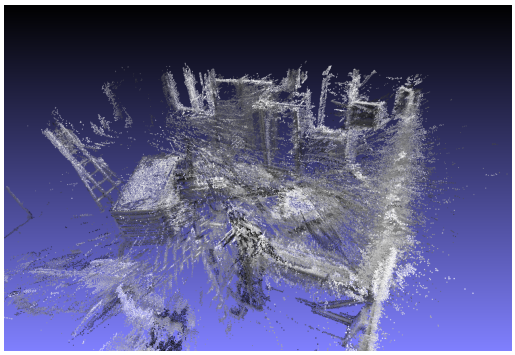
(e) Edge-aided fitting without clustering



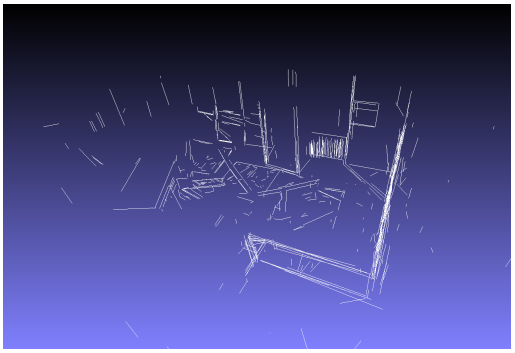
(f) Edge-aided fitting with clustering

Figure 4.1: Results of sequence EuRoC MAV Vicon Room 101 using ORB-SLAM. As shown in (c), a large number of outlier line segments are produced by Line3D++ near the wall area. Our method, on the other hand, produce fewer outliers as shown in (e) and (f). The noisy semi-dense point cloud shown in (b) is effectively simplified and filtered by our method to produce a accurate line segment based model with few outliers.

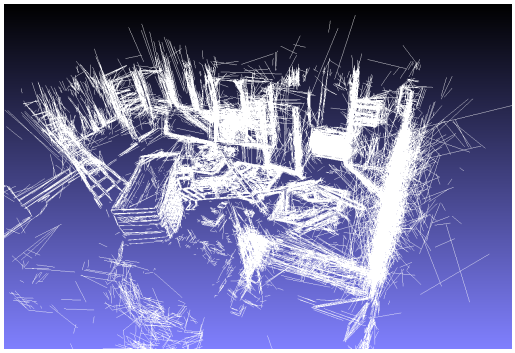
(a) Sample original image



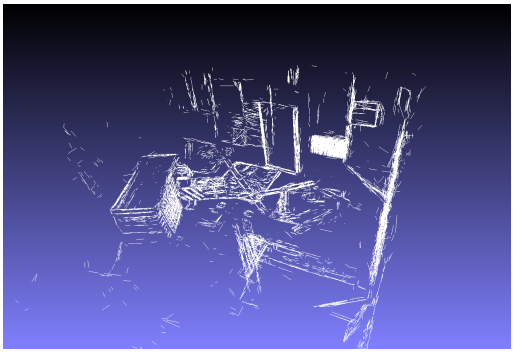
(b) Semi-dense point cloud



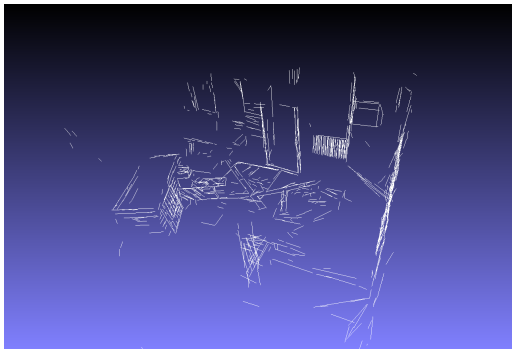
(c) Line3d++



(d) Decoupled fitting using EDLines



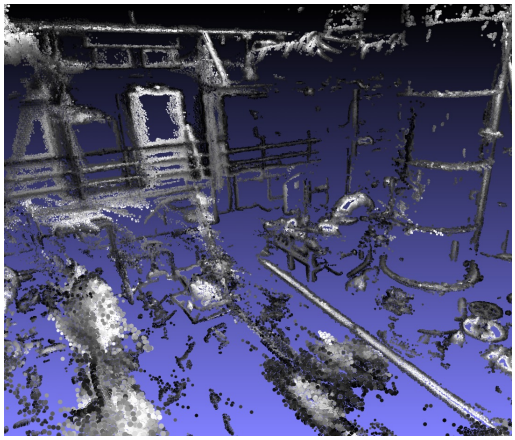
(e) Edge-aided fitting without clustering



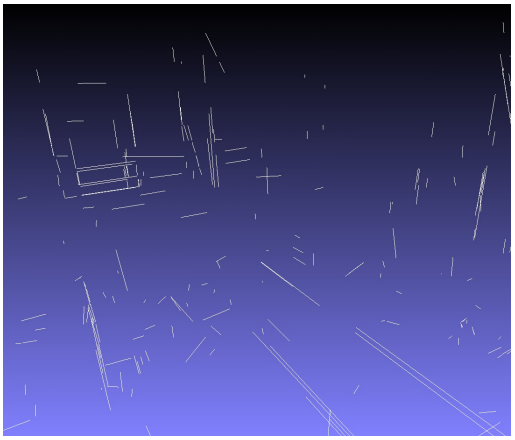
(f) Edge-aided fitting with clustering

Figure 4.2: Results of sequence EuRoC MAV Vicon Room 101 using LSD-SLAM. Notice the large number of outlier line segments on the right side of (c) produced by Line3D++. On the contrary, our method utilizes the semi-dense depth map and produce results with fewer outliers, as shown in (e) and (f). Although the semi-dense point cloud produced by LSD-SLAM is much noisier than the one from semi-dense ORB-SLAM, our method can still effectively reduces the amount of outliers and produce a line segment based model of the environment.

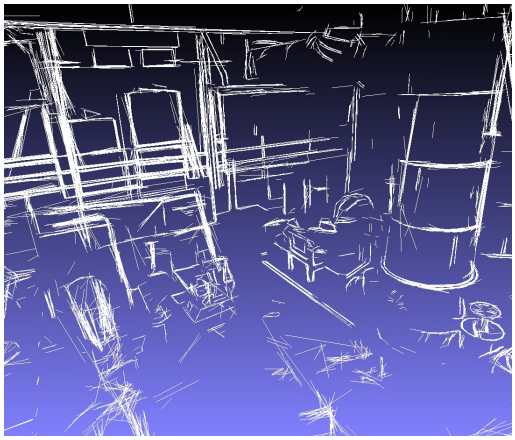
(a) Sample original image



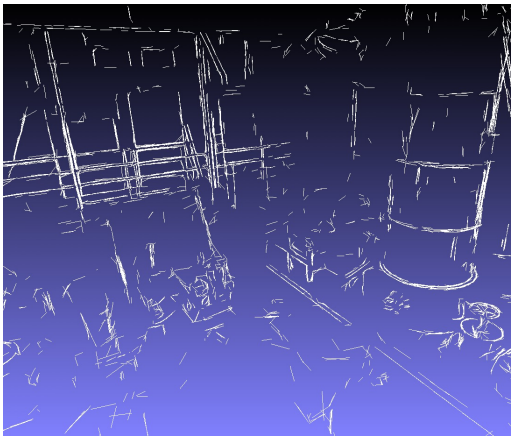
(b) Semi-dense point cloud



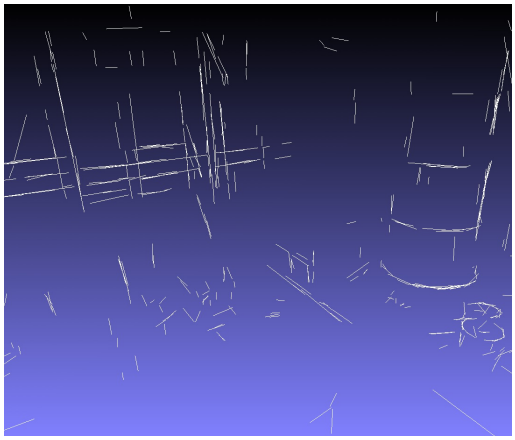
(c) Line3d++



(d) Decoupled fitting using EDLines

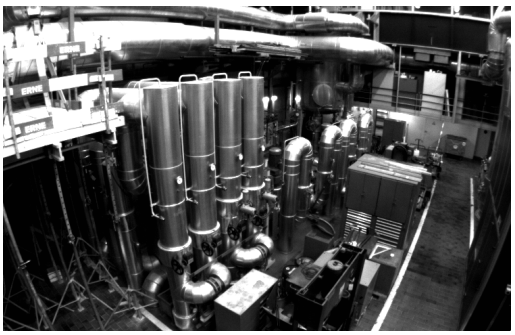


(e) Edge-aided fitting without clustering

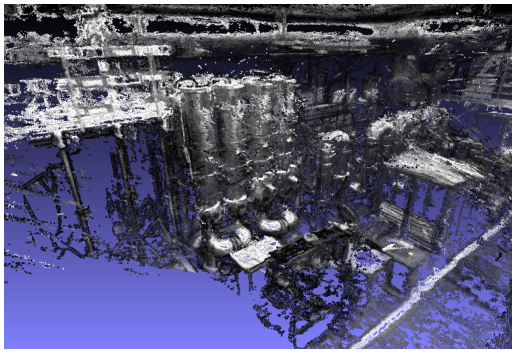


(f) Edge-aided fitting with clustering

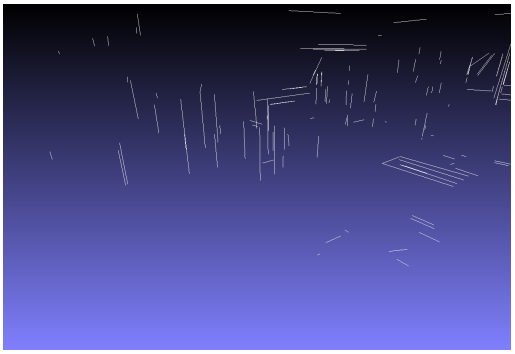
Figure 4.3: Results of sequence EuRoC MAV Machine Hall 01 using ORB-SLAM. Comparing (c) and (e), on the right side of the images, some curved structural lines are not reconstructed by Line3D++, while they are clearly visible in result of our method. Because of the incremental line segment extraction, our method is capable of capturing finer details of the scene compared to Line3D++.



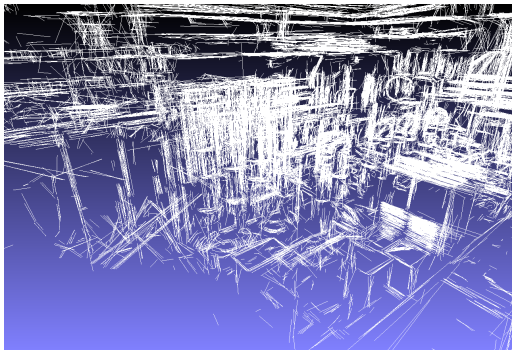
(a) Sample original image



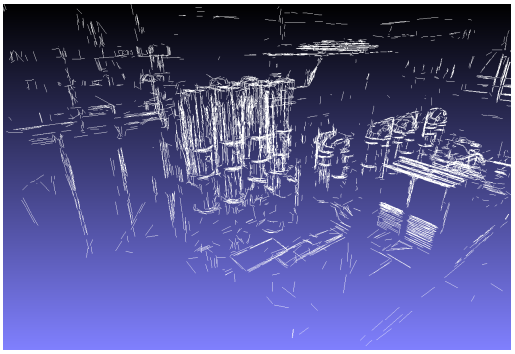
(b) Semi-dense point cloud



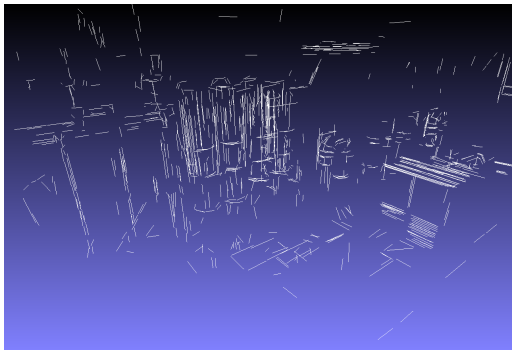
(c) Line3d++



(d) Decoupled fitting using EDLines



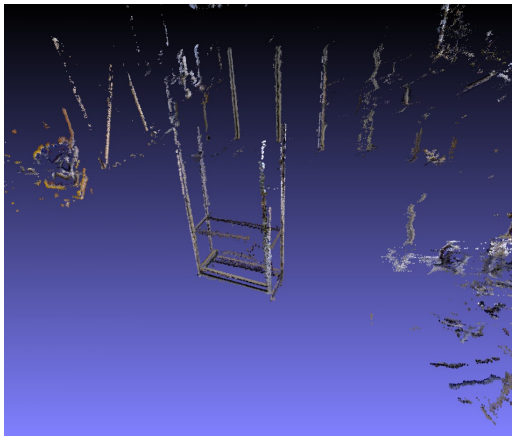
(e) Edge-aided fitting without clustering



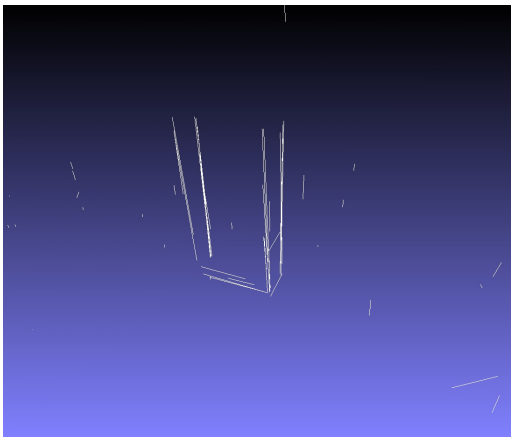
(f) Edge-aided fitting with clustering

Figure 4.4: Results of sequence EuRoC MAV Machine Hall 01 using LSD-SLAM. Compared to the result of Line3D++ in (c), our method, result shown in (e), can fit line segments to the point cloud from areas where Line3D++ failed to extract line segments due to the limitation of 2D line segment detector. Comparing (c) and (f), it is obvious that our method captures much more structural information than Line3D++.

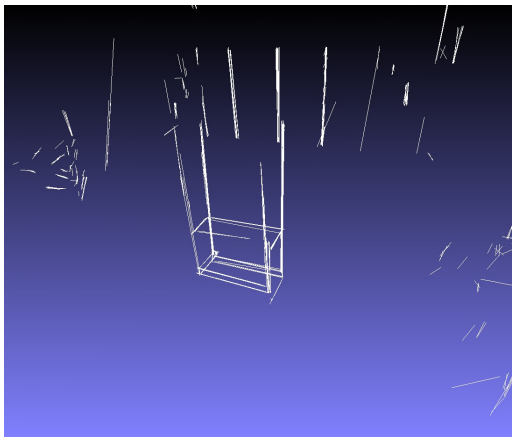
(a) Sample original image



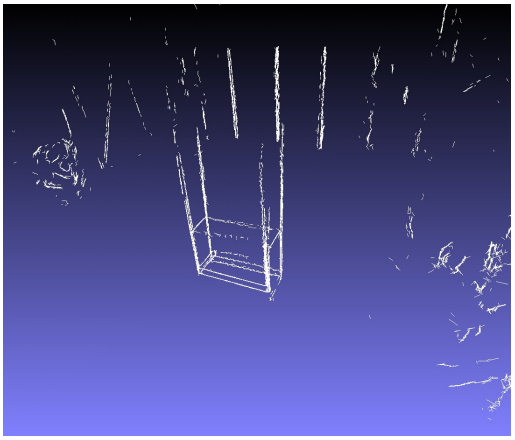
(b) Semi-dense point cloud



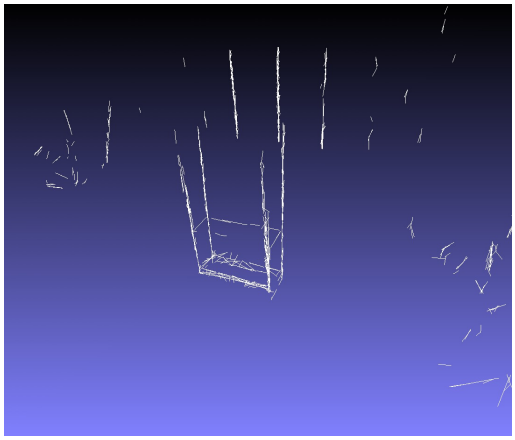
(c) Line3d++



(d) Decoupled fitting using EDLines

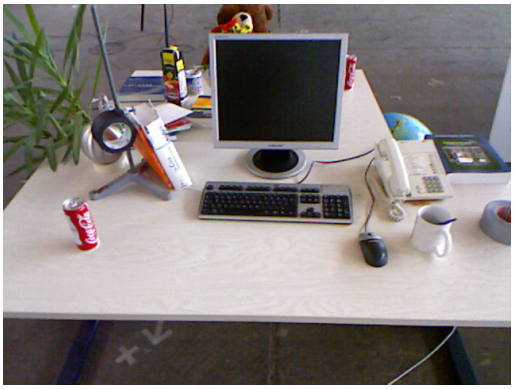


(e) Edge-aided fitting without clustering

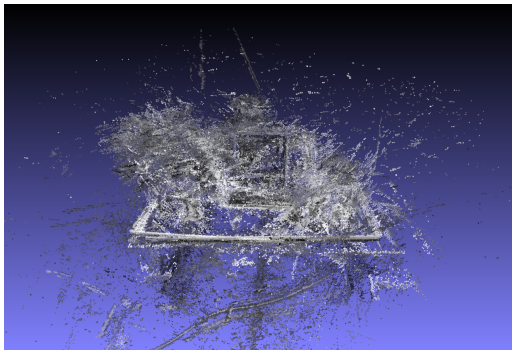


(f) Edge-aided fitting with clustering

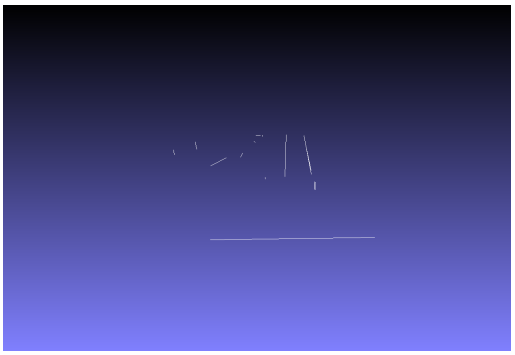
Figure 4.5: Results of sequence TUM RGBD fr3-large-cabinet using ORB-SLAM. Compared to other methods, our method, shown in (e) and (f), captures more structural information in the scene while keeping the number of elements small.



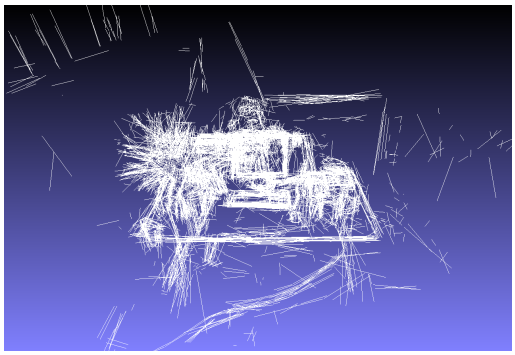
(a) Sample original image



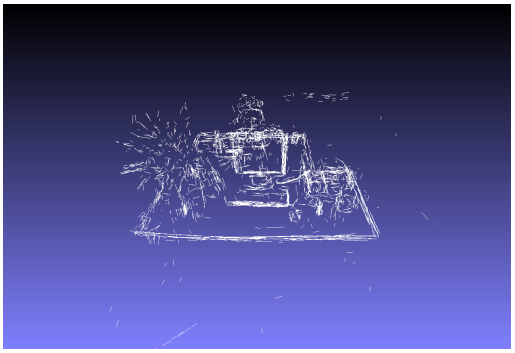
(b) Semi-dense point cloud



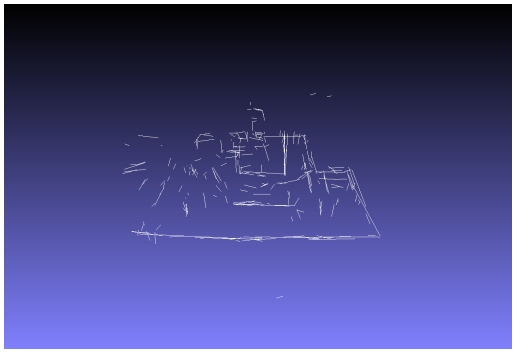
(c) Line3d++



(d) Decoupled fitting using EDLines



(e) Edge-aided fitting without clustering



(f) Edge-aided fitting with clustering

Figure 4.6: Results of sequence TUM RGBD fr2-desk using LSD-SLAM. As shown in (e) and (f), our method captures more major structural information compared to (c). Compared to (d), our proposed approach produces much fewer outliers.

To further demonstrate the capability of our method, we compare it to a decoupled 3D line segment fitting method using 2D line segment given by EDLines [1]. Given detected line segments and the depth information on some of the pixels along the line segments, we can easily estimate the 3D line segment position by performing a single 2D line fitting on the $\mathbf{p_1}$-**xz** plane. In this case, there are a fixed number of pixels on the line segment since we do not need to iteratively search along pixel chains and extend line segments. Therefore we can efficiently perform RANSAC in 2D to remove outliers before the line fitting process. With the fitted line, we compute the 3D location of the endpoints and reconstruct the 3D line segment. Note the result of this method is equivalent to directly performing a RANSAC in 3D to fit all 3D points on the line segment. However, fitting a line in 2D is faster because fewer parameters are required to represent the line and the search space is much smaller.

The results of decoupled line segment fitting are presented in the (d) subfigures of figures from Figure 4.1 to Figure 4.8. We can observe that the results of decoupled line segment fitting can contain a large amount of erroneous line segments. Outliers in this case can have large displacements from the ground truth, which is not ideal for our surface reconstruction method. Compared to the edge aided 3D line fitting which tries to utilize pixel position and depth simultaneously, the decoupled fitting essentially fits lines in the image plane and depth plane in two steps. The error from line fitting in the image plane will be propagated to the error of 3D line segment position, which result in an inaccurate reconstruction compared to our method, as shown in Figure 4.9. It is worth mentioning that the decoupled fitting tends to generate longer segments since only the pixel position is considered in the image plane line fitting process. Longer segments will make the error propagation even worse because the total error of line segments in image space might be larger. Another source of error is that EDLines may detects a long line segment which is not a continuous line in 3D space. Trying to fit a single 3D line segment onto the 2D segment in this case will result in a large error. On the other hand, in our method, if either of the two errors of line fitting grows higher

Table 4.1: Average distance of vertices to ground truth surface (ORB-SLAM)

Representation	Vicon Room 101	Vicon Room 201
Semi-dense point cloud	22.03 mm	22.49 mm
Line3D++	84.10 mm	78.58 mm
Decoupled 3D fitting	21.48 mm	23.43 mm
Edge aided w/o clustering	13.91 mm	17.45 mm
Edge aided w/ clustering	13.93 mm	17.69 mm

Table 4.2: Average distance of vertices to ground truth surface (LSD-SLAM)

Representation	Vicon Room 101	Vicon Room 201
Semi-dense point cloud	43.44 mm	45.58 mm
Line3D++	95.55 mm	270.56 mm
Decoupled 3D fitting	61.44 mm	60.44 mm
Edge aided w/o clustering	30.46 mm	40.76 mm
Edge aided w/ clustering	30.46 mm	39.07 mm

than the threshold, we stop the line fitting and start a new line fitting process. In this way, the errors accumulated from image plane and depth are bounded, therefore it prevents the line segments from being far away from the 3D points.

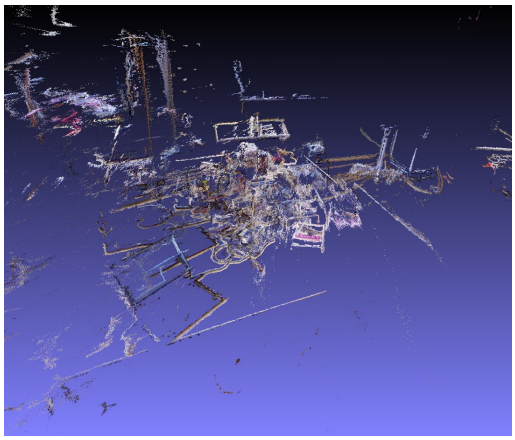
4.3 Quantitative Comparison

In this section, we present some quantitative results comparing our method with other methods.

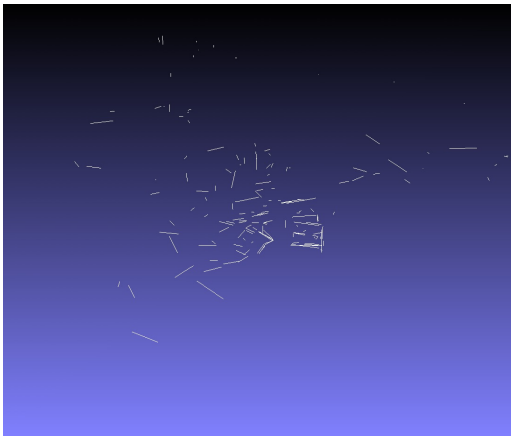
4.3.1 Distance to surface

To demonstrate the accuracy of our method, we compute the average distance of line segment endpoints to the ground truth surface in two EuRoC MAV sequences, as shown in Table 4.1 and Table 4.2. We take the provided precise 3D scanning of environment as ground truth. Since the output of monocular SLAM systems have coordinates different from the ground truth surface data, we estimate the global Euclidean transform and scale change by performing ICP to align the semi-dense point cloud to the ground truth point cloud. The

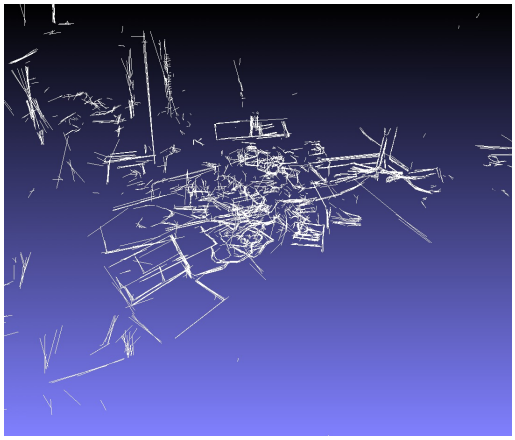
(a) Sample original image



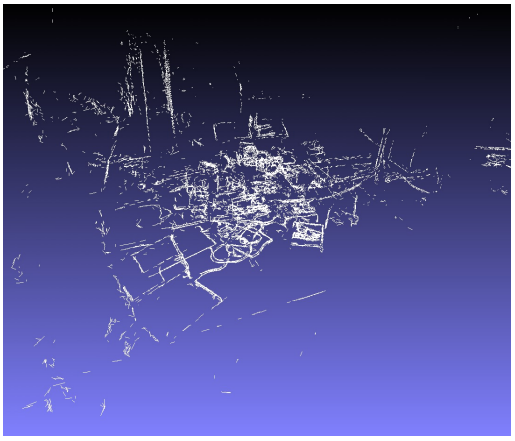
(b) Semi-dense point cloud



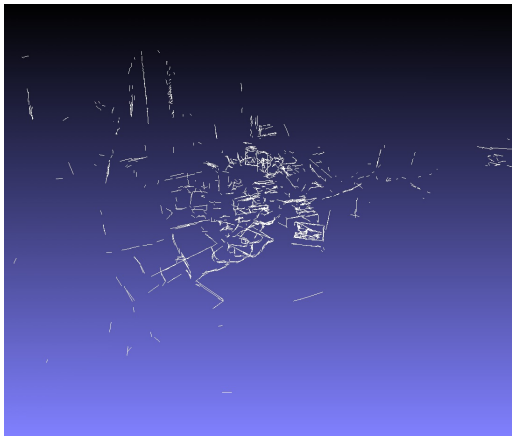
(c) Line3d++



(d) Decoupled fitting using EDLines

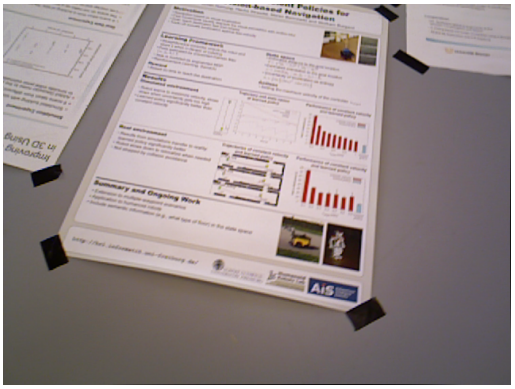


(e) Edge-aided fitting without clustering

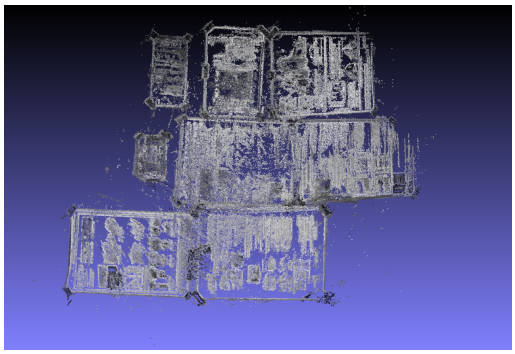


(f) Edge-aided fitting with clustering

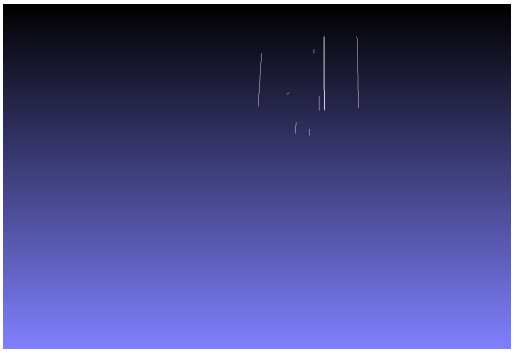
Figure 4.7: Results of sequence TUM RGBD fr1-room using ORB-SLAM.



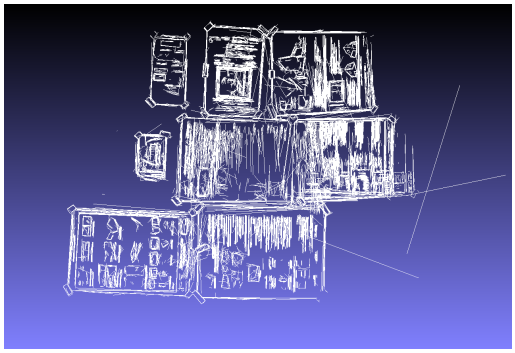
(a) Sample original image



(b) Semi-dense point cloud

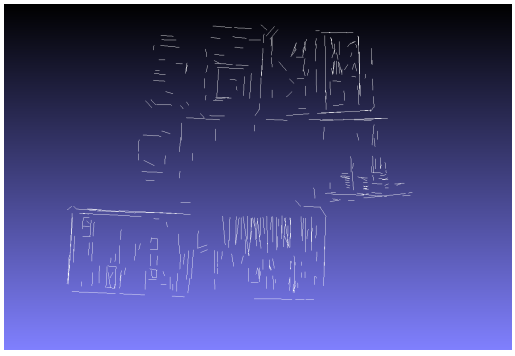


(c) Line3d++



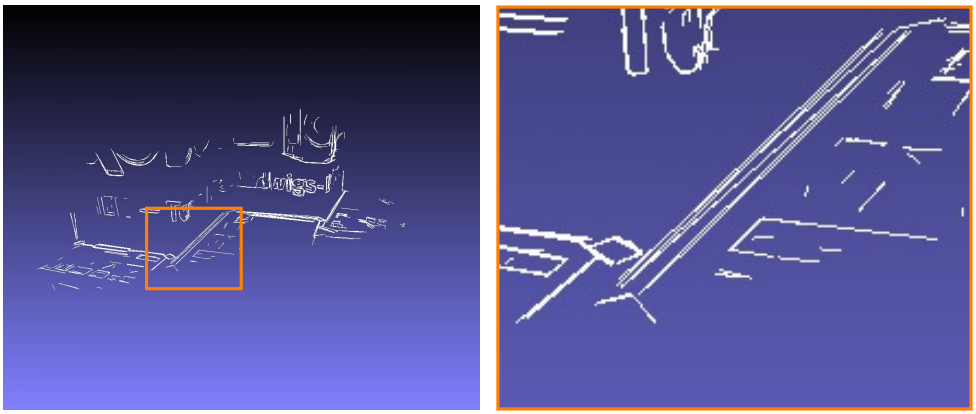
(d) Decoupled fitting using EDLines

(e) Edge-aided fitting without clustering

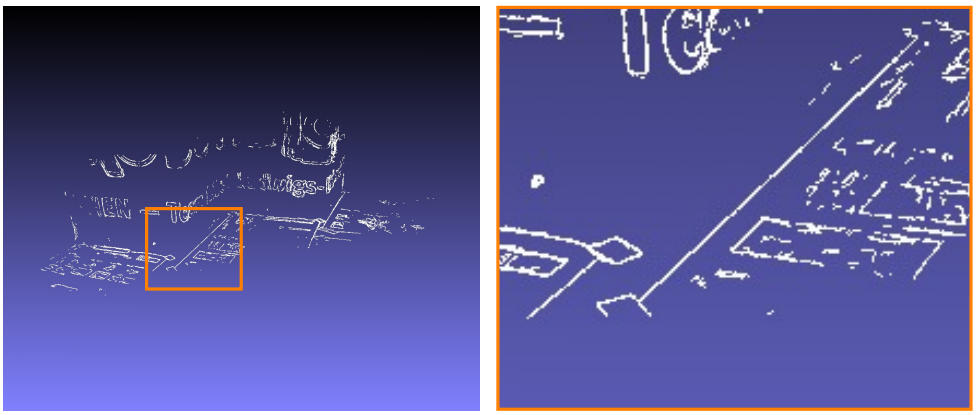


(f) Edge-aided fitting with clustering

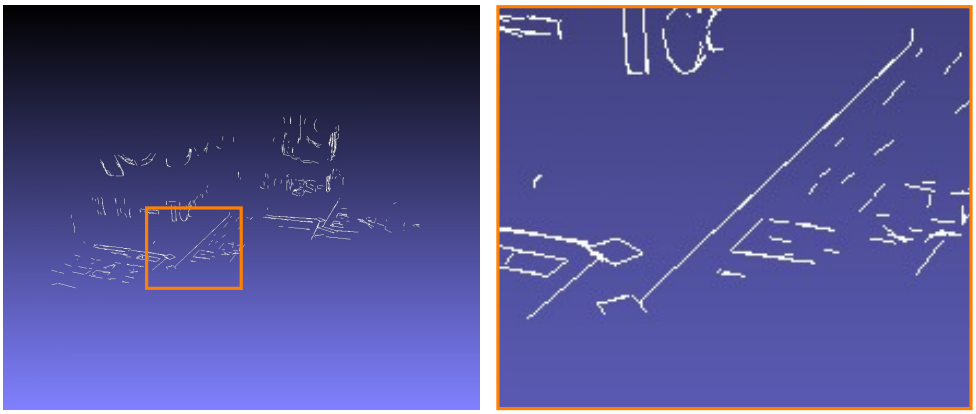
Figure 4.8: Results of sequence TUM RGBD fr3-nostructure-texture-near-withloop using LSD-SLAM.



(a) Decoupled fitting using EDLines



(b) Edge-aided fitting without clustering



(c) Edge-aided fitting with clustering

Figure 4.9: Comparison of results of sequence fr3-structure-texture-near using ORB-SLAM. Compared to our method, shown in (b) and (c), which tries to utilize pixel position and depth simultaneously, the decoupled fitting, shown in (a), essentially fits lines in the image plane and depth plane in two steps. The error from line fitting in the image plane will be propagated to the error of 3D line segment position, resulting in noisy and inconsistent 3D line positions from different keyframes.

Table 4.3: Number of vertices in sequences of EuRoC MAV dataset (ORB-SLAM)

Representation	Vicon Room 101	Machine Hall 01
Semi-dense point cloud	2361598	3252467
Line3D++	2832	2354
Decoupled 3D fitting	15994	42416
Edge aided w/o clustering	34958	41718
Edge aided w/ clustering	2396	2810

Table 4.4: Number of vertices in sequences of EuRoC MAV dataset (LSD-SLAM)

Representation	Vicon Room 101	Machine Hall 01
Semi-dense point cloud	8520281	11584044
Line3D++	1262	376
Decoupled 3D fitting	96020	165224
Edge aided w/o clustering	25784	33084
Edge aided w/ clustering	1994	3318

same Euclidean transform and scale change are applied to all the output line segment data before calculating distances, so that all the distances calculated are in the coordinates of the ground truth data. It can be seen in Table 4.1 and Table 4.2 that the result of our method fit to the surface better than other methods. Compared to the points in semi-dense point cloud, our method effectively filters out the outliers and reduces the error.

4.3.2 Compactness

For easier handling and manipulation, it is desired to have fewer 3D elements while they can still represent most of the environment. In the surface reconstruction pipeline, a smaller number of vertices will also greatly reduce the running time. As shown in Table 4.3, Table 4.4, Table 4.5 and Table 4.6, the point clouds are greatly simplified with our edge aided 3D line fitting algorithm. The results are simplified further to present a clean structure of the scene using our 3D line segment clustering process. Note that although

Table 4.5: Number of vertices in sequences of TUM RGB-D dataset (ORB-SLAM)

Representation	fr3-large-cabinet	fr1-room
Semi-dense point cloud	263637	1044752
Line3D++	124	330
Decoupled 3D fitting	1106	9966
Edge aided w/o clustering	15760	42624
Edge aided w/ clustering	1304	3334

Table 4.6: Number of vertices in sequences of TUM RGB-D dataset (LSD-SLAM)

Representation	fr3-nostructure-texture-near	fr2-desk
Semi-dense point cloud	1442942	2187987
Line3D++	18	30
Decoupled 3D fitting	22738	9966
Edge aided w/o clustering	9354	24306
Edge aided w/ clustering	746	526

Line3D++ produces the fewest number of vertices in the reconstruction, the completeness of reconstruction is generally worse than our method as shown in the figures from Figure 4.1 to Figure 4.8.

4.3.3 Running Time

Table 4.7 presents the average running time of 3D line segments fitting using ORB-SLAM on the sequences shown in Figure 4.1, Figure 4.3, Figure 4.5 and Figure 4.7. Similarly, Table 4.8 presents the average running time using LSD-SLAM on the sequences shown in Figure 4.2, Figure 4.4, Figure 4.6 and Figure 4.8. Our line segment fitting method is run-time efficient while utilizing large amount of depth information. Compared to the running time of edge aided 3D fitting, decoupled 3D fitting requires additional computation time for performing RANSAC. Because the segments are usually much longer in decoupled 3D line segments fitting, RANSAC is necessary in order to obtain a good fit for the larger pixel set on the line segments. Our 3D line segment

Table 4.7: Running time per keyframe (ORB-SLAM)

Method	Average Time (ms)
Decoupled 3D fitting	10.42
Edge aided 3D fitting	7.40

Table 4.8: Running time per keyframe (LSD-SLAM)

Method	Average Time (ms)
Decoupled 3D fitting	83.58
Edge aided 3D fitting	43.88

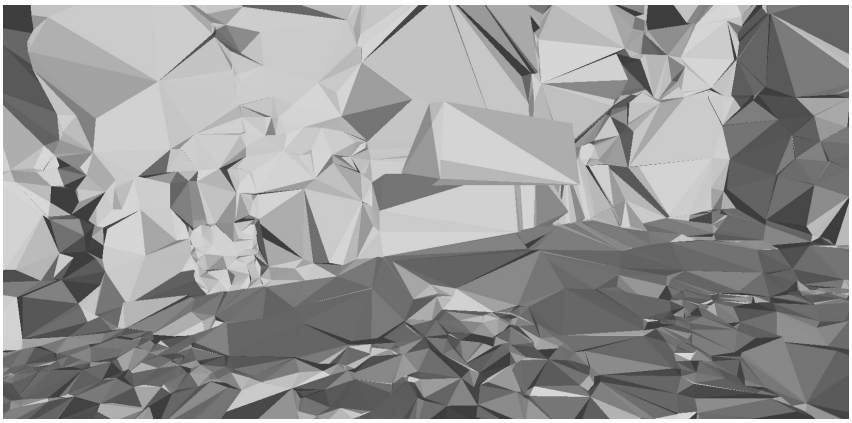
fitting algorithm is linear in the number of pixels on detected edges. Since LSD-SLAM generally produce much more pixels with depth compared to semi-dense ORB-SLAM, the methods take longer time to process each keyframe for LSD-SLAM. Although the fitting algorithm is fast enough to be real-time, our clustering process is relatively slower, which takes about 300 millisecond per keyframe on a 144 second long sequence. The complexity of clustering a single line segment is $O(C)$, where C is the number of existing clusters. Therefore, the complexity of the clustering process in a sequence can be generally considered to be $O(N^2)$, where N denotes the number of line segments.

4.4 Surface Reconstruction

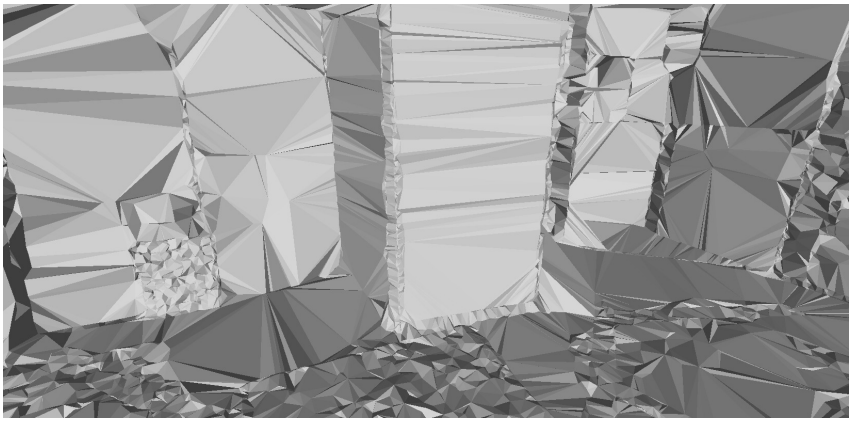
The resulting line segments of our method can be used to improve the quality of surface reconstruction. In our surface reconstruction method, we use end points of the line segments extracted from the semi-dense point cloud. The clustering process is omitted for fast performance.

We first compare the surface reconstructed using the sparse map points of ORB-SLAM with the surface reconstructed using the line segment end points from our proposed method using semi-dense ORB-SLAM. The result running on EuRoC Vicon Room 101 sequence is shown in different views in Figure 4.10, Figure 4.11 and Figure 4.12. Taking advantage of the semi-dense nature

(a) Sample original image



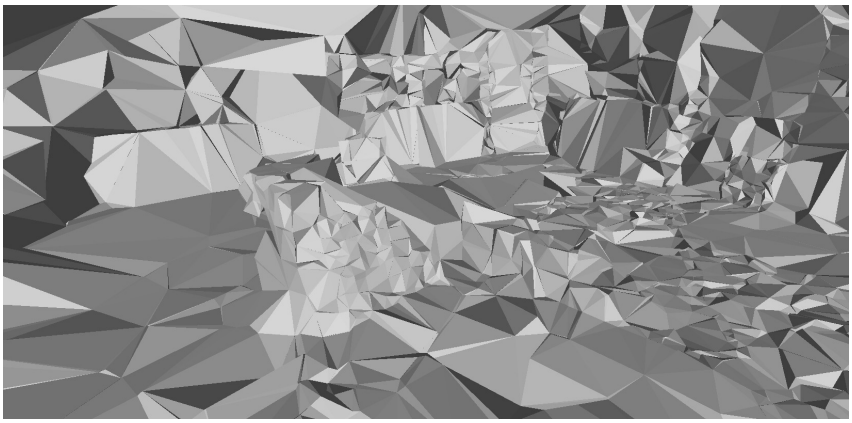
(b) Surface reconstructed with map points of ORB-SLAM



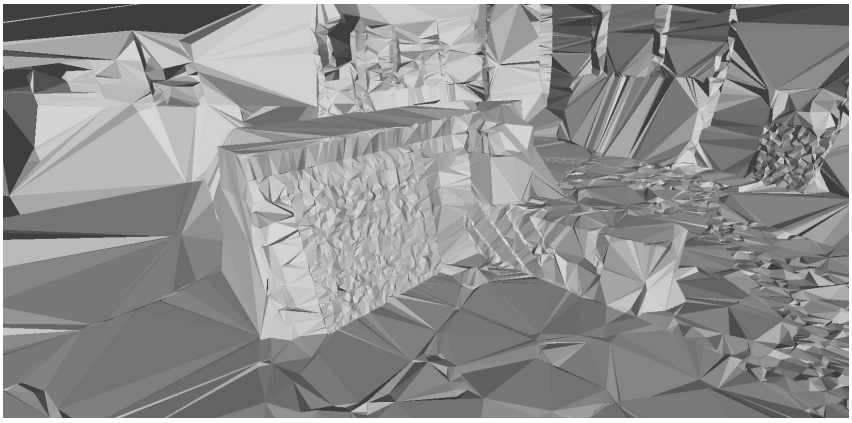
(c) Surface reconstructed with our line segments endpoints

Figure 4.10: Reconstructed surface of sequence Vicon Room 101: View 1

(a) Sample original image

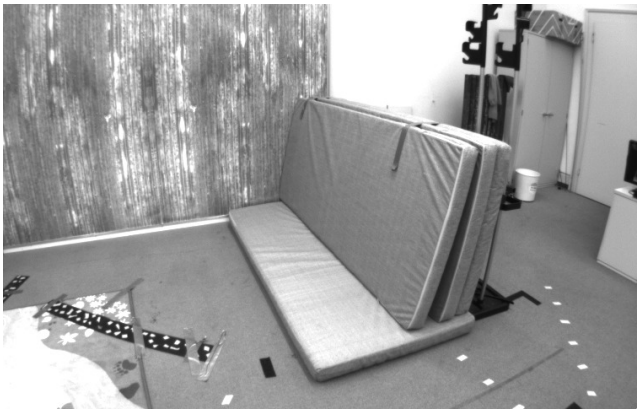


(b) Surface reconstructed with map points of ORB-SLAM

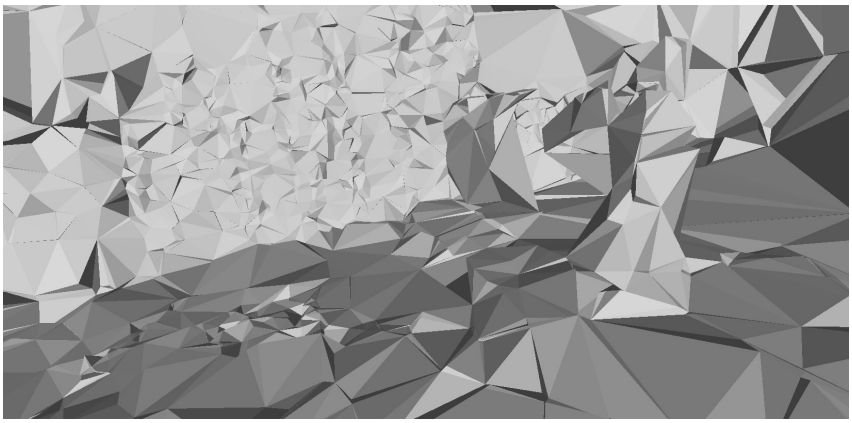


(c) Surface reconstructed with our line segments endpoints

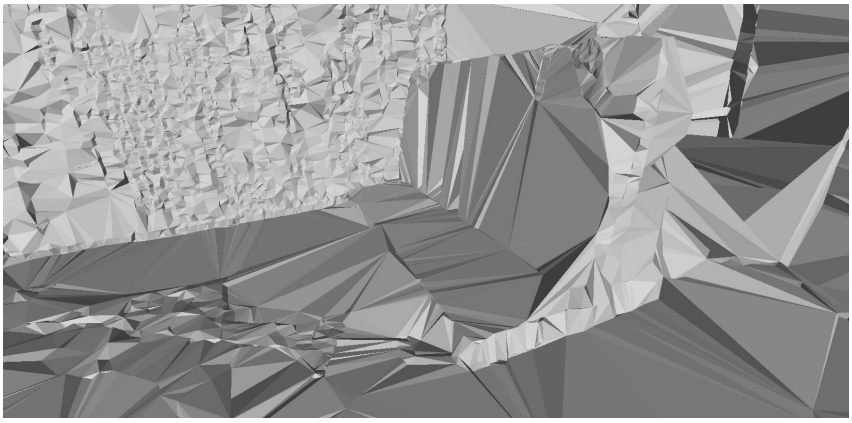
Figure 4.11: Reconstructed surface of sequence Vicon Room 101: View 2



(a) Sample original image



(b) Surface reconstructed with map points of ORB-SLAM



(c) Surface reconstructed with our line segments endpoints

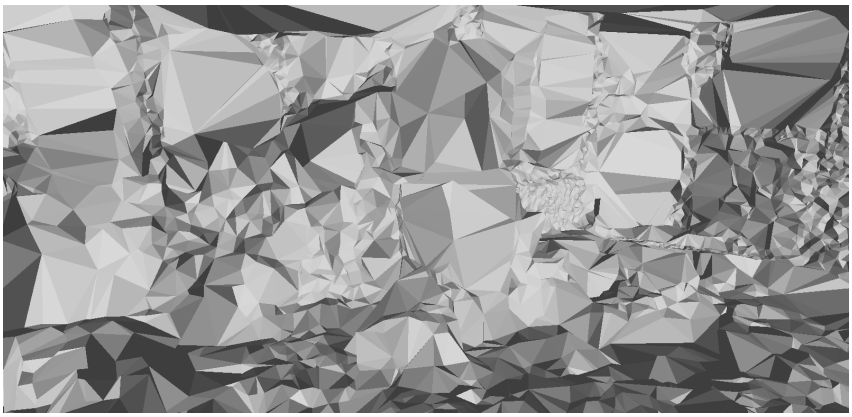
Figure 4.12: Reconstructed surface of sequence Vicon Room 101: View 3

of the point cloud, we have more points available, thus our method yields much smoother surfaces. Although the number of points we used is still larger than the number of map points, compared to the semi-dense point cloud, our method reduce the number of points to a level that the surface reconstruction method can handle. Also thanks to the structural information provided by the line segments, major structures in the room are much more obvious.

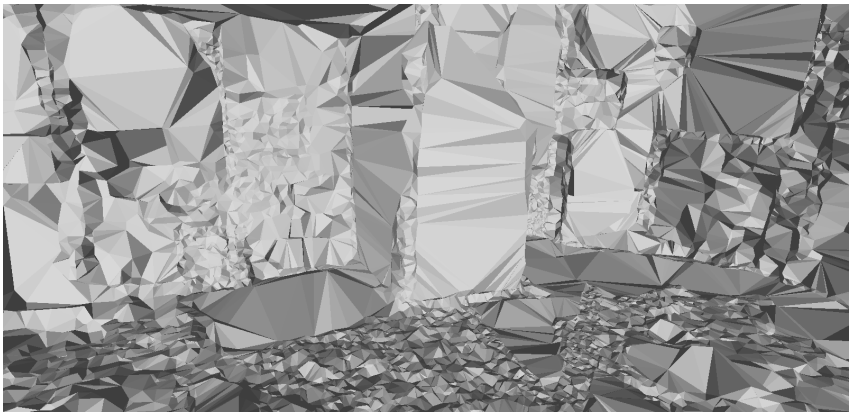
An important benefit our line segment extraction method provides is the reduced number of outliers, comparing to the map points reconstructed in ORB-SLAM. The outliers significantly affect the quality of the surface, especially outliers in visibility rays. Outlier in visibility rays can cause the surface reconstruction method to falsely carve out large regions of the space. In ORB-SLAM, in order to obtain robust performance, features are matched across many different keyframes. It is inevitable to have mismatches of feature points, resulting in outlier visibility rays which can see 3D points that they should not be able to see. On the contrary, in our method, each end point is only associated with a single visibility ray. Thus, the chance of having an outlier visibility ray is much lower for our method. Thanks to the larger number of end points compared to map points, there is still enough visibility information to carve out the tetrahedra in order to reconstruct the surface of the environment.

We also perform the comparison using LSD-SLAM. Here we compare the surface reconstructed using the line segment end points from our method and randomly sampled points from the original semi-dense point cloud. In order to be fair on the comparison, the number of points randomly sampled from each keyframe is the same with the number of line segment end points from the same keyframe. In this way, we use the exact same amount of points in two approaches. The results are shown in Figure 4.13, Figure 4.14 and Figure 4.15. It is clear to see that the surface reconstructed using our line segment end points presents more major structures in the scene compared to using random points. As seen previously, the raw semi-dense point cloud produced by LSD-SLAM is quite noisy. Since the points from our method contain less outliers and more structural information, it is easier to observe the major structures.

(a) Sample original image

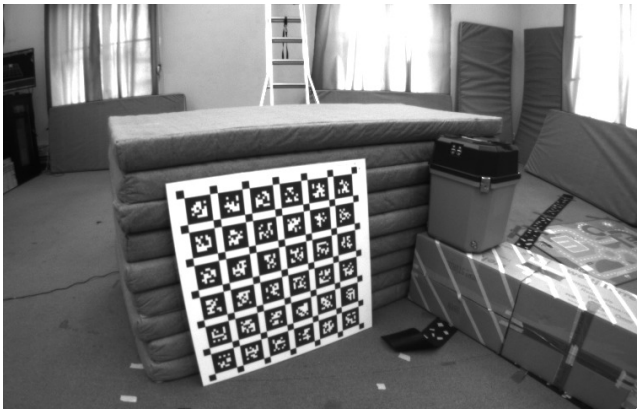


(b) Surface reconstructed with random points of LSD-SLAM

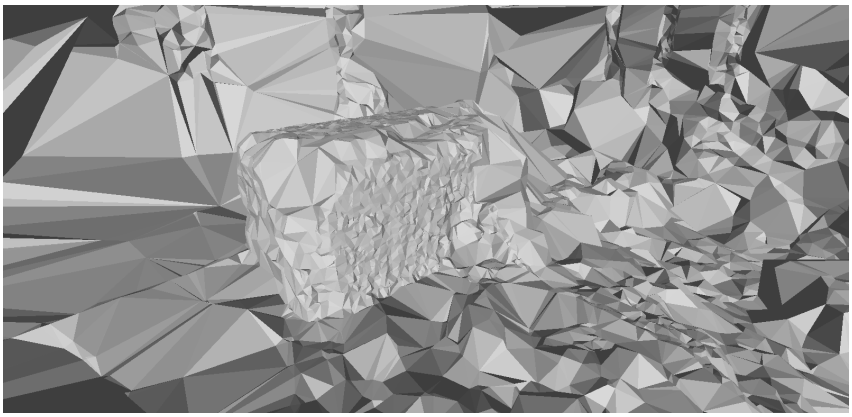


(c) Surface reconstructed with our line segments endpoints

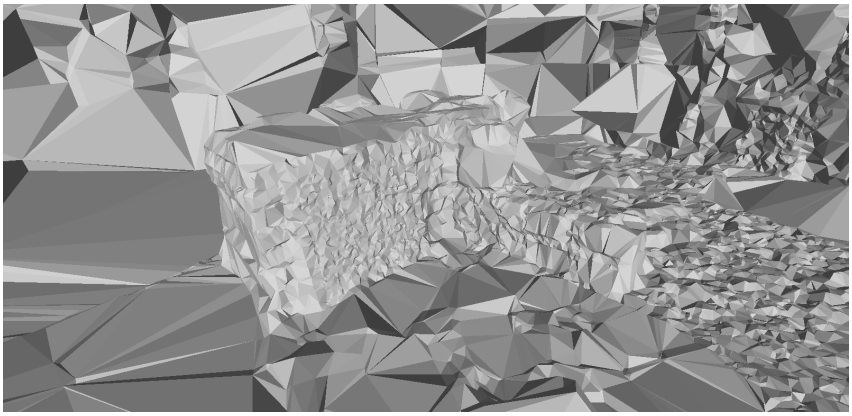
Figure 4.13: Reconstructed surface of sequence Vicon Room 101: View 1



(a) Sample original image

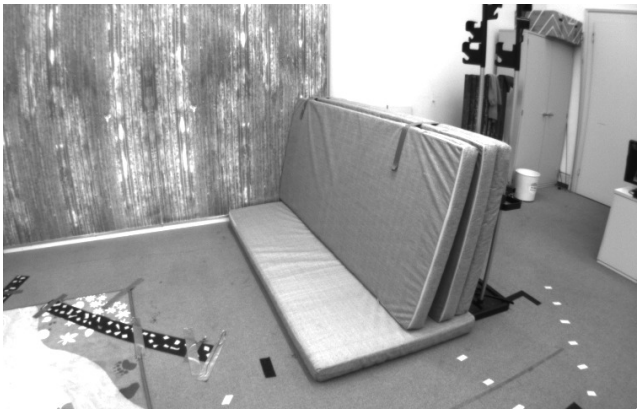


(b) Surface reconstructed with random points of LSD-SLAM

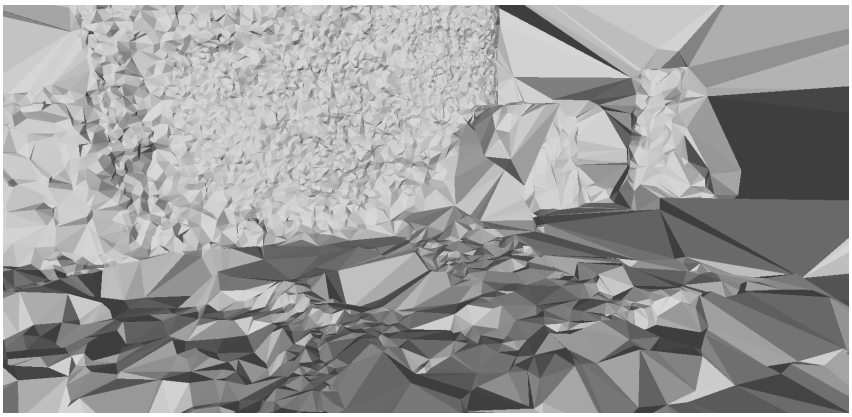


(c) Surface reconstructed with our line segments endpoints

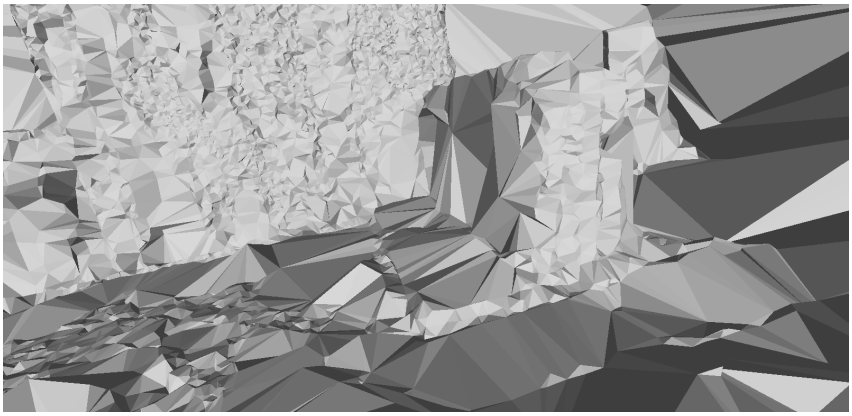
Figure 4.14: Reconstructed surface of sequence Vicon Room 101: View 2



(a) Sample original image



(b) Surface reconstructed with random points of LSD-SLAM



(c) Surface reconstructed with our line segments endpoints

Figure 4.15: Reconstructed surface of sequence Vicon Room 101: View 3

Chapter 5

Conclusion

In this thesis, we present an incremental 3D line segment based method that uses underlying structural information to simplify the semi-dense point cloud output by keyframe-base SLAM system. The main contribution lies in the novel edge aided 3D line segment extraction algorithm which solely relies on the image and the semi-dense depth map of individual keyframes. Our method is fully incremental. It tries to minimize the line fitting error on both image plane and depth plane simultaneously as the line segment grows. By incrementally clustering the line segments detected on each keyframe, we can obtain a compact and complete 3D line segment reconstruction for the scene. Compared to using the line segments produced by 2D image detectors and minimizing the line fitting error on the depth plane afterwards, our method achieves better accuracy in terms of the location of the reconstructed vertices. We show that the result of our method can be used in incremental surface reconstruction to improve the quality of 3D surfaces. By greatly simplifying the semi-dense point cloud while keeping major structures, our method enables real-time surface reconstruction with semi-dense SLAM systems.

The major purpose of our method is reconstructing surface with semi-dense SLAM. The essential idea behind our method is that when reconstructing surfaces using point clouds, the accuracy of the surface has a decreasing increase as the density of point increase. In other words, a subset of the points in a point cloud may be sufficient to reconstruct an accurate enough approximation of the surface. By selecting "critical points" that contributes to the accuracy

of surface the most, it is possible to greatly reduce the computation demand for reconstructing the surface. In a way, our method is similar to selecting the points located on line segments out of the point cloud, based on the simple heuristic that those points are the critical points. We show with experiments that the surface can be approximately reconstructed using points on line segment in the scene. However, obviously, the simple heuristic is not perfect. We plan to explore other heuristics. Data-driven approaches such as machine learning are also promising directions.

References

[1] C. Akinlar and C. Topal, "Edlines: A real-time line segment detector with a false detection control," *Pattern Recognition Letters*, vol. 32, no. 13, pp. 1633–1642, 2011. 14, 47

[2] N. Amenta and M. W. Bern, "Surface reconstruction by voronoi filtering," *Discrete & Computational Geometry*, vol. 22, no. 4, pp. 481–504, 1999. 19

[3] N. Ayache and B. Faverjon, "Efficient registration of stereo images by matching graph descriptions of edge segments," *International Journal of Computer Vision*, vol. 1, no. 2, pp. 107–131, 1987. 13

[4] A. Bartoli and P. F. Sturm, "Structure-from-motion using lines: Representation, triangulation, and bundle adjustment," *Computer Vision and Image Understanding*, vol. 100, no. 3, pp. 416–441, 2005. 2, 14, 15

[5] H. Bay, T. Tuytelaars, and L. J. V. Gool, "SURF: speeded up robust features," in *Computer Vision - ECCV 2006, 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006, Proceedings, Part I*, 2006, pp. 404–417. 14

[6] M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva, "A survey of surface reconstruction from point clouds," *Comput. Graph. Forum*, vol. 36, no. 1, pp. 301–329, 2017. 18

[7] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *I. J. Robotics Res.*, vol. 35, no. 10, pp. 1157–1163, 2016. 39

[8] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. D. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Trans. Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016. 4

[9] A. Chauve, P. Labatut, and J. Pons, "Robust piecewise-planar 3d reconstruction and completion from large-scale unstructured point data," in *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*, 2010, pp. 1261–1268. 2

[10] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1996, New Orleans, LA, USA, August 4-9, 1996*, 1996, pp. 303–312. 22

[11] A. J. Davison, I. D. Reid, N. Molton, and O. Stasse, "Monoslam: Real-time single camera SLAM," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1052–1067, 2007. 5

[12] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, no. 1, pp. 11–15, 1972. 14

[13] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *CoRR*, vol. abs/1607.02565, 2016. 8

[14] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: large-scale direct monocular SLAM," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part II*, 2014, pp. 834–849. 11, 12, 26, 27, 39

[15] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981. 16

[16] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: fast semi-direct monocular visual odometry," in *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, 2014, pp. 15–22. 8

[17] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski, "Towards internet-scale multi-view stereo," in *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*, 2010, pp. 1434–1441. 15

[18] R. G. von Gioi, J. Jakubowicz, J. Morel, and G. Randall, "LSD: A fast line segment detector with a false detection control," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 4, pp. 722–732, 2010. 14, 16, 24

[19] G. H. Golub and C. F. van Loan, "An analysis of the total least squares problem," *SIAM Journal on Numerical Analysis*, vol. 17, no. 6, pp. 883–893, 1980. 29

[20] M. Hofer, M. Donoser, and H. Bischof, "Semi-global 3d line modeling for incremental structure-from-motion," in *British Machine Vision Conference, BMVC 2014, Nottingham, UK, September 1-5, 2014*, 2014. 23, 24

[21] M. Hofer, M. Maurer, and H. Bischof, "Improving sparse 3d models for man-made environments using line-based 3d reconstruction," in *2nd International Conference on 3D Vision, 3DV 2014, Tokyo, Japan, December 8-11, 2014, Volume 1*, 2014, pp. 535–542. 23

[22] ——, "Efficient 3d scene abstraction using line segments," *Computer Vision and Image Understanding*, vol. 157, pp. 167–178, 2017. 2, 14, 15, 40

[23] C. Hoppe, M. Klopschitz, M. Donoser, and H. Bischof, "Incremental surface extraction from sparse structure-from-motion point clouds," in *British Machine Vision Conference, BMVC 2013, Bristol, UK, September 9-13, 2013*, 2013. 1, 33

[24] ——, "Incremental surface extraction from sparse structure-from-motion point clouds," in *British Machine Vision Conference, BMVC 2013, Bristol, UK, September 9-13, 2013*, 2013. 19, 21

[25] S. Ikehata, I. Boyadzhiev, Q. Shan, and Y. Furukawa, "Panoramic structure from motion via geometric relationship detection," *CoRR*, vol. abs/1612.01256, 2016. 2

[26] S. Ikehata, H. Yang, and Y. Furukawa, "Structured indoor modeling," in *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, 2015, pp. 1323–1331. 2

[27] A. Jain, C. Kurz, T. Thormählen, and H. Seidel, "Exploiting global connectivity constraints for reconstruction of 3d line segments from images," in *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*, 2010, pp. 1586–1593. 2, 14

[28] P. Jenke, B. Krückeberg, and W. Straßer, "Surface reconstruction from fitted shape primitives," in *Proceedings of the Vision, Modeling, and Visualization Conference 2008, VMV 2008, Konstanz, Germany, October 8-10, 2008*, 2008, pp. 31–40. 2

[29] A. Khatamian and H. R. Arabnia, "Survey on 3d surface reconstruction," *JIPS*, vol. 12, no. 3, pp. 338–357, 2016. 18

[30] G. Klein and D. W. Murray, "Parallel tracking and mapping for small AR workspaces," in *Sixth IEEE/ACM International Symposium on Mixed and Augmented Reality, ISMAR 2007, 13-16 November 2007, Nara, Japan*, 2007, pp. 225–234. 5, 6

[31] P. Labatut, J. Pons, and R. Keriven, "Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts," in *IEEE 11th International Conference on Computer Vision, ICCV 2007, Rio de Janeiro, Brazil, October 14-20, 2007*, 2007, pp. 1–8. 19, 21, 24

[32] M. Lhuillier and S. Yu, "Manifold surface reconstruction of an environment from sparse structure-from-motion data," *Computer Vision and Image Understanding*, vol. 117, no. 11, pp. 1628–1644, 2013. 19

[33] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987, Anaheim, California, USA, July 27-31, 1987*, 1987, pp. 163–169. 22

[34] D. I. Lovi, "Incremental free-space carving for real-time 3d reconstruction," Master's thesis, University of Alberta, 2011. 1, 19, 20, 22, 33, 35

[35] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004. 14

[36] B. Micusík and H. Wildenauer, "Structure from motion with line segments under relaxed endpoint constraints," *International Journal of Computer Vision*, vol. 124, no. 1, pp. 65–79, 2017. 2, 14, 15

[37] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: A versatile and accurate monocular SLAM system," *IEEE Trans. Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015. 6–8, 26, 39

[38] R. Mur-Artal and J. D. Tardós, "Probabilistic semi-dense mapping from highly accurate feature-based monocular SLAM," in *Robotics: Science and Systems XI, Sapienza University of Rome, Rome, Italy, July 13-17, 2015*, 2015. 12–14, 27, 28, 39

[39] ——, "ORB-SLAM2: an open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017. 39

[40] Y. Nakayama, H. Saito, M. Shimizu, and N. Yamaguchi, "3d line segment based model generation by RGB-D camera for camera pose estimation," in *Computer Vision - ACCV 2014 Workshops - Singapore, Singapore, November 1-2, 2014, Revised Selected Papers, Part III*, 2014, pp. 459–472. 2, 17, 18

[41] R. A. Newcombe and A. J. Davison, "Live dense reconstruction with a single moving camera," in *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*, 2010, pp. 1498–1505. 10

[42] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. W. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *10th IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2011, Basel, Switzerland, October 26-29, 2011*, 2011, pp. 127–136. 22, 23

[43] R. A. Newcombe, S. Lovegrove, and A. J. Davison, "DTAM: dense tracking and mapping in real-time," in *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, 2011, pp. 2320–2327. 9

[44] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3d reconstruction at scale using voxel hashing," *ACM Trans. Graph.*, vol. 32, no. 6, 169:1–169:11, 2013. 23

[45] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. I. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board MAV planning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada, September 24-28, 2017*, 2017, pp. 1366–1373. 23

[46] P. Ondruska, P. Kohli, and S. Izadi, "Mobilefusion: Real-time volumetric surface reconstruction and dense tracking on mobile phones," *IEEE Trans. Vis. Comput. Graph.*, vol. 21, no. 11, pp. 1251–1258, 2015. 23

[47] V. Pradeep, C. Rhemann, S. Izadi, C. Zach, M. Bleyer, and S. Bathiche, "Monofusion: Real-time 3d reconstruction of small scenes with a single web camera," in *IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2013, Adelaide, Australia, October 1-4, 2013*, 2013, pp. 83–88. 10, 23

[48] A. Pumarola, A. Vakhitov, A. Agudo, A. Sanfeliu, and F. Moreno-Noguer, "PL-SLAM: real-time monocular visual SLAM with points and lines," in *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, 2017, pp. 4503–4508. 2, 16

[49] K. S. Roberts, "A new representation for a line," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 1988, 5-9 June, 1988, Ann Arbor, Michigan, USA.*, 1988, pp. 635–640. 16

[50] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "ORB: an efficient alternative to SIFT or SURF," in *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, 2011, pp. 2564–2571. 6

[51] R. F. Salas-Moreno, B. Glocker, P. H. J. Kelly, and A. J. Davison, "Dense planar SLAM," in *IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2014, Munich, Germany, September 10-12, 2014*, 2014, pp. 157–164. 2

[52] C. Schmid and A. Zisserman, "Automatic line matching across views," in *1997 Conference on Computer Vision and Pattern Recognition (CVPR '97), June 17-19, 1997, San Juan, Puerto Rico*, 1997, pp. 666–671. 13

[53] K. Snow and B. Schaffrin, "Line fitting in euclidean 3d space," *Studia Geophysica et Geodaetica*, vol. 60, no. 2, pp. 210–227, 2016. 2, 16, 17

[54] F. Steinbrücker, J. Sturm, and D. Cremers, "Volumetric 3d mapping in real-time on a CPU," in *2014 IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China, May 31 - June 7, 2014*, 2014, pp. 2021–2028. 23

[55] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, 2012, pp. 573–580. 39

[56] T. Sugiura, A. Torii, and M. Okutomi, "3d surface reconstruction from point-and-line cloud," in *2015 International Conference on 3D Vision, 3DV 2015, Lyon, France, October 19-22, 2015*, 2015, pp. 264–272. 24, 25, 33

[57] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual SLAM algorithms: A survey from 2010 to 2016," *IPSJ Trans. Computer Vision and Applications*, vol. 9, p. 16, 2017. 4, 5

[58] E. Tola, V. Lepetit, and P. Fua, "DAISY: an efficient dense descriptor applied to wide-baseline stereo," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 5, pp. 815–830, 2010. 24

[59] C. Topal and C. Akinlar, "Edge drawing: A combined real-time edge and segment detector," *J. Visual Communication and Image Representation*, vol. 23, no. 6, pp. 862–872, 2012. 2, 28

[60] Z. Wang, F. Wu, and Z. Hu, "MSLD: A robust descriptor for line matching," *Pattern Recognition*, vol. 42, no. 5, pp. 941–953, 2009. 14

[61] D. Woo, S. S. Han, Y. Jung, and K. Lee, "Generation of 3d building model using 3d line detection scheme based on line fitting of elevation data," in *Advances in Multimedia Information Processing - PCM 2005, 6th Pacific-Rim Conference on Multimedia, Jeju Island, Korea, November 13-16, 2005, Proceedings, Part I*, 2005, pp. 559–569. 17

[62] J. Xiao and Y. Furukawa, "Reconstructing the world's museums," *International Journal of Computer Vision*, vol. 110, no. 3, pp. 243–258, 2014. 2

[63] S. Yang, Y. Song, M. Kaess, and S. Scherer, "Pop-up SLAM: semantic monocular plane SLAM for low-texture environments," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2016, Daejeon, South Korea, October 9-14, 2016*, 2016, pp. 1222–1229. 2

[64] L. Zhang and R. Koch, "An efficient and robust line segment matching approach based on LBD descriptor and pairwise geometric consistency," *J. Visual Communication and Image Representation*, vol. 24, no. 7, pp. 794–805, 2013. 14, 16

[65] ——, "Structure and motion from line correspondences: Representation, projection, initialization and sparse bundle adjustment," *J. Visual Communication and Image Representation*, vol. 25, no. 5, pp. 904–915, 2014. 2, 14

[66] L. Zhang, C. Xu, K. Lee, and R. Koch, "Robust and efficient pose estimation from line correspondences," in *Computer Vision - ACCV 2012 - 11th Asian Conference on Computer Vision, Daejeon, Korea, November 5-9, 2012, Revised Selected Papers, Part III*, 2012, pp. 217–230. 14