Dipl.-Ing. Manuel Hofer

Building with Lines: Efficient 3D Scene Abstraction for the Built Environment

DOCTORAL THESIS

to achieve the university degree of
Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Prof. Dr. Horst Bischof
Institute for Computer Graphics and Vision

Prof. Dr.-Ing. Reinhard Koch
Institute of Computer Science, University of Kiel

Graz, Austria, July 2016

Abstract

Extracting 3D information from a moving camera is traditionally based on interest point detection and matching. This is especially challenging in urban indoor- and outdoor environments, where the number of distinctive interest points is naturally limited. While common Structure-from-Motion (SfM) approaches usually manage to obtain the correct camera poses, the number of accurate 3D points is very small due to the low number of matchable features. Subsequent Multi-view Stereo approaches may help to overcome this problem, but suffer from a high computational complexity.

We propose a novel approach for the task of 3D scene abstraction from oriented image sequences (e.g. SfM results), which uses straight line segments as underlying image features. We use purely geometric constraints to match 2D line segments from different images, and omit any kind of appearance constraints (such as color or image gradients) altogether. This enables us to find line correspondences for highly non-planar scenes, and under changing illumination conditions, for which traditional descriptor- or color-based line matching methods would often fail. This results in a one-to-many matching procedure, where each individual 2D segment can have multiple spatially disparate correspondences in each neighboring image. We then triangulate all potential correspondences in 3D space, and evaluate their likelihood of being correct by employing a scoring procedure based on the mutual support between them. We now assign a 3D estimate to each 2D segment individually, which simply corresponds to the triangulation of its matching hypothesis with the highest score. As a final reconstruction step, we cluster corresponding 2D segments based on the similarity between their 3D location estimates, using an efficient graph-clustering formulation. The final line-based 3D model and its 2D residuals can further be used to refine the input SfM result, by using combined point- and line-based bundle adjustment at the end. We show that our method generates accurate 3D models with low computational costs, which makes it especially useful for large-scale urban indoor- and outdoor datasets.

Since the quality and completeness of image-based 3D reconstructions heavily depends on the available image set, online SfM methods have been introduced to guide the user during the image acquisition process, by directly computing a sparse 3D model while the images are being recorded. As an application for our method, we show how our line-based 3D reconstruction principles can be easily reformulated to allow an incremental reconstruction, which enables its integration into any online SfM pipeline. We show how additional 3D lines in combination with the sparse point cloud from the SfM significantly improve the visual impression of the emerging 3D model, which in turn helps the user to better judge whether the images already taken are sufficient for 3D reconstruction or not.

Kurzfassung

Das Extrahieren von 3D Information aus Bildern einer sich bewegenden Kamera wird traditionell durch so genannte Feature Punkte gelöst, die zuerst in den einzelnen Bildern detektiert und anschließend über die Bildgrenzen hinaus abgeglichen werden. Dieser Vorgang ist speziell in urbanen Umgebungen (innerhalb, sowie auch außerhalb von Gebäuden) mit immensen Schwierigkeiten verbunden, da die Anzahl der visuell gut unterscheidbaren Feature Punkte hier naturgemäß eher beschränkt ist. Während gängige Structure-from-Motion (SfM) Ansätze im Regelfall zumindest die korrekten Kameraposen zu den verwendeten Bildern finden können, ist die Anzahl der rekonstruierten 3D Punkte aufgrund der wenigen korrekten Feature Punkt Korrespondenzen im Allgemeinen nur sehr klein. Nachfolgende reine 3D Rekonstruktions (*Multi-view Stereo*) Algorithmen können dieses Problem zwar häufig beseitigen, sind dafür aber mit einem hohen Rechenaufwand verbunden.

In dieser Dissertation stellen wir eine neuartige Methode zur dreidimensionalen Abstraktion einer Szene aus einer orientierten Bildsequenz (z.B. eines SfM Resultates) vor, wobei wir gerade Liniensegmente als zugrundeliegende Bild Merkmale verwenden. Wir verwenden rein geometrische Eigenschaften um potentiell korrespondierende Liniensegmente zwischen Bildern abzugleichen und verzichten gänzlich auf visuelle Merkmale, wie z.B. Farbe oder Bildgradienten. Dies ermöglicht es uns Linien Korrespondenzen auch in höchst nicht-planaren Szenen und unter sich verändernder Beleuchtung zu finden, was mit traditionellen auf Deskriptoren- oder auf Farbe basierenden Korrespondenzfindungsmethoden kaum möglich wäre. Das Resultat sind eine Vielzahl an potentiellen Korrespondenzen für jedes einzelne 2D Segment, in jedem benachbarten Bild. Wir triangulieren nun alle Korrespondenzen in den 3D Raum und evaluieren deren Grad an Korrektheit durch eine Beurteilungsprozedur, welche auf der gegenseitigen Unterstützung dieser Korrespondenzen in 3D beruht. Wir weisen nun jedem 2D Liniensegment individuell eine geschätzte 3D Position zu, welche aus der Korrespondenzhypothese mit der höchsten Korrektheitswahrscheinlichkeit abgeleitet ist. Als letzten Schritt verwenden wir Ähnlichkeitskriterien unter potentiell korrespondierenden 2D Segmenten, basierend auf der räumlichen Nähe ihrer geschätzten 3D Positionen, um zusammengehörige 2D Segmente aus allen Bildern zu gruppieren. Das finale 3D Linienmodel kann im Anschluss auch dazu verwendet werden die berechneten Kameraposen aus dem SfM Resultat weiter zu verfeinern, indem man eine kombinierte Optimierung über alle rekonstruierten Punkte und Linien durchführt. Wir demonstrieren dass unsere Methode in der Lage ist mit geringem Rechenaufwand akkurate 3D Modelle zu erzeugen, was sich speziell für große Datensätze urbaner Umgebungen als höchst nützlich erweist.

Da die Qualität und die Vollständigkeit einer bildgestützten 3D Rekonstruktion sehr stark von den zugrundeliegenden Bildern abhängen, besteht die Möglichkeit eine online SfM Methode zu verwenden, die den Benutzer während der Bildaufnahme unterstützt, indem die aufgenommenen Bilder direkt zum Berechnen einer dünnen Punktwolke herangezogen werden. Als Anwendungsbeispiel für unsere Methodik zeigen wir wie unsere Rekonstruktionsprinzipien mit wenig Aufwand so umformuliert werden können, dass eine inkrementelle Rekonstruktion möglich ist, was in weiterer Folge die Integration in jede beliebige online SfM Applikation ermöglicht. Wir streichen hervor wie sehr die Verfügbarkeit von 3D Linien zusätzlich zur dünnen Punktwolke der SfM Applikation den entstehenden visuellen Eindruck des generierten 3D Models verbessert, was natürlich auch dem Benutzer zugute kommt, da dadurch eine bessere Abschätzung darüber möglich ist, ob die bereits aufgenommen Bilder zum Erstellen eines zufriedenstellenden 3D Models geeignet sind oder nicht.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date

Signature

Acknowledgments

First I would like to thank Prof. Horst Bischof for giving me the opportunity to pursue a PhD degree under his supervision, while also being paid for the work I very much enjoyed during the last few years. I am very thankful for the fruitful work environment at ICG, where I was welcomed with open arms right from the start. In addition, I also want to thank my second supervisor, Prof. Reinhard Koch, for taking the time to review my PhD thesis, and to come down to Graz for my PhD defense.

I also want to thank all my colleagues from ICG, who made this working place very special, and helped to overcome some rarely occurring less pleasant periods (e.g. the incredible heat in the offices during the summer). A special thanks goes to the *Aerial Vision Group*, the reading group, and all members of the *3D-pitoti* project, for all the fruitful discussions, collaborations, the long drives to Italy, and of course all the social events that took place.

I want to thank my family, especially my parents and my brother, for all their (not just financial) support during all my years of studying, and for the healthy environment in which I was lucky enough to grow up. I would not have gotten that far without you, and I hope I can do the same for my own kids someday. Last, but definitely not least, I want to thank my fiancée Anna for putting up with we, and for always being there for me. I love you, and I always will.

This work has partially been supported by the Austrian Research Promotion Agency (FFG) projects *FreeLine* (Bridge1, 843450) and *PEGASUS* (FIT-IT, 825841/10397), in collaboration with OMICRON electronics GmbH. In addition, I received funding from the European Union for working on the *3D-pitoti* project (Seventh Framework Programme, 600545) during my time as a PhD student.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Contribution and Overview . . . 4

2 Related Work **7**
2.1 Point-based Structure-from-Motion . . . 9
2.2 Line Matching . . . 12
2.3 Line-based Structure-from-Motion . . . 18
2.4 Line-based SLAM . . . 23
2.5 Line-based Multi-View Stereo . . . 26
2.6 Summary . . . 31

3 Line3D++: A Line-based 3D Reconstruction Framework **33**
3.1 Prerequisites . . . 33
3.1.1 Camera Model . . . 34
3.1.2 Epipolar Geometry . . . 37
3.2 Pipeline Overview . . . 40
3.3 Line Segment Detection . . . 41
3.4 Establishing Line Segment Correspondences . . . 43
3.4.1 Visual Neighbor Selection . . . 44
3.4.2 Epipolar-guided Line Matching . . . 46
3.4.3 Improving the Matching Precision . . . 47
3.4.4 Creating 3D Hypotheses from Matched Segments . . . 48
3.5 Evaluating Line Segment Correspondences . . . 49
3.5.1 Scale-Invariant Spatial Regularization . . . 53
3.6 Assigning 3D Locations to 2D Segments . . . 53

3.7 Clustering Corresponding 2D Segments . . . 55
3.7.1 Final 3D Lines from Clustered Segments . . . 56
3.8 Combined Bundle Adjustment . . . 57
3.9 Summary . . . 59

4 Evaluation and Results 61
4.1 Testing Environment & Implementation Details . . . 61
4.1.1 SfM Pipeline . . . 62
4.2 Default Parameters . . . 63
4.3 Test Datasets . . . 64
4.3.1 Groundtruth Sequences . . . 64
4.3.2 Real-World Sequences . . . 65
4.4 Reconstruction Results . . . 67
4.4.1 Results on Real-World Sequences . . . 67
4.4.2 Results on Groundtruth Sequences . . . 68
4.5 Parameter Evaluation . . . 70
4.5.1 Line Segment Detection . . . 70
4.5.2 Line Matching . . . 73
4.5.2.1 Evaluating τ . . . 74
4.5.2.2 Evaluating M and k . . . 76
4.5.3 Match Evaluation . . . 77
4.6 Bundle Adjustment Evaluation . . . 79
4.6.1 Camera Pose Accuracy . . . 81
4.6.2 Line Model Accuracy . . . 82
4.7 Runtime Evaluation: GPU vs. CPU . . . 82
4.8 Summary . . . 84

5 Application: Online SfM using Points and Lines 85
5.1 Online Structure-from-Motion . . . 85
5.2 Incremental Line3D++ . . . 89
5.2.1 Line Segment Matching & Depth Estimation . . . 89
5.2.1.1 Line Segment Matching . . . 89
5.2.1.2 Depth Estimation . . . 90
5.2.2 3D Model Update . . . 91
5.2.2.1 Segment Clustering . . . 91
5.2.2.2 Cluster Verification . . . 92
5.2.3 Bundle Adjustment . . . 93
5.3 Experimental Results . . . 93
5.3.1 Test Setup . . . 94
5.3.2 Quantitative Evaluation . . . 95
5.4 Summary . . . 99

6 Conclusion **101**
6.1 Summary 101
6.2 Future Work: Beyond Lines 102

A List of Acronyms **107**

B List of Publications **109**
B.1 2013 109
B.2 2014 110
B.3 2015 111
B.4 2016 112

Bibliography **115**

List of Figures

1.1 Example SfM reconstruction for the *Pylon* sequence. 2
1.2 Close-up views of the *Pylon* sequence. 2
1.3 Comparison between sparse SfM-, and dense MVS point clouds. 3
1.4 Three different 3D representations of the *Pylon* and the *Building* sequence. 5

2.1 Illustrations from (Baillard et al., 1999). 13
2.2 Illustrations from (Wang et al., 2009) and (Wang et al., 2012). 14
2.3 Illustrations from (Zhang and Koch, 2013). 15
2.4 Illustrations from (Wang et al., 2009). 16
2.5 Illustrations from (Kim and Lee, 2010). 16
2.6 Matching of wiry structures. 17
2.7 Illustrations from (Bay et al., 2006). 20
2.8 Illustrations from (Schindler et al., 2006). 20
2.9 Illustrations from (Elqursh and Elgammal, 2011). 21
2.10 Illustrations from (Zhang and Koch, 2014). 22
2.11 Illustrations from (Micusik and Wildenauer, 2014). 22
2.12 Illustrations from (Ramalingam et al., 2015). 23
2.13 Illustrations from (Zhang et al., 2012). 25
2.14 Illustrations from (Flint et al., 2010). 26
2.15 Illustrations from (Werner and Zisserman, 2002). 27
2.16 Illustrations from (Ok et al., 2011). 28
2.17 Illustrations from (Chen and Wang, 2010). 29
2.18 Illustrations from (Jain et al., 2010). 30
2.19 Illustrations from (Jain et al., 2010). 30

3.1 The pinhole camera model. 35

3.2 The epipolar geometry. 38
3.3 Limitations of the epipolar geometry. 40
3.4 Line3D++ pipeline overview. 42
3.5 Line segment detection using LSD. 43
3.6 Visualization of the distance-based visual neighbor score. 46
3.7 The epipolar-based matching procedure. 47
3.8 A visualization of the obtained matches. 48
3.9 Computation of a 3D hypothesis from matched 2D line segments. 49
3.10 Inlier and outlier correspondences. 51
3.11 Derivation of the slope μ_σ of the spatial regularizer. 53
3.12 All accepted 3D line hypotheses, and the best hypothesis for each 2D segment. 54
3.13 Final 3D models for the *Building* and the *Pylon* sequence using two different clustering methods. 58

4.1 The *Timberframe* sequence. 65
4.2 The *Strecha* sequences. 65
4.3 Example images from the real-world sequences. 66
4.4 Example images from the *Dubrovnik6K* dataset. 67
4.5 Reconstruction results for the real-world datasets. 68
4.6 Line-based 3D reconstruction of the crowdsourced *Dubrovnik6K* dataset. . . 69
4.7 Quantitative evaluation on the *Herz-Jesu-P8*, and the *Fountain-P11* datasets. 70
4.8 Quantitative evaluation on the *Timberframe* dataset. 71
4.9 Reconstruction results for the *Herz-Jesu-P25*, and the *Castle-P30* datasets. 71
4.10 Reconstruction results for the line segment detection evaluation (real-world). 73
4.11 Reconstruction results for the line segment detection evaluation (groundtruth). 74
4.12 Reconstruction results for the line matching evaluation (real-world). 75
4.13 Reconstruction results for the line matching evaluation (*Pylon*). 78
4.14 Reconstruction results for the match scoring evaluation. 80
4.15 Reconstruction results for extreme σ values. 81
4.16 Reconstruction results for the combined bundle adjustment evaluation. . . . 83

5.1 An example of an unsuitable image set for SfM. 86
5.2 A schematic overview of the online SfM pipeline by (Hoppe et al., 2012). . . 87
5.3 An online SfM result using points and lines. 88
5.4 An illustration of the incremental reconstruction procedure. 94
5.5 A visual comparison between the offline- versus the online method (groundtruth). 96
5.6 A visual comparison between the offline- versus the online method (real-world). 97
5.7 The basic online/offline 3D reconstruction workflow. 100

6.1 Line-based vs. curve-based 3D reconstruction. 104

List of Tables

4.1 List of default parameters. 64
4.2 List of real-world test sequences. 66
4.3 Reconstruction results for the real-world datasets. 69
4.4 Evaluation results for the line segment detection step. 72
4.5 Quantitative evaluation results for the line segment detection step. 72
4.6 Evaluation results for the line matching step. 76
4.7 Quantitative evaluation results for k-nn line matching. 77
4.8 Evaluation results for the line matching step (*Pylon*). 79
4.9 Evaluation results for the combined bundle adjustment. 82
4.10 Quantitative evaluation results for the bundling step. 83
4.11 Runtime evaluation results (GPU vs. CPU). 84

5.1 A comparison between the offline- and the online algorithm. 95
5.2 Camera pose evaluation results (offline vs. online). 98
5.3 Line model evaluation results (offline vs. online). 98

1 Introduction

Contents

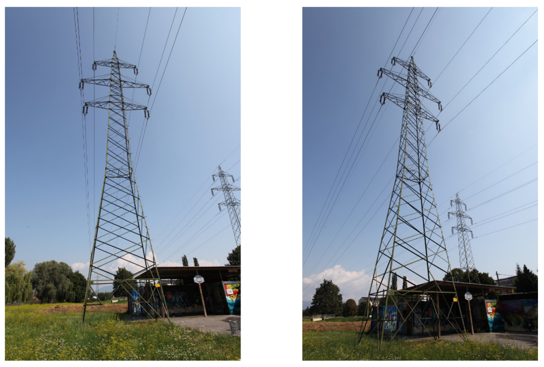
1.1 Motivation 1
1.2 Contribution and Overview 4

1.1 Motivation

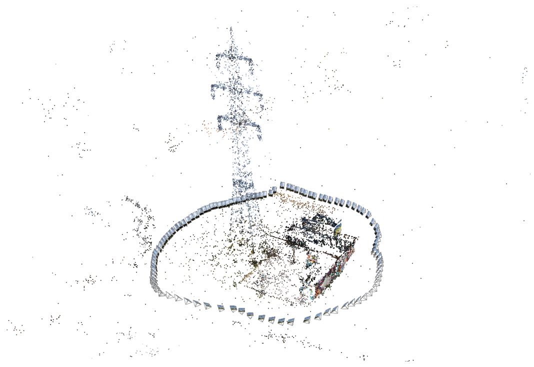
Not too long ago, automatic recovery of 3D information from an image sequence used to be a very challenging and time consuming task. Today, thanks to freely available software [114, 136, 137, 143, 149, 172], as well as sophisticated professional tools [3, 24, 147], even non-expert users are able to generate accurate (georeferenced) 3D models from arbitrary scenes within hours.

The first step of any traditional image-based 3D reconstruction pipeline, is the so-called Structure-from-Motion (SfM) step [2, 47, 143]. Here, the goal is to simultaneously solve for the unknown camera poses of the images, and the underlying 3D structure of the scene. To be able to handle very large datasets consisting of thousands of images, all common *SfM* pipelines make use of a sparse set of distinctive feature points (key points) to find connections between images, which then constrain the camera poses and give an initial 3D representation of the scene. Figure 1.1 shows a typical *SfM* result for an image sequence around a power pylon. The sparse point cloud represents the matched feature points in 3D, and the small image thumbnails at the bottom of the pylon represent the camera poses of the respective images.

As we can see in this example, the 3D point density is not equally distributed throughout the whole reconstruction. The amount of reconstructed 3D points for each part of the scene depends heavily on its visual appearance (see Figure 1.2 for close-up views). This is due to the fact that matchable feature points can only be extracted from non-homogeneous and preferably unique image regions, such that they can be mathematically



(a) Example images

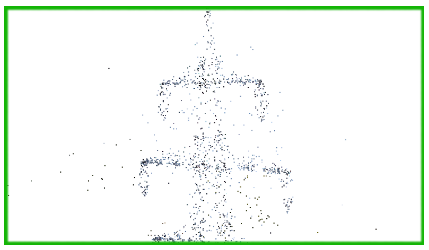


(b) *SfM* result (18, 623 points)

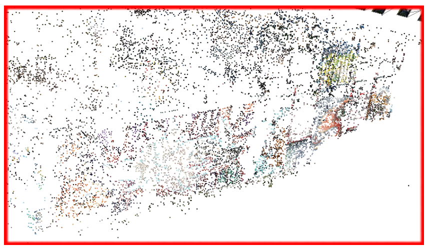
Figure 1.1: Example *SfM* reconstruction for the *Pylon* sequence (106 images): (a) Two example input images from the sequence, (b) visualization of the *SfM* result (camera poses and sparse point cloud) using the *ICG3D* library [72].

encoded in a compact and meaningful way, through one of the many available feature point descriptors (e.g. Scale-Invariant Feature Transform (SIFT) [101] or Speeded Up Robust Features (SURF) [18]), which allow scale-, rotation and illumination invariant comparison of features from different images. Hence, the 3D output obtained by any *SfM* pipeline is usually quite sparse in any case (even when a high number of features per image is used), and only provides 3D information in textured parts of the scene.

(a) Overview



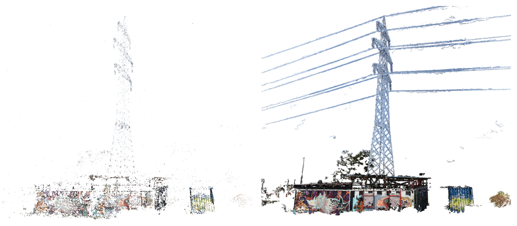
(b) Untextured region (more sparse)



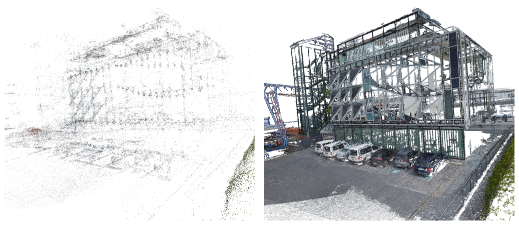
(c) Textured region (more dense)

Figure 1.2: Close-up views of the sparse point cloud from the *Pylon* sequence: (a) The whole model with the close-up areas highlighted (approximately), (b) the sparsely occupied top-part of the pylon, and (c) the more densely occupied wall.

The more important part of the *SfM* output are actually the obtained camera poses. The availability of camera pose information significantly eases the recovery of 3D information from an image sequence, and especially the directly computable *epipolar geometry* between different images simplifies the correspondence problem to a one dimensional search along the epipolar lines [54]. This enables subsequent Multi-View Stereo (MVS) pipelines, such as *PMVS* [49] or SURE [131], to create accurate (semi-) dense point clouds, which also provide 3D information for less-textured parts of the scene, and significantly improve



(a) *Pylon* (18, 623 → 1, 393, 124 pts.)



(b) *Building* (97, 699 → 14, 516, 368 pts.)

Figure 1.3: Comparison between sparse *SfM*-, and dense *MVS* point clouds for (a) the *Pylon* sequence (106 images), and (b) the *Building* sequence (344 images). The dense models were obtained using *PMVS* [49].

the overall visual impression and semantic meaning of the obtained 3D model. In addition, watertight surface meshes can be directly extracted from such dense point clouds (e.g. [69, 91]), and realistically textured using the underlying image sequence [160]. Figure 1.3 illustrates the difference in terms of the visual impression between a sparse *SfM*-, and a dense *MVS* point cloud. As we can see, the dense model significantly improves the level of detail in both cases. This is especially notable for the *Pylon* sequence, were for example non of the power lines are visible in the sparse point cloud at all.

While the *SfM* process can be performed quite efficiently even for large-scale datasets (e.g. by exploiting massive parallelism [47] and efficient feature matching strategies [58, 120]), computing a dense 3D model is still computationally expensive, and can easily take up to several days on standard desktop computers. Moreover, the resulting model often consists of millions of points (or triangles), which makes all kinds of subsequent data analysis tasks very tedious and challenging. In many cases, even viewing such a 3D model in common 3D viewers can bring modern computers to their limits, due to the huge amount of main- and *GPU* memory that is needed to render large point clouds or meshes.

The benefit of point clouds (or also meshes) is their flexibility, since objects of arbitrary shapes and complexities can be theoretically expressed by such a data structure. However, as stated above, in terms of point clouds the number of points that is produced by common *MVS* algorithms is usually very large, and since pretty much every state-of-the-art 3D reconstruction software (e.g. Pix4D [147]) uses such a point cloud to derive the 3D surface from it, the produced meshes have a very high complexity as well. While there are of course scenes with an inherently high geometrically complexity (e.g. non-urban areas, such as forests or mountains), which definitely require a lot of 3D entities to be accurately reconstructed, there are also a large number of scenes where the relevant 3D information can be encoded much more efficiently, due to a high amount of geometric regularities.

The most prominent example are man-made indoor and outdoor environments, such as urban areas. Here, we encounter a high amount of piecewise planar and linear structures (e.g. buildings, roads, or power pylons), which can be efficiently represented by a set of 3D planes or lines. Since it is not very efficient to first compute a dense point cloud or mesh (which might take a long time) just to reduce this data to a more abstract 3D model afterwards, it would be beneficial to have efficient algorithms that can directly extract such semantically meaningful and simple 3D models from (oriented) image sequences. To achieve this goal, we propose a novel method to generate 3D line models from *SfM* results in an efficient and robust way. We show how such an approach can effectively capture the 3D structure of man-made environments significantly faster than existing dense *MVS* methods, and with much lower memory requirements.

1.2 Contribution and Overview

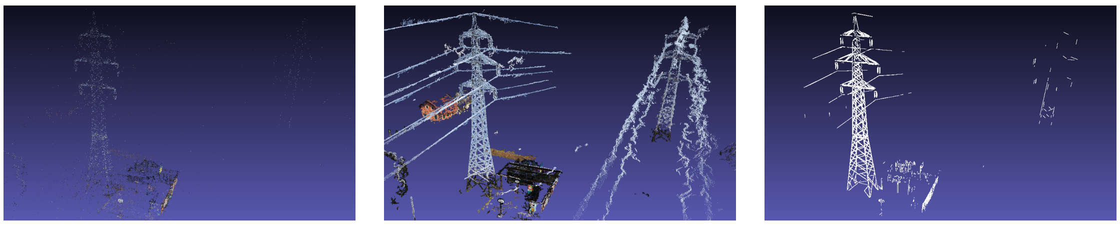
In this thesis, we focus on the task of efficient 3D scene abstraction for the built environment, by making use of 2D line segments as complementary image features to traditional key points. We propose a novel line-based 3D reconstruction method, which can operate directly on the output of any traditional *SfM* pipeline. Our method is based on epipolar-guided line segment matching, which does not require any kind of appearance information, to be able to handle illumination changes and wiry structures, for which traditional line matching methods are not suitable. We derive 3D estimates for all 2D segments in all images individually, by analysing the mutual support between their pairwise matches, and formulate the final 3D reconstruction as a graph-clustering problem. Our algorithm is very runtime efficient, and orders of magnitude faster than standard dense 3D reconstruction algorithms, while still providing a significant amount of valuable and semantically meaningful 3D information in a very compact manner.

Figure 1.4 shows a comparison between a sparse-, dense-, and a line-based 3D model for the two datasets introduced in Section 1.1. As we can see, our line-based reconstruction provides a high degree of semantically meaningful 3D information, despite its sparsity compared to the dense point model (only several thousand lines vs. several million points). Moreover, our method is much more efficient than computing a dense 3D model, with an average runtime of approximately one second per image on both datasets.

The work presented in this thesis resulted in a publicly available reconstruction framework called *Line3D++* [1], which enables the accurate and fast line-based 3D reconstruction of arbitrary scenes, without excessive parameter tuning. It comes with native support for several major *SfM* pipelines (free and non-free), and can be easily linked to any other kind of *SfM*-like application. We further present how our method can be incorporated into an online *SfM* pipeline, to jointly generate sparse point clouds and 3D line models on-the-fly, already during image acquisition.

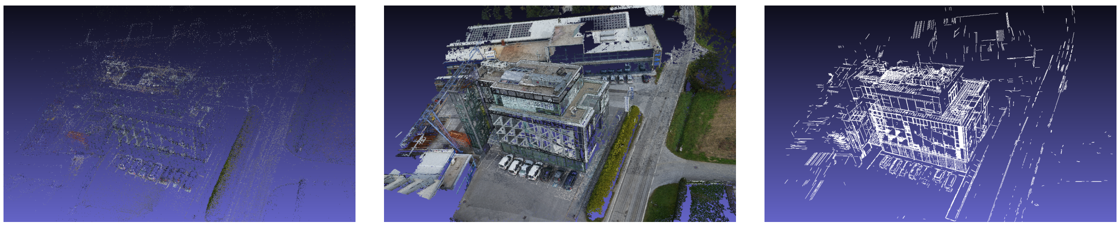
The remainder of this thesis is structured as follows. In Chapter 2 we give an extensive overview over all related methods that deal with the task of line-based 3D reconstruction, and put them in relation to our proposed method. In Chapter 3 we introduce our reconstruction principles for the offline case, and explain all necessary reconstruction steps in

[1] `https://github.com/manhofer/Line3Dpp`



$18,623$ points	$1,393,124$ points	**$5,319$ lines**
$t = 660$ s	$t = 7,980$ s	**$t = 75.27s$**
$\varnothing t = 6.23$ s	$\varnothing t = 75.28$ s	**$\varnothing t = 0.71s$**

Pylon, 106 images



$97,699$ points	$14,516,368$ points	**$11,238$ lines**
$t = 2,640$ s	$t = 51,660$ s	**$t = 361.17s$**
$\varnothing t = 7.67$ s	$\varnothing t = 150.17$ s	**$\varnothing t = 1.05s$**

Building, 344 images

Figure 1.4: Three different 3D representations of the *Pylon* and the *Building* sequence: *Left:* *SfM* result [72] (*SIFT* [101] features), *Middle:* dense *PMVS* [49] point cloud, *Right:* line-based 3D model by our proposed method (*Line3D++*).

more detail. In Chapter 4 we perform an extensive evaluation of all parts of the pipeline separately, and show experimental results on several real-world and synthetic datasets, with and without groundtruth. Finally, we show how our approach can be integrated into an online *SfM* system in Chapter 5, and conclude with several ideas for future works in Chapter 6.

2 Related Work

Contents

2.1 Point-based Structure-from-Motion 9
2.2 Line Matching 12
2.3 Line-based Structure-from-Motion 18
2.4 Line-based SLAM 23
2.5 Line-based Multi-View Stereo 26
2.6 Summary 31

Line segments have been used in the context of 3D vision since the early 1980s, with various applications such as establishing correspondences between maps and aerial images [108], stereo matching [8, 106, 109], tracking [32], robot navigation [7], or Structure-from-Motion (SfM) [57, 153]. However, even though many of these methods already introduced promising concepts for image-based 3D reconstruction, processing realistic and possibly large-scale datasets still remained virtually impossible. This changed with the introduction of efficiently detectable and reproducible image feature points (e.g. Harris corners [53]), and highly invariant feature point descriptors (e.g. the well-known *SIFT*- [101], or *SURF* [18] descriptors), which started the successful era of *SfM* for the masses [2, 47, 143], and subsequently shifted the momentum towards point-based methods.

Later on, the core concepts of modern feature point descriptors have been adopted for line segments as well, resulting in so called line descriptors (e.g. [163, 179, 182]), which enabled the robust and efficient matching of line segments for the wide baseline case, and even under severe illumination and viewpoint changes. However, line-based 3D reconstruction methods that could handle realistic datasets remained surprisingly rare. Most of the presented algorithms either work only in very constrained environments (e.g. Manhattan-worlds [132]), or suffer from an unnecessarily high computational complexity (e.g. [73]). Only in recent years, more general and more efficient methods have been introduced, which finally caught up with point-based *SfM* and Multi-View Stereo (MVS), in terms of usability and robustness [63, 110, 180].

In this chapter we first give a general overview of traditional point-based SfM (Section 2.1), followed by an extensive survey of all related methods that deal with straight line segments in a 3D context. In general, all relevant literature in line-based 3D vision can be put in one of the following categories:

1. **Line Matching** (Section 2.2):
   Matching 2D line segments between different images, with appearance and/or geometric constraints. The aim is to create inter-image correspondences between line segments that originate from the same physical 3D structure. Having such correspondences is a necessary requirement for most subsequent 3D vision tasks, such as line-based *SfM*, *SLAM*, or *MVS*.

2. **Line-based Structure-from-Motion** (Section 2.3):
   Performing all parts of an *SfM* pipeline with line segments only. This includes feature matching, camera pose estimation, sparse 3D reconstruction, and bundle adjustment. The result is usually a 3D line model and the camera poses of the underlying images.

3. **Line-based SLAM** (Section 2.4):
   Performing real-time visual navigation with line segments as image features. Similar to *SfM*, the output is a 3D map of the environment with the current and past positions of the camera. However, the goal is to get an on-the-fly response in real-time, to use the obtained information for localization and obstacle avoidance (e.g. of a moving robot).

4. **Line-based Multi-View Stereo** (Section 2.5):
   Performing line-based 3D reconstruction with already given camera poses. As in point-based *MVS*, the goal is to get an accurate 3D model of the scene that is as complete as possible. Hence, the runtime is usually not a critical point, which enables more complex matching and reconstruction strategies, as well as the usage of more 2D features (e.g. in contrast to time-critical *SLAM* applications). In addition, the availability of the camera poses significantly eases line matching, since geometric (e.g. epipolar) matching constraints can be used in addition to (or instead of) appearance constraints.

Our work itself belongs to the fourth category (Line-based *MVS*), since we use camera poses obtained by traditional (point-based) *SfM* pipelines as input. However, parts of our algorithm are related to approaches throughout all of the aforementioned categories, which we will introduce and discuss in the following sections.

2.1 Point-based Structure-from-Motion

What we know today as *SfM* has its origins in early image-based 3D reconstruction methods, which have also been known under the term *photogrammetry* since the late nineteenth century [6]. An excellent survey that covers the historic development from these early methods to modern *SfM* pipelines can be found in [68]. Basically, most of the common *SfM* pipelines today follow six steps:

1. **Image Acquisition:** Several (≥ 2) images from different viewpoints are taken from the target object or scene. All relevant parts of the scene should be visible in at least two images, and the camera movements between subsequent shots should be small. This step is technically not part of the *SfM* pipeline itself, but rather a necessary prerequisite (unless online *SfM* is considered; see Chapter 5).

2. **Feature Detection:** Distinctive key points are extracted from all images, and mathematically described by one of the numerous descriptors available (e.g. *SIFT* [101] or *SURF* [18]). These key points with their corresponding descriptors are often also referred to as *image features*, or *feature points*.

3. **Feature Matching:** The image features are now *matched*, i.e. by finding visually similar pairs of features (based on their descriptor similarity) from two different images, that fulfill some minimum similarity criterion. The matching process is usually a *one-to-one* matching, so each feature from one image can only have (at most) one correspondence in another image.

4. **Geometric Verification:** Matched image pairs are now verified by computing the relative pose between them, using the obtained feature correspondences and a robust *RANSAC* [45] procedure. All correspondences are then verified using basic epipolar constraints, and outlier matches are subsequently removed.

5. **Reconstruction:** Given the relative poses between matched images from the previous step, a consistent 3D model is created by merging all cameras to one world-coordinate frame. This can either be done incrementally (starting from an initial camera pair; e.g. [47, 143, 172]), or in a global way (e.g. [28, 150]). During this procedure, all matched feature points are triangulated to 3D space, to form a sparse point-cloud.

6. **Bundle Adjustment:** To ensure a consistent 3D model, the last step is to run a global bundle adjustment procedure [156], which is basically a non-linear optimization, that optimizes camera poses, 3D points, and potentially also the distortion coefficients for all images simultaneously, by minimizing the reprojection error over all reconstructed 3D points and their 2D residuals.

Of course, all available *SfM* pipelines are slightly different from each other. However, these six steps can be considered the basics of *SfM*, and should help the reader to better understand the methods which are going to be introduced in the following paragraphs. For a more complete overview of the *SfM* basics, as well as multiple-view geometry in general, we kindly refer to the excellent book by Hartley & Zisserman [55].

Among the first to make *SfM* mainstream were Snavely et al. [143] in 2006. They demonstrated how large and unordered image collections (e.g. from the Internet) can be used to interactively explore the depicted sites of interest in 3D, directly on ones home computer (see the *Photo Tourism* project for more details [1]). The core part of their method is an *SfM* pipeline called *bundler*, which is freely available, and arguably became the state-of-the-art in *SfM* for many years. They use *SIFT* [101] features to create correspondences between all image pairs, and robustly estimate the relative poses between them using the eight-point algorithm [55] in a *RANSAC* [45] loop. They finally use the pairwise relative poses and all consistent *tracks* of matched key-points to jointly estimate a global sparse 3D model, as well as the camera poses of the aligned images.

While their method produces very accurate 3D models, it is computationally quite expensive, since all possible image pairs are matched. However, in many cases there are far more image pairs which do not have a visual overlap (i.e. which do not *see* the same part of the scene), than pairs which actually do. This is especially notable in large image sets, which e.g. span a whole city. Hence, it is beneficial to determine which image pairs should be matched *before* the matching procedure actually takes place. For this purpose, a popular solution is to use a vocabulary tree [120], which basically inserts the *SIFT* (or other) descriptors into a tree structure, and determines which images are visually similar by analyzing how many of their descriptors fall into similar leaf notes. A *SfM* pipeline that makes use of this method was e.g. presented by Irschara et al. [72]. They showed how this relatively simple procedure results in a massive speed-up, without negatively influencing the accuracy or completeness of the reconstruction results.

With the quasi permanent availability of high-resolution cameras in mobile devices, as well as the increasing popularity of uploading images to social media (e.g. Flickr [2]), *SfM* methods which are capable of handling thousands of uncalibrated images at a time, to reconstruct popular and frequently photographed landmarks, came more and more into focus. Similar to the *Photo Tourism* project mentioned above, Agarwal et al. [2] published a paper called *Building Rome in a Day*, where they demonstrated how the city of Rome in Italy could be reconstructed from more than two million crowd-sourced photographs in just one single day, on a massive computing cluster. They basically used the *bundler* software [143], but introduced a novel parallel distributed matching system, and a highly efficient bundle adjustment procedure. A year later, Frahm et al. [47] further improved the efficiency of the method and managed to reconstruct the same scene in one day on one single machine. They make use of GIST [124] features, to capture the global image

[1]`http://phototour.cs.washington.edu/`
[2]`http://www.flickr.com/`

appearances, and use a k-medoids [79] clustering algorithm to divide the image set into visually similar subsets. For each set, they derive one iconic view which is then used as a representative for each cluster of images. Finally, they use a vocabulary tree based matching procedure on all iconic views to compute a sparse 3D model of the scene. In subsequent years, various other approaches to very large-scale (or *Internet-scale*) *SfM* have been presented, which aim at increasing the number of images that can be realistically processed towards the billion mark [59, 128, 138].

Most *SfM* pipelines introduced so far work under the assumption that all necessary pieces of information (i.e. all images) are already available during execution time. However, if the final 3D model is not satisfying, because it e.g. shows some holes at parts where not enough images have been taken, it can not be further improved when more information (i.e. additional images) becomes available. However, conceptionally speaking it is quite straightforward to just add new images to the reconstruction by performing the basic matching procedure for each new image, and then adding the corresponding camera poses using absolute pose estimation (e.g. [89]). Finally, a new round of bundle adjustment should ensure the global consistency of the obtained model. This procedure could easily be repeated as long as new images come along, and as long as these images have some visual overlap to the existing (already integrated) images. This concept is often referred to as *incremental SfM*.

Changchang Wu [172] released a very efficient incremental *SfM* pipeline, which is known as *VisualSfM*. It also uses *SIFT* features (as most *SfM* pipelines) and is highly efficient due to a preemptive matching strategy, and a multicore bundle adjustment implementation [173]. In their corresponding paper [172], they showed that in practice the runtime complexity of incremental *SfM* can be considered linear-, despite being in fact quadratic in the number of images. This holds as long as the number of images is not too large ($\approx 15K$ in their evaluations).

Related to incremental *SfM*, *online SfM* goes one step further and allows to compute sparse 3D models from high-resolution images on-the-fly, already while the images are being recorded. Hoppe et al. [70] proposed a sophisticated online *SfM* system (based on [72]), which starts from an initial image pair, and incrementally adds new images to the model using absolute pose estimation [89]. It also allows rapid position changes, since it uses a vocabulary tree [120] to identify the already integrated images to which a new image should be aligned. This of course only works if the part of the scene depicted in the new image has already been *seen* before (i.e. there actually is an existing image to which the new image can be matched). Evaluations on groundtruth datasets showed that their method has a comparable accuracy to *bundler*, however with a much lower runtime and with the benefit of being fully interactive. More details about online *SfM* can be found in [68]. In Chapter 5, we will demonstrate how our proposed method can be efficiently integrated into this online *SfM* system, to generate both a sparse 3D point- and a 3D line model on-the-fly, without negatively influencing the overall runtime (compared to point-based *SfM* alone).

In recent years, several new *SfM* pipelines have been proposed by various research groups (e.g. [28, 114, 136, 137, 149, 167]). For the most part, the core concepts among them are quite similar. The differences lie often in the details, and a general statement on which of these pipelines is best is hard (if not impossible) to make. Since full *SfM* pipelines have been published quite frequently already, many researchers focus on improving a specific part of *SfM* separately, which leads to interesting algorithms that can subsequently be integrated into existing (or new) *SfM* methods. For instance, publications about e.g. direct structure estimation [75], optimizing the viewing graph [150], or stitching visually disconnected *SfM* sub-models via symmetry information [27] have been released recently, to just name a few.

All the *SfM* pipelines mentioned above ultimately result in the same kind of output, which consists of the camera poses (for all successfully integrated images) and a sparse set of 3D points (with visibility information). As already stated above, some methods in addition also provide the distortion coefficients for the images, which allows us to undistort all images before 2D line segments are being extracted later on. For our line-based *MVS* approach it does not matter which *SfM* pipeline was executed to obtain the camera poses. However, it is recommended to use a state-of-the-art method (e.g. *colmap* [137]) since it in general produces more accurate and more complete results than older methods. Our framework comes with native support for the following *SfM* pipelines:

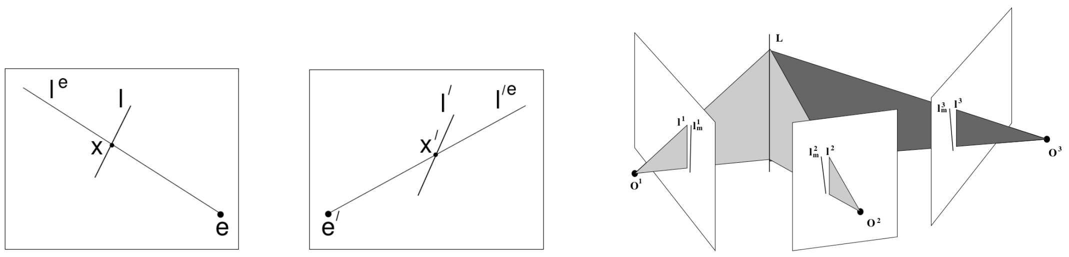
- *bundler* [143] - `http://www.cs.cornell.edu/~snavely/bundler/`
- *VisualSfM* [172] - `http://ccwu.me/vsfm/`
- *Pix4D* [147] - `https://pix4d.com/` [3]
- *mavmap* [136] - `https://github.com/mavmap/mavmap/`
- *OpenMVG* [114] - `https://github.com/openMVG/openMVG/`
- *colmap* [137] - `https://github.com/colmap/colmap`
- *ICG3D* [72] - internal (not publicly available)

We want to emphasize again that our method is not limited to these approaches, and additional interfaces can be created straightforward at any time.

2.2 Line Matching

Line matching is the process of establishing pairwise correspondences between two sets of line segments. Common tasks are 2D line segment matching between two images, or establishing 3D to 2D correspondences between a 3D model and an image (e.g. for camera

[3]commercial software, not freely available



(a) Epipolar geometry **(b)** Trifocal constraint

Figure 2.1: Illustrations taken from [9]. (a) Establishing a point-wise correspondence $X \leftrightarrow X'$ between two line segments l and l' from two images, by intersecting the epipolar line l'^e of X with l' (and vice versa). (b) The projective relationship between a 3D line segment L and its images (2D segments) l^1, l^2, and l^3 in three different views. If l^1 and l^2 are correctly matched, the intersection of their viewing planes coincides with their real pre-image L, which can be projectively verified by l^3.

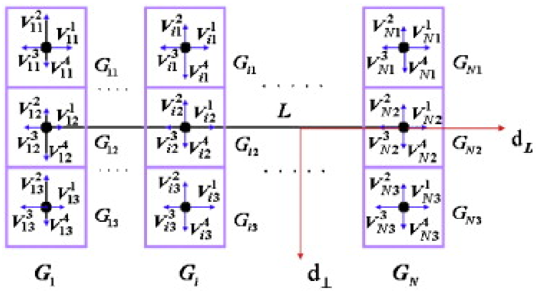
pose estimation [181]). In this section, we focus on the former task, since we do not assume a given 3D model in our work, and we do not focus on line-based pose estimation as well.

Basically, line matching methods can be divided into two categories. Those that match line segments individually by exploiting only local pairwise similarities (geometric or appearance), and those that match the whole set at once, e.g. by using a graph matching formulation that exploits local- as well as global matching constraints. The former usually benefit from a lower computational complexity (similar to classic local feature point matching in *SfM* pipelines), while the latter are generally more robust to repetitive patterns and weak visual distinctiveness in the target images, by also exploiting geometric relationships between segment pairs (e.g. collinearity, intersections, co-planarity,...).

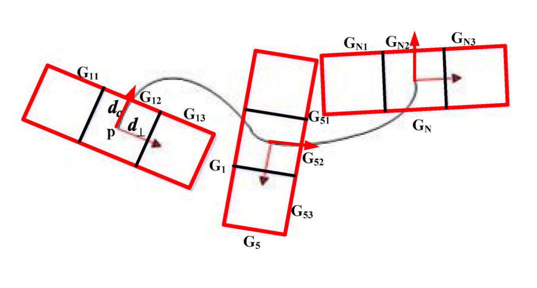
In the earlier days of Computer Vision, line segment matching was mostly done by establishing potential correspondences using geometric- (e.g. length and orientation differences, or epipolar overlap) and intensity constraints (e.g. intensity- or contrast differences), followed either by a local matching procedure [32, 106, 109], or graph matching [8, 108]. All of these methods already achieved quite impressive results, given the limited computational resources available at that time.

A few years later, Schmid and Zisserman [134] proposed an automatic line matching approach for image triplets. Assuming known camera poses, they exploit the epipolar geometry to establish point-wise line segment correspondences for intensity based matching (Figure 2.1a). To verify potential matches, they make use of the trifocal tensor [57], which encodes all projective geometric relationships between three images (Figure 2.1b). This two step procedure enables virtually outlier free line matching results, but also requires knowledge about the extrinsic camera parameters, and at least three images. They further applied their method to curve matching [135], as well as piecewise planar 3D reconstruction of buildings [9].

In 2005, Bay et al. [17] proposed the first line descriptor for uncalibrated wide-baseline



(a) *MSLD* descriptor [163]



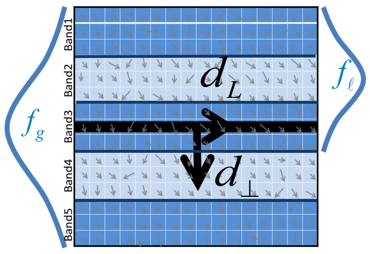
(b) *MSHS* descriptor [162]

Figure 2.2: Illustrations taken from [162, 163]. (a) The *SIFT*-like [101] structure of the *MSLD* line descriptor [163]. Several gradient histograms are computed orthogonal to ($d_\perp$) and along the line (d_L), and transformed into a line descriptor. (b) The same principle can be used to compute a descriptor for arbitrary 2D curves, resulting in the Mean-Standard Deviation of the Hue and Saturation (MSHS) descriptor [162].

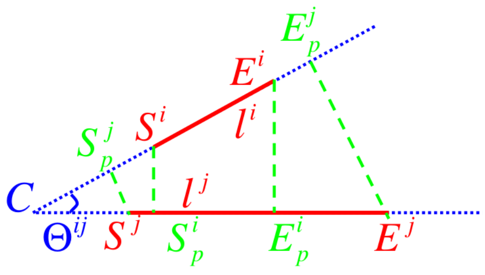
stereo images, by computing two HSV color histograms from image pixels directly adjacent to each line segment on either side. They match line segments by comparing the respective histograms using a robust dissimilarity measure, and further improve the matching accuracy by applying a topological filter which takes into account the relative locations between pairs of segments.

In contrast to a histogram-based descriptor, several patch-based line descriptors have been proposed in the following years, inspired by powerful feature point descriptors, such as *SIFT* [101] or *SURF* [18]. Khaleghi et al. [80] proposed the Scale Invariant Line Transform (SILT) descriptor, which uses Haar-like features and a Principle Component Analysis (PCA). The patch-based support region for their descriptor is scaled with the length of the line segment, to achieve scale invariance. However, this does not take potential occlusions and imprecisely detected or broken line segments into account. In the same year, the popular Mean-Standard Line Deviation (MSLD) line descriptor was introduced by Wang et al. [163]. It is closely related to *SIFT* [101], and builds up on the same principles (gradient histograms; see Figure 2.2a). They reported excellent matching results even under severe viewpoint and illumination changes, but no invariance to scale variations. Their method was further adapted and refined in [182], and extended for the task of curve matching [162] (Figure 2.2b).

Zhang and Koch [179] proposed the so-called Line Band Descriptor (LBD), which is constructed in a similar way as *MSLD*. The core difference is that they do not compute gradient histograms, but rather accumulate gradients within several band-like regions adjacent to- and parallel to the line segment (Figure 2.3a). They further improve the matching performance by enforcing pairwise geometric consistencies between spatially related line segments, using a spectral clustering approach (Figure 2.3b). They showed how a combination of a robust line descriptor with additional geometric constraints significantly boosts the line matching performance, compared to using appearance constraints only. A similar observation was made in [100], where they also combined appearance- and geometric (neighbourhood) constraints to an effective iterative line matching algorithm.



(a) *LBD* descriptor

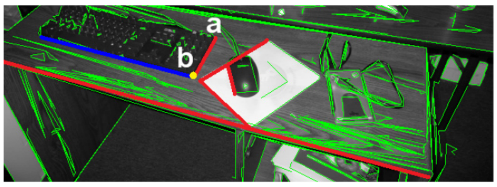


(b) Pairwise geometric relations

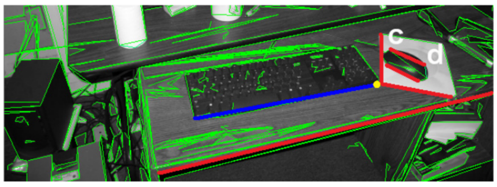
Figure 2.3: Illustrations taken from [179]. (a) The structure of the *LBD* line descriptor [179]. In contrast to *MSLD* (Figure 2.2a), the gradients are computed within several bands parallel to the line segment (Band 1 to 5), and not within quadratic patches. (b) An illustration of pairwise relationships (intersection point C, angle Θ^{ij}, relative lengths l^i and l^j, and pairwise endpoint projections $\{S_p^j, E_p^j\}$ and $\{S_p^i, E_p^i\}$) between two segments i and j from the same image. These relations are taken into account when building a relational graph, which is then used to match all line segments simultaneously by a spectral clustering approach.

Wang et al. [161] proposed an interesting approach to line matching, where they clustered groups of spatially close line segments to so-called Line Signatures (LSs). They measure similarities between two *LSs* from different images, by analysing the geometric configurations of their underlying line segments. Even though these configurations are only invariant under perspective transformations if all segments in the *LS* are coplanar, they showed that their approach also works well when this is not the case (given that the camera motion is not too extreme). Figure 2.4 illustrates two matched *LSs* from different images. While their approach achieves a fairly high matching accuracy, it is rather computationally expensive. A similar approach has been proposed by Fan et al. [41], where they also exploited neighbourhood information for matching, but replaced the neighbouring line segments with matched feature points, which might be coplanar with the lines. They originally tackled affine invariants by using one line- and two point correspondences, and later added projective invariants by using one line- and four point correspondences [42].

A different approach to the line matching problem was proposed by Kim and Lee [82]. They make use of the perspective invariance of coplanar line intersections, and match them by using a Normalized Cross Correlation (NCC) score on a patch around the intersection points. They show how their approach can effectively be used for various task such as fundamental matrix estimation, which can be done straightforwardly since the matched intersections are just ordinary point correspondences. They further refined their method to be more robust to intersections which occur on spatial discontinuities [83] (i.e. where only parts of the extracted intersection patch are on a planar surface; see Figure 2.5). A similar approach has been introduced by Li et al. [95, 96], where they replaced the *NCC* matching

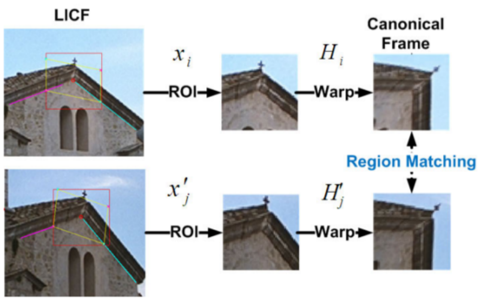


(a) Image 1

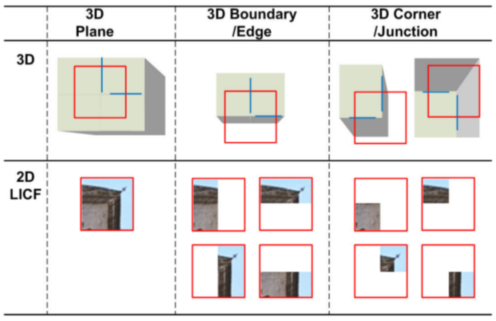


(b) Image 2

Figure 2.4: Illustrations taken from [161]. (a) A *LS* computed from a central segment (blue) and its neighboring segments (red). (b) Its match in the second image. As we can see, all segments have been matched correctly, apart from segment $\overline{ab}$ which has been erroneously matched with $\overline{cd}$.



(a) Matching line intersections



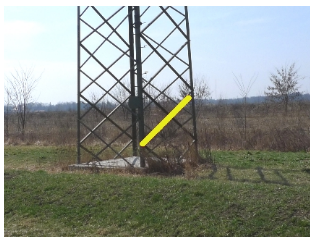
(b) Intersection scenarios

Figure 2.5: Illustrations taken from [83]. (a) Two matching line intersections from different views. The patches are rectified to a canonical frame to allow appearance-based matching. (b) The possible intersection scenarios and the resulting invariant regions within the corresponding patches. Only when the intersection occurs on a 3D plane (left column) the full patch can be used.

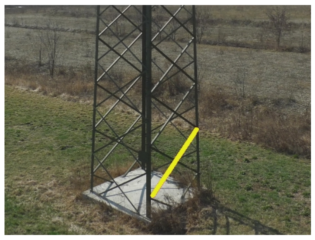
with a *SIFT*-like descriptor on the line junctions, and by Bauer et al. [14, 15], where they introduced so called *Zwickel* features, which are basically image regions enclosed by two intersecting coplanar line segments.

While the aforementioned methods produce satisfying results for a wide range of scenarios, their main problem is that virtually all of these methods are not invariant to scale changes. For instance, the *SILT* [80] descriptor promises scale invariance in his name, but this only holds when the line segment endpoints are the same in both views. For realistic scenarios, with occlusions and imprecisions in the line segment detection process, this usually does not hold. Fathi and Brilakis [43] proposed a method which determines the scale of a line by using Laplacian zero-crossings, to compute a scale invariant version of the *MSLD* [163] descriptor. While their method still cannot handle occlusions, it is invariant to imprecise line segment endpoints. Verhagen et al. [157] recently showed how two popular line descriptors (Bay et al. [17] and *MSLD* [163]) can be made scale invariant, and performed several evaluations undermining their proposed modifications.

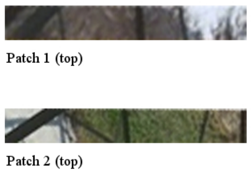
To sum up, line matching has been a very active field of research in Computer Vision,



(a) Image 1



(b) Image 2



(c) Extracted Patches

Figure 2.6: (a-b) Two images of a power pylon taken from different perspectives. The same part of the structure is highlighted in both views. The surroundings of the segments vary significantly between the two shots, since they are not coplanar with the segments (grass, sky, ...). (c) Two same-size patches extracted from the top part of the respective line segments. As we can see, the patches have a completely different appearance, which renders descriptor-based line matching almost impossible.

with a lot of different proposed solutions. The most recent and most promising methods all combine appearance- and geometric constraints, which is a very reasonable thing to do given the rich amount of intra-image relationships that line segments usually have (e.g. collinearity, or pairwise intersections), and which benefit graph-matching rather than individual local matching. However, for our case the situation is a little bit different. Since we operate in a multi-view scenario ($\geq$ 3 images) with given camera poses, we benefit from even more prior information. That is, given a pairwise match in two images, we can use other neighboring images to verify this match, by simply triangulating and backprojecting it into these images. The discussed line matching algorithms above are more general, and do not assume additional images to be available. Hence, they need to be as robust and accurate as possible, to ensure a high amount of correct matches even when just two images are available.

While the discussed line matching algorithms report very high matching scores for their evaluation testcases, in practice of multi-view 3D reconstruction their performance is often rather limited. For example, all patch-based line descriptors are not able to handle thin wiry structures (such as power pylons or fences), which frequently occur in man-made environments. The problem is that a patch drawn around a segment located on such a structure would almost always contain pixels which do not originate from the same planar surface on which the line segment is located, which results in a non-meaningful descriptor. Figure 2.6 illustrates this situation for a power pylon. As we can see, descriptor-based line matching is hard to achieve for such objects.

To handle all kinds of structures, we decided to omit appearance-based line matching completely, and simply rely on the more robust geometric properties that line segment matches over multiple images have. In particular, we use the epipolar overlap between the endpoints of a segment in one image, with a potentially matching segment in the other image as a similarity measure. This has the benefit that we do not have to compute and match any kind of line descriptors, which significantly boosts the runtime performance and also makes our method less sensitive to illumination changes. However, since we only use weak epipolar constraints, we cannot perform a robust one-to-one matching for pairs of images, since for one segment in the first image it can easily happen that several segments from the second image fulfill the underlying overlap constraints. Just taking the one match that maximizes the epipolar overlap is also not a good solution for the general case, especially when we have to deal with a high amount of occlusions. Therefore, we simply keep *all* matches that fulfill the criteria (one-to-many matching), and efficiently score and verify them using other neighboring images, after the matching procedure. We will show how this approach successfully enables the virtually outlier free 3D reconstruction of wiry as well as solid objects, with quasi real-time performance.

2.3 Line-based Structure-from-Motion

Structure-from-Motion (*SfM*) means to simultaneously recover the motion parameters of a moving camera, and 3D information about the scene it observes. Most commonly, this is done by extracting a discrete set of image features that are then matched or tracked throughout multiple views, which allows to solve for the unknown camera parameters and the 3D coordinates of these image features.

State-of-the-art *SfM* pipelines [72, 114, 143, 147, 172] usually use very distinctive 2D image points (also called *interest points* or *key points*) as features, which has two main benefits. First, a small number of matched points over as little as two images already provides enough information to compute the camera poses (five for relative pose- [119], and three for absolute pose estimation [89]). And second, feature points can be efficiently detected and matched using one of the numerous interest point detectors and descriptors that are available (e.g. *SIFT* [101]). However, while point-based *SfM* pipelines have been shown to work well in numerous kinds of environments, they heavily rely on the existence of distinctive texture in the scene. While this is usually no problem in outdoor environments (especially when dealing with aerial images), this can be problematic in indoor- and monotonic urban environments. And even if the *SfM* pipeline manages to create enough feature matches to successfully compute the correct camera poses, the resulting 3D model is usually very sparse and quite meaningless (for more information about point-based *SfM* please see Section 2.1).

To overcome this drawback, one could use different kinds of image features which are more suited for such untextured environments. A popular choice are straight line segments, which occur frequently in man-made environments. However, while line features can provide a much better sparse 3D representation of an almost textureless urban scene, they are not as convenient to use as point features. This is due to the fact that matched lines between two images do not put enough constraints on the fundamental matrix such that relative pose estimation could be performed. Therefore, one always needs to consider at least three views at a time to compute the relative camera poses for the general case

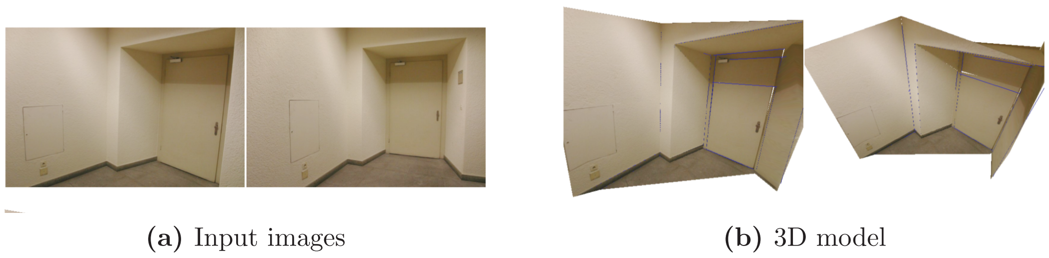
[71].

Yen and Huang [174] and Liu and Huang [99] were among the first to propose line-based *SfM* for the task of estimating the motion of a rigid body observed by a static camera in three frames (which is virtually the same as a rigid scene and a moving camera, given that the camera only observes the one moving object, or multiple objects with the same motion). In the following years, several methods have been proposed based on different principles such as Kalman Filters [140, 158], a unified statistical framework [107], a globally visible reference plane [130], affine camera models [78, 127], matrix factorization [105, 151, 155], solving linear equation systems [52, 56, 57, 159, 165], or non-linear optimization of a global objective function [12, 13, 112, 153, 183].

What all these methods have in common is that they do not consider the task of matching line segments across images, and assume that all matches are already given. Hence, they omit one of the most challenging and time consuming steps of *SfM*: the feature matching. While several line matching algorithms were already introduced at the time of the aforementioned methods (see Section 2.2), full line-based *SfM* pipelines which include the matching process were surprisingly rare.

Bartoli et al. [11] proposed to use a combination of line segments and feature points, so called Pencil-of-Points (PoPs). They basically attach interest points to adjacent 2D line segments to form a *PoP*, and match them based on cross-correlation scores between their corresponding feature points. Hence, they break down line matching to point matching, which in addition gives them point-wise correspondences between matched lines. They have shown that the epipolar geometry between two images can be derived from just three pairs of matched *PoPs*, which is beneficial especially for hypothesize and test frameworks, such as Random Sample Consesus (RANSAC) [45]. However, their method requires reliable feature points within close proximity of candidate lines, which is often not the case in untextured environments, and when the lines in question correspond to depth discontinuities.

Bay et al. [16] proposed a stereo *SfM* method for two uncalibrated images, where they utilize their own histogram-based line matching method [17] to compute pairwise line correspondences. They continue by finding junctions between potentially coplanar lines, and segment the images into planar regions using a *Binary Partitioning Tree*. They estimate the fundamental matrix using corresponding junctions, which are of course valid point-wise correspondences given that the underlying lines are actually coplanar. Finally, they derive a piecewise planar 3D model of the scene using the detected planar image regions, and their matched borderlines (see Figure 2.7). While their approach is able to compute impressive dense 3D models especially for indoor scenes, it is not straightforwardly extendible to more than two images. In addition, the underlying color-based line-descriptor [17] is not robust to large illumination changes, which makes an outdoor use very challenging. A similar approach was presented by Kim and Lee [84], where they also utilize the intersections of coplanar lines to compute the epipolar geometry. The difference is that they do not match the line segments themselves, but rather their intersection points using



(a) Input images **(b)** 3D model

Figure 2.7: Illustrations taken from [16]. (a) Two uncalibrated input images of an untextured indoor scene. (b) The resulting piecewise planar 3D model obtained from matched line segments, and automatically detected planar image regions using line junctions.

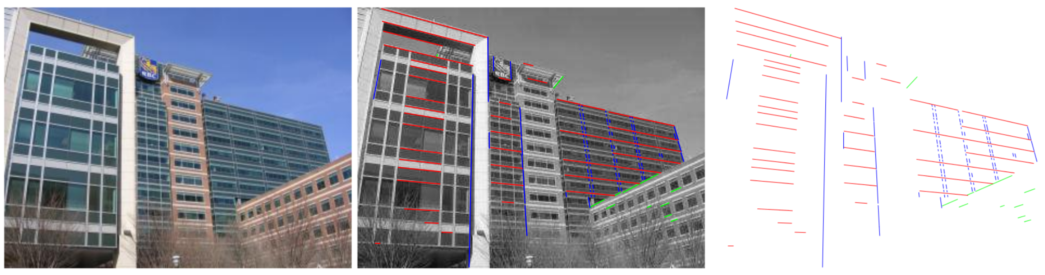


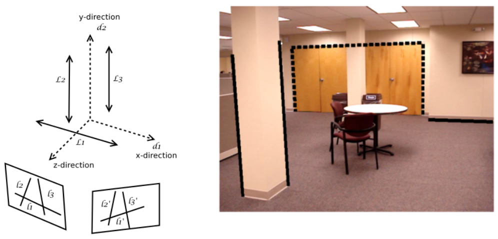
Figure 2.8: Illustrations taken from [132]. *Left:* An example input image from an urban scene. *Middle:* Detected 2D line segments color coded with their assigned vanishing direction. *Right:* The reconstructed 3D model rendered from the same perspective as the input image.

an *NCC* score [82]. However, they do not compute dense piecewise planar 3D models at the end, as it is done in [16].

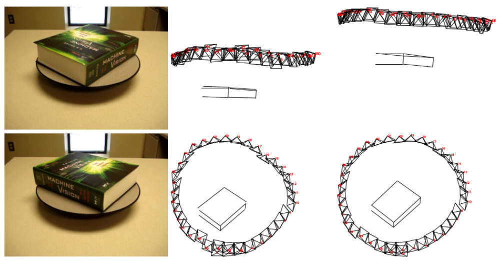
Schindler et al. [132] presented a line-based *SfM* method specifically designed for Manhattan-world like scenarios. They cluster image lines with common vanishing directions, and use this information to derive camera motion and 3D lines very conveniently (given that ≥ 3 distinct vanishing directions are present). However, while in the vast majority of urban scenes at least three different vanishing directions should be observable, the underlying strong assumptions somewhat limit the general applicability (i.e. lines that could not be assigned to one of the detected vanishing directions are not reconstructed). Figure 2.8 shows some results obtained by their algorithm. A related method was presented by Kim and Manduchi [81] for smartphone applications. Here they make use of the phones' Inertial Measurement Unit (IMU) for rotation estimation, and compute three dominant and mutually orthogonal planes from matched lines (matched with the *MSLD* line descriptor [163]).

Elqursh and Elgammal [37] introduced a line-based *SfM* framework, which allows relative pose estimation between just two images rather than three. They achieve this by considering triplets of 2D lines, where two of the lines are parallel and one is orthogonal to both (parallelism and orthogonality refer to the pre-images of the 2D lines in 3D; see

Figure 2.9a). They decouple the relative pose estimation into two separate steps. First, they solve for the rotation using matched line triplets, and second, they obtain the relative translation from two matched line intersection points. Their approach is designed for image sequences with a small relative motion between consecutive frames (see Figure 2.9b). While the possibility of determining the relative camera motion between two images using lines instead of points is interesting and potentially useful, finding suitable line triplets and producing reliable line matches at the same time can be challenging, especially for larger baselines.



(a) Line triplets

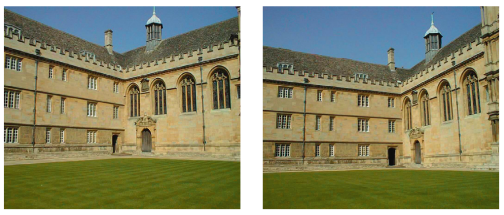


(b) Final 3D model and camera trajectory

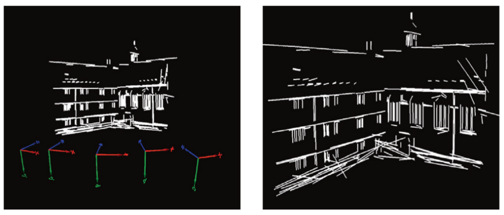
Figure 2.9: Illustrations taken from [37]. (a) The general geometric configuration of a suitable line triplet in 3D (*left*), and two real-world examples visualized with dashed- and solid black lines (*right*). (b) An example *SfM* result (3D lines and camera poses) of a small image sequence around a book.

Zhang and Koch [180] proposed a very sophisticated line-based *SfM* pipeline, which is centered around their *LBD* line descriptor [179] for matching, and their Perspective-n-Line (PnL) [181] algorithm for absolute pose estimation. They initialize the reconstruction from matched lines across three images in closed from, and incrementally add new images by establishing 3D-2D line correspondences, which can then be used to compute the absolute camera pose [181]. As a core novelty, they introduce the *Cayley* 3D line representation, which can be efficiently derived from Plücker coordinates through the Cayley transform [90]. This representation allows to encode 3D lines using the minimum of four parameters, without unavoidable singularities. This significantly eases the optimization of the 3D line parameters during bundle adjustment. Figure 2.10 shows an exemplary reconstruction result using their proposed method.

Micusik and Wildenauer [110] proposed a similar approach, which is also built on an initial reconstruction from three images, and a sequential alignment of new images. However, their matching procedure is fundamentally different. They propose to compute *SIFT* [101] features on the endpoints of detected 2D line segments, and use these feature points for matching. This of course only works for small camera motions, such that strong deviations in the line segment endpoints from frame to frame are less likely. To get the camera poses, they follow [37] for rotation estimation and compute the translation afterwards using a linear formulation. They have shown that their approach has a simi-



(a) Exemplary input images

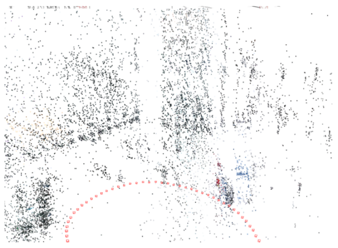


(b) Final 3D reconstruction

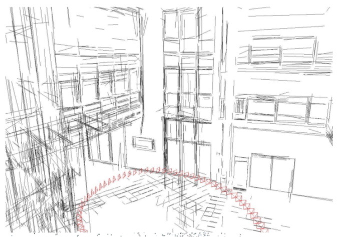
Figure 2.10: Illustrations taken from [180]. (a) Two exemplary images from the Wadham College dataset[a]. (b) The final reconstruction result rendered from two perspectives, with (*left*) and without (*right*) camera poses.

[a]`http://www.robots.ox.ac.uk/~vgg/data/data-mview.html`

(a) Exemplary image



(b) Point-based *SfM*



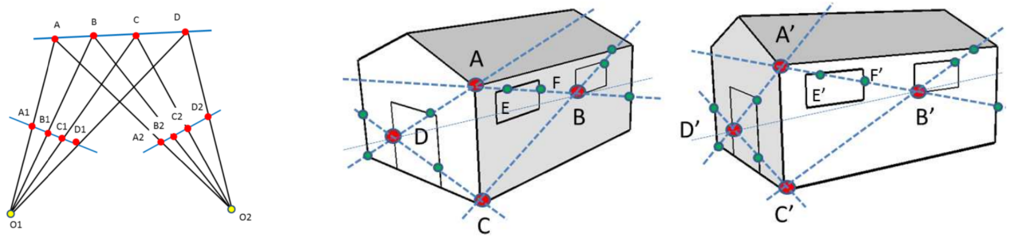
(c) Line-based *SfM*

Figure 2.11: Illustrations taken from [110]. (a) An exemplary image from the sequence used for the reconstruction (fisheye lens). (b) A classic point-based *SfM* reconstruction. (c) The result obtained using their proposed line-based 3D reconstruction pipeline. As we can see, the line model provides much more semantic information about the scene structure, despite being also sparse.

lar performance than traditional point-based *SfM*, but delivers a much more meaningful sparse 3D model (see Figure 2.11). However, their documented reconstructions appear to be relatively noisy (see experiments in [110]).

Recently, Ramalingam et al. [129] proposed an interesting hybrid approach for point- and line features. They create virtual lines between coplanar feature points, and intersect these virtual lines with actually detected 2D line segments in the images. Given that the virtual- and the actual line are coplanar, the newly generated intersection point is a well defined and invariant feature point, that can be used for pose estimation and to densify the sparse 3D point model. Using the well defined *Cross-Ratio* [55] constraint (see Figure 2.12), the intersection points (as well as the line segments) can be matched between images, which significantly improves the visual appearance of the resulting 3D model. However, their approach is strictly seen not a real line-based *SfM* pipeline, it mainly utilizes lines to increase the number of interest points.

As we can see, a lot of different attempts at line-based *SfM* have been made over the years. While especially in recent years very promising and flexible methods have been



(a) Cross-Ratio **(b)** Applying Cross-Ratio to obtain new features

Figure 2.12: Illustrations taken from [129]. (a) A visualization of the perspective Cross-Ratio invariant, defined by four collinear points (see [55] and [129] for a detailed explanation). (b) Obtaining additional invariant feature points (*green*), by intersecting virtual lines (*dashed blue*) spanned by two coplanar interest points (*red*) with actual 2D lines (coplanar with the virtual ones). The Cross-Ratio constraint can be efficiently used to identify corresponding intersection points between two images, which subsequently leads to matched line segments and additional feature points.

introduced [110, 180], they are still not as general and out-of-the-box usable as classic point-based *SfM* pipelines. This is mainly due to the fact that line-based camera pose estimation is much more complicated than when using points, which is often compensated by using stronger constraints on the scene structure, or by assuming small camera motions (as mentioned above). In addition, processing unordered image sequences with potentially large baselines has not yet been demonstrated by any line-based *SfM* method, while point-based methods can handle such cases quite efficiently (e.g. by using descriptor vocabularies [58, 120]). However, there are scenarios in which point-based methods will fail due to the lack of texture, and the presented line-based *SfM* approaches provide a valuable set of tools to compensate this issue, given that the respective underlying assumptions can be applied.

In our approach we rely on given camera poses, most likely obtained by any traditional point-based *SfM* pipeline that is available. Hence, our work is a line-based *MVS* method, rather than line-based *SfM*. We have seen in our research that point-based *SfM* only rarely fails to obtain the correct camera poses, which in turn significantly eases the task of line-based 3D reconstruction. Since point-based *SfM* is widely spread in the Computer Vision community, and also widely used in the industry, we believe that having a robust method at hand that can efficiently post-process and enhance an *SfM* result with 3D lines can be very valuable for numerous tasks.

2.4 Line-based SLAM

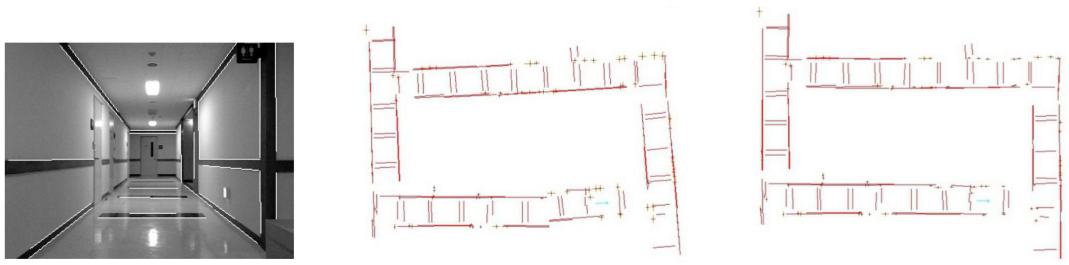
Simultaneous Localization and Mapping (SLAM) is a common name for a system which enables a device (e.g. a robot, a drone, or simply a camera) to localise itself within an (potentially) unknown environment, while simultaneously creating or extending a map of its surroundings. Originally, *SLAM* systems where introduced by the robotics community to enable (semi-) autonomous robots to navigate through any given environment, by making use of various measurement devices, such as ultrasonic range finders [36, 164], laser range finders [102, 177], and of course also camera systems [31, 87]. Due to their high flexibility and their low cost and weight, cameras are particularly useful for *SLAM* systems, especially when dealing with Unmanned Aerial Vehicles (UAVs), where weight is a critical factor.

In our research, we did not focus on *SLAM* applications directly. However, we did investigate the online reconstruction of 3D lines within an online *SfM* framework [61, 65], which can be seen as a *SLAM*-like system (see Chapter 5). While arguably there is no real conceptional difference between online *SfM* and *SLAM* (both compute camera poses and 3D structure on-the-fly in quasi real-time), we follow the idea that *SLAM* is usually focused on fast navigation (with a low-resolution continuous video- or image stream), while online *SfM* is generally more focused on computing an accurate 3D model (most commonly with high-resolution still images, and potentially wider baselines). For the sake of completeness, we present a short overview of available line-based *SLAM* approaches in the remainder of this section.

The majority of all line-based *SLAM* systems are based on an Extended Kalman Filter (EKF), which is able to handle non-linear systems. Among these methods several different core principles are applied, such as the usage of vanishing point information [20, 46, 92, 125, 184], omnidirectional cameras [20], stereo cameras [7, 30], a mixture of point- and line features [74, 93, 142], Manhattan-world assumptions [178], undelayed line initialization [145], or special line configurations [37, 175, 176]. In addition, several model-based *SLAM* approaches have been presented [4, 50, 86]. However, since these methods assume a given 2D or 3D model of the environment to which they localize themselves, they are strictly speaking not real *SLAM* systems (since they do not map anything).

In addition to pure line-based *SLAM*, several methods have been presented which aim at combining line features with different types of landmarks within one consistent framework. As mentioned above for *EKF*-based *SLAM*, a popular choice are a combination of points and lines [74, 93, 142], which are known to highly complement each other, since feature points represent texture while lines represent structural outlines. Klein and Murray [88] showed how the integration of locally straight edgelets (which are basically straight line segments) into a point-based *SLAM* method significantly improves the agility of the system, which allows more rapid camera motions. However, essential for the success of combining different feature types is a suitable landmark parametrization. An extensive survey about this issue was published by Solà et al. [144], for monocular *EKF*-*SLAM*.

In contrast to monocular *SLAM*, several stereo methods using line features have been proposed as well (e.g. [7, 30]). The benefit of stereo *SLAM* is that just two different imaging positions ($\geq$ 4 images) are enough to compute accurate pose estimates from matched lines, which is in general not possible for the monocular two-view case (unless special line configurations are considered, as e.g. shown in [37]). In addition, a calibrated



(a) Exemplary input **(b)** W.o. loop closure **(c)** With loop closure

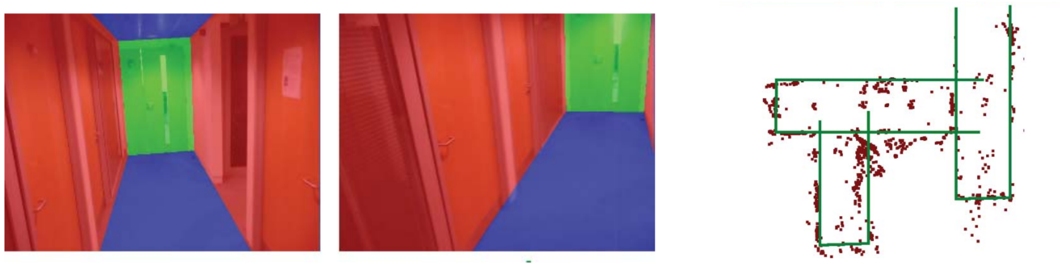
Figure 2.13: Illustrations taken from [175]. (a) An exemplary input frame with detected vertical- and floor lines (white). (b) Their original result without the explicit loop closure procedure [176] (top view). (c) Their improved results with loop closure through vanishing point information [175]. As can be seen, the reconstructed map is significantly more accurate and consistent when loop closure is performed.

stereo setup directly provides the correct metric scale for the reconstructed map, which is especially useful for robot navigation. Sophisticated line-based stereo *SLAM* methods were - among others - proposed by Chandraker et al. [25], and very recently by Holzmann et al. [67]. Both approaches show remarkable results, especially in low-texture environments.

What has been widely used throughput the literature, is the exploitation of structural properties of the surrounding environment. Since most line-based *SLAM* methods were motivated by indoor applications, the Manhattan-world- or similar assumptions are often deployed. Zhang et al. [176] proposed to use a combination of vertical- and floor lines within a monocular *EKF*-*SLAM* framework. They show how knowledge about the line orientations significantly eases both localization and mapping, as well as it reduces potential drift. They further extended their work to perform loop closure, by exploiting vanishing points [175] (see Figure 2.13). A similar method has been recently proposed by Zhou et al. [184]. Here, they make use of the so-called *building structure lines*, which are essentially horizontal and vertical lines on the ground and on the walls, following the Manhattan-world assumption.

Flint et al. [46] went one step further and showed how semantically meaningful 3D models can be extracted in real-time within a *SLAM* framework. They use the classic Parallel Tracking and Mapping (PTAM) approach by Klein and Murray [87] as a basis, and demonstrate how a semantically labelled piecewise planar 3D model can be derived from lines that follow dominant vanishing directions. As an example of the usefulness of such higher-order semantic information, Figure 2.14 illustrates how it can be used to derive floor plans of buildings fully automatically.

In recent years, *SLAM* approaches that use affordable depth sensors (e.g. the Microsoft Kinect) rather than ordinary cameras have become increasingly popular (e.g. KinectFusion [118]), especially for indoor environments. The core benefit is that these sensors provide not only an RGB image, but also a depth map corresponding to its pixels.



(a) Semantically labelled 3D model **(b)** Floor plan

Figure 2.14: Illustrations taken from [46]. (a) The reconstructed semantically labelled piecewise planar 3D model projected into two input frames. The different colors identify different structural planes in the scene (*red and green*: walls, *blue*: floor and ceiling). (b) The automatically extracted floor plan. The green lines are the identified wall structures, and the red dots are the 3D points reconstructed by *PTAM* [87] (top view). As can be seen, the points alone do not provide enough information to robustly derive the correct floor plan.

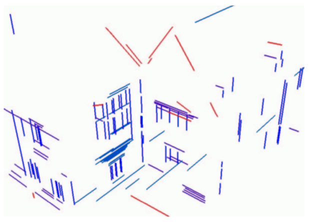
The underlying depth estimation method is usually based on active sensing technologies (e.g. structured light or time-of-flight), which allows depth computation also in very low-textured environments. A line-based *SLAM* approach based on KinectFusion [118] has been presented by Nakayama et al. [117], showing improved reconstruction results for man-made scenes and piecewise linear objects. However, the downside of most active depth sensors is that they only have a very limited depth range, and do not work well under direct sunlight.

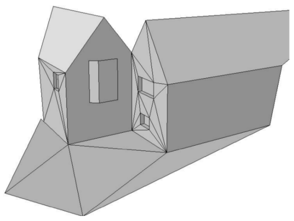
All in all, a lot of excellent work has been done in the field of line-based *SLAM* so far. Recently, the observable trend throughout the *SLAM* community is to move from feature-based methods to so-called direct *SLAM* methods, which means that no explicit features (neither points, nor lines) are computed, but consecutive images are aligned by minimizing a global photometric- or geometric error between them. Several examples for this paradigm shift can be found in the literature for monocular- [38, 152], or also stereo camera systems [39]. How this development affects the future of line-based *SLAM* remains to be seen.

2.5 Line-based Multi-View Stereo

The final category of the related work section is Line-based Multi-View Stereo (*MVS*). In contrast to Line-based *SfM* (Section 2.3), *MVS* algorithms assume given camera poses, and focus only on the reconstruction part. Since our method fits into this category, the approaches mentioned in the following paragraphs are more closely related to our work than the ones from the previous sections.

Among the available line-based *MVS* methods, a significant amount uses the extracted 3D lines as prior to create a piecewise planar 3D model of the scene, which can be done


(a) Reconstructed 3D lines


(b) Piecewise planar model

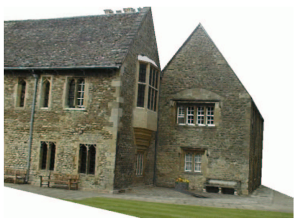

(c) Piecewise planar model (textured)

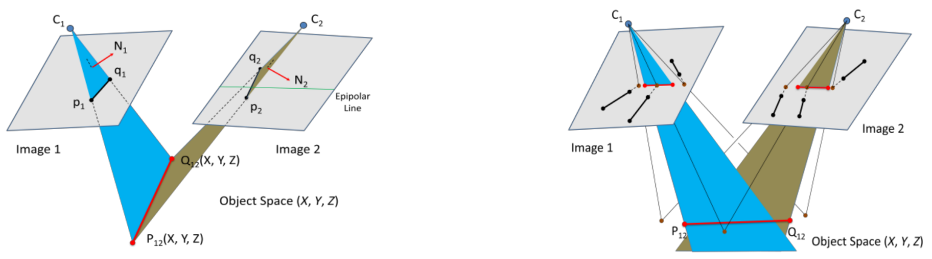
Figure 2.15: Illustrations taken from [166]. (a) The reconstructed 3D line segments by using [134]. (b-c) The derived piecewise planar 3D model with and without photo-realistic texture.

very reliably and efficiently for urban indoor and outdoor scenes. The benefit compared to point-based 3D plane reconstruction methods is, that two 3D lines are enough to create a plane hypothesis, while at least three points would be needed. Furthermore, it is much easier to reduce the set of potentially coplanar 3D features when lines are used, since only intersecting- or parallel lines need to be considered (whereas all 3D point triplets form valid 3D planes, unless the three points are collinear). Many of the presented algorithms focus on the 3D reconstruction of buildings from aerial images, by either using 2D line segments [9, 19, 85, 113, 121, 139], or general *edgels* [76] (i.e. chains of edge pixels) as image features.

In addition to these aerial methods, several more general approaches to piecewise planar 3D modeling through lines have been presented as well. Werner and Zisserman [166] proposed to use vanishing direction information obtained from 2D lines in the images, to find dominant planes in a scene. They show quite impressive results for non-trivial buildings, e.g. with oriels, protrusions, and complex façades (see Figure 2.15). Similar methods were presented by Schindler and Bauer [133], and Sinha et al. [141]. The core differences are that in [133] a dense 3D point-cloud is needed as well (in addition to the 3D lines), and in [141] the plane estimation is formulated as a pixel-wise labelling problem using a Markov Random Field (MRF). All three methods make use of the line matching approach by Schmid and Zisserman [134], to create a 3D line model.

Pure line-based *MVS* approaches (without creating a 3D plane model) have of course been proposed as well. Many of them focus on aerial sequences, to reconstruct outlines of buildings, roads, or other man-made structures. Woo et al. proposed several line-based *MVS* methods for this task, which either use Digital Elevation Models (DEMs) [169], or disparity maps [170, 171] in conjunction with aerial imagery. They only aim at reconstructing rooftop outlines, and discard all line features that correspond to other structures, or noise. Ok et al. [123] proposed a different approach to aerial line-based 3D reconstruction, where they separated the pairwise line matching and triangulation into two categories. The first category deals with 2D lines in a general configuration (Figure 2.16a), and the second one handles lines that are nearly aligned with the epipolar lines (Figure

2.16b). By treating the latter as a special case they manage to successfully reconstruct 3D lines which are approximately aligned with the camera motion, a case which is hard to handle for the majority of the related methods.



(a) General case **(b)** Epipolar-aligned case

Figure 2.16: Illustrations taken from [123]. (a) Matching line segments in a general configuration. The line in the target image (right) is not aligned with the epipolar line, which makes a robust matching and triangulation possible. (b) Matching line segments when the target segment (right) is aligned with the epipolar line. Here, an accurate intersection between the target line and the epipolar line is not possible, which makes triangulation almost impossible. The algorithm of Ok et al. [123] can handle such cases, in contrast to most related methods.

A different category of line-based *MVS* approaches deals with the problem of extracting 3D line models not just from an oriented image sequence, but requires a dense point-cloud of the scene as well. Chen and Wang [26] used *bundler* [143] and *PMVS* [49] to create a dense 3D point-cloud, which is then used to determine matches between 2D line segments from different images. A similar method was proposed by Fu et al. [48], where they improved the line matching by using spectral graph analysis of a pairwise matching graph for all 2D line segments. Figure 2.17 shows one of the impressive reconstruction results obtained by [26] . However, the need for a dense 3D point-cloud renders such methods quite inefficient, since obtaining it usually takes a lot of time and memory. A related approach was proposed by Lin et al. [98], with the difference that they use a point-cloud obtained with a Light Detection and Ranging (LiDAR) device.

All of the aforementioned approaches provide very reasonable results within their respective doamin (e.g. aerial image data), but are not as flexible and out-of-the-box usable as common point-based *MVS* methods (e.g. *PMVS* [49]). Most of them rely quite heavily on a set of core assumptions, that prevent a proper generalization to arbitrary scenes. Among the discussed approaches, methods that build on the line matching algorithm by Schmid and Zisserman [134] (which already showed basic line-based 3D models) are arguably the most versatile. They only rely on epipolar- and intensity-based matching constraints, and do not enforce further constraints on the line-based 3D model. However, matching lines using gray-value similarities is in general not very robust to illumination changes, and using more sophisticated line-descriptors does not work well for wiry structures (see Section 2.2). To overcome this drawback, several methods that do not rely on

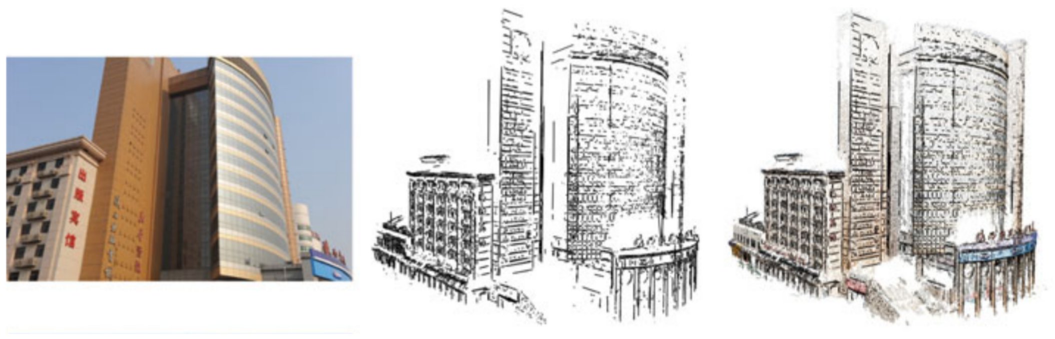


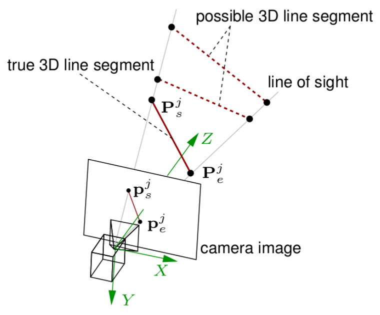
Figure 2.17: Illustrations taken from [26]. *Left:* An example image of their *Press Building* dataset (100 images, 2128 × 1416 px). *Middle:* Obtained 3D line model. *Right:* The 3D line model and the dense *PMVS* [49] point-cloud combined.

appearance constraints at all have been presented over the years.

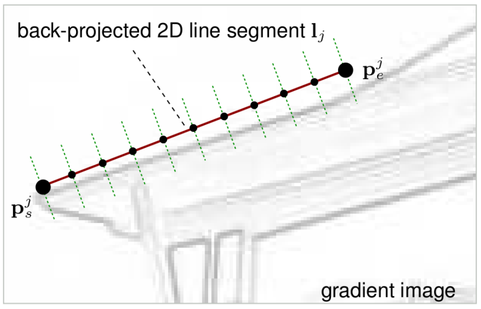
Heuel and Förstner [60] presented a purely geometric approach to line-based *MVS*, based on uncertain projective geometry. They created an extensive framework to match 2D line segments from multiple images, and optimally reconstruct 3D lines from these correspondences, which can further be grouped to 3D corners. They represent all geometric entities (points, lines, and planes) and their respective uncertainties in homogeneous coordinates (in 2D and 3D), and show how new entities can be easily reconstructed, hand in hand with their propagated uncertainties. However, sadly they only evaluated their approach on few aerial sequences, which do not look ultimately convincing.

The most relevant work for our research was presented by Jain et al. [73]. Their approach to line-based *MVS* is fundamentally different from all the methods discussed above, since they do not just omit appearance-based line matching, but line matching all together. What they do is relatively simple, they unproject all 2D line segments from all images to all possible spatial positions (within a certain range; see Figure 2.18a), and then reproject all of these spatial 3D line hypotheses into nearby images. They now compute an edge-based score for each reprojected 3D hypothesis, which simply analyzes how well the reprojection fits with the image gradients (see Figure 2.18b). This is a reasonable thing to do, since image gradients (or in extension also image edges) are used to detect 2D lines in images in any available line segment detector. Afterwards, they compute a 3D position for each 2D line segment, which is simply defined by the 3D hypothesis with the highest score. To verify their estimates, and to discard outliers, they perform a spatial clustering procedure, which fuses together 2D line segments with spatially close 3D positions (by using a cylindrical grouping radius; see Figure 2.19a). They reconstruct a final set of 3D lines by merging the selected 3D hypotheses of clustered 2D lines, and discard 2D segments that could not be clustered with at least one other segment. In addition, they also perform a loopy belief propagation procedure to enforce connections between 3D lines. A final reconstruction result using their method can be seen in Figure 2.19b.

The absence of an explicit line matching step (either appearance- or geometry-based)

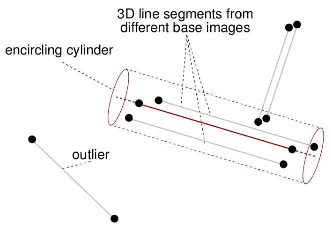


(a) Potential 3D locations

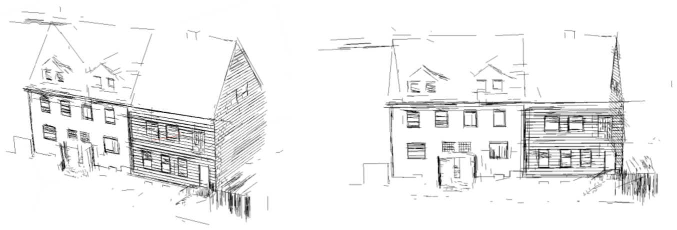


(b) Projective gradient scoring

Figure 2.18: Illustrations taken from [73]. (a) All potential 3D locations of the pre-image of the 2D line segment $l = \{p_s^j, p_e^j\}$ have to be located on the lines-of-sight of its two endpoints p_s^j and p_e^j. All potential 3D locations are equally likely, unless more images from different viewpoints exist to verify them. (b) All 3D hypotheses (locations) are backprojected into neighboring images, and scored based on their alignment with the image gradients. The better the alignment, the more likely a certain 3D hypothesis corresponds to the true location of the pre-image of the underlying 2D segment.



(a) Spatial line clustering



(b) Exemplary reconstruction result

Figure 2.19: Illustrations taken from [73]. (a) Clustering lines based on the spatial proximity of 3D line hypotheses, with a fixed cylindrical grouping radius. All segments which can not be clustered with at least one other segment are considered outliers and removed. (b) An exemplary reconstruction result of their *Street* [a] sequence (20 images).

[a]`http://resources.mpi-inf.mpg.de/LineReconstruction/`

allows [73] to reconstruct scenes of an arbitrary complexity, e.g. including wiry objects or high illumination changes. However, the core problem of their work is that the computational complexity is incredibly high. In their paper they do not state the actual runtime, but they mention that their approach needs several hours even for small-scale datasets. This is mostly due to the very expensive 3D hypothesis estimation step, where *all* possible 3D locations for every 2D line segment have to be analyzed by backprojection and gradient scoring. Given the potentially very large depth range in which valid 3D lines might occur in front of the cameras, their is no straightforward way to reduce the computational complexity, unless some prior knowledge about the scene structure is available.

To overcome this issue, we proposed to combine their projective 3D hypothesis scoring with purely geometric line matching [66]. The core idea is that it is not necessary to analyze all potential 3D locations for each 2D segment, but rather a limited set based on the information that is provided by nearby images. We therefore compute a set of line matches for each 2D segment in its neighboring images, based solely on weak epipolar matching constraints. We then triangulate all these matches, and only backproject and score this discrete and limited set of 3D locations. We show how this simple procedure can significantly boost the performance of the overall reconstruction algorithm, without negatively influencing the completeness or the accuracy of the resulting 3D models. The insights gained from these initial experiments have formed the basis of our subsequent research in line-based 3D reconstruction, where we have successfully increased the performance and the accuracy of our method even more [62, 63]. In addition, we have shown how our principles can be used within online *SfM* frameworks, for the robust and fast on-the-fly reconstruction of 3D line models [61, 65] (see Chapter 5).

2.6 Summary

We have presented an extensive overview of all closely related fields in the area of line-based 3D vision, divided into the four categories Line Matching, Line-based *SfM*, Line-based *SLAM*, and Line-based *MVS*. While arguably there are more applications for lines in 3D vision (e.g. camera calibration [23, 34]), these are the ones that are directly related to our work, and have influenced it the most.

Throughout this chapter, we tried to emphasize why we do not use classic appearance-based line matching, and why we use given camera poses from traditional *SfM* pipelines, rather than performing full line-based *SfM*. First, we want to be able to reconstruct wiry objects (e.g. power pylons), which is virtually impossible when state-of-the-art patch-based line descriptors are used. And second, line-based *SfM* is by far more challenging than point-based *SfM*, given that either special requirements need to be met such that relative poses can be computed from matched line segments across just two images, or matches across at least three images need to be considered at al times. This becomes even more severe given that it is in many cases not trivial to come up with a sufficiently large set of correct matches in the first place, e.g. when dealing with wiry objects, as mentioned above. Hence, we decided to rely on traditional *SfM* pipelines to deliver the correct camera poses, and focus more on the reconstruction part. This enables our work to be out-of-the-box usable with most available *SfM* pipelines, which are frequently used by a very large community (both in academia and industry). In the following chapter, we give an extensive in-depth overview of our proposed line-based 3D reconstruction pipeline, and describe all necessary processing steps in detail.

3 Line3D++: A Line-based 3D Reconstruction Framework

Contents

3.1 Prerequisites 33
3.2 Pipeline Overview 40
3.3 Line Segment Detection 41
3.4 Establishing Line Segment Correspondences 43
3.5 Evaluating Line Segment Correspondences 49
3.6 Assigning 3D Locations to 2D Segments 53
3.7 Clustering Corresponding 2D Segments 55
3.8 Combined Bundle Adjustment 57
3.9 Summary 59

In this chapter we introduce our line-based 3D reconstruction framework, and explain all underlying processing steps in detail. For a better visual understanding of our pipeline, we show graphical illustrations of the various processing steps on the example of our *Building* and *Pylon* datasets, introduced in Figure 1.4. All following steps are part of our open-source implementation *Line3D++*, which is available online [1]. All parameters that are introduced throughout the algorithmic explanations are kept abstract, and their default values as well as their influence on the reconstruction results are discussed during the evaluations in Chapter 4.

The algorithm introduced in this chapter is based on several of our earlier publications on this topic [62, 63, 66], and has been recently published in compressed form in [64].

3.1 Prerequisites

The input to our algorithm is always an oriented (and not necessarily ordered) set of calibrated images $I = \{I_1, \ldots, I_N\}$, i.e. images with given intrinsic- (K) and extrinsic

[1] `https://github.com/manhofer/Line3Dpp`

(R, t) camera parameters (see Section 3.1.1 for an explanation), as well as their distortion coefficients (if necessary). The extrinsics can be easily obtained by any given Structure-from-Motion (SfM) pipeline available, e.g. the freeware tools VisualSfM [172] or Open-MVG [114], and the intrinsics and distortion coefficients can be obtained by any camera calibration toolbox (e.g. [29]). Alternatively, all this information (extrinsics, intrinsics, and distortion coefficients) might potentially be estimated by the *SfM* application alone, without prior knowledge (depending on what pipeline is used).

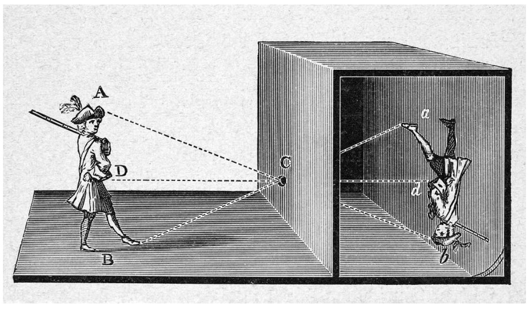
Since our method only needs the images and the corresponding camera poses, one could technically also use an external tracking system to compute these poses. However, in the remainder of this thesis we always assume without loss of generality that an *SfM* pipeline has been run beforehand. This means that we assume to also have a sparse set of 3D points $X = \{\mathcal{X}_1, \ldots, \mathcal{X}_T\}$, which we will need later to define which images are *visual neighbors* (i.e. see the same thing). We further define $X_i \subset X$ to be the set of 3D points which are visible in image I_i, where *visible* means that they have a 2D residual in this particular image. For a better understanding of our algorithm, we explain the underlying camera model and its parameters, as well as the principle of epipolar geometry in the following sections.

3.1.1 Camera Model

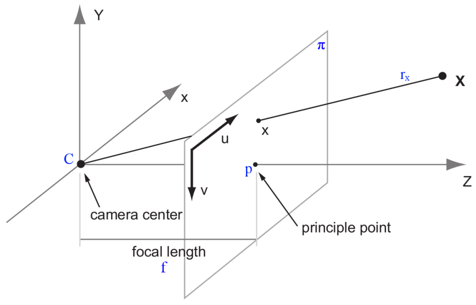
To understand the relation between a 3D entity (e.g. a point) and its 2D projection in an image, we need to analyze the camera that was used to create this image. Even though each physical camera is build from slightly different components, the underlying *camera model* (i.e. the abstract principal behind the actual camera) is usually quite similar. In this thesis, we focus on a very simple model, called a *pinhole camera*, which is frequently used to model common compact-, Digital Single-Lens Reflex (DSLR), or also smartphone cameras in 3D reconstruction applications. However, there exist of course also other camera models, such as e.g. affine-, or non-central cameras.

The basic principle of a pinhole camera can probably be best explained by the so called *camera obscura* (Latin for *dark room*), which was already familiar to the Greek philosopher Aristotle more than 300 years BC. It is basically a dark room (or a box) where a small hole is placed in one of the walls (hence, the term *pinhole*). Light from an external light source, that is reflected from objects outside of the box, now travels through the hole, and projects an image of these objects onto the back surface (opposite of the hole). The projected image is upside down, but color as well as perspective is preserved. Figure 3.1a shows an illustration of this principle.

Geometrically speaking, a pinhole camera consists of a center of projection C, an image plane π, and the focal length f. The center of projection (also called *camera center*) is basically the pinhole itself, while the image plane π is the equivalent to the wall of the camera obscura on which the image is projected. The focal length f is simply the distance between the camera center and the image plane, which basically defines how large the projected image will appear. Figure 3.1b shows a graphical illustration of the relationship between these three components (for more detailed explanations we refer to [55]).



(a) The *camera obscura*



(b) The pinhole camera model

Figure 3.1: An illustration of the pinhole camera model. (a) The camera obscura, which is an early version of an actual physical camera that directly follows the pinhole camera model [a]. (b) The geometric principles behind the pinhole camera model (image taken from [68]).

[a]illustration taken from `https://prezi.com/4y2ggpyifukg/camera-obscura/`

In this example, the image plane π is placed in front of the camera center C, which means that the resulting images are not upside down. Please note that this is only a theoretical model, which could not be build exactly like this. However, the 3D-to-2D projection principles are just as valid for this scenario. In addition to the camera center (C), the focal length (f), and the image plane (π), the illustration also shows a so-called principle point ($p = [p_u, p_v]^T$), and a coordinate frame (X, Y, and Z). The coordinate frame defines the local camera coordinate system, i.e. the 3D world as seen from the cameras' viewpoint. It has its origin at the camera center C, with the Z axis (the *optical axis*) going into the scene. The 2D point in which the optical axis intersects π is the principle point.

Now, if a 3D point $\mathcal{X} = [X, Y, Z]^T \in \mathbb{R}^3$ is defined in the local camera coordinate system, its projection $x = [u, v]^T \in \mathbb{R}^2$ in the image π can be written as

$$x = \begin{pmatrix} u \\ v \end{pmatrix} = f \begin{pmatrix} X/Z \\ Y/Z \end{pmatrix}, \tag{3.1}$$

which basically means that for a point in space with given X and Y coordinates, its position in the image depends linearly on its depth (i.e. its Z coordinate), and the focal length f.

This projection of course only works if the 3D point is defined in the local camera coordinate system. However, in a multi-view scenario with multiple cameras, each 3D point should only have *one* globally consistent 3D coordinate triplet, and not one per observing camera. Hence, we need to define a global coordinate system for our reconstructed (or observed) 3D world. To easily define the projection of globally defined 3D points into an arbitrary 2D image plane, we first switch to the *projective space* by making use of homogeneous coordinates. Hence, our 3D point $\tilde{\mathcal{X}} = [X, Y, Z, 1]^T \in \mathbb{P}^3$ has now four coordinates, and the corresponding 2D point $\tilde{x} = [u', v', w]^T \in \mathbb{P}^2$ has three (for an easier understanding, $\tilde{\cdot}$ symbolizes homogeneous coordinates).

The projection of $\tilde{\mathcal{X}}$ into π can now be rewritten as

$$\tilde{x} = \begin{pmatrix} u' \\ v' \\ w \end{pmatrix} = \begin{pmatrix} f_x & 0 & p_u \\ 0 & f_y & p_v \\ 0 & 0 & 1 \end{pmatrix} [R_{3\times3}|t_{3\times1}] \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = K[R_{3\times3}|t_{3\times1}]\tilde{\mathcal{X}}, \tag{3.2}$$

where K holds the *intrinsic* parameters of the camera (i.e. its internal characteristics), and R (rotation) and t (translation) are the *external* camera parameters. The external parameters (often also called *extrinsics*) basically describe the transformation from the global coordinate system to the camera coordinate systems. In addition, instead of having only one focal length f in the intrinsics K, we now have two potentially different values f_x and f_y (in x- and y direction). This allows a more realistic modelling of physical cameras, and generally produces more accurate measurements. The camera center C itself can be expressed in global (*world*) coordinates as

$$C = -R^T t, \tag{3.3}$$

where $R^T = R^{-1}$, since it is a rotation matrix.

The 3D-to-2D projection can be seen as a dual process, where

$$\mathcal{X}_C = [R|t]\tilde{\mathcal{X}} \tag{3.4}$$

first transforms a 3D point $\mathcal{X}$ from global- to local camera coordinates (still in 3D), and

$$\tilde{x} = K\mathcal{X}_C \tag{3.5}$$

finally projects the local 3D point into the image plane. The whole projection process is often expressed in terms of one single matrix-vector multiplication

$$\tilde{x} = P\tilde{\mathcal{X}} = K[R|t]\tilde{\mathcal{X}}, \tag{3.6}$$

where $P = K[R|t]$ is the 3×4 *projection matrix.* To *de-homogenize* $\tilde{x}$ (i.e. to transform it back to $\mathbb{R}^2$), we can simply use

$$x = \begin{pmatrix} u'/w \\ v'/w \end{pmatrix}, \tag{3.7}$$

which basically means that the first two coordinates of $\tilde{x}$ are divided by the third.

The reverse process, which is the unprojection of a 2D image point $\tilde{x} = [u, v, 1]^T \in \mathbb{P}^2$ into 3D space, is of course defined as well. However, during the 3D-to-2D projection (i.e. the image acquisition) all depth information inevitably gets lost by design. Hence, we can only unproject 2D points as infinite 3D rays, on which the real 3D entity that created this 2D image point must be located. For the point x in Figure 3.1b, this would be the ray $r_x \in \mathbb{R}^3$, passing through the camera center C, the image point x itself, and its true 3D pre-image $\mathcal{X}$. It can be simply computed as

$$r_x = R^T K^{-1} \tilde{x}, \tag{3.8}$$

and can then be used to reconstruct the actual 3D point

$$\mathcal{X} = C + d \cdot r_x, \tag{3.9}$$

given that its depth $d = \|C - \mathcal{X}\|_2$ along the ray is known, and r_x is normalized to unit length ($\|r_x\| = 1$).

What is missing to actually use this camera model for *SfM* applications from consumer cameras, is the modelling of potential image distortion. Especially wide angle lenses do not strictly follow the pinhole camera model, but introduce a certain amount of radial- as well as tangential distortion, especially further away from the image center. However, this issue can be easily fixed by calibrating the camera first, which means to estimate its intrinsics and its distortion coefficients by following a defined calibration procedure, which usually involves taking pictures of a planar calibration target from multiple view points. An overview of common calibration strategies, as well as a freely available calibration tool, can be found in [29]. For the rest of this work, we assume that the calibration coefficients are available (either by calibration, or estimated by the *SfM* pipeline), which allows us to undistort the images such that the pinhole camera model can be used directly.

3.1.2 Epipolar Geometry

Given our input image set I, the underlying *SfM* pipeline provides us with intrinsics K_i, as well as extrinsics R_i and t_i, for each image $I_i \in I$ (and in extension, of course also $P_i = K_i[R_i|t_i]$). This information fully defines the spatial camera configuration of this multi-view scenario, as well as the sparse 3D point-cloud (given that feature matches are available, which can then be straightforwardly triangulated to 3D points).

However, even if all this information is available there is no explicit direct mapping between all corresponding pixels in pairs of images I_i and I_j, i.e. in form of a globally coherent transformation matrix $T_{i,j}$, that fully maps one image onto the other. That means, given a pair of corresponding 2D points $\tilde{x_i} = P_i\mathcal{X}$ and $\tilde{x_j} = P_j\mathcal{X}$ in π_i and π_j (which are the images of the same 3D point $\mathcal{X}$), a well defined relationship such as

$$\tilde{x_j} = T_{i,j}\tilde{x_i} \quad \text{and} \quad \tilde{x_i} = T_{i,j}^{-1}\tilde{x_j}, \tag{3.10}$$

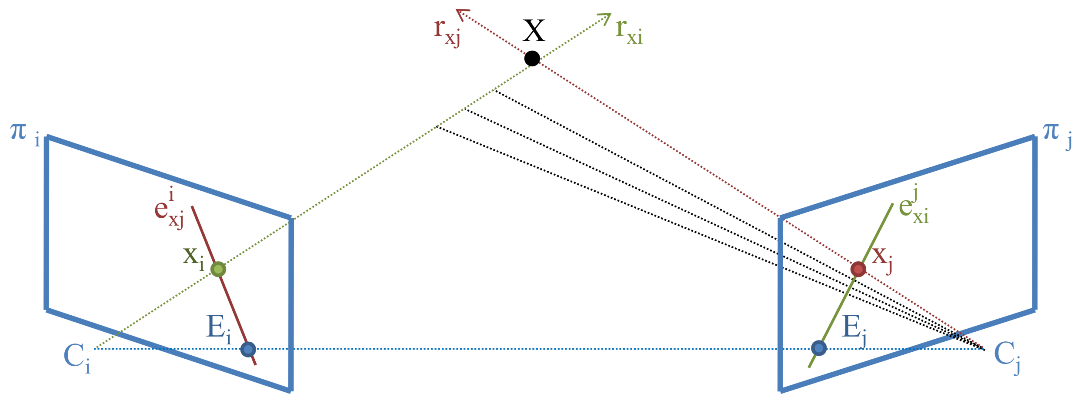


Figure 3.2: An illustration of the epipolar geometry between two images I_i and I_j. The 3D point $\mathcal{X}$ and the two camera centers C_i and C_j span a triangle, whose lines of intersection with the two image planes π_i and π_j are the two epipolar lines $e^j_{x_i}$ and $e^i_{x_j}$.

that maps every pixel from one image to its correct correspondence in the second image, does not exist. This is true for the general case, but there are some special scenarios for which a global transformation does indeed exist (see [55] for an introduction on *homographies*). The reason why such a transformation is not possible, is because of the loss of the depth information when the 3D world is mapped to a 2D image. Only for those 2D points for which depth information is available (i.e. the 2D feature points that have been successfully matched and reconstructed to a 3D point $\mathcal{X}$) a direct mapping between images is possible, since $\mathcal{X}$ can be projected into any image, given that the corresponding projection matrix is available.

As we have discussed above, each 2D point $\tilde{x}_i$ can be at least unprojected as a 3D ray

$$r_{x_i} = R_i^T K_i^{-1} \tilde{x}_i, \tag{3.11}$$

even if its depth is not known. Since we could technically generate an infinite number of 3D points

$$\mathcal{X}_{x_i}(d) = C_i + d \cdot r_{x_i} \tag{3.12}$$

along this ray (for all possible depths d), we could project all these points into another image I_j. The geometric structure that would emerge in π_j is again a straight line, and more specifically, the 2D image of the infinite ray r_{x_i} in I_j. This resulting 2D line is called an *epipolar line*, denoted as $e^j_{x_i}$. Figure 3.2 illustrates the geometric principle of the so-called *epipolar geometry* graphically.

As we can see, the 3D point $\mathcal{X}$ and the two camera centers C_i and C_j span a triangle. This triangle intersects the two image planes π_i and π_j in two straight lines, which are the epipolar lines $e^j_{x_i}$ (in I_j) and $e^i_{x_j}$ (in I_i). These lines coincide with the projections of r_{x_i} in image I_j, and r_{x_j} in image I_i respectively. As we can also see, all potential unprojections $\mathcal{X}_{x_i}(d)$ of x_i (with different depths d; visualized as black dashed lines) would be projected exactly onto the epipolar line $e^j_{x_i}$ in π_j. In addition, the intersections of the line between C_i and C_j with the image planes are the two *epipoles* E_i and E_j. These epipoles are not depending on any 3D points, but only on the relative spatial positions between the two cameras.

The epipolar geometry between two cameras can be efficiently encoded in a 3×3 matrix, which is called the *essential matrix* ($E_{i,j}$) for the calibrated-, and the *fundamental matrix* ($F_{i,j}$) for the uncalibrated case [55]. Both matrices can be estimated from point correspondences between the two images, where in general eight correspondences are needed to derive the essential-, and seven to derive the fundamental matrix. However, several methods exist that have shown that less correspondences are also sufficient in many cases (e.g. the five-point relative pose problem [119]). In the rest of this section we will only consider the fundamental matrix $F_{i,j}$. The relationship between $E_{i,j}$ and $F_{i,j}$ is defined as

$$E_{i,j} = K_j^T F_{i,j} K_i, \tag{3.13}$$

which means that $E_{i,j}$ can always be derived from $F_{i,j}$, as long as the intrinsics of the two cameras (K_i and K_j) are known.

Given $F_{i,j}$ for an image pair I_i and I_j, the epipolar line $e^j_{x_i}$ in I_j of a 2D point x_i in I_i can be simply computed as

$$e^j_{x_i} = F_{i,j}\tilde{x}_i, \tag{3.14}$$

by multiplying the fundamental matrix with the homogenized point coordinates $\tilde{x}_i$. The same holds for the opposite direction, where

$$e^i_{x_j} = F_{j,i}\tilde{x_j} = F^T_{i,j}\tilde{x}_j, \tag{3.15}$$

and $F_{j,i}$ being equivalent to $F^T_{i,j}$. Hence, the fundamental matrix is not bidirectional, but once $F_{i,j}$ is known, $F_{j,i}$ follows straightforward.

When we again have a look at Figure 3.2, we see that the projection x_j of $\mathcal{X}$ in I_j has to lie on the epipolar line $e^j_{x_i}$ defined by its counterpart x_i in I_i (and vice versa). This is the case because $e^j_{x_i}$ is the projection of the ray r_{x_i} on which $\mathcal{X}$ must be located (otherwise x_i could not be an image of $\mathcal{X}$, given the underlying camera model). This means that if we only have x_i and $F_{i,j}$, and we do not know the corresponding point x_j in the other image, we only have to look for a correspondence along the epipolar line $e^j_{x_i}$, which effectively reduces the correspondence search (also commonly known as the *correspondence problem*) from a 2D to a 1D problem. This is arguably the most important property of the epipolar geometry in the context of image-based 3D reconstruction, and has been widely used especially in multi-view stereo algorithms (e.g. [49, 131]). As a consequence, it also follows that

$$\tilde{x_j}^T F_{i,j}\tilde{x_i} = 0, \tag{3.16}$$

for all corresponding point pairs x_i and x_j. However, please note that Equation 3.16 is

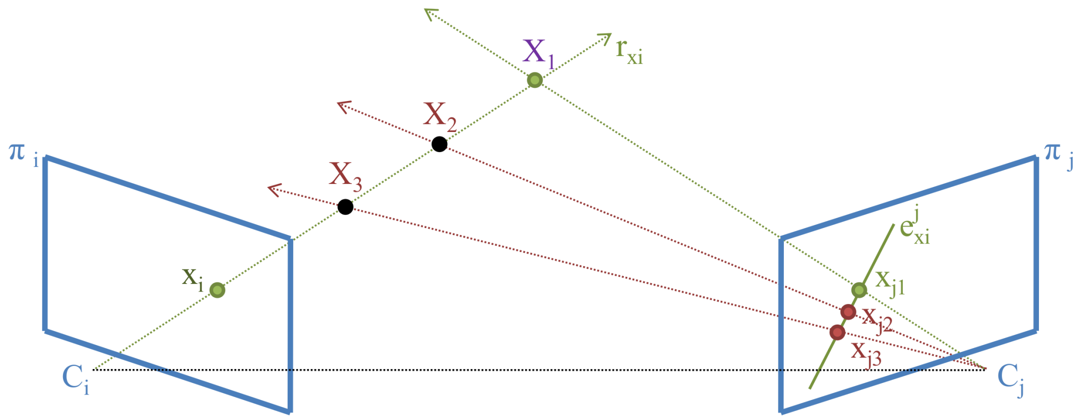


Figure 3.3: An illustration on how different 3D points $\mathcal{X}_1$, $\mathcal{X}_2$, and $\mathcal{X}_3$ that project to x_i in π_i, all fulfill the basic epipolar constraint (Equation 3.16), since their projections x_{j1}, x_{j2}, and x_{j3} (in π_j) all lie on the epipolar line $e^j_{x_i}$ of x_i.

fulfilled for a certain point x_i and *all* other points $z \in \pi_j$, which happen to lie on the epipolar line $e^j_{x_i}$ (even if they are not really corresponding to the same 3D entity!). Figure 3.3 illustrates this issue. As we can see, all three potential correspondences (x_i, x_{jk}) (with $k \in \{1, 2, 3\}$) fulfill the epipolar constraint in Equation 3.16, despite the fact that only one of these correspondences is correct (in terms that both 2D points are images of the same physical 3D entity).

With a proper camera model and the concept of the epipolar geometry at hand, we have now all the tools we need to proceed to our actual method, which aims at using the (undistorted) input images and the corresponding camera parameters to generate line-based 3D models in an efficient and robust way. In the remainder of this chapter we will explain all the necessary processing steps in full detail.

3.2 Pipeline Overview

Our line-based 3D reconstruction method consists of several steps:

1. **Line Segment Detection** (Section 3.3)
   An out-of-the-box line segment detector is used to detect 2D line segments in all available images. These image features are the basis of our 3D reconstruction approach.

2. **Establishing Line Segment Correspondences** (Section 3.4)
   The detected 2D segments are matched between related images, i.e. images that see the same part of the scene. Here, we focus on maximizing the recall by employing a purely geometric one-to-many matching procedure, so that ideally no correct matches are missed even when severe illumination changes or occlusions are present.

3. **Evaluating Line Segment Correspondences** (Section 3.5)
   We triangulate all matches to 3D line hypotheses, and evaluate them based on their mutual support across several different images in a fast local scoring procedure (directly in 3D space).

4. **Assigning 3D Locations to 2D Segments** (Section 3.6)
   We use the obtained scores from the previous step to remove matches that are most likely incorrect, and to assign an initial 3D location to each 2D segment individually, based on its triangulated match with the highest score.

5. **Clustering Corresponding 2D Segments** (Section 3.7)
   In the final reconstruction step, we compute a global sparse affinity matrix for all 2D segments across all images, based on the spatial proximity of their estimated 3D locations. We then cluster corresponding 2D segments using an efficient graph-clustering formulation, and derive their underlying 3D line using their assigned 3D locations.

6. **Combined Bundle Adjustment** [optional] (Section 3.8)
   As an optional step, we show how the resulting line-based 3D model and the given *SfM* result can be further improved by a combined bundle adjustment procedure over all features (points and lines).

Figure 3.4 illustrates all processing steps in a graphical way, for a better understanding. In the following sections we will describe all these steps in more detail.

3.3 Line Segment Detection

As a first step, we have to detect a set of 2D line segments $L_i = \{\ell_1^i, \ldots, \ell_{m_i}^i\}$ for each (undistorted) image $I_i \in I$, which will function as our 2D features for the 3D reconstruction procedure. For this task, any available 2D line segment detector can be used. Please note that we are dealing with *straight line segments*, and not curves. From this point forward, a *line* is always meant to be a *straight line*, unless stated otherwise. From the various existing 2D line representations (e.g. polar coordinates), we chose to represent each 2D line segment

$$\ell_m^i = \{p_m^i, q_m^i\} \tag{3.17}$$

simply by its two endpoints p_m^i and q_m^i ($p_m^i, q_m^i \in \mathbb{R}^2$). Since the amount of detected line segments obtained by a line segment detector can be very high (especially when image noise is present), we limit ourselves to the κ longest detected line segments in each image. This is motivated by the fact that long line segments are more likely to correspond to important structural parts of the scene (and are more robustly fitted), while very short segments often originate from image noise and less relevant structural elements. A similar scheme is often applied in point-based *SfM* as well, where usually only a subset of all detected feature

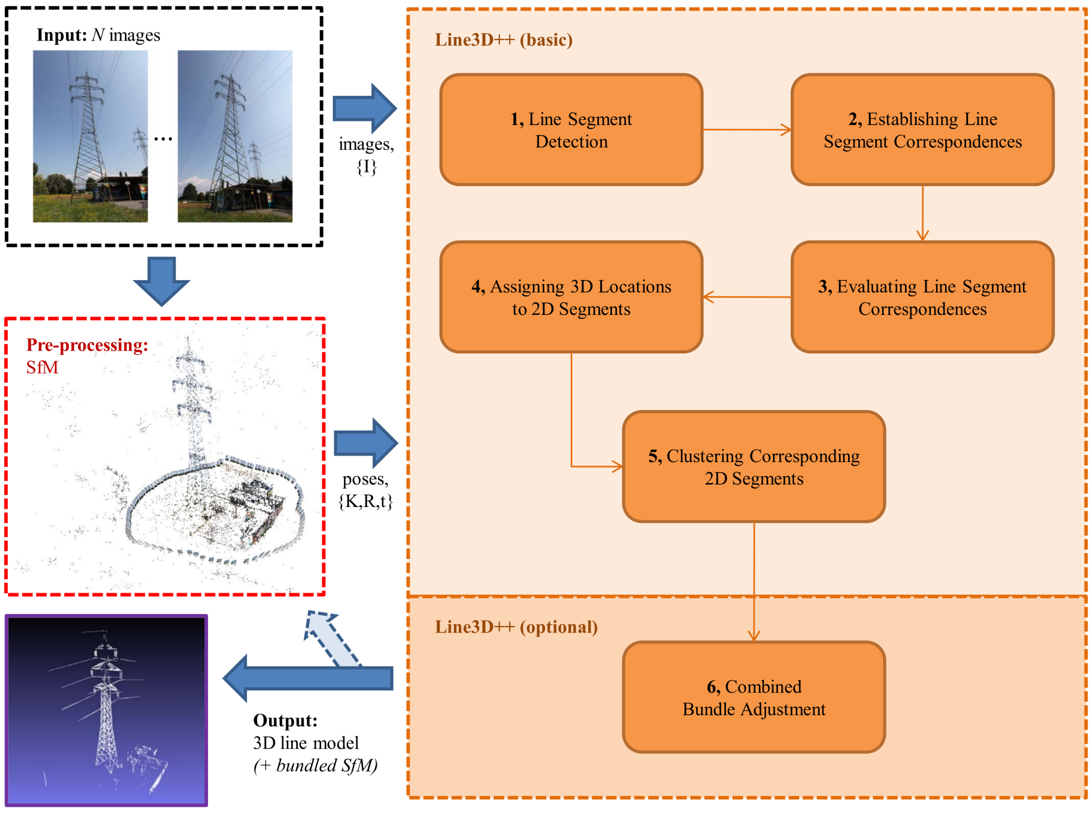

Input: N images
...
images, {I}
Pre-processing: SfM
poses, {K,R,t}
Line3D++ (basic)
1, Line Segment Detection
2, Establishing Line Segment Correspondences
3, Evaluating Line Segment Correspondences
4, Assigning 3D Locations to 2D Segments
5, Clustering Corresponding 2D Segments
Line3D++ (optional)
6, Combined Bundle Adjustment
Output: 3D line model (+ bundled SfM)

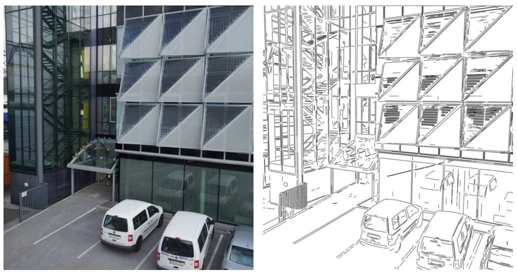
Figure 3.4: A schematic overview of our *Line3D++* line-based 3D reconstruction pipeline, including all necessary pre-processing steps (image acquisition and *SfM*).

points is actually used for the reconstruction (e.g. the k largest *SIFT* [101] features in [72]). In addition, we only consider line segments with a length

$$\|\ell_m^i\| = \|p_m^i - q_m^i\|_2 \tag{3.18}$$

of more than ρ times the image diagonal to be valid.

The choice of the line segment detector is in general arbitrary. However, it is of course recommended to use a robust algorithm, that is not easily influenced by image noise. Among the first approaches to line segment detections, Ballard [10] used the famous Hough voting scheme to transform an edge image (obtained using the Canny edge detector [22]) into a binary parameter space for infinite 2D lines, where the final line detections can be derived from local maxima. While Hough-based methods are in general very well suited to detect most of the relevant 2D lines, their precision is usually very low, because they are highly sensitive to noise and high-frequency texture. In addition, they usually have a comparably high runtime due to their computational complexity, which has partly been solved e.g. by using only a randomly selected subset of edge points [104].


(a) Building (detail)

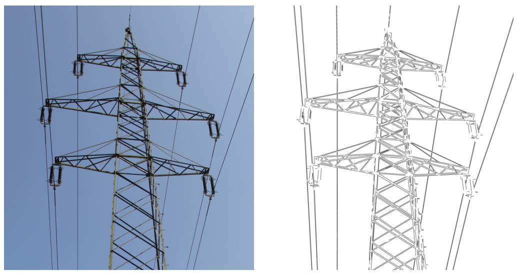

(b) Pylon (detail)

Figure 3.5: Line segment detection using the *LSD* algorithm [51]. (a) Example image from the *Building* sequence (3801 segments). (b) Example image from the *Pylon* sequence (6875 segments).

More recent line segment detectors, such as the Line Segment Detector (LSD) [51] or the *EDLines* [5] algorithm, do not suffer from such problems. Both methods are based on iteratively chaining together edge pixels with a similar orientation, and evaluating these straight edgels by an *a contrario* model, based on the so called Helmholtz principle [33]. They both have the benefit that they do not require any parameter tuning for various images, with the obtained results being fairly similar. Very recently, some more sophisticated methods such as *CannyLines* [103] or a saliency-based method [21] have been presented, which do not require excessive parameter tuning as well. Overall, the results for all state-of-the-art methods do not deviate by a large extend, which makes all of them perfectly usable for line-based 3D reconstruction. In our implementation, we use the *LSD* [51] algorithm, for which source code is available (e.g. within OpenCV 3 [2]). Figure 3.5 shows two example images with their respective line segments, obtained by *LSD*.

3.4 Establishing Line Segment Correspondences

To generate a line-based 3D model we need to establish correspondences between the detected 2D line segments from all images. Theoretically, this could be done by one of the numerous line-matching approaches presented in the past (e.g. [163, 181, 182]). However, we have already discussed in Section 2.2 that most of these approaches are appearance-based, and therefore not suitable for highly non-planar objects such as power pylons, or fences. To be able to reconstruct all kinds of linear structures, we use purely geometric matching constraints which are based solely on the epipolar geometry between two images.

The matching algorithm is divided into three main parts. First, we need to establish which image pairs (*visual neighbors*) should be matched, since especially in big scenes not all possible image pairs share a common field-of-view (Section 3.4.1). Then, we perform our weak epipolar-guided matching procedure for all selected image pairs, which returns

[2] `http://opencv.org/opencv-3-0.html`

multiple potential matches for each source 2D segment in each target image (*one-to-many matching*; Section 3.4.2). And last, we perform an optional filtering step which aims at increasing the matching precision (for a better performance in all subsequent steps), without negatively affecting the recall (*k-nn matching*; Section 3.4.3). After the matching is completed, we can triangulate all matching 2D segment pairs to 3D line segment hypotheses (Section 3.4.4), which will be needed to discard outlier matches and to reconstruct the 3D line model later on.

3.4.1 Visual Neighbor Selection

Especially for large scenes, it is unlikely that all image pairs I_i and $I_j \in I$ have some *visual overlap*, i.e. it is unlikely that they see the same part of the scene. Hence, it would be completely unnecessary to match all images with each other, which would result in a quadratic runtime complexity ($\mathcal{O}(N^2)$). In traditional *SfM*, this problem is often solved by making use of the computed feature-point descriptors, e.g. by utilizing a vocabulary tree [120]. Since we do not have such descriptors (that would not be reliable anyway), we have to think of another way of determining a meaningful subset of all image pairs for matching.

In our earlier work, we simply used the Euclidean distance $\|C_i - C_j\|_2$ between the camera centers C_i and $C_j \in \mathbb{R}^3$ of two images, and the angle between their optical axes as a similarity measure [65, 66]. However, this is not necessarily a good indicator for visual overlap, since it does not take occlusions into account. Since we have more information from the *SfM* result than just camera poses, we can also use visibility information from the sparse 3D point set X as an indicator for a common field-of-view [62, 63]. That means, if it holds for two images I_i and I_j that there exists a 3D point $\mathcal{X}_t$ such that

$$\mathcal{X}_t \in X_i \quad \wedge \quad \mathcal{X}_t \in X_j, \tag{3.19}$$

it follows that there is at least a small part of the scene which is visible in both images (unless $\mathcal{X}_t$ is an outlier). As a consequence, the more worldpoints the two images share, the higher is the probability that they have a large visual overlap, and the more sense does it make to match lines between them.

Building up on this assumption, we first define a pairwise view similarity score

$$S_I(i,j) = \begin{cases} \frac{2\cdot|X_i \cap X_j|}{|X_i|+|X_j|} & \text{if } i \neq j \\ -\infty & \text{else} \end{cases}, \tag{3.20}$$

which is based on Dice's similarity coefficient (the higher the more similar). It basically sets the number of the worldpoints that both images have in common in relation to the total number of worldpoints each image sees, and forbids self-similarity. We use this score to compute a sorted visual neighbor set

$$V_i = \{j_1, j_2, \dots \mid S_I(i, j_\cdot) > 0 \;\cap\; x < y \Rightarrow S_I(i, j_x) \geq S_I(i, j_y)\} \tag{3.21}$$

for each image I_i individually, which just contains all indices of all other images I_j (which share at least one worldpoint with I_i), sorted by their view similarity $S_I(i, j)$ in descending order. For a meaningful matching procedure we could now e.g. simply use the top M elements in V_i for each image I_i individually, which we have done previously in [63]. This results in a fast matching algorithm, with a linear time complexity $\mathcal{O}(N \cdot M)$ (with respect to the number of images N), given that M is kept constant (usually $M \ll N$, especially for large-scale datasets).

However, the problem with this formulation is that it may have a slight bias towards image pairs with a small baseline, since those pairs often have a lot of sparse 3D points in common (given the moderate rotational invariance of classic feature point descriptors). Since line segments are more robust to larger camera displacements, it would be reasonable to try to create more line correspondences between spatially more distant views, since resulting 3D triangulations would be more accurate and stable.

To achieve this, we define a second distance-based visual neighbor score

$$\hat{S}_I(i, j) = \begin{cases} \sum_{h \in \{1,2\}} |\langle R_i \cdot C_j + t_i, n_h \rangle| & \text{if } S_I(i, j) > 0.8 \cdot \max\{S_I(i, \cdot)\} \\ -\infty & \text{else} \end{cases}, \tag{3.22}$$

where $R_i \cdot C_j + t_i$ transforms the camera center C_j of image I_j into the camera coordinate frame of I_i, and $|\langle P, n \rangle|$ is the Euclidean distance between a 3D point P and a plane going through the origin (which equals C_i in the local camera coordinate frame) and with a normal vector n ($\langle \cdot, \cdot \rangle$ is the dot product). The two normal vectors are defined as

$$n_1 = (1, 0, 0)^T \quad \text{and} \quad n_2 = (0, 1, 0)^T, \tag{3.23}$$

which means that the two planes are the (y, z)- and the (x, z)-plane of the camera coordinate system (with the z-axis pointing along the optical axis into the scene; see Figure 3.6).

What we want to achieve with this formulation is, that views I_j which have a large spatial displacement (baseline) to I_i in both the *vertical*- as well as the *horizontal* direction have a higher score, which ideally means that vertical as well as horizontal lines in L_i can be more easily matched and triangulated with lines from L_j, since the epipolar lines of the endpoints of a specific 2D segment ℓ_m^i are less likely to collapse to one single line (which frequently happens especially for horizontal lines when the camera only moves in one constant height e.g. sideways along a façade).

However, since it would not make sense to consider a distance-based score on its own (because it would favour image pairs that are as far away from each other as possible, with potentially no visual overlap at all), the formulation in Equation 3.22 only returns positive

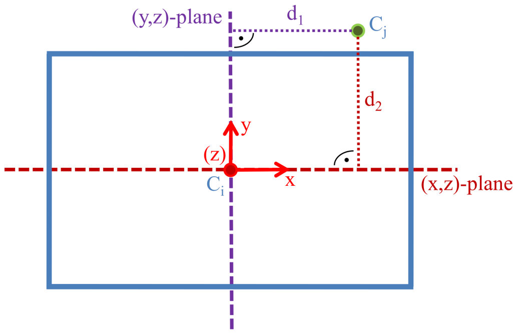


Figure 3.6: Visualization of the distance-based visual neighbor score $\hat{S}_I(i,j)$ between two images I_i and I_j (Equation 3.22). The image I_i is shown from the back, i.e. such that its optical axis (z-axis) is only visible as a point ($(z) \equiv C_i$). The two distances d_1 and d_2 of the camera center C_j of another image to the (y,z)- (*purple*) and the (x,z)-plane (*red*) of the camera coordinate frame are simply added to form the final score.

scores for view pairs with a worldpoint-based similarity score $S_I(i,j)$ that is bigger than 80% of the maximum score for image I_i (i.e. the score corresponding to the first index in the sorted set V_i). This ensures that only images which already have a high number of common worldpoints (and hence, probably a large shared field-of-view) are considered for distance-based scoring.

We now compute a second sorted visual neighbor set

$$\hat{V}_i = \left\{ j_1, j_2, \dots \mid \hat{S}_I(i,j_.) > 0 \ \cap \ x < y \Rightarrow \hat{S}_I(i,j_x) \geq \hat{S}_I(i,j_y) \right\} \tag{3.24}$$

which is a subset of V_i, but now sorted by the distance score $\hat{S}_I(i,j)$. We now select the first half ($M/2$) of the final visual neighbors from $\hat{V}_i$, and the second half from V_i (skipping potential duplicates). This ensures that we have a more balanced visual neighbor set consisting of images that have a high visual overlap with I_i ($\rightarrow V_i$), as well as those that have a large baseline to I_i ($\rightarrow \hat{V}_i$). We then store the final visual neighbor set in V_i^M and proceed to the matching step.

3.4.2 Epipolar-guided Line Matching

After the visual neighbor estimation, we match all segments in L_i to all segments in L_j (if $j \in V_i^M$), to determine potential correspondences. For a specific segment pair, $\ell_m^i \in L_i$ and $\ell_{\bar{m}}^j \in L_j$, we first compute the epipolar lines of the endpoints p_m^i and q_m^i of ℓ_m^i in the opposite image I_j. We then simply intersect the infinite line passing through the segment $\ell_{\bar{m}}^j$ with these epipolar lines, and obtain two intersection points x_p and x_q, which have to be collinear with the endpoints $p_{\bar{m}}^j$ and $q_{\bar{m}}^j$ of the segment $\ell_{\bar{m}}^j$. We then define a matching score between ℓ_m^i and $\ell_{\bar{m}}^j$ as

$$s(\ell_m^i, \ell_{\bar{m}}^j) = \frac{inner\left(\left\{p_{\bar{m}}^j, q_{\bar{m}}^j, x_p, x_q\right\}\right)}{outer\left(\left\{p_{\bar{m}}^j, q_{\bar{m}}^j, x_p, x_q\right\}\right)}, \tag{3.25}$$

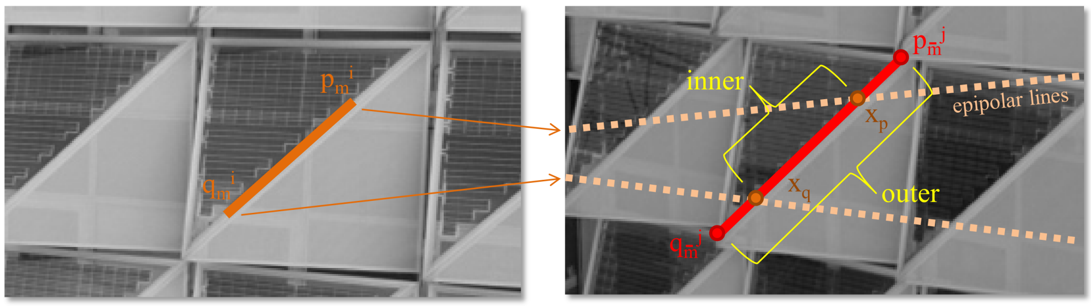


Figure 3.7: An illustration of the epipolar-based matching procedure. *Left:* image I_i with one specific 2D line segment ℓ_m^i, and its two endpoints p_m^i and q_m^i (*orange*). *Right:* The epipolar lines of the endpoints of ℓ_m^i in image I_j are shown as dashed lines. The two intersection points x_p and x_q (*brown*) of the epipolar lines and the infinite line passing through $\ell_{\bar{m}}^j$ (*red*) form a collinear quadruple, and the matching score $s(\ell_m^i, \ell_{\bar{m}}^j)$ is simply the Euclidean distance between the two inner points (x_p and x_q), divided by the distance between the two outer points ($p_{\bar{m}}^j$ and $q_{\bar{m}}^j$).

where $inner(\{\cdots\})$ and $outer(\{\cdots\})$ stand for the Euclidean distance between the inner- and the outer pair of the four collinear points ($p_{\bar{m}}^j$, $q_{\bar{m}}^j$, x_p, and x_q) respectively. Figure 3.7 illustrates this process graphically. What we basically do is to measure the *mutual support* between two 2D segments, given the geometric constraints which are defined by the epipolar geometry. If both segments would be ideal measurements (un-occluded and neither too long, nor too short) of the same physical 3D structure, the matching score $s(\ell_m^i, \ell_{\bar{m}}^j)$ would be exactly 1 (given that the camera poses are noise free as well).

If the matching score is above a fixed threshold τ, we consider ℓ_m^i and $\ell_{\bar{m}}^j$ to be potentially matching. However, we only consider constellations as valid matches where an overlap between the line segment $l_{\bar{m}}^j$ and the virtual segment defined by the intersection points x_p and x_q actually exists (i.e. when either one or both of the points x_p and x_q are in between $\ell_{\bar{m}}^j$, or vice versa).

3.4.3 Improving the Matching Precision

A matching procedure like this naturally results in a high recall but a very low precision, especially when the threshold τ is small. This means that for each 2D segment we generate a large set of potential matches, from which only few are correct (in most cases only one per neighboring view is correct). Figure 3.8 illustrates a potential matching situation for one specific 2D segment ℓ_m^i, and one visual neighbor image I_j. As we can see, several matching candidates $\ell_{\bar{m}}^j \in L_j$ fulfill the geometric matching requirements, which means that we end up with a quite large set of potential matches for this one segment ℓ_m^i. However, increasing the threshold does not lead to satisfying matching results for the general case, since the robustness to occlusions and imprecise line segment detections would quickly vanish.

We have already faced this issue in our previous work [61], where we tried to improve the matching precision by incorporating appearance information. We used a modified

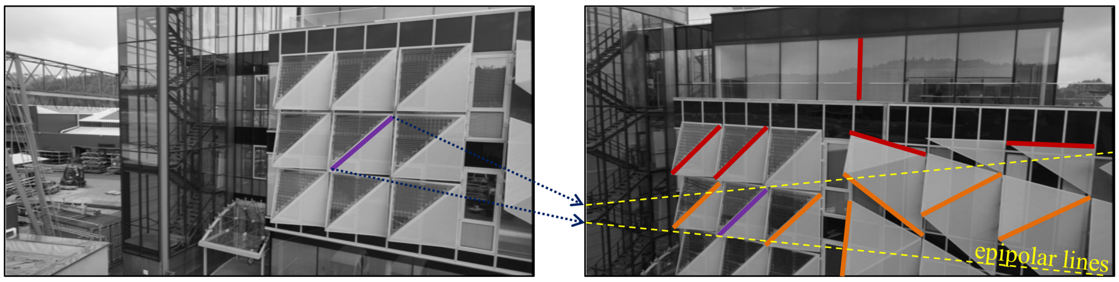


Figure 3.8: An illustration of how many matches per neighboring image I_j ($j \in V_i^M$) are to be expected. *Left:* image I_i with one specific 2D line segment ℓ_m^i (purple). *Right:* The epipolar lines of the endpoints of ℓ_m^i in image I_j are shown in yellow. Accepted matches are shown in orange and purple (correct match), and rejected matches in red. As we can see, more than one potential match is found by the matching procedure, since it only depends on weak epipolar constraints ($\tau = 0.25$). Please note that this is just a visualization and not necessarily the complete set of accepted/rejected matches!

version of the HSV histogram-based line descriptor by Bay et al. [17], which is not patch-based, and therefore also works for thin wiry structures. However, a color-based descriptor like this has the drawback that it is not very robust to illumination changes. In addition, color is often not enough to distinguish structural elements in urban environments, since such structures very often share the same color (as can be clearly seen for the potential matching candidates in Figure 3.8). Hence, we need to come up with a different way to narrow down the number of outlier matches.

When we look at the right part of Figure 3.8, we can see that although most of the accepted matches (*orange*) have a fairly high overlap score $s(\ell_m^i, \ell_{\bar{m}}^j)$, the correct match (*purple*) has definitely the highest score (close to the optimum). Given that we do not have any occlusions, or imprecise line segment detections, one could always just use the one match with the highest score per neighboring image, which has to be the correct one, unless it is not visible in this image. Since this assumption is unrealistic, it would of course not make a lot of sense to just use the one best match, which would result in a lot of missing correspondences. However, even if we consider occlusions and slightly imprecise 2D segments, it is unlikely that the correct match will not be among the top scoring matches for the vast majority of the cases. Hence, we propose to just keep the best k matches for one specific 2D segment ℓ_m^i, in each neighboring image. This is an optional step, and we will evaluate how different values for k effect the reconstruction results and the runtime in Chapter 4.

3.4.4 Creating 3D Hypotheses from Matched Segments

Since we know all camera poses, we can transform each 2D correspondence $\ell_m^i \rightarrow \ell_{\bar{m}}^j$ into a 3D line $H_{m,\bar{m}}^{i,j}$, which is the intersection of the two planes defined by the respective camera centers $C_i, C_j \in \mathbb{R}^3$ and the 2D segments ℓ_m^i and $\ell_{\bar{m}}^j$. We compute two 3D line segment hypotheses ($h_{m,\bar{m}}^{i,j}$ and $h_{\bar{m},m}^{j,i}$) for each correspondence, which are defined as 3D line

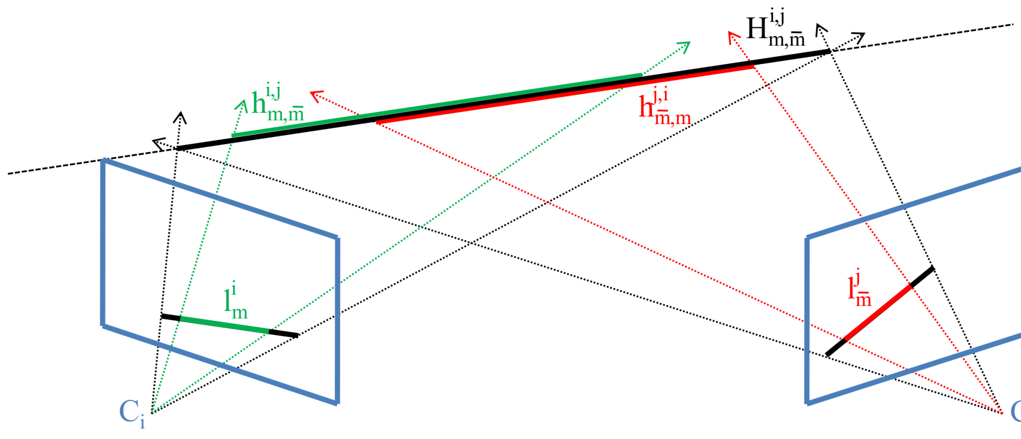


Figure 3.9: An illustration how the two 3D hypotheses $h^{i,j}_{m,\bar{m}}$ and $h^{j,i}_{\bar{m},m}$ are computed from two matched 2D segments ℓ^i_m (green) and $\ell^j_{\bar{m}}$ (red). Both hypotheses are collinear on the underlying (infinite) 3D line $H^{i,j}_{m,\bar{m}}$ (black), but in general not identical, since the endpoints of ℓ^i_m and $\ell^j_{\bar{m}}$ must not necessarily coincide.

segments on $H^{i,j}_{m,\bar{m}}$, whose projected endpoints coincide with the endpoints of the 2D line segments ℓ^i_m and $\ell^j_{\bar{m}}$ respectively. Similar to the 2D case, a 3D line segment consists of two 3D points $h^{i,j}_{m,\bar{m}} = \{P^{i,j}_{m,\bar{m}}, Q^{i,j}_{m,\bar{m}}\}$. Note that $H^{i,j}_{m,\bar{m}} = H^{j,i}_{\bar{m},m}$, while in general $h^{i,j}_{m,\bar{m}} \neq h^{j,i}_{\bar{m},m}$ (due to occlusions and imprecise 2D segment detections). However, the segments $h^{i,j}_{m,\bar{m}}$ and $h^{j,i}_{\bar{m},m}$ are always collinear on the infinite line $H^{i,j}_{m,\bar{m}}$. For a visualization see Figure 3.9.

With all potential correspondences and their respective 3D hypotheses at hand we can now proceed to the task of correspondence evaluation, where we aim at distinguishing correct from incorrect matches by analyzing spatial coherences between their estimated 3D hypotheses.

3.5 Evaluating Line Segment Correspondences

As stated above, the matching procedure gives us a potentially very large set of correspondences, most of which are of course incorrect. To distinguish between correct and incorrect matches, we want to compute some kind of confidence value for each of them, after all images have been matched with all visual neighbors. What can be done to check whether a matching hypothesis $h^{i,j}_{m,\bar{m}}$ between two segments ℓ^i_m and $\ell^j_{\bar{m}}$ makes sense or not, is to analyze how well this hypothesis *fits* to the neighboring images (apart from I_i and I_j, which created this hypothesis). This can for example be done using gradient-based scoring of the 3D hypothesis $h^{i,j}_{m,\bar{m}}$ over multiple images [66, 73], which basically means to project the 3D line segment into these images, and to compute how well the projection aligns with the image gradients. While this procedure is in general quite effective, it is also very time consuming. To speed-up this procedure, one could check how well the projection aligns just with the detected 2D line segments rather than all image gradients [63].

However, both methods just measure the re-projection error, which can be small despite a potentially large displacement between the reprojected 3D hypothesis and the pre-image of the reference 2D feature (either a gradient pixel [66, 73], or a 2D segment [63]). To overcome this problem, we want to evaluate the likelihood a correspondence and its 3D hypothesis directly in 3D space. This of course means that we need some reference data in 3D, to which we can evaluate our 3D hypotheses. Since we do not want to assume to have groundtruth available (i.e. in form of a dense surface model), we came up with a different idea for correspondence evaluation. The core idea behind our evaluation procedure is quite similar to the backprojection and scoring mentioned above. We also want to measure the deviation of a 3D hypothesis $h^{i,j}_{m,\bar{m}}$ from some other *feature*, but directly in 3D rather than in 2D.

Consider the following situation. Given a 2D segment ℓ^i_m that is correctly matched to a segment $\ell^j_{\bar{m}}$ in image I_j, and also correctly matched to another segment $\ell^b_{\tilde{m}}$ in image I_b, than the resulting 3D hypotheses $h^{i,j}_{m,\bar{m}}$ and $h^{i,b}_{m,\tilde{m}}$ *must* be spatially close to each other (in an ideal noise-free scenario, they should be perfectly collinear). However, if ℓ^i_m is matched incorrectly in both I_j and I_b, then the resulting 3D hypothesis will *most likely* not be spatially close, since their is no geometric consistency between triangulated outliers. Figure 3.10 illustrates this observation graphically. As we can see, triangulated inlier correspondences always support each other, while outlier correspondences do not have this property (unless by chance, on some rare occasions).

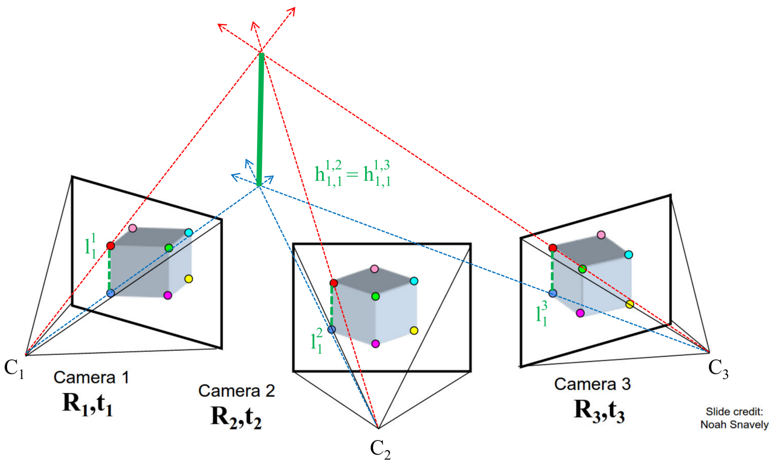
We now use this consistency property to our advantage, to compute a correctness score for each correspondence in 3D space. The main idea is to find for each 3D segment $h^{i,j}_{m,\bar{m}}$ (associated with the correspondence between $\ell^i_m \in L_i$ and $\ell^j_{\bar{m}} \in L_j$) all other 3D segments $h^{i,\cdot}_{m,\cdot}$ (originating from ℓ^i_m as well), that are spatially close to $h^{i,j}_{m,\bar{m}}$ (i.e. that *support* it). The number of different views from which these underlying correspondences emerge is a good indicator whether a 3D segment is likely to exist in reality or not, since it is unlikely that random incorrect correspondences are triangulated to the same spatial position, while all correct matches for one specific 2D segments always end up at the (approximate) same position in space.

In our method, we use a similarity measure based on spatial- and angular errors between 3D hypotheses. We assign a confidence

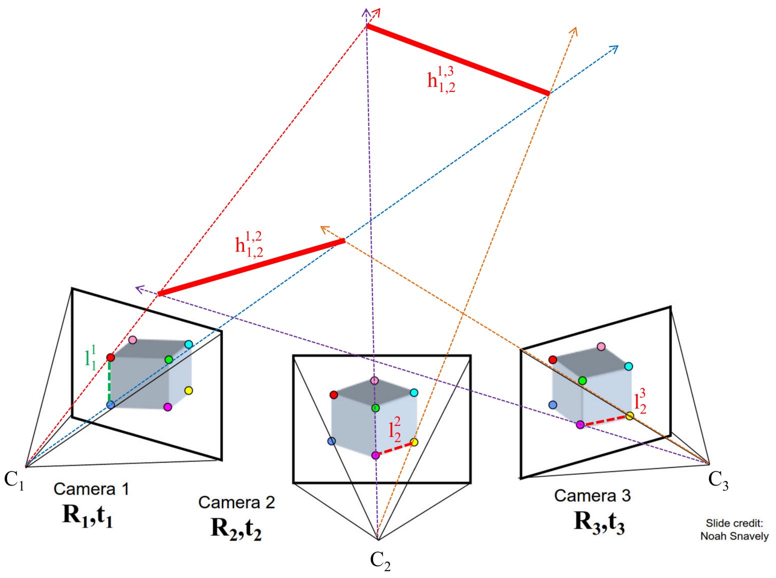
$$c(h^{i,j}_{m,\bar{m}}) = \sum_{x \in V^M_i \setminus \{j\}} \max_{y \in \{1,\ldots,m_x\}} \left\{ A(h^{i,j}_{m,\bar{m}}, h^{i,x}_{m,y}) \right\}, \tag{3.26}$$

to a correspondence $h^{i,j}_{m,\bar{m}}$, where A computes an affinity between two 3D hypotheses originating from the same source segment (ℓ^i_m). This affinity is defined as

$$A(h^{i,j}_{m,\bar{m}}, h^{i,x}_{m,y}) = \begin{cases} \min\left\{ S^a(h^{i,j}_{m,\bar{m}}, h^{i,x}_{m,y}), S^p(h^{i,j}_{m,\bar{m}}, h^{i,x}_{m,y}) \right\} & \text{if } \min\{\cdots\} > \frac{1}{2} \\ 0 & \text{else} \end{cases}, \tag{3.27}$$



(a) Inlier consistency



(b) Outlier inconsistency

Figure 3.10: An illustration how multiple inlier correspondences for one specific 2D segment ℓ_1^1 in image I_1 are spatially consistent, while this does not hold for outlier correspondences. (a) If ℓ_1^1 is correctly matched with ℓ_2^1 and ℓ_3^1 in the images I_2 and I_3, the resulting 3D hypotheses $h_{1,1}^{1,2}$ and $h_{1,1}^{1,3}$ are (ideally) identical. (b) If ℓ_1^1 is incorrectly matched with ℓ_2^2 and ℓ_2^3, the resulting 3D line segments $h_{1,2}^{1,2}$ and $h_{1,2}^{1,3}$ are not spatially consistent at all, despite the fact that both ℓ_2^2 and ℓ_2^3 are images of the same physical 3D structure. The original slide was taken from `http://slideplayer.com/slide/7085888/` (slide credit: Noah Snavely).

with S^a being an *angular* similarity in 3D, and S^p being a *positional* similarity in 3D, defined as

$$S^a(h^{i,j}_{m,\bar{m}}, h^{i,x}_{m,y}) = \exp\left(-\frac{\angle(h^{i,j}_{m,\bar{m}}, h^{i,x}_{m,y})^2}{2\sigma_a^2}\right) \tag{3.28}$$

and

$$S^p(h^{i,j}_{m,\bar{m}}, h^{i,x}_{m,y}) = \min_{Z \in h^{i,j}_{m,\bar{m}}} \exp\left(-\frac{d_\perp(Z, h^{i,x}_{m,y})^2}{\sigma_p(d_i(Z))^2 + \sigma_p(d_j(Z))^2}\right), \tag{3.29}$$

where $\angle(h_1, h_2)$ denotes the angle between the two line segments (in degrees), $d_\perp(Z, h_2)$ is the normal distance between the 3D point Z and the infinite line passing through h_2, and $d_i(Z) = \|C_i - Z\|_2$ is the Euclidean distance between the camera center C_i of I_i and Z (i.e. its depth along its corresponding camera ray). To prevent that the confidence gets high if we only have several weak supporters (i.e. many non-zero elements in the sum in Equation 3.26, but no strong support among them), we truncate the affinity and only accept values above 1/2.

Both formulas require a regularization parameter σ. For the angular case (σ_a; Equation 3.28), we can chose a reasonable value very easily, since angles are scale invariant. However, for the spatial case (σ_p; Equation 3.29) this is non trivial. Even if we know the scale of the reconstruction, choosing a constant value for all 3D hypotheses is not recommended, since the positional uncertainty of 3D line hypotheses h also depends strongly on the distances to the cameras from which they were triangulated. Hence, choosing a value that is too large will produce a bias towards hypotheses that are very close to the cameras, while a small value will prevent that hypotheses farther away from the cameras can reach reasonable scores.

To prevent this, we use a *depth adaptive* spatial regularization function $\sigma_p(d_i(Z))$, similar to the one we first proposed in [62]. This regularization function is defined as

$$\sigma_p(d) = d \cdot \frac{\mu}{d_{\text{med}}}, \tag{3.30}$$

which is a linear function in the depth d. The slope, μ/d_{med}, of the function consists of a user specified spatial regularizer μ (e.g. 0.05 for 5 cm in a metric reconstruction), and a regularization depth d_{med}, which in our case is simply the median worldpoint-to-camera distance (however, a different regularization depth can be chosen by the user as well). This formulation basically *allows* a spatial uncertainty of μ when the distance to the camera is exactly d_{med}, and a lower/larger uncertainty when the distance is smaller/bigger. This allows us to reliably score correspondences that are triangulated closer to their respective cameras, as well as those that are triangulated farther away.

However, this formulation requires knowledge about the reconstruction scale, since μ needs to be chosen by the user. Since in *SfM* applications scale information is not always present, we came up with a scale-invariant formulation to handle such cases [61–63, 65].

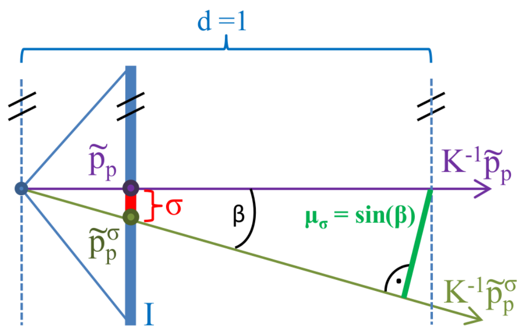


Figure 3.11: Derivation of the slope μ_σ of the spatial regularization function $\sigma_p^i(d)$, based on the camera geometry and the regularization parameter σ. The image plane I is shown from the top.

3.5.1 Scale-Invariant Spatial Regularization

To be scale invariant, we slightly modify the spatial regularizer $\sigma_p(d)$ (Equation 3.30) for each image individually as

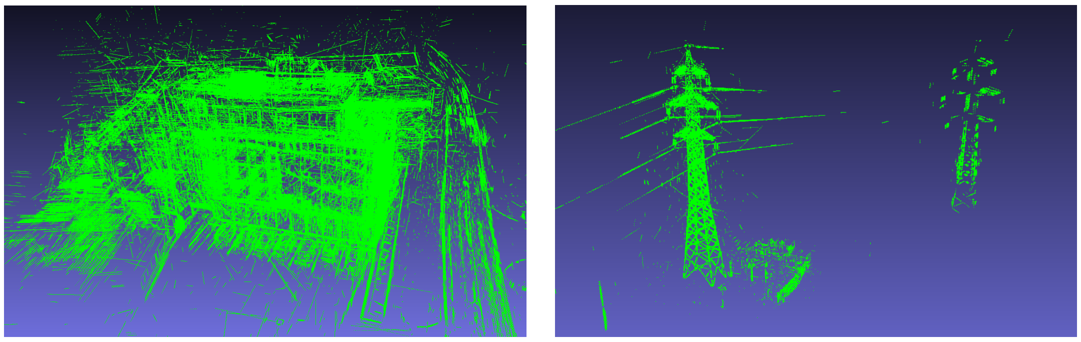
$$\sigma_p^i(d) = d \cdot \mu_\sigma, \tag{3.31}$$

which is again a linear function in the depth d. However, this time the slope, μ_σ, is derived from the underlying camera geometry. Given a standard pinhole camera model, we do this by shifting the principle point $\tilde{p_p}$ horizontally by a user defined 2D regularizer σ (in pixels), to obtain a second point $\tilde{p_p^\sigma}$ (in homogeneous coordinates), and then computing the angle β between the two 3D rays $K^{-1} \cdot \tilde{p_p}$ and $K^{-1} \cdot \tilde{p_p^\sigma}$, where K^{-1} are the cameras' inverse intrinsics. We then simply compute $\mu_\sigma = \sin(\beta)$, which is basically the maximum distance we can move the unprojection of the principal point at depth $d = 1$, such that the distance between the reprojection of the moved point and the principal point in the image is less or equal to σ (see Figure 3.11). This formulation ensures that higher 3D point-to-line distances ($d_\perp(Z, h)$ in Equation 3.29) are punished less when the respective hypotheses are farther away from their observing cameras, and vice versa. We now simply replace $\sigma_p(d)$ with our new $\sigma_p^i(d)$ in Equation 3.29, for the scale invariant case.

With the confidence formulation from Equation 3.26 we are now able to determine whether a matching hypothesis makes sense or not. We only keep hypotheses for further processing for which $c(h_{m,\bar{m}}^{i,j}) > 1$, which means that at least two segments from two additional images (apart from I_i and I_j) have to support $h_{m,\bar{m}}^{i,j}$. We end up with a much sparser set of correspondences, with a significantly lower number of outliers, while correct hypotheses are only seldom removed (e.g. when a segment is occluded in many visual neighbors). Figure 3.12 (top row) shows a visualization of all verified correspondences, that remained after the previous verification step. The building and the pylon can be clearly recognized, even though several outlier hypotheses remain (e.g. in the sky).

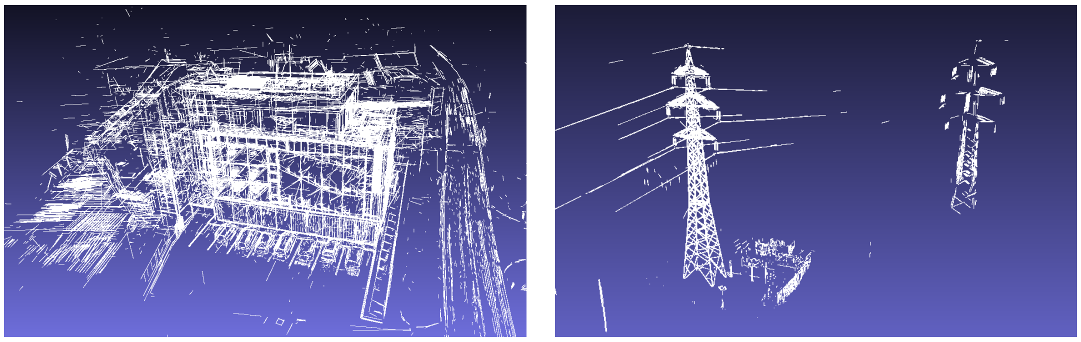
3.6 Assigning 3D Locations to 2D Segments

Given all remaining (verified) hypotheses $h_{m,\bar{m}}^{i,j}$ for a 2D segment ℓ_m^i, we want to estimate its most probable 3D position, since each 2D segment can only be a projection of one



$649,388$ $292,798$

(a) All scored hypotheses $\{h^{i,j}_{m,\bar{m}}\}$



$101,074$ $55,587$

(b) Best hypotheses $\{\hat{h}^i_m\}$

Figure 3.12: The result of the scoring and filtering procedure. (a) All hypotheses $h^{i,j}_{m,\bar{m}}$ for which $c(h^{i,j}_{m,\bar{m}}) > 1$. (b) The set of all selected hypotheses $\{\hat{h}^i_m\}$ (i.e. the one hypothesis with the highest confidence for each 2D segment across all images). The 3D structure is clearly recognizable in all images, while the majority of the remaining outliers have been removed below.

specific 3D structure. We use this 3D information for the following clustering procedure, as we have first shown in [61, 62]. For each 2D segment ℓ^i_m we define its 3D location as

$$\hat{h}^i_m = \underset{h^{i,j}_{m,\bar{m}}}{\operatorname{argmax}} \left\{ c(h^{i,j}_{m,\bar{m}}) \right\}, \tag{3.32}$$

which is simply its 3D hypothesis with the highest confidence. As stated above, we only keep hypotheses with a confidence bigger than one, which means that each remaining hypothesis is supported by ≥ 4 images (see Section 3.5). We have seen that 3D segment hypotheses verified by 4 or more images are rarely incorrect, which can also be observed for *SfM* point-clouds on the 3D point level. Figure 3.12 (bottom row) shows all selected 3D hypotheses $\{\hat{h}^i_m\}$ for the *Building* and the *Pylon* sequence. As we can see, the majority of the 2D segments selects its correct 3D position, with only few isolated outliers remaining.

3.7 Clustering Corresponding 2D Segments

After the previous step, we end up with an individual 3D estimate $\hat{h}^i_m$ for each 2D line segment ℓ^i_m. To compute one consistent 3D model, we have to fuse corresponding 2D segments (i.e. segments that originate from the same 3D entity) and their 3D estimates together. As we have shown in our previous work [62, 63], this can be efficiently done by employing a graph clustering procedure, which operates on a global affinity matrix W, encoding the pairwise similarities between potentially corresponding 2D segments across all images. Since we only need to consider segment pairs which have been successfully matched (i.e. for which a valid 3D hypothesis exists), this matrix is usually very sparse. To compute these similarities, we use the same metric as for the hypothesis confidence above (see Equation 3.27). Hence, the affinity between two potentially matching 2D segments ℓ^i_m and $\ell^j_{\bar{m}}$ can be computed straightforwardly as

$$W(\ell^i_m, \ell^j_{\bar{m}}) = \begin{cases} \min\left\{S^a(\hat{h}^i_m, \hat{h}^j_{\bar{m}}), \bar{E}^p(\hat{h}^i_m, \hat{h}^j_{\bar{m}})\right\} & \text{if } \min\{\cdots\} > \frac{1}{2} \\ 0 & \text{else} \end{cases}, \tag{3.33}$$

where S^a is the basic angular similarity (Equation 3.28), and $\bar{E}^p$ is a symmetric positional similarity defined as

$$\bar{E}^p(\hat{h}^i_m, \hat{h}^j_{\bar{m}}) = \min\left\{\bar{S}^p(\hat{h}^i_m, \hat{h}^j_{\bar{m}}), \bar{S}^p(\hat{h}^j_{\bar{m}}, \hat{h}^i_m)\right\}, \tag{3.34}$$

which is computed using a slightly modified version of S^p (referring to Equation 3.29)

$$\bar{S}^p(\hat{h}^i_m, \hat{h}^j_{\bar{m}}) = \min_{Z \in \hat{h}^i_m} \exp\left(-\frac{d_\perp(Z, \hat{h}^j_{\bar{m}})^2}{\bar{\sigma}_p(d_i(Z))^2 + \bar{\sigma}_p(d_j(Z))^2}\right), \tag{3.35}$$

with a different spatial regularizer

$$\bar{\sigma}_p(d) = \begin{cases} d \cdot \frac{\mu}{d_{\text{med}}} & \text{if } d < d^L_{\text{med}} \\ d^L_{\text{med}} \cdot \frac{\mu}{d_{\text{med}}} & \text{otherwise} \end{cases}, \tag{3.36}$$

which is truncated at a certain depth d^L_{med}, to avoid the possibility that the allowed spatial uncertainty grows too large for points far away from the camera center. This is not necessary during the scoring procedure, but for the reconstruction we want to prevent potentially imprecise line clusters that might occur far away from the observing cameras. To obtain a reasonable estimate for d^L_{med} (even when the scale is unknown), we define it as the median depth over all final 3D hypotheses $\hat{h}^i_m$ to their respective camera I_i, by using both segment endpoints (similar as for d_{med} in Equation 3.30, but for 3D line hypotheses instead of worldpoints).

The resulting affinity matrix W could now be directly fed to an arbitrary graph clustering algorithm, which takes a simple pairwise affinity matrix as an input. In our earlier approaches [61, 62], we used the popular method by Felzenszwalb and Huttenlocher [44] as a clustering algorithm, which delivers visually pleasant results for the general case, without any kind of parameter tuning. In our most recent work [63, 64], we experimented with a different clustering strategy [35], which is based on diffusing the given affinity matrix W, by implicitly considering the underlying data manifold. The diffused matrix is then segmented by the same graph clustering method as before [44]. In both cases, we only accept clusters if they contain 2D segments from at least three different images.

In practice the choice of the clustering method has a minor influence on the resulting 3D model. In general, the result obtained using [35] manages to cluster together more 3D segments and obtains a slightly denser result, at virtually no additional cost regarding computational time. Hence, we recommend to use the diffusion-based method [35] by default.

Regardless of the method, the output of the clustering algorithm is always a set of 3D line clusters $\Pi = \{\Pi_1, \cdots, \Pi_T\}$, where each cluster Π_t consists of a set of 2D residuals $L_{\Pi_t} = \{\ell_1, \ell_2, \cdots\}$, a representative 3D line H_{Π_t}, and one or more collinear 3D line segments $h_{\Pi_t} = \{h_1, \cdots\}$ (which are a part of the infinite line H_{Π_t}). To obtain the underlying 3D line H_{Π_t} for a cluster Π_t, we simply use the 3D depth estimates ($\hat{h}$) of its 2D residuals.

3.7.1 Final 3D Lines from Clustered Segments

To estimate one representative 3D line H_{Π_t} for a cluster Π_t, we need two components: the line direction $v_{\Pi_t} \in \mathbb{R}^3$, and a 3D point $Z_{\Pi_t} \in \mathbb{R}^3$ on the line. Other representations are of course possible as well, e.g. Plücker coordinates or the Cayley representation [180], but the point-direction representation can be straightforwardly derived from the 3D hypotheses $\hat{h}$ of the clustered 2D segments L_{Π_t}, as first shown in [73].

To obtain the line direction v_{Π_t} we first compute a matrix

$$\mathcal{P} = \begin{bmatrix} P_m^i & Q_m^i & \cdots \end{bmatrix}, \quad P_m^i, Q_m^i \in \hat{h}_m^i \text{ s.t. } \ell_m^i \in L_{\Pi_t}, \tag{3.37}$$

which simply contains all 3D endpoints of the 3D estimations $\hat{h}_m^i$, over all 2D residuals of the cluster (L_{Π_t}). We then compute the scatter matrix

$$M = \mathcal{P} \cdot \left(I - \frac{1}{2 \cdot |L_{\Pi_t}|} \cdot \mathbb{C}_1 \right) \cdot \mathcal{P}^T, \tag{3.38}$$

where I is the identity matrix, and $\mathbb{C}_1$ is a constant matrix containing only 1's. Finally, we perform a Singular Value Decomposition

$$\{U, \Sigma, V\} = \text{SVD}(M), \tag{3.39}$$

and define v_{Π_t} to be the column of U corresponding to the highest singular value (stored in Σ), normalized to unit length. A point on the line (Z_{Π_t}) can easily be obtained by computing the center of gravity of all the 3D segment endpoints stored in $\mathcal{P}$. The final 3D line H_{Π_t} of the cluster can now be written in parametric form as

$$H_{\Pi_t}(s) = Z_{\Pi_t} + s \cdot v_{\Pi_t}. \tag{3.40}$$

The estimated 3D line H_{Π_t} represents the clustered 2D segments as an infinite line in 3D space. However, not every part of the line is actually physically existing (i.e. corresponds to a real structural element in the scene). To obtain the actual 3D line segment set h_{Π_t} on H_{Π_t} (i.e. the actually visible part of the infinite line H_{Π_t}, with respect to its 2D observations L_{Π_t}), we project the 3D estimates ($\hat{h}$) of all 2D residuals L_{Π_t} onto the newly estimated 3D line H_{Π_t}, and compute a minimal set of non overlapping 3D line segments on this line, such that each of these segments is fully covered by at least three of the projected hypotheses (from at least three different images). This ensures that we only reconstruct 3D line segments that actually correspond to physically existing parts of the observed scene. Figure 3.13 shows the final 3D line models (i.e. all final sets h_{Π_t}, for all valid clusters Π_t) for the *Building* and the *Pylon* sequence, using the two different clustering approaches mentioned above.

3.8 Combined Bundle Adjustment

The resulting set of 3D line clusters Π obtained in the previous section can be used as the final output of our pipeline, and is in general very accurate (with respect to the used regularization parameters μ or σ, and σ_a). However, since the obtained 3D lines are more or less an average over several 3D hypotheses, which are only two-view triangulations, there is no guarantee that they fit the 2D observations in an optimal way. To overcome this issue, one could easily formulate a bundle adjustment procedure [156], to optimize the 3D lines with respect to their 2D residuals (i.e. minimizing the reprojection error). In addition, we could go even further and don't just optimize the 3D lines alone, but also the whole *SfM* result (points and camera poses) using the newly obtained 2D to 3D correspondences, that are provided by the reconstructed line clusters.

While there are slight differences in the exact formulation in various *SfM* pipelines, bundling is usually done by minimizing a non-linear least squares problem, consisting of the reprojection error of the reconstructed 3D points to their 2D residuals, with respect to the camera poses. Analogue to a 3D line cluster Π_t, a reconstructed 3D point $\mathcal{X}_t \in X$ from the *SfM* can be seen as a cluster Ω_t of 2D points $P_{\Omega_t} = \{p_1, p_2, \cdots\}$ ($p \in \mathbb{R}^2$), and an associated 3D position $\mathcal{X}_t \in \mathbb{R}^3$. For points and lines, the combined optimization problem can be defined as

$$f^* = \min_{\{R,t\},\Omega,\Pi} \sum_{\Omega_t \in \Omega} f^P(\Omega_t) + \lambda \sum_{\Pi_t \in \Pi} f^L(\Pi_t), \tag{3.41}$$

where R, t are the rotation and translation of the cameras, and f^P and f^L are error functions for 3D points and lines respectively, with a scalar weighting factor λ. For the

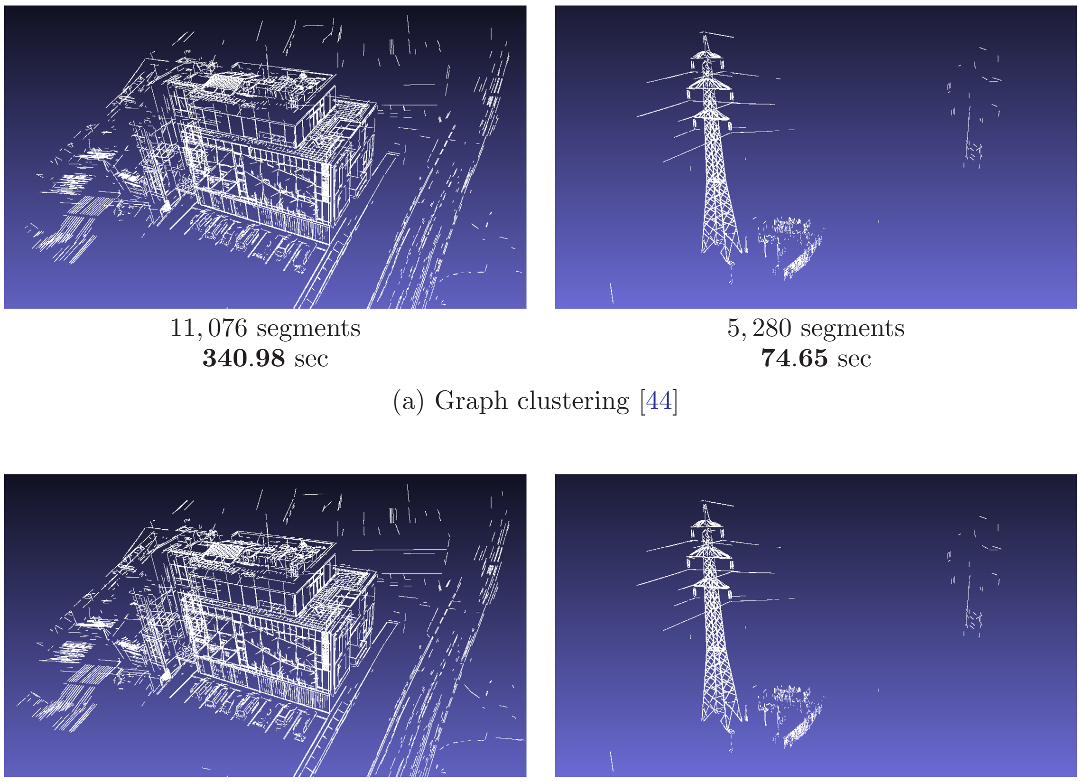


Figure 3.13: Final line-based 3D models of the *Building* and the *Pylon* sequence obtained by *Line3D++* when (a) directly using the graph-clustering method by Felzenszwalb and Huttenlocher [44], or when (b) pre-processing the affinity matrix W by using matrix diffusion [35]. Both results are almost identical. However, the diffusion based approach often manages to reconstruct slightly more parts of the scene.

sake of simplicity, we do not include the cameras' intrinsics and distortion coefficients into the bundle adjustment (but they can be added straightforwardly). In our case, the error functions are defined as

$$f^P(\Omega_t) = \sum_{p^x \in P_\Omega} \rho_\epsilon \left(d^P_{2D}(\Gamma_x(\mathcal{X}_t), p^x) \right) \tag{3.42}$$

and

$$f^L(\Pi_t) = \sum_{\ell^x \in L_\Pi} \rho_\epsilon \left(d^L_{2D}(\Gamma_x(H_\Pi), \ell^x) \right), \tag{3.43}$$

where ρ_ϵ is the robust Huber loss function (linearised at ϵ), Γ_x projects a 3D point or line into image I_x, and d^P_{2D} and d^L_{2D} are error functions for 2D points and lines respectively.

For the point case, a common choice is the Euclidean distance

$$d_{2D}^{P}(p_1, p_2) = \|p_1 - p_2\|_2, \tag{3.44}$$

between the projected ($\Gamma_x(h_\Omega) \equiv p_1$) and the observed ($p^x \equiv p_2$) 2D point.

For lines this is slightly more complicated. We define the error function as

$$d_{2D}^{L}(\ell_1, \ell_2) = \left(\sum_{p \in \ell_2} d_{\perp}^{2D}(p, \ell_1) \right) \cdot \exp(w_L \cdot \angle(\ell_1, \ell_2)), \tag{3.45}$$

where $d_{\perp}^{2D}(p, \ell)$ is the normal distance from an endpoint p to an infinite line ℓ (analogue to Equation 3.29 for the 3D case), and $\angle(\ell_1, \ell_2)$ is the angle between the two lines ℓ_1 and ℓ_2 ($\in [0, \pi/2]$). The second term can basically be seen as a weighting factor for the first term, and punishes directional deviations by increasing the positional error if the angle between the projected and the observed 2D segment is large. We have seen that this additional weighting leads to a much faster convergence, without negatively affecting the results. To avoid over-parametrization, we use the Cayley 3D line representation for bundling, as suggested in [180]. It has the benefit that it only needs four parameters to define a 3D line, which is an optimal encoding, since a 3D line has exactly four degrees of freedom. We solve the optimization problem by using the powerful *Ceres* solver [1], which offers built-in support for bundle adjustment and is very efficient even for large-scale problems.

3.9 Summary

Throughout this chapter we have introduced our line-based 3D reconstruction pipeline, and all its necessary computing steps. For each step, we have introduced a set of parameters, which have to be set properly in order to ensure satisfying reconstruction results. In the following chapter, we will discuss these parameters and their effects on both the 3D model and the runtime, for each step of the algorithm individually. We test our method on several challenging datasets, with and without groundtruth 3D surfaces and/or camera poses. In addition, we will evaluate the effect of the combined bundle adjustment procedure on both the accuracy of the 3D line model, as well as the accuracy of the underlying camera poses.

4 Evaluation and Results

Contents

4.1 Testing Environment & Implementation Details 61
4.2 Default Parameters 63
4.3 Test Datasets 64
4.4 Reconstruction Results 67
4.5 Parameter Evaluation 70
4.6 Bundle Adjustment Evaluation 79
4.7 Runtime Evaluation: GPU vs. CPU 82
4.8 Summary 84

In this chapter, we show various reconstruction results using our proposed method. Furthermore, we evaluate all parameters introduced in the different pipeline steps throughout Chapter 3, and their effect on the reconstruction results. To have a meaningful evaluation, we use several challenging test sequences (synthetic and real-world) with and without groundtruth camera poses, and for some cases also with a groundtruth 3D surface. We will conclude this chapter with an evaluation of the combined bundle adjustment procedure for points and lines, and with a runtime analysis which highlights the benefits of parallel computing using the *GPU*.

4.1 Testing Environment & Implementation Details

Our test system is an ordinary mid-range desktop PC equipped with the following components:

- **CPU:** Intel Core i5-3570, 4×3.4 GHz
- **GPU:** nVidia Geforce GTX 580, 512 *CUDA* cores

- **Main Memory:** 16 *GB*
- **Hard Drive:** 2 *TB HDD*

Our algorithm is implemented in C++ and *CUDA* (if an nVidia graphics card is available). To ensure an efficient computation, we make use of parallel computation whenever possible. For less computationally expensive steps (e.g. line segment detection or creating the affinity matrix), and when no suitable *GPU* is available, we make us of the *OpenMP* [1] framework for parallelization. If possible, the most runtime intense steps, which are line matching and hypotheses scoring, are executed on the *GPU*. We will analyze the achieved speed-up in Section 4.7.

4.1.1 SfM Pipeline

As explained in Section 3.1, we obtain the camera poses of our input images by running an off-the-shelf Structure-from-Motion (SfM) pipeline beforehand. While our code directly supports multiple available *SfM* pipelines (free and non-free), we use our in-house *ICG3D* library for all test sequences in this chapter (apart from one Internet-scale crowd-sourced dataset, which already comes with a *bundler* [143] result file). The *SfM* part of the library was first introduced in [72], and often extended and improved in subsequent years. It offers state-of-the-art performance in terms of accuracy, and has a very similar runtime performance as the recently proposed *colmap* [137].

The core principles of this *SfM* pipeline are quite straightforward. All images are first undistorted based on an initial calibration estimated using [29]. Then, *SIFT* [101] features are extracted from the undistorted images, and matched using an accelerated vocabulary tree based matching scheme [120]. By default, we extract the 5000 largest features per image, and use

$$v_r = \max(\min(0.2 \cdot N, 20), 5) \tag{4.1}$$

as a voctree radius, where N is the total number of images in the sequence. After the matching step, relative poses are computed between matched images, and verified using epipolar constraints based on the matched feature points. Then, an epipolar graph is created, and the largest connected chain of cameras is extracted. The first two images are used as an initialization, with the origin being located at the camera coordinate frame of the first image, and the distance between the two images set to one. Now, the remaining images are added incrementally using absolute pose estimation [89]. After every tenth newly added image (and at the very end), bundle adjustment is performed using the Ceres solver [1]. All processing steps are executed in parallel (whenever possible), either on the *GPU* or on the *CPU*.

For all real-world test sequences, we do not perform geo-referencing or manual scaling, and keep the reconstruction in the local (non-metric) coordinate system. For the

[1] `http://openmp.org/wp/`

groundtruth sequences, we rigidly align the *SfM* result to the groundtruth poses, using a basic similarity transformation (rotation, translation, and scale). We could of course directly use the groundtruth camera poses without the *SfM* as well. However, then we would not have feature correspondences for visual neighbor estimation.

4.2 Default Parameters

Throughout Chapter 3, we have introduced several parameters for each of the reconstruction steps. For the algorithm to work as expected, these parameters need to be set properly. In Section 4.5 we evaluate different values for all these parameters, and their effect on the final output. However, we evaluate the parameters step-by-step, which means that we will only vary the parameters corresponding to one specific algorithmic step (e.g. matching) at a time, and keep all the others fixed. Hence, we have to specify the default values for all steps in advance.

We have performed numerous experiments over the last years, which finally resulted in this basic default parameter set for our open-source implementation. Please note that these values do not necessarily produce the *best* results for *all* possible scenes (which would be virtually impossible), but are set such that reasonable results are to be expected for previously unprocessed datasets, without the necessity of excessive parameter tuning. Of course, tuning the parameters individually for each dataset might potentially improve the completeness and the accuracy of the resulting 3D models even more, but this would be a very tedious task with no scientific value. Table 4.1 shows all default values, for all the individual reconstruction steps.

Two of these values are not stated as a specific value in the table. The first is the spatial regularizer (μ [m] or σ [px]), which is by default set to $\sigma = 2.5$ pixels, since we do not assume to have a metrically scaled *SfM* reconstruction at hand. However, this parameter obviously heavily depends on the image sizes (see Section 3.5.1, on how the spatial regularizer is derived from σ). The selected value of 2.5 pixels usually delivers satisfying reconstruction results for high-resolution images ($>$ 10 Megapixels), and for non close-up views (e.g. outdoor recordings with a *UAV*). For image sequences where the camera is closer to the target objects (e.g. indoor scenes), this value generally needs to be increased. We will illustrate this issue for the various test datasets in Section 4.4, and specify a spatial regularizer for all of them individually.

The second non-constant parameter is the weighting term λ of the objective function in the bundle adjustment (see Section 3.8). We compute it dynamically by setting $\lambda = |\Omega|/|\Pi|$ (Equation 3.41), which balances the weight between points and lines such that both parts contribute equally.

Line Segment Detection (Section 3.3)		
Parameter	*Value*	*Description*
ρ	0.005	min. 2D line segment length (relative to the image diagonal)
κ	3000	max. number of 2D segments per image
Establishing Line Segment Correspondences (Section 3.4)		
M	10	number of visual neighbors for matching
τ	0.25	min. epipolar overlap
k	10	number of matches (per 2D segment and neighbor image) that are kept
Evaluating Line Segment Correspondences (Section 3.5)		
σ_a	10°	angular regularizer
μ / σ	var.	spatial regularizer
Clustering Corresponding 2D Segments (Section 3.7)		
Cl. mode	[35]	clustering algorithm (→ matrix diffusion)
Combined Bundle Adjustment (Section 3.8)		
ϵ	2 px	linearizer for the Huber loss function
w_L	2	influence of the angular line reprojection error
λ	dyn.	line weight in objective function

Table 4.1: The default parameters of the *Line3D++* algorithm (see Chapter 3).

4.3 Test Datasets

To show the capabilities of our method and to evaluate its accuracy, we use several challenging test datasets. We divide the set of test sequences into two categories: publicly available *groundtruth sequences*, for which camera poses and optionally also a 3D surface model are available, and *real-world sequences*, most of which we have recorded ourselves, by either using a high-resolution hand-held camera or an *UAV*.

4.3.1 Groundtruth Sequences

We use five groundtruth sequences in our evaluations that come with accurate camera poses, three of which also have a 3D surface mesh available. The first sequence is the *Timberframe*[2] dataset from [73]. It is a synthetic sequence consisting of 240 images (1280×960 px), with groundtruth camera poses and a Computer Aided Design (CAD) model. Figure 4.1 shows an example image from the sequence and the surface model.

The other four sequences are *Herz-Jesu-P8*, *Herz-Jesu-P25*, *Castle-P30*, and *Fountain-P11*, which are all part of the *Strecha* [146] dense Multi-View Stereo (MVS) datasets [3]. They have different numbers of images (specified by the "*-P**" at the end of the sequence name), and a fixed resolution of 3072×2048 pixels. Figure 4.2 shows example images for all sequences, and the two surface meshes (laserscans) of *Herz-Jesu-P8* and *Fountain-P11*.

[2] `http://resources.mpi-inf.mpg.de/LineReconstruction/`
[3] `http://cvlabwww.epfl.ch/data/multiview/denseMVS.html`

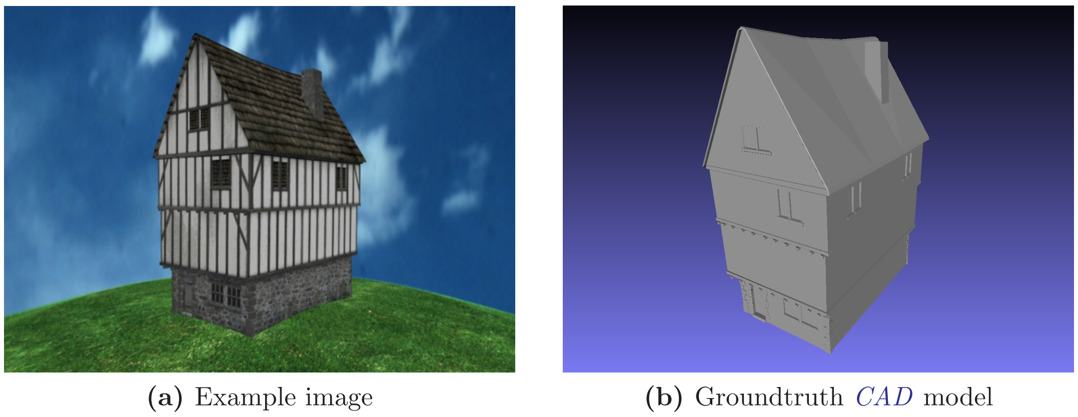
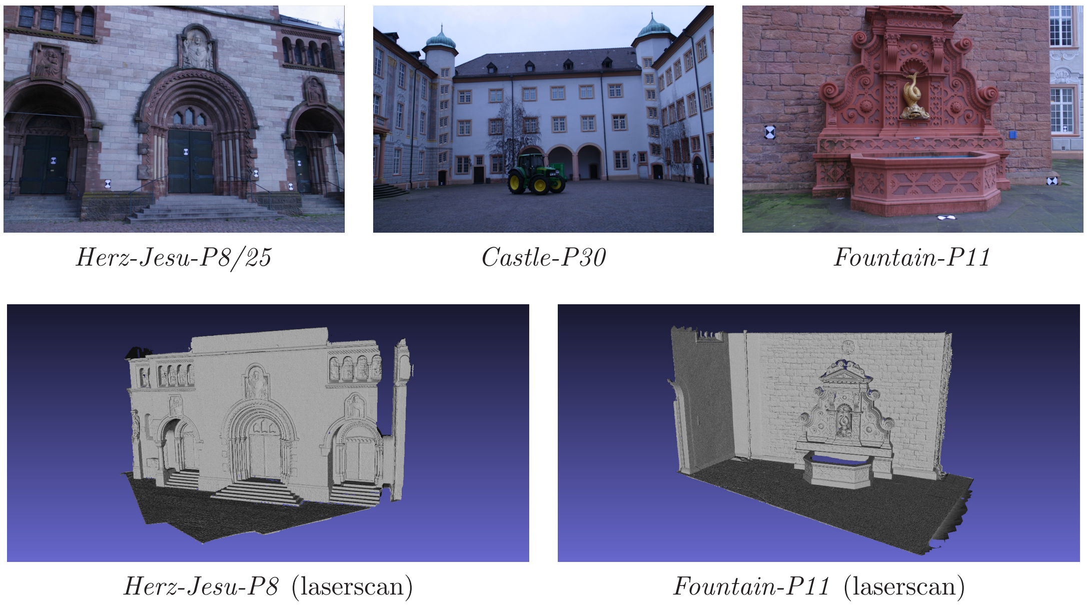

(a) Example image **(b)** Groundtruth *CAD* model

Figure 4.1: The synthetic *Timberframe* sequence [73] (240 images, 1280×960 px). (a) An example image from the sequence. (b) The groundtruth *CAD* model. Please note that the texture of the house, as well as the ground on which it stands, are not part of the groundtruth.


Herz-Jesu-P8/25 *Castle-P30* *Fountain-P11*

Herz-Jesu-P8 (laserscan) *Fountain-P11* (laserscan)

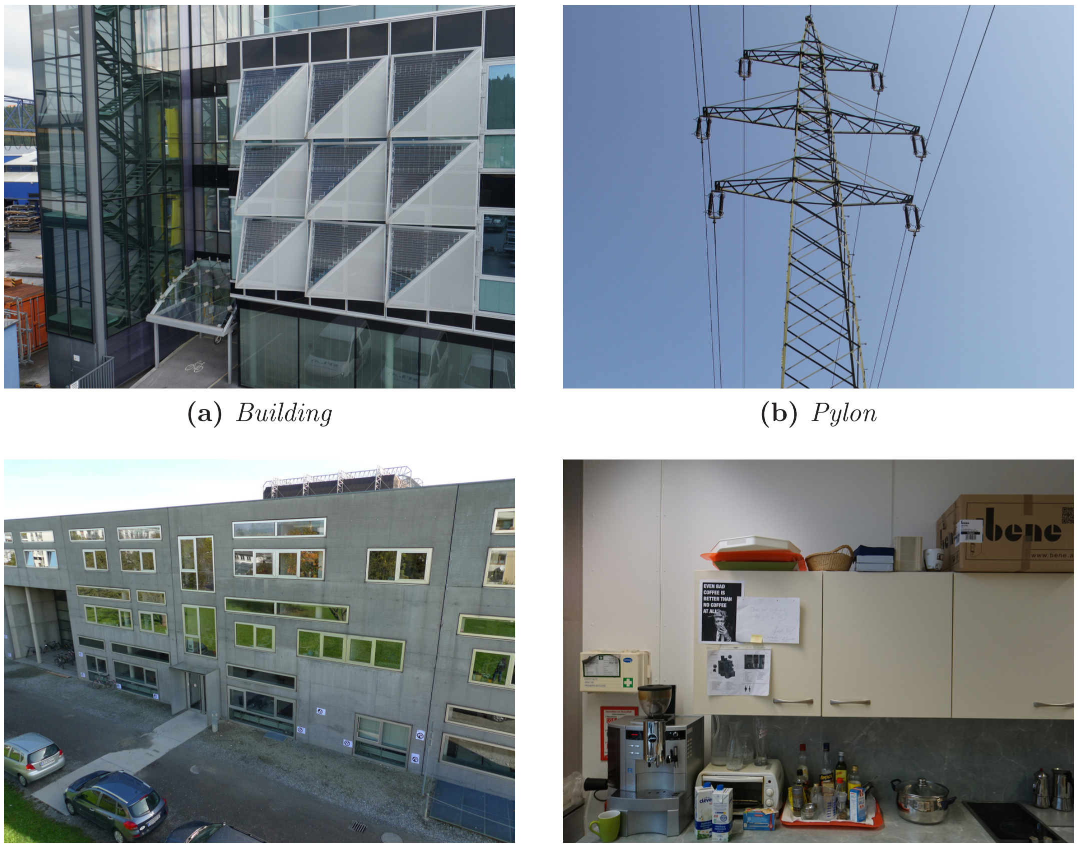
Figure 4.2: The used *Strecha* [146] dense *MVS* datasets (3072 × 2048 px). *Top row:* Example images of all sequences (*Herz-Jesu-P8* and *Herz-Jesu-P25* have similar images, only a different amount). *Bottom row:* The two groundtruth surface meshes (obtained by a laserscanner) for *Herz-Jesu-P8* and *Fountain-P11*. Please note that some actually existing parts of the depicted objects are not contained in the groundtruth, e.g. the railings at the entrance stairs in *Herz-Jesu-P8*.

4.3.2 Real-World Sequences

In addition to the five groundtruth datasets introduced above, we also use five real-world datasets for a qualitative evaluation. The first four datasets are the *Building*, *Pylon*, *Façade*, and the *Kitchen* sequence, which we have recorded ourselves. We used different cameras and recording platforms (hand-held or *UAV*), as well as different kinds of scenes (e.g. wiry objects, or indoor scenes). Table 4.2 gives some more information about the scenes, such as resolution or number of images. Figure 4.3 shows an example image from each sequence.

Name	**Camera**	**Platform**	**Resolution**	**# images**
Building	Sony NEX5	*UAV*	4912×3264	344
Pylon	Canon 5D	Hand-held	3744×5616	106
Façade	Panasonic DMC-LX3	*UAV*	3648×2736	317
Kitchen	Sony Alpha 6000	Hand-held	6000×4000	62

Table 4.2: The characteristics of our real-world test sequences.



(a) *Building* **(b)** *Pylon*

(c) *Façade* **(d)** *Kitchen*

Figure 4.3: Example images from the real-world sequences (a) *Building*, (b) *Pylon*, (c) *Façade*, and (d) *Kitchen*. Information about the datasets can be found in Table 4.2.

Finally, we also tested our algorithm on the Internet-scale *Dubrovnik6K* [4] [97] dataset,

[4]`http://www.cs.cornell.edu/projects/p2f/`

which consists of more than 6000 crowd-sourced images (see Figure 4.4 for some examples). It comes with a metrically scaled *SfM* reconstruction obtained using *bundler* [143], which we directly use as an input for our method. Here, most images have a different resolution and originate from different cameras, and were taken at different times as well. These facts make a dataset like this especially challenging to reconstruct, since the intrinsic camera- and distortion parameters have to be estimated for each image individually. In addition, most of the images have a comparably low resolution (below 5 Megapixels).

Figure 4.4: Example images from the crowd-sourced *Dubrovnik6K* dataset [97]. The images where obtained from Flickr (`https://www.flickr.com/`).

4.4 Reconstruction Results

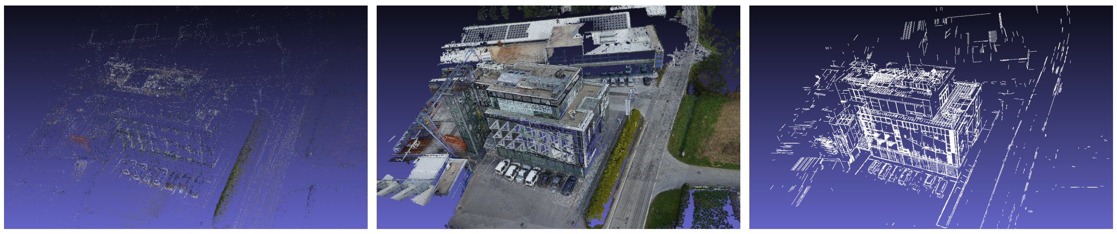
In this section, we show the reconstruction results for all our test sequences, using the default parameters introduced in Section 4.1. The spatial regularizer (μ for metric-, and σ for unscaled *SfM* results) is specified for each dataset individually.

4.4.1 Results on Real-World Sequences

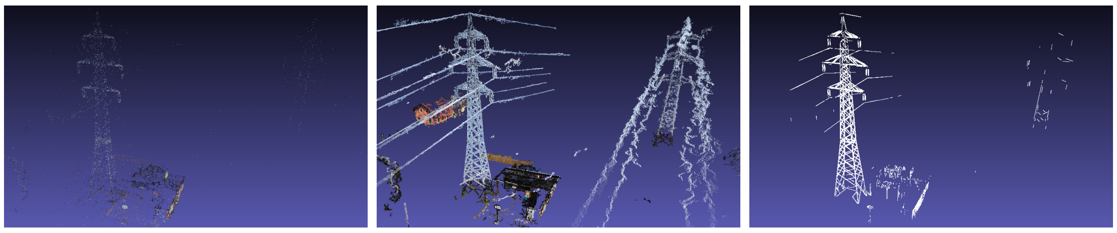
Figure 4.5 shows the final line-based 3D models for the real-world sequences, in context with their corresponding sparse *SfM*- [72], and their dense *PMVS* [49] point clouds. All relevant numbers (e.g. runtime, or number of 3D entities) can be found in Table 4.3.

As we can see, the line-based 3D models deliver a high amount of semantically meaningful information compared to basic *SfM*, despite being a sparse 3D representation as well. In addition, our algorithm is on average around 9.5 times faster than *SfM*, and more than 160 times faster than *PMVS* (however, *SfM* has to be performed beforehand, unless the camera poses are already available from some other source).

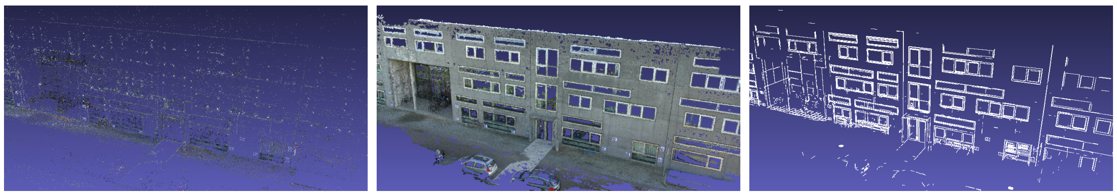
To further emphasize the efficiency of our algorithm, we show a line-based 3D reconstruction for the crowdsourced *Dubrovnik6K* [97] dataset in Figure 4.6. As we can see, our method is able to successfully process this very large-scale dataset, with a total runtime of just 90 minutes ($\varnothing t \approx 0.79$ s). Since the scale is metric, and the images have various different resolutions, we use a spatial regularizer $\mu = 0.025$ (which is 2.5 cm) defined in world coordinates. The obtained result shows satisfying 3D models, especially in densely photographed areas of the city (e.g. churches). However, due to the generally low image resolutions, the amount of noise is slightly higher compared to the results shown in



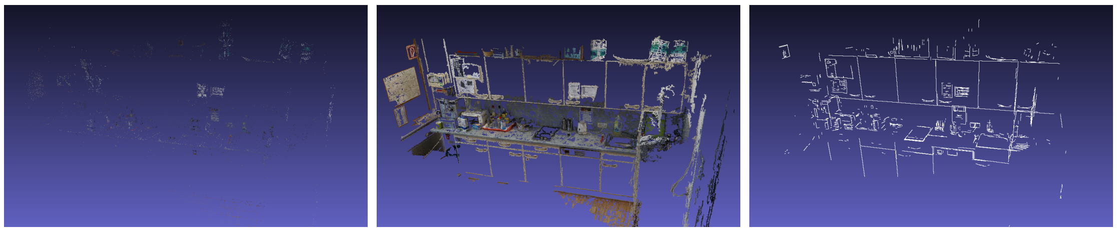
Building (344 images)



Pylon (106 images)



Façade (317 images)



Kitchen (62 images)

Figure 4.5: Reconstruction results for the real-world datasets. *Left column:* The sparse *SfM* reconstruction [72] using *SIFT* [101] feature points. *Middle column:* A dense point cloud using *PMVS* [49]. *Right column:* A line-based 3D model obtained using our proposed *Line3D++* algorithm. The corresponding numbers can be found in Table 4.3.

Figure 4.5. In general, we recommend to use one consistent high-resolution camera over crowdsourced data, but this experiment shows that our method can handle such cases as well.

4.4.2 Results on Groundtruth Sequences

Figure 4.7 shows a quantitative evaluation of our method on the *Herz-Jesu-P8*, and the *Fountain-P11* datasets [146]. The lines are color-coded by their Root-Mean-Square Error

	SfM [72]			*PMVS* [49]			**Line3D++ (proposed)**			
Dataset	time t [s]	$\varnothing t$ [s]	# points	time t [s]	$\varnothing t$ [s]	# points	time t [s]	$\varnothing t$ [s]	# lines	σ [px]
Building	2,640	7.67	97,699	51,660	150.17	14,516,368	**361.17**	**1.05**	**11,238**	2.5
Pylon	660	6.23	18,623	7,980	75.28	1,393,124	**75.27**	**0.71**	**5,319**	2.5
Façade	1,980	6.25	79,870	25,380	80.06	6,520,863	**162.02**	**0.51**	**6,767**	4.0
Kitchen	420	6.77	24,401	9,960	160.65	1,237,824	**35.53**	**0.57**	**1,118**	15.0
Mean		6.73			116.54			**0.71**		

Table 4.3: Corresponding numbers to the reconstruction results of our real-world datasets, seen in Figure 4.5. For more information about the datasets see Table 4.2.

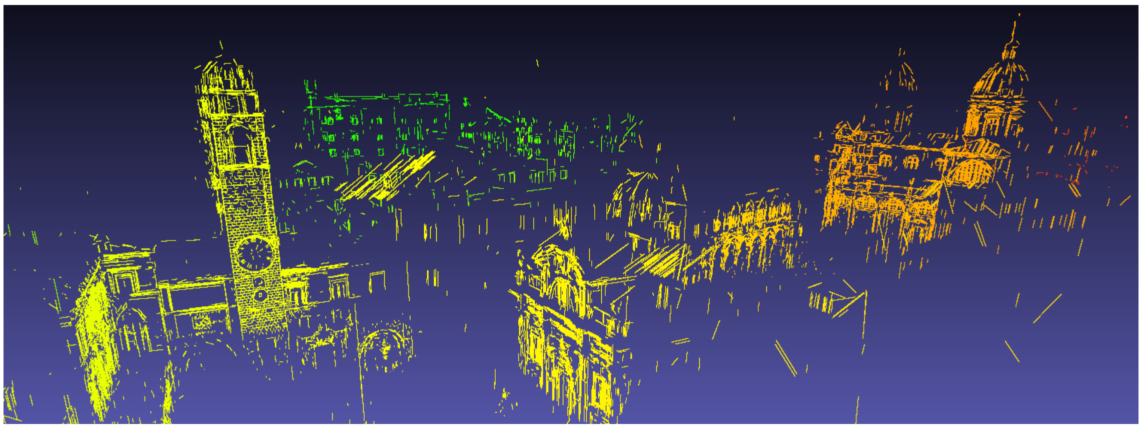
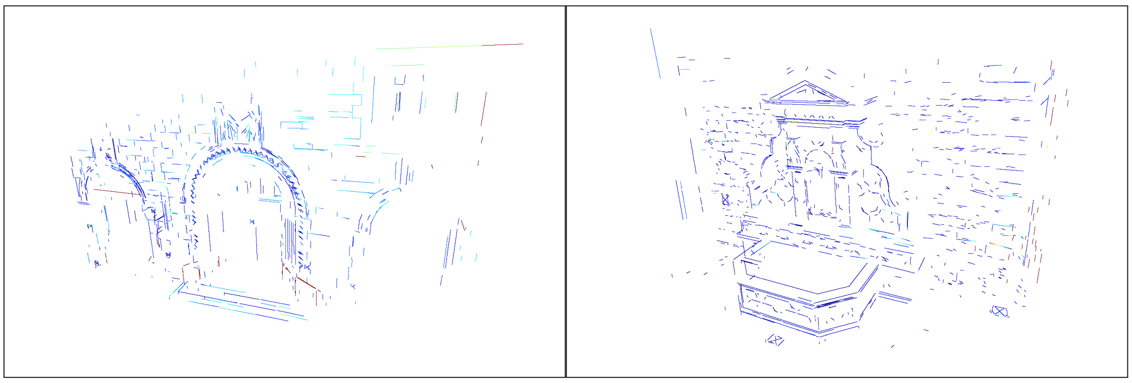


Figure 4.6: Reconstruction result for the large crowdsourced *Dubrovnik6K* [97] dataset (6844 images; $\mu = 0.025$; 21,040 3D lines). The total runtime was approximately 90 minutes (excluding *SfM*).

(RMSE) to the ground truth surface, using the Hausdorff distance. Please note that not all valid 3D lines are actually contained in the ground truth. This is especially notable on the railings at the main entrance of *Herz-Jesu-P8* (colored in red).

Figure 4.8 shows an evaluation on the synthetic *Timberframe* dataset, as well a comparison to the original reconstruction result by Jain et al. [73]. As we can see, our method manages to reconstruct the house more densely and with a lower *RMSE* to the ground truth surface. In their paper no explicit runtimes are given, but it is stated that the reconstruction can easily take several hours. Our method finishes in less than a minute, which is mostly due to the more efficient matching procedure and the massive parallelism.

Results for the remaining two datasets (*Herz-Jesu-P25* and *Castle-P30*), which do not have a groundtruth surface mesh available, can be seen in Figure 4.9. Please note that all our reconstruction results above where obtained with bundle adjustment enabled. We will analyze how the optimization procedure affects the quality of the line-based 3D models, as well as the accuracy of the camera poses, in Section 4.6.



Herz-Jesu-P8 [$\mu = 0.025$]
RMSE: 5.21 cm
870 segments
4.84 seconds

Fountain-P11 [$\mu = 0.025$]
RMSE: 3.34 cm
1,029 segments
7.19 seconds

Figure 4.7: Quantitative evaluation on the *Herz-Jesu-P8*, and the *Fountain-P11* datasets [146]. The *RMSE* is computed for densely sampled points on the 3D lines to the ground truth mesh. The colors indicate the *RMSE* using a linear color mapping (blue: 0.0, green: 0.05, red: ≥ 0.1 m).

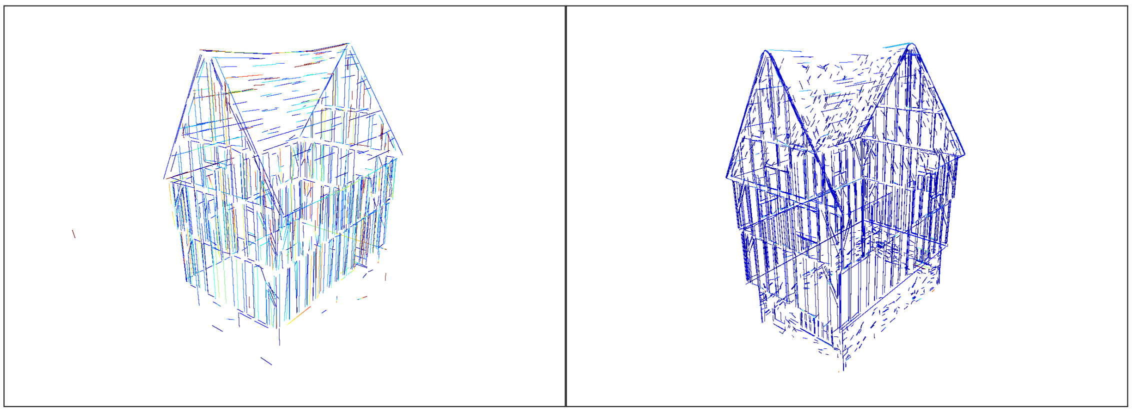
4.5 Parameter Evaluation

In this section, we evaluate different parameters for each reconstruction step individually, and analyze how different choices effect the reconstruction output. All remaining parameters (apart from the one being analyzed) are set to their default values, as discussed in Section 4.2.

4.5.1 Line Segment Detection

As line segment detector we chose the popular *LSD* algorithm [51], which does not require any parameter tuning, and for which several efficient and robust implementations are available (e.g. in OpenCV 3). As stated in Section 3.3, we only use the κ longest 2D line segment for reconstruction, but only if they are longer than ρ times the image diagonal ($\kappa = 3000$, $\rho = 0.005$; see Table 4.1). In most cases, the number of detected line segments that are long enough is well below 3000, which means that κ is not a very important parameter, and just ensures a certain worst-case performance and memory consumption. Therefore, we do not perform an evaluation on κ, since the results do not really change when it is modified, unless we would set it to an unreasonably small number.

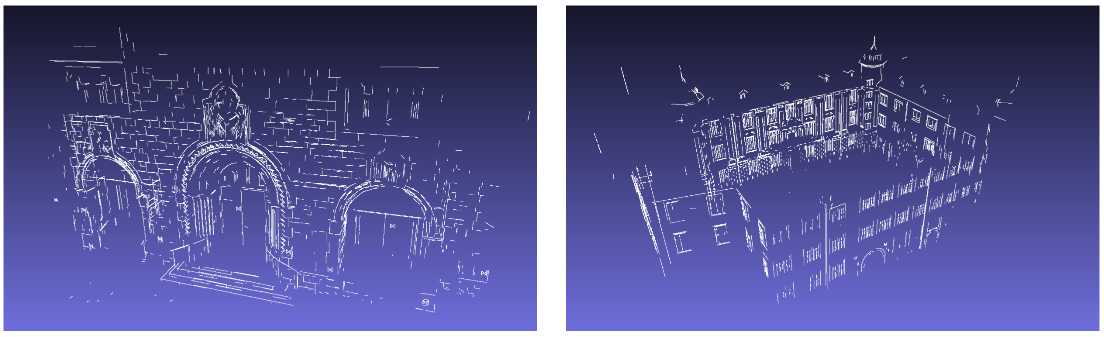
However, since we have proposed to scale down high-resolution images to smaller sizes for a more efficient line segment detection in our previous work [61–63, 65], we perform an evaluation how line segment detection on the full-size versus the reduced images affects the completeness of the results, as well as the runtime. For this experiment, we reduced the images to a constant *width* of 1280 ($\approx$ HD ready) and 1920 ($\approx$ FullHD) pixels, and compared to the full resolution results from Section 4.4.1 and 4.4.2 (*width* in this con-



Jain et al. [73]
RMSE: 16.14 cm
828 segments
n.a. (≈ several hours)

Line3D++ [$\mu = 0.05$]
RMSE: 3.85 cm
3,694 segments
37.70 seconds

Figure 4.8: Quantitative evaluation on the *Timberframe* [73] synthetic dataset (240 images). The *RMSE* is computed for densely sampled points on the 3D lines to the ground truth *CAD* model. The colors indicate the *RMSE* using a linear color mapping (blue: 0.0, green: 0.25, red: ≥ 0.5 m). On the left, we see the original result by [73]. On the right, we see the result obtained using *Line3D++*.



Herz-Jesu-P25 [$\mu = 0.025$]
2,210 segments
21.53 seconds

Castle-P30 [$\mu = 0.05$]
3,188 segments
22.02 seconds

Figure 4.9: Reconstruction results for the *Herz-Jesu-P25* (25 images), and the *Castle-P30* (30 images) datasets [146].

text always refers to the larger image dimension). After the line segment detection, the coordinates are always upscaled to the full image dimensions.

Table 4.4 shows the evaluation result for all datasets, except the *Timberframe* sequence, where all images already have a width of 1280 pixels. The speed-up is simply defined as

$$\text{speed-up}_w = \frac{t_{ref}}{t_w}, \tag{4.2}$$

where t_{ref} is the reconstruction time using the full-sized images, and t_w is the reconstruction time using the resized images. The reconstruction results can be seen in Figure 4.10 for the real-world-, and in Figure 4.11 for the ground-truth datasets.

	w = 1280 (≈ HD ready)				**w = 1920** (≈ FullHD)				**Full Size**			
Dataset	time t [s]	$\varnothing t$ [s]	# lines	Speed-up	time t [s]	$\varnothing t$ [s]	# lines	Speed-up	time t [s]	$\varnothing t$ [s]	# lines	*width*
Building	129.18	0.38	2,346	2.80×	206.16	0.60	6,268	1.75×	361.17	1.05	11,238	4912
Pylon	20.63	0.19	1,340	3.65×	31.42	0.30	3,071	2.40×	75.27	0.71	5,319	5616
Façade	57.19	0.18	3,537	2.83×	84.84	0.27	5,462	1.91×	162.02	0.51	6,767	3648
Kitchen	9.89	0.16	708	3.59×	13.38	0.22	1,062	2.66×	35.53	0.57	1,118	6000
HJ-P25 [146]	13.12	0.52	1,679	1.64×	17.47	0.70	2,223	1.23×	21.53	0.86	2,210	3072
HJ-P8 [146]	2.96	0.37	580	1.64×	4.01	0.50	838	1.21×	4.84	0.61	870	3072
C-P30 [146]	8.67	0.29	1,610	2.54×	16.98	0.57	2,927	1.30×	22.02	0.73	3,188	3072
F-P11 [146]	4.42	0.40	833	1.63×	6.37	0.58	1,086	1.13×	7.19	0.65	1,029	3072
Mean		**0.29**		**2.37**×		**0.43**		**1.62**×		**0.65**		

Table 4.4: Evaluation results for the line segment detection step (see Section 3.3). The corresponding reconstruction results can be seen in Figures 4.10 and 4.11.

As we can see, for the majority of the cases the completeness of the resulting 3D models does not suffer largely when the images are resized to approximate FullHD resolution, while the quality significantly drops when the images are made even smaller. This is especially noticeable for the *Pylon* and the *Building* model (Figure 4.10), where large parts of the reconstruction are missing when the images are reduced.

The average speed-up is about 2.4 times for HD ready, and about 1.6 times for FullHD resolution. However, if we look at the absolute numbers we see that this is only relevant for the larger image sequences, such as *Building*, where the total execution time drops from 361.17 ($\varnothing t$ = 1.05 s) to 129.18 seconds ($\varnothing t$ = 0.38 s). But even here the absolute difference is rather negligible, compared to the execution time of the *SfM* pipeline that is executed beforehand.

Table 4.5 shows a quantitative accuracy evaluation on the *Herz-Jesu-P8* and the *Fountain-P11* groundtruth datasets [146]. As we can see, the accuracy of the 3D model is very similar for all image sizes, with only small differences in the millimeter range. To conclude, in general it is beneficial to detect the line segments on higher resolutions, which benefits the completeness and the accuracy of the reconstructions, without severely affecting the runtime efficiency. However, when the available execution time is limited it makes sense to slightly reduce the image sizes, as long as the reduction is not too severe.

	w = 1280 (≈ HD ready)		**w = 1920** (≈ FullHD)		**Full Size** (w = 3072)	
Dataset	*RMSE* [cm]	mean err. [cm]	*RMSE* [cm]	mean err. [cm]	*RMSE* [cm]	mean err. [cm]
Herz-Jesu-P8	6.83	3.22	6.04	2.92	**5.21**	**2.07**
Fountain-P11	3.97	1.30	**3.32**	**1.08**	3.34	1.14

Table 4.5: Quantitative evaluation results for the line segment detection step, on the two groundtruth datasets *Herz-Jesu-P8* and *Fountain-P11* [146].

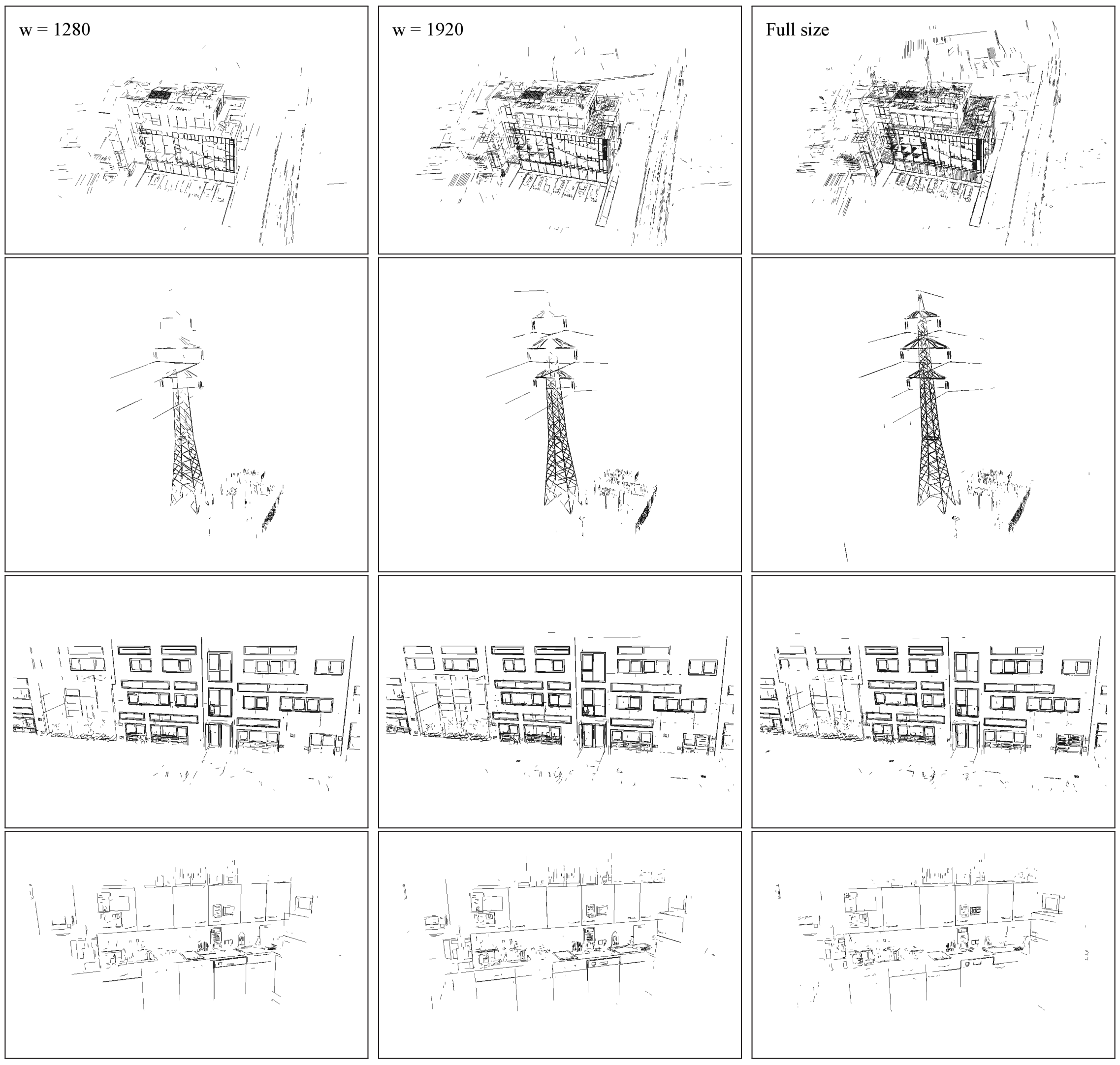


Figure 4.10: Reconstruction results on the real-world datasets, for different image-sizes during line segment detection. The corresponding numbers can be found in Table 4.4.

4.5.2 Line Matching

The line matching step (see Section 3.4) depends on three parameters, which are the maximum number of visual neighbors ($M = 10$), the epipolar overlap threshold ($\tau = 0.25$), and the number of matches per 2D segment and neighboring image which are kept ($k = 10$). We divide the parameter evaluation into two steps. First, we evaluate the epipolar overlap threshold τ on its own, and second, we evaluate M and k in combination. Therefore, we perform a quantitative evaluation on how changing k affects recall and precision of the matching procedure on all datasets, followed by a demonstration on how various M and k pairs influence the reconstruction results for the *Pylon* sequence visually.

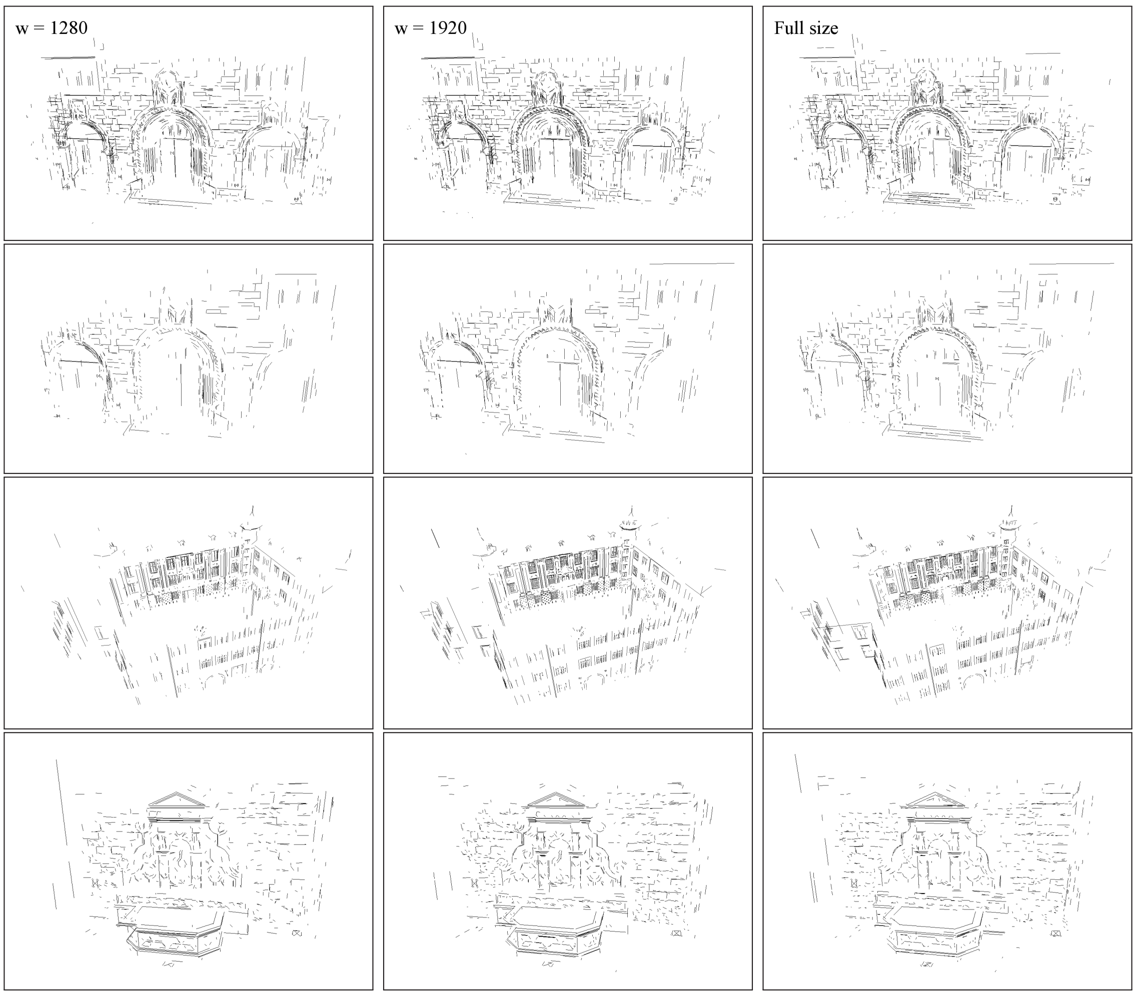


Figure 4.11: Reconstruction results on the groundtruth datasets, for different image-sizes during line segment detection. The corresponding numbers can be found in Table 4.4.

4.5.2.1 Evaluating τ

The default value for τ is 0.25, which means that the mutual support of two potentially matching 2D line segments has to be at least 25% (with respect to the epipolar geometry). Our experiments showed that this value is a fair compromise between robustness to occlusions, and matching precision, since it does not result in too many outlier matches. As an evaluation, we run our algorithm with four different values ($\tau \in \{0.01, 0.25, 0.50, 0.90\}$), which range from a very low threshold ($\tau = 0.01$), that accepts almost everything as long as there is at least some overlap, to a very high threshold ($\tau = 0.90$), that expects an almost complete compliance, and hence no severe occlusions are tolerated. For this experiment, we set $k = \infty$ (which means no k-nearest-neighbor matching; see Section 3.4.3), such that the effects of modifying τ can be properly analyzed.

Figure 4.12 shows the resulting 3D models, and Table 4.6 the corresponding numbers.

As expected, the number of reconstructed 3D lines decreases when τ is increased, since a lower number of matches is accepted. This essentially means that the tolerance to imprecise line segment detections and occlusions goes down with higher values for τ. It is noticeable that up to $\tau = 0.5$ (i.e. 50% occlusion tolerance) the results are fairly stable, and the completeness of the 3D models does not change dramatically. However, when a very extreme value is chosen ($\tau = 0.9$), the 3D models are very unsatisfying, and a lot of relevant parts of the scene are missing. As a conclusion, it is beneficial to chose lower values for τ since the chance of missing correct matches is decreased, while the risk of introducing outliers is still very small. Analogue to the number of reconstructed 3D lines, the runtime is of course higher when τ is smaller, since more matches have to be processed in later stages of the algorithm. However, the mean per-image runtime only varies between 0.88 and 0.64 seconds, which further undermines that τ should be kept low, unless one needs to process a very large dataset with a low time budget.

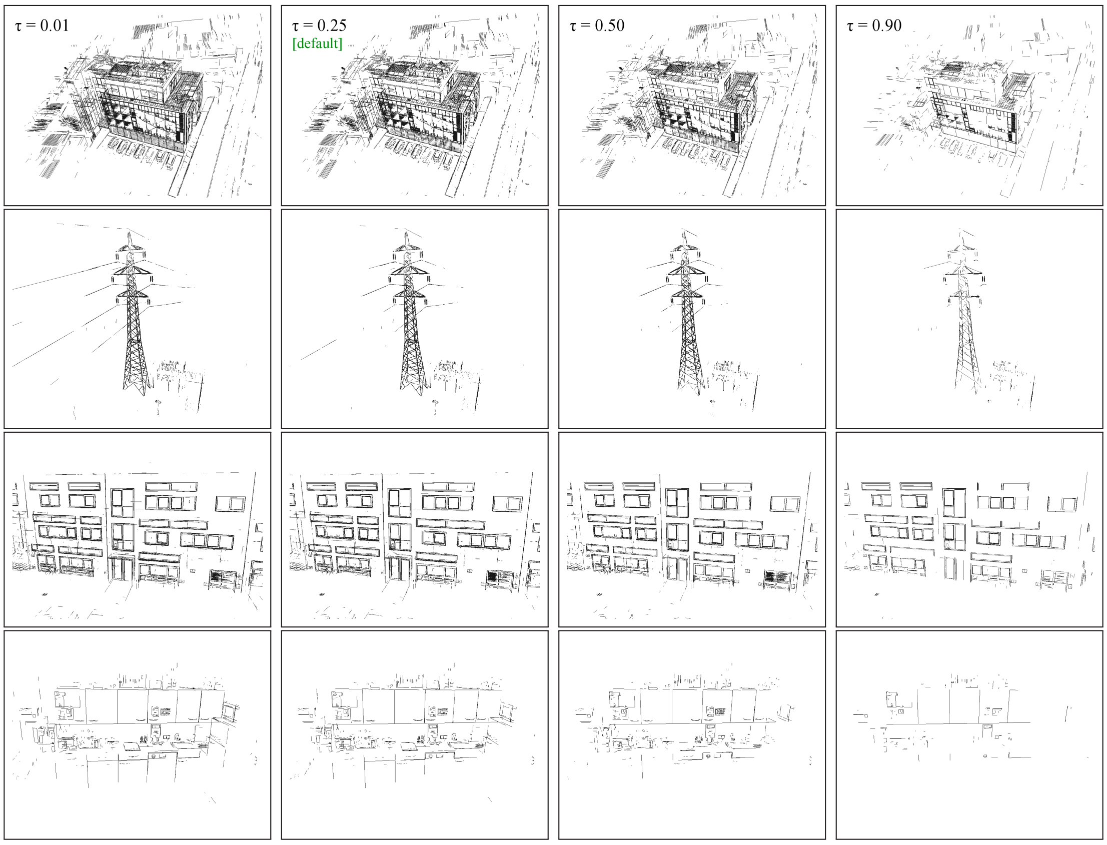


Figure 4.12: Reconstruction results for different τ values during the line matching evaluation, on the real-world datasets. The corresponding numbers can be found in Table 4.6.

	$\tau = \mathbf{0.01}$			$\tau = \mathbf{0.25}$ (default)			$\tau = \mathbf{0.50}$			$\tau = \mathbf{0.90}$		
Dataset	time t [s]	$\varnothing t$ [s]	# lines	time t [s]	$\varnothing t$ [s]	# lines	time t [s]	$\varnothing t$ [s]	# lines	time t [s]	$\varnothing t$ [s]	# lines
Building	497.32	1.45	15,580	385.36	1.12	15,248	338.18	0.98	12,021	319.41	0.93	5,227
Pylon	87.83	0.83	5,280	78.87	0.74	5,663	72.34	0.68	5,292	68.71	0.65	1,497
Façade	208.61	0.66	8,157	163.96	0.52	7,730	146.80	0.46	5,673	143.31	0.45	2,241
Kitchen	35.92	0.58	1,176	34.93	0.56	1,180	33.84	0.55	919	32.84	0.53	223
Mean		**0.88**			**0.74**			**0.67**			**0.64**	

Table 4.6: Line matching evaluation results for various τ values, with $k = \infty$ (see Section 3.4). The corresponding reconstruction results can be seen in Figure 4.12.

4.5.2.2 Evaluating M and k

The default value for M as well as k is 10, which means that each image is matched with its $M = 10$ nearest visual neighbors (see Section 3.4.1), and for each 2D segment the $k = 10$ best matches are kept in each of the selected visual neighbor images (see Section 3.4.3). Hence, each 2D segment has at most $M \cdot k = 100$ potential matches.

As a qualitative evaluation of the parameter k, we perform our geometric matching procedure for randomly selected image pairs from all datasets (by only allowing pairs which are actually visual neighbors), for various k values ($k \in \{1, 5, 10, 20, \infty\}$). We then calculate the

$$\text{recall} = \frac{\#\text{true\_positives}}{\#\text{true\_positives} + \#\text{false\_positives}}, \tag{4.3}$$

and the

$$\text{precision} = \frac{\#\text{true\_positives}}{\#\text{true\_positives} + \#\text{false\_negatives}}, \tag{4.4}$$

considering only the 100 longest line segments in each image (since manually evaluating all 3000 segments would be quite exhausting, and we do not have labelled groundtruth matches available). Also, we only include 2D segments into the measurements which are not occluded in the opposite image, such that the correct match could actually be found. This is a reasonable thing to do, since occluded segments cannot be matched by any line matching method. Table 4.7 shows the evaluation results.

As we can see, when $k = 1$ (nearest-neighbor matching) we end up with a fairly low recall (and precision) of only 55%, which means that the miss the correct match almost half of the time. However, the speed-up compared to the baseline ($k = \infty$) is with more than 30 times quite large, where speed-up simply relates to the number of resulting matches which need to be post-processed later on. When we increase k to 5, the recall quickly increases to almost 80% (with a precision of around 16%), which is already quite reasonable for proper reconstruction results. The speed-up drops to approximately 6 times, which is of course to be expected, since we have (at most) five times more matches to process. When we further increase k to 10 (default) and 20, the recall jumps to 90% and 93% respectively. Compared to the 11% increase in recall between $k = 5$ and $k = 10$, the difference of just 3% between $k = 10$ and $k = 20$ is quite insignificant in practice. When we finally set $k = \infty$ we only gain additional 3% in recall, which suggests that there is not much to be gained from setting $k > 20$. As a conclusion, we use $k = 10$ by default, which is still about three times faster than the baseline, with a reasonably high recall. However, if runtime is not an issue one can always use larger k values (or $k = \infty$), to minimize the risk of missing a correct match.

	k = 1		**k = 5**			**k = 10** (default)			**k = 20**			**k = ∞**	
Dataset	Rec./Prec.	Speed-up	Rec.	Prec.	Speed-up	Rec.	Prec.	Speed-up	Rec.	Prec.	Speed-up	Rec.	Prec.
HJ-P25 [146]	0.64	31.4×	0.79	0.16	6.4×	0.88	0.09	3.4×	0.93	0.05	1.7×	0.95	0.03
C-P30 [146]	0.45	31.9×	0.63	0.14	7.1×	0.79	0.09	4.0×	0.83	0.05	2.2×	0.89	0.03
F-P11 [146]	0.64	20.3×	0.94	0.19	4.1×	1.00	0.10	2.1×	1.00	0.06	1.3×	1.00	0.05
TF [73]	0.41	25.4×	0.80	0.17	5.3×	0.96	0.09	2.7×	1.00	0.06	1.4×	1.00	0.04
Building	0.75	64.4×	0.82	0.16	12.9×	0.93	0.09	6.4×	0.98	0.05	3.2×	1.00	0.02
Pylon	0.48	27.1×	0.86	0.17	5.4×	0.96	0.10	2.7×	0.98	0.06	1.6×	0.98	0.04
Façade	0.33	32.4×	0.67	0.14	6.3×	0.81	0.08	3.2×	0.85	0.05	1.7×	0.96	0.03
Kitchen	0.72	11.9×	0.81	0.18	2.7×	0.88	0.11	1.5×	0.88	0.08	1.1×	0.88	0.07
Mean	**0.55**	**30.6×**	**0.79**	**0.16**	**6.3×**	**0.90**	**0.09**	**3.2×**	**0.93**	**0.06**	**1.8×**	**0.96**	**0.04**

Table 4.7: Quantitative line matching evaluation results for various k values (see Section 3.4.3). *Rec.* stands for recall, and *Prec.* for precision.

To give a visual impression, we also evaluate the combined effect of M and k on the *Pylon* sequence, by selecting $M \in \{5, 10, 20, \infty\}$, and $k \in \{1, 5, 10, \infty\}$. The reconstruction results are shown in Figure 4.13 , and the corresponding numbers in Table 4.8. As we can see, varying k does not affect the runtime by a very large margin for this mid-sized dataset (106 images). However, the reconstruction results are significantly better and more complete when higher values for k are used. It is noticeable that setting $k = 10$ (default) and $k = \infty$ produces nearly identical results (also regarding the number of reconstructed segments), with a slight runtime benefit for $k = 10$. In addition, we can also see here that nearest-neighbor matching ($k = 1$) is not very suitable for our matching procedure, since we often encounter severe occlusions, which means that the correct match might not have the highest epipolar overlap.

Similar observations can be made for the choice of M. The results are noticeably better when each image is matched with more visual neighbors, but the runtime of course increases with each additional matching procedure. When $M = 10$ (default), the results are virtually identical to $M = 20$ and $M = \infty$, with a lower runtime. As a final conclusion, given a high enough time budget it is always beneficial to increase M and k, to get the best result possible. However, the default values deliver results close to the optimum, with a bounded computational complexity.

4.5.3 Match Evaluation

The evaluation (scoring) of the obtained matches (see Section 3.5) depends on two parameters. The first is the angular regularizer σ_a (by default set to 10°), and the second is a depth-adaptive spatial regularizer, which is either defined directly in 3D space (μ), or scale-invariantly in pixels (σ). As we have stated in Section 4.2, the angular regularizer can be kept constant since angles are inherently scale invariant. The more challenging

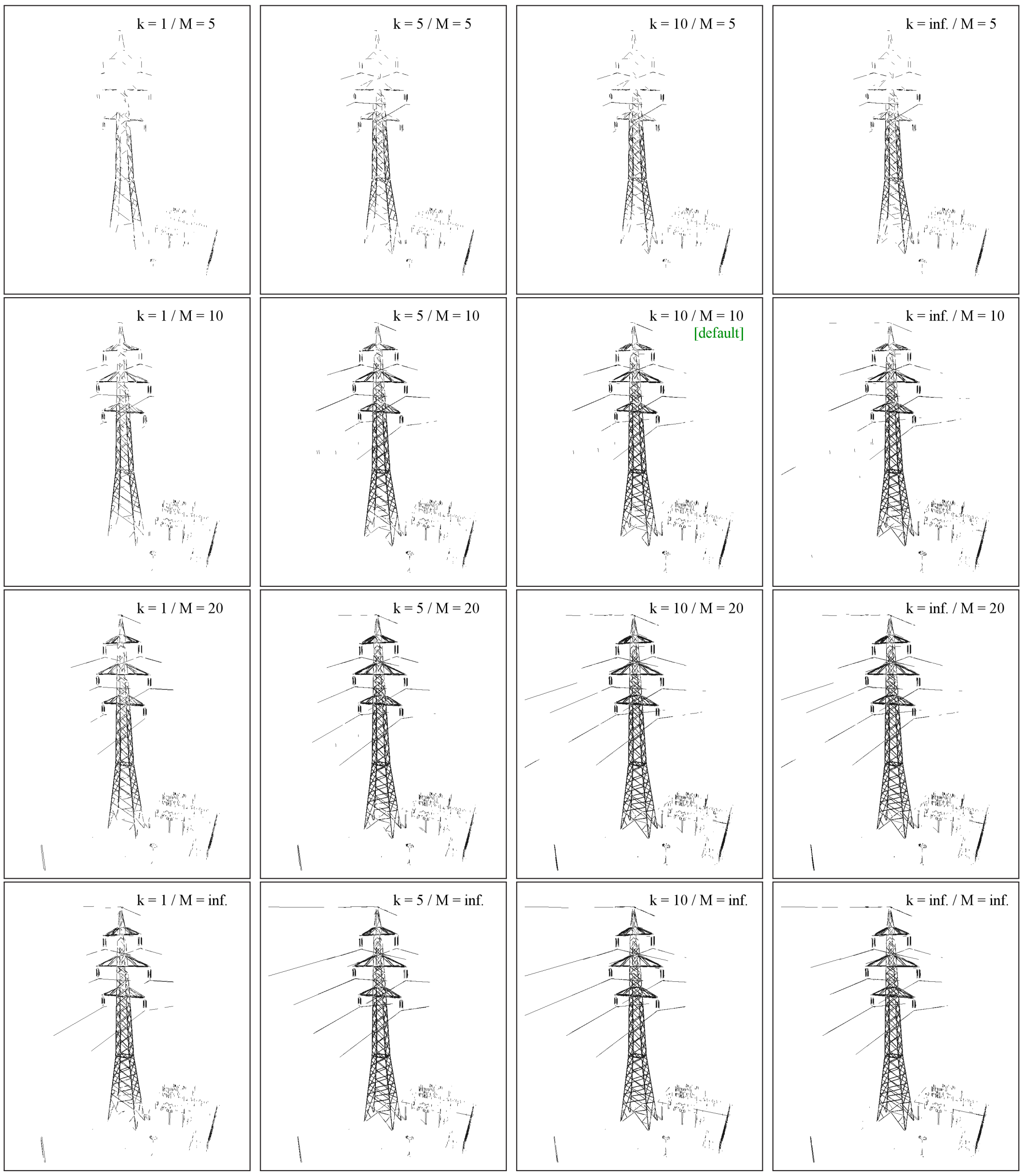


Figure 4.13: Reconstruction results for four different M and k values, demonstrated on the *Pylon* sequence. The corresponding numbers can be found in Table 4.8.

task is to select a proper spatial regularizer, since it depends on multiple factors (scale of the reconstruction, image sizes, accuracy of the camera poses, ...). Hence, we cannot simply define a default parameter that makes perfect sense for all scenarios.

When the scale is known, selecting a proper value is fairly straightforward (e.g. setting μ to a few centimeters usually works fine for the reconstruction of buildings and other outdoor structures). However, when we do not know the reconstruction scale, the selection of a proper value for σ is slightly more complicated. In Table 4.3, we have specified working

	$\mathbf{k=1}$ (nearest-neigh.)			$\mathbf{k=5}$			$\mathbf{k=10}$ (default)			$\mathbf{k=\infty}$		
Pylon	time t [s]	$\varnothing t$ [s]	# lines	time t [s]	$\varnothing t$ [s]	# lines	time t [s]	$\varnothing t$ [s]	# lines	time t [s]	$\varnothing t$ [s]	# lines
$\mathbf{M=5}$	60.10	0.57	866	60.09	0.57	1,984	60.10	0.57	2,402	61.73	0.58	2,782
$\mathbf{M=10}$ (def.)	70.63	0.67	2,100	72.85	0.69	4,584	75.27	0.71	5,319	78.56	0.74	5,663
$\mathbf{M=20}$	92.70	0.87	2,848	99.14	0.94	5,215	101.87	0.96	5,317	110.13	1.04	5,160
$\mathbf{M=\infty}$	110.37	1.04	3,248	118.07	1.11	4,590	127.51	1.20	4,560	142.83	1.35	4,385

Table 4.8: Evaluation results on the *Pylon* sequence, for four different M and k values in the line matching step (see Sections 3.4.1 and 3.4.3). The corresponding reconstruction results can be seen in Figure 4.13.

σ values for our real-world datasets, which provide reasonable results. As we can see, the values vary from 2.5 to as much as 15 pixels. For the general case, a value in the one digit pixel range is usually fine, especially when we deal with high-resolution outdoor datasets, where the distance between the objects and the images is generally larger than in indoor datasets. However, to demonstrate the affect of modifying σ, we perform an experiment on our real-world datasets, where we set the previously chosen value for σ from Table 4.3 to $\sigma/2$ and 2σ. Figure 4.14 shows the corresponding reconstruction results.

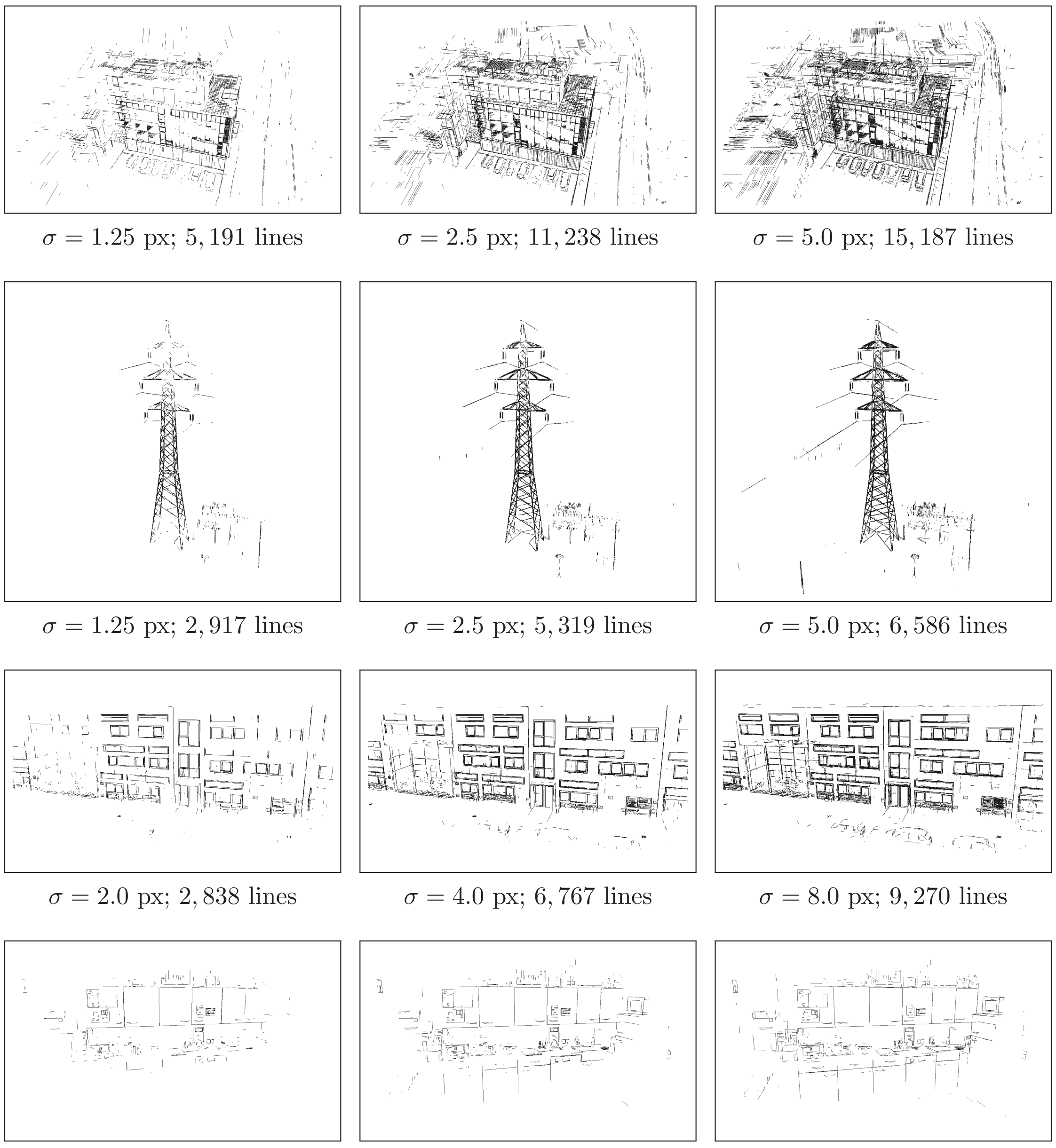
As we can see, lower values for σ naturally lead to sparser 3D models, while higher values produce a more complete output. In all cases, no outliers were introduced, even when $\sigma = 30$ pixels for the *Kitchen* sequence. However, this is not very surprising since all the chosen values are still in a very reasonable range (given the image sizes). To demonstrate the effect of unreasonably large σ values, we ran the *Pylon* testcase with $\sigma \in \{10, 25, 50\}$. A detail view on the top of the pylon can be seen in Figure 4.15 (a-c).

As we can observe, the models are still fairly accurate, with hardly any severe outliers present. However, the results do get quite noisy, especially when $\sigma = 50$ pixels. What is interesting is, that the number of reconstructed 3D lines actually decreases with higher values for σ. This is due to the fact that a higher spatial regularizer might fuse 2D segments that are actually not originating from the same 3D structure, but are just close together (i.e. the main structural parts of the pylon). A full view of the scene can be seen in Figure 4.15 (d).

So far, there is no automatic way to derive σ (or μ) for every possible scenario. However, as a rule of thumb, one digit pixel values are a good initial guess for σ (lower for outdoor-, and higher for indoor scenes), and a few centimeters should be reasonable for μ. If the initial guess was incorrect, re-running *Line3D++* with a different value is usually not a problem, due to the algorithm's high efficiency.

4.6 Bundle Adjustment Evaluation

To evaluate the potential influence of the reconstructed line segments on the accuracy of the overall 3D reconstruction (including the *SfM* step), we re-bundle the whole model using both the 3D points from the *SfM*, and the 3D lines from our proposed method. We then perform two separate evaluations. First, we evaluate the accuracy of the optimized



$\sigma = 1.25$ px; 5,191 lines — $\sigma = 2.5$ px; 11,238 lines — $\sigma = 5.0$ px; 15,187 lines

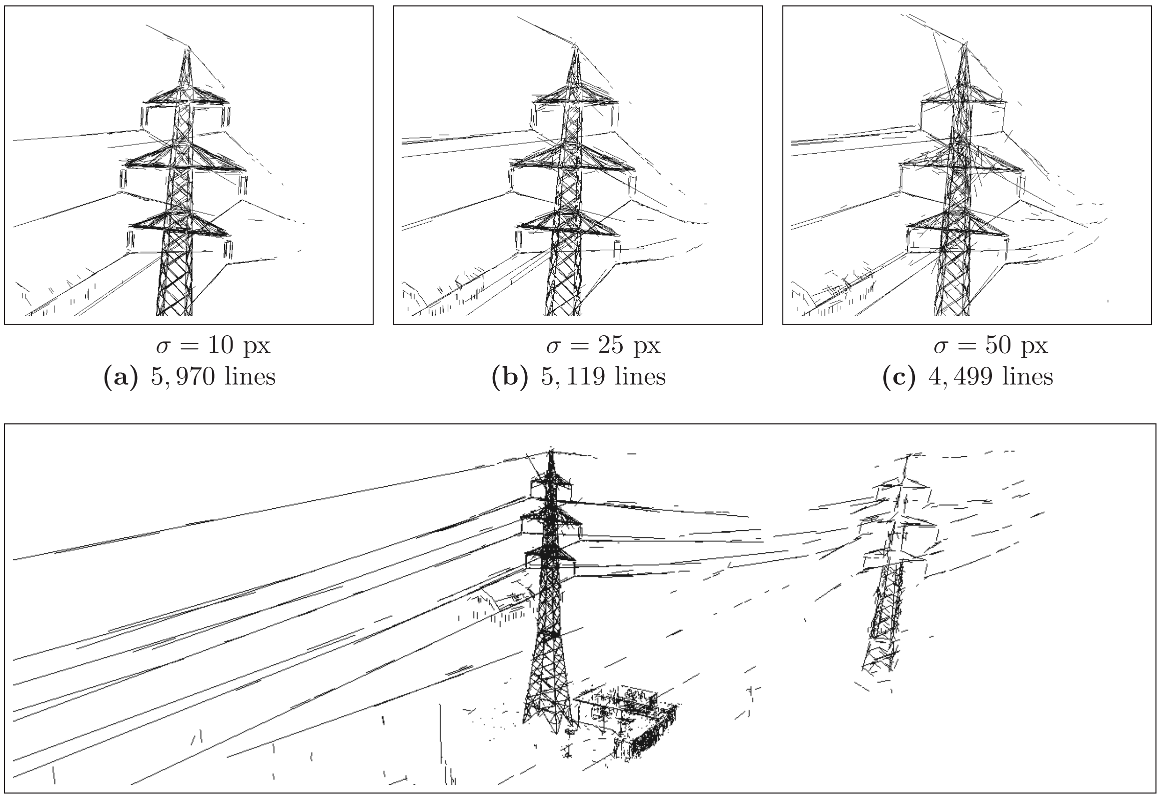
$\sigma = 1.25$ px; 2,917 lines — $\sigma = 2.5$ px; 5,319 lines — $\sigma = 5.0$ px; 6,586 lines

$\sigma = 2.0$ px; 2,838 lines — $\sigma = 4.0$ px; 6,767 lines — $\sigma = 8.0$ px; 9,270 lines

$\sigma = 7.5$ px; 649 lines — $\sigma = 15.0$ px; 1,118 lines — $\sigma = 30.0$ px; 1,180 lines

Figure 4.14: Reconstruction results for different σ values during the match evaluation step.

camera poses in comparison to the purely point-based *SfM*. And second, we evaluate the accuracy of the optimized 3D line models in comparison to unoptimized reconstructions (i.e. stopping after the clustering step described in Section 3.7.1). As a reference, we use the true camera poses provided by our groundtruth datasets, as well as the groundtruth surface models.



$\sigma = 10$ px
(a) $5,970$ lines

$\sigma = 25$ px
(b) $5,119$ lines

$\sigma = 50$ px
(c) $4,499$ lines

(d) Full scene ($\sigma = 50$ px)

Figure 4.15: Reconstruction results for extreme σ values on the *Pylon* sequence.

4.6.1 Camera Pose Accuracy

To evaluate the effect of the combined bundling procedure on the camera poses, we compare the re-bundled poses to the poses from the basic (points only) *SfM* result [72] (5000 *SIFT* [101] features; see Section 4.1.1), as well as to a different *SfM* result with more feature points per image (# *SIFT* $= 5000 + \kappa = 8000$). This additional comparison evaluates how the appearance as well as the accuracy of the *SfM* results changes when simply more feature points are used, rather than when points and lines are used in combination.

Table 4.9 shows the evaluation results. The error computation is dived in two parts, the positional error (in centimeters) and the angular error (in degrees). The positional error is simply the absolute deviation of the reconstructed camera centers from the groundtruth camera positions, and the angular error is the angle between the optical axes of the reconstructed- and the groundtruth camera poses. As we can see, for most cases (apart from *Fountain-P11*) the combined bundle adjustment results in the lowest positional errors. This might of course also be a result of the higher number of 3D entities to be bundled (i.e. more 3D information). It is observable throughout all testcases that a higher number of *SIFT* [101] features does not really result in denser point clouds (i.e. the number of reconstructed 3D points stays roughly the same), while adding the reconstructed lines significantly increases the available 3D entities for optimization. However, all in all the improvement on the positional errors is rather small, and lies somewhere in the millimeter or low centimeter range for all datasets, since the underlying *SfM* pipeline is already quite accurate [72].

For the angular error, the results are less conclusive. All three methods achieve the best scores (lowest errors) for some of the testcases, but with very small absolute differences among them. All in all, the angular errors are extremely low, with the maximum error being always well below one degree (apart from *Castle-P30*, where the maximum errors are slightly above one degree). Figure 4.16 shows comparative reconstruction results for *Herz-Jesu-P25* and *Castle-P30*, for all three reconstruction methods. As we can see, the point clouds with 5000 versus 8000 *SIFT* features are virtually identical, while points and lines in combination give a much better visual impression of the scene, with a higher semantic meaning.

	SfM [72] (# *SIFT* = 5,000)					***SfM*** [72] (# *SIFT* = 8,000)					**Comb.** (# *SIFT* = 5,000; # LINES = 3,000)				
	Pos. Err. [cm]		Ang. Err. [deg]		$\|\Omega\|$	Pos. Err. [cm]		Ang. Err. [deg]		$\|\Omega\|$	Pos. Err. [cm]		Ang. Err. [deg]		$\|\Omega\| + \|\Pi\|$
Dataset	max	mean	max	mean		max	mean	max	mean		max	mean	max	mean	
HJ-P25 [146]	1.611	0.611	0.155	0.070	12,520	1.435	0.599	**0.123**	**0.049**	12,311	**1.207**	**0.495**	0.139	0.074	14,730
HJ-P8 [146]	0.692	0.425	0.279	0.253	4,924	0.669	0.424	**0.273**	0.253	4,924	**0.658**	**0.404**	0.276	**0.252**	5,794
C-P30 [146]	62.244	8.093	**1.217**	0.263	17,386	65.461	8.760	1.269	**0.252**	17,688	**62.077**	**7.707**	1.219	0.265	20,574
F-P11 [146]	**0.472**	**0.309**	0.101	0.084	7,365	0.476	0.324	0.113	0.084	7,357	0.572	0.354	**0.091**	**0.070**	8,394
TF [73]	8.453	3.709	**0.205**	**0.061**	43,413	9.930	3.393	0.227	0.064	43,579	**7.910**	**2.817**	0.245	0.082	47,107

Table 4.9: Evaluation results for the combined bundle adjsutment optimization (see Section 3.8). The corresponding reconstruction results for *Herz-Jesu-P25* and *Castle-P30* can be seen in Figure 4.16.

4.6.2 Line Model Accuracy

Table 4.10 shows a quantitative evaluation of the bundled- and unbundled 3D line models, with respect to the groundtruth surfaces of *Herz-Jesu-P8*, *Fountain-P11*, and *Timberframe*. For *Herz-Jesu-P8* and *Timberframe* the bundling procedure improves the accuracy of the 3D reconstruction, while the unbundled result is more accurate for *Fountain-P11*. This is consistent with the analysis of the bundled camera poses in Section 4.6.1, where *Fountain-P11* is the only dataset which does not benefit from the combined bundle adjustment. However, the differences are at most in the low millimeter range, which further shows that the obtained camera poses using purely point-based *SfM* are already very accurate, and do not benefit by a large amount from a combined optimization at the end.

4.7 Runtime Evaluation: GPU vs. CPU

Several parts of our method can be efficiently executed in parallel. Most notably, the computationally most expensive line segment matching- (Section 3.4) and hypotheses scoring

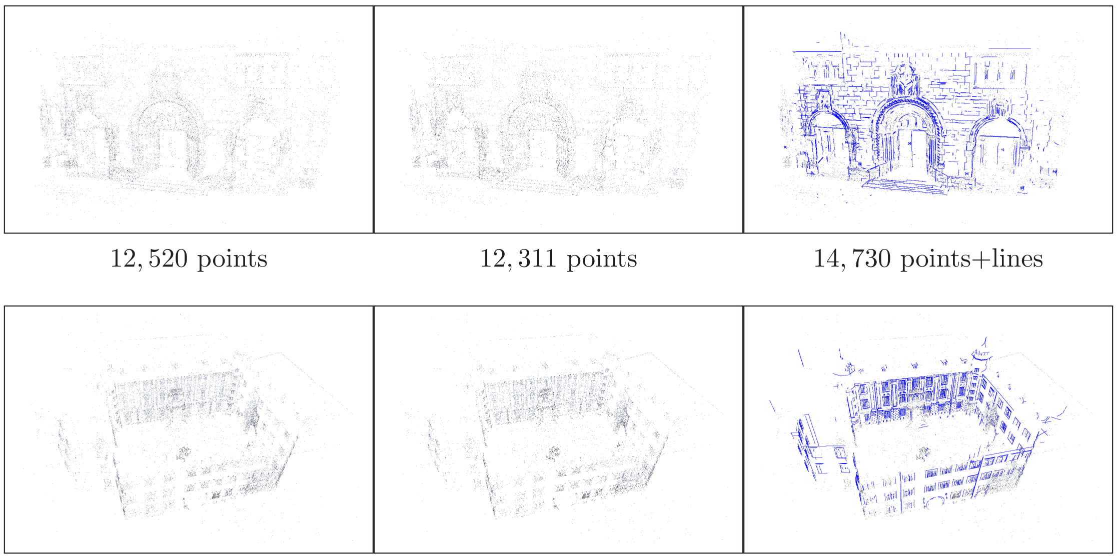


Figure 4.16: Reconstruction results for *Herz-Jesu-P25* (top) and *Castle-P30* (bottom). *Left:* *SfM* [72] with 5000 *SIFT* features, *Middle:* *SfM* with 8000 *SIFT* features, and *Right:* *SfM* and Line3D++ combined. The corresponding numbers can be found in Table 4.9.

	Line3D++ (raw)		**Line3D++** (bundled)	
Dataset	*RMSE* [cm]	mean err. [cm]	*RMSE* [cm]	mean err. [cm]
Herz-Jesu-P8	5.56	2.81	**5.21**	**2.07**
Fountain-P11	**3.30**	**1.11**	3.34	1.14
Timberframe	3.88	2.88	**3.85**	**2.81**

Table 4.10: Quantitative evaluation results for the combined bundle adjustment, on the three groundtruth datasets *Herz-Jesu-P8* [146], *Fountain-P11* [146], and *Timberframe* [73].

steps (Section 3.5) can be easily parallelized, which significantly boosts the performance. In our case, we make use of the *CUDA* [5] framework by nVidia, which of course means that *GPU* support is only available when an nVidia graphics card is installed.

Table 4.11 shows the total execution times on the *CPU* versus the *GPU*, for all test-cases. As we can see, the average speed-up when using the *GPU* is approximately 2.4 times, and the average execution time drops from 1.49 to 0.65 seconds per image. The reconstruction results are virtually identical, there are only slight variations in the number of reconstructed 3D lines. This is mainly due to the different datatypes that are used on the *CPU* and the *GPU*, as well as due to the *CPU*'s higher floating-point precision.

[5] `http://www.nvidia.com/object/cuda_home_new.html`

	Line3D++ (GPU)				**Line3D++** (CPU)		
Dataset	time t [s]	$\varnothing t$ [s]	# lines	Speed-up	time t [s]	$\varnothing t$ [s]	# lines
Building	**361.17**	**1.05**	$11,238$	$2.27\times$	821.37	2.39	$11,096$
Pylon	**75.27**	**0.71**	$5,319$	$1.87\times$	140.97	1.33	$5,299$
Façade	**162.02**	**0.51**	$6,767$	$2.00\times$	324.51	1.02	6.735
Kitchen	**35.53**	**0.57**	$1,118$	$1.46\times$	45.20	0.84	$1,100$
HJ-P25 [146]	**21.53**	**0.86**	$2,210$	$2.81\times$	60.50	2.42	$2,203$
HJ-P8 [146]	**4.84**	**0.61**	870	$2.28\times$	11.04	1.38	871
C-P30 [146]	**22.02**	**0.73**	$3,188$	$2.70\times$	59.55	1.99	$3,224$
F-P11 [146]	**7.19**	**0.65**	$1,029$	$2.21\times$	15.86	1.44	$1,035$
TF [73]	**37.70**	**0.16**	$3,694$	$3.60\times$	135.87	0.57	$3,638$
Mean		**0.65**		$2.36\times$		1.49	

Table 4.11: Runtime evaluation results for using the *GPU* versus the *CPU*.

4.8 Summary

In this chapter, we have shown a variety of line-based 3D reconstructions from several challenging real-world- and groundtruth datasets, using our proposed method. We have further analyzed all individual steps of our line-based 3D reconstruction pipeline separately, and have shown how the respective parameters affect the accuracy and completeness of the resulting 3D models. In addition, we have evaluated the use of a combined bundle adjustment procedure, to further optimize the camera parameters and the 3D data (points and lines) after the reconstruction.

While dense point clouds of course provide more 3D information for most of our datasets (see Figure 4.5), the amount of data that is needed to encode this information is considerably larger than the set of 3D lines obtained by our method. For each of the datasets, more than a million 3D points are reconstructed by *PMVS* [49], which makes further processing of any kind (e.g. meshing and texturing, or semantic scene understanding) relatively tedious on ordinary desktop machines. In contrast, the 3D line models do not exceed more than a few thousand 3D entities. While for some scenarios the dense scene coverage is definitely necessary, there are a lot of applications where a compact 3D line model would be just as (or probably even more) suitable than a large unordered set of points, such as e.g. plane detection, piecewise planar 3D reconstruction, or more specific tasks, such as window detection.

In the following chapters, we will talk about potential applications for our line-based 3D reconstruction algorithm. For the main part, we will show how our method can be easily integrated into an online *SfM* pipeline, for real-time reconstructions already during image acquisition (Chapter 5), followed by a discussion about several potential extensions of our method for future projects (Chapter 6).

5 Application: Online SfM using Points and Lines

Contents

5.1 Online Structure-from-Motion 85
5.2 Incremental Line3D++ 89
5.3 Experimental Results 93
5.4 Summary 99

In the previous chapters, we have introduced and evaluated our line-based Multi-View Stereo (MVS) pipeline for the offline case (i.e. we assumed that a consistent and finalized Structure-from-Motion (SfM) result is available). As a practical application of our work, we now show how a combination of online *SfM* and an incremental version of our algorithm can be easily used to compute camera poses and a combined point-and-line cloud on-the-fly, already during the image acquisition process. We first motivate the use of online *SfM* altogether, by discussing its benefits compared to traditional offline *SfM* pipelines. Afterwards, we show the necessary modifications to make our approach capable of incremental 3D reconstruction, and conclude with several experiments to evaluate accuracy and efficiency of the online- versus the offline version.

The core principles of our incremental reconstruction procedure are based on our related publications on this topic [61, 65]. The final algorithm (as it is presented in this chapter) is not yet published.

5.1 Online Structure-from-Motion

As we have discussed in Section 2.1, there are numerous *SfM* pipelines available to reconstruct unordered and potentially very large image sequences in 3D. However, whether an image set is suitable for an accurate and complete reconstruction does not primarily depend on which *SfM* pipeline is used, but rather on the configuration of the images themselves. To ensure that all (relevant) images are correctly integrated into the 3D model,

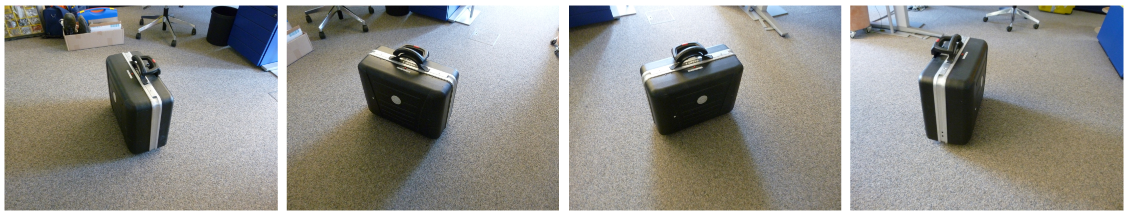


Figure 5.1: An example of an unsuitable image set for *SfM* reconstruction, as taken by a non-expert user. As we can see, the relative camera motion between consecutive images is too large for reliable feature matching.

several constraints must be fulfilled. The most important ones are a *high redundancy-* and a *small camera motion* between consecutive images, as well as a *sufficient coverage* of all parts of the scene that should be reconstructed.

A high redundancy means that matching candidates need to have a high visual overlap, which increases the probability of finding enough corresponding feature points between them. Similarly, a small camera motion between nearby images ensures that the detected features can be reliably matched, since virtually all modern feature point descriptors are only robust to moderate out-of-plane camera rotations (e.g. *SIFT* [101] is known to be quite unreliable for affine transformations beyond 50° [111]). Both of these constraints are of course not completely independent, since a high redundancy somewhat implies a similar camera pose, and vice versa. Sufficient coverage means that all important parts of the scene (i.e. those parts that should be included in the 3D model) need to be covered by more than one image. Ideally, the coverage should be at least three images per target area (the higher the better).

As discussed at length in [68], taking an image set that fulfils these constraints is often hard to achieve even for experienced users. Especially for complex objects it is virtually impossible to judge whether enough images have been taken for a complete reconstruction. This is even more severe when inexperienced users are involved, that are potentially not familiar with the underlying processing steps of an *SfM* pipeline, and therefore lack basic judgement about the usability of the obtained images. Figure 5.1 shows an image set for 3D reconstruction, that has been taken by a non-expert user. Apart from the fact that the object itself is highly unsuitable for *SfM* approaches, due to its textureless surface with almost no distinctive image features, the camera movements between consecutive images are way too large for reliable feature matching, which causes a standard *SfM* pipeline to fail. However, if the *SfM* pipeline is not directly employed on-site (i.e. by using a portable computer), it can easily happen that upon returning to the workstation the user realizes that the image set is insufficient, which means that he has to return to the target scene (which might have changed in the meantime; i.e. movable objects, illumination, ...) to take additional pictures.

Even if the *SfM* pipeline is executed directly on-site, this would still be a highly non-interactive process, which includes long unproductive waiting periods during which the

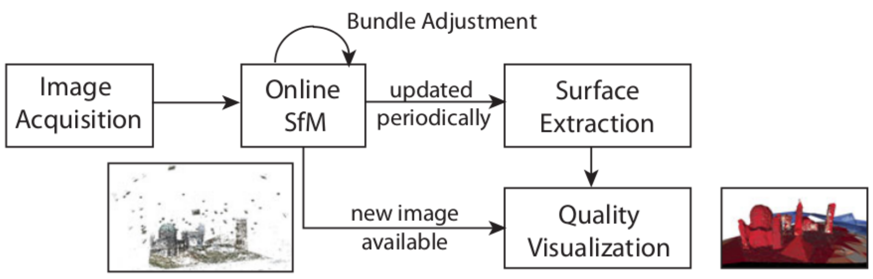


Figure 5.2: A schematic overview of the online *SfM* pipeline proposed by Hoppe et al. [70]. The illustration was taken from the original paper.

computer processes the obtained pictures, and the user has to wait for some feedback. Hence, it would be beneficial if the user would be guided by the *SfM* system, such that he immediately knows if the picture he just took is suitable or not. This was the core motivation for the paper by Hoppe et al. [70], where they introduced the first real online *SfM* pipeline which can not only perform classic point-based *SfM* on-the-fly, but also directly compute a dense 3D surface model, which can then be used to directly visualize the amount of coverage and redundancy for each part of the observed scene. Figure 5.2 shows a schematic illustration of their proposed method.

Basically, the *SfM* part of their approach is an online-capable version of [72]. They start from an initial image pair I_1 and I_2, for which the relative camera poses are computed automatically using *SIFT* [101] feature matches (with the origin being defined as the camera coordinate frame of the first image). For each incoming new image I_t, they use a vocabulary tree to determine a set of similar images $V_t \subset \{1, \cdots, t-1\}$, and try to align I_t to the existing reconstruction by using absolute pose estimation from *SIFT* correspondences [89]. In addition, a periodic bundle adjustment procedure is performed to keep the reconstruction globally consistent [156]. Since they always use the visual neighbors from the vocabulary tree, and do not assume a strict image sequence, they implicitly perform loop closure, and allow a non continuous image acquisition process (i.e. one can stop taking images at any time, and continue at some other part of the scene, as long as this part has already been seen before).

In addition to the sparse *SfM* part of their work, they also compute an incremental 3D surface from the sparse point cloud [69]. They use the obtained triangle mesh to visualize the quality of the reconstruction using a scalable color map. The two possible quality measures are *redundancy* (i.e. the number of cameras that see a certain triangle) and *resolution* (i.e. the ground sampling distance). These measures are especially useful to guide an inexperienced user during the image acquisition, since he directly sees in which parts of the scene he needs to take additional pictures, or in which parts he already has enough pictures for a successful 3D reconstruction.

While there have been other attempts to online *SfM* before (e.g. [115, 116]), the

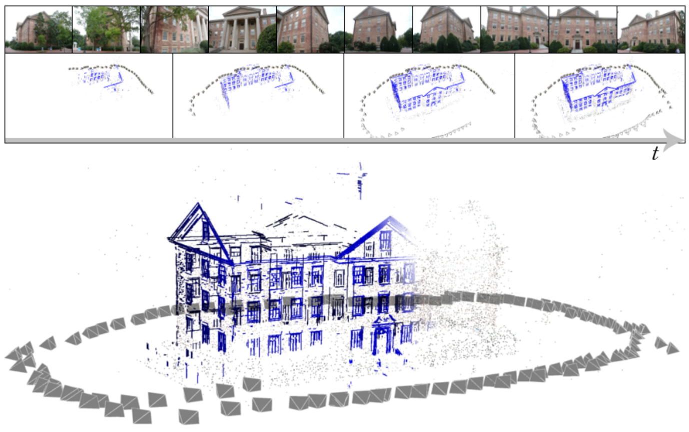


Figure 5.3: Online *SfM* result for a bakestone house (128 images). *Top row*: A visualization of the incremental reconstruction process. *Bottom right*: *SIFT* [101] features only [70], $\varnothing t = 1.07s$. *Bottom left*: Points and lines combined, $\varnothing t = 1.23s$.

method by Hoppe et al. [70] is the only available real online *SfM* pipeline, capable of directly reconstructing an accurate sparse 3D model from high-resolution still images (to the best of our knowledge). However, online *SfM* is of course closely related to visual *SLAM*, which aims et directly estimating 3D structure and camera motion as well. While the core principles between many *SLAM* algorithms and *SfM* pipelines are similar (e.g. *PTAM* [87], where also matched feature points are used), *SLAM* algorithms are usually tuned for real-time computation with high frame rates. Hence, they often operate on a video stream of a lower resolution than e.g. still images from a modern *DSLR* camera. This has the benefit that the baselines between consecutive images are relatively small, which allows an efficient direct camera pose estimation without any kind of explicit and hand-crafted image features [38, 152]. However, the accuracy and level of detail of the obtained 3D models is usually limited by the smaller image resolutions, and potential motion blur. Hence, if the goal is to create the most accurate 3D model rather than navigating through the scene in real-time, it is preferred to use an online capable *SfM* pipeline rather than *SLAM*. This also has the additional benefit that the user in the end has a manageable set of high-quality images for post-processing (e.g. for computing a dense point-cloud), rather than a large video stream.

In the remainder of this chapter, we will demonstrate how we can efficiently integrate our line-based 3D reconstruction method into the online *SfM* pipeline presented in [70]. However, our method could potentially be integrated into any online *SfM* (or potentially also *SLAM*) scheme in the same way, as long as an on-the-fly image alignment and a periodical bundle adjustment are performed. In the next section we will discuss the necessary adaptions to our reconstruction principle presented in Chapter 3, to make our method incremental and quasi real-time capable. For more information about online *SfM* itself we kindly refer to the PhD thesis of Christof Hoppe [68]. Figure 5.3 shows an example reconstruction using a combination of [70] and our proposed method. As we can see, the visual impression is greatly enhanced when line segments and point features are used together, while the average (per image) runtime is only slightly increased.

5.2 Incremental Line3D++

In this section, we describe the process of incremental line-based 3D reconstruction step-by-step, and discuss the necessary adaptions compared to the offline version. In the past, we have published two similar approaches [61, 65], which were also integrated into the online *SfM* pipeline by Hoppe et al. [70]. The core differences to our current work lie mainly in the details, e.g. the usage of a greedy direct clustering approach [65], or additional appearance-based matching constraints [61]. The method described here in this section closely follows the principles of our offline algorithm from Chapter 3, and uses the same basic steps. First, 2D line segments are detected and matched between each new image and its visual neighbors among the already processed images, followed by an assignment of the most probable 3D location for each 2D segment (Section 5.2.1). This procedure is adaptive, i.e. the depth estimate can change when more information is available. Afterwards, the 3D line model is updated using an incremental graph clustering approach (Section 5.2.2). A combined bundle adjustment procedure is performed periodically, to keep the 3D point- and line clouds consistent (Section 5.2.3).

5.2.1 Line Segment Matching & Depth Estimation

Given an already processed image sequence $I = \{I_1, \dots, I_{t-1}\}$, we want to integrate a new incoming image I_t and update the 3D model accordingly. As in Section 3.3, we start by running the *LSD* [51] line segment detector, to obtain a set of line segments $L_t = \{\ell_1^t, \dots, \ell_{m_t}^t\}$ from image I_t, where each line segment ℓ_m^t is again simply defined by two 2D points $p_m^t, q_m^t \in \mathbb{R}^2$. We now match L_t with existing images, to obtain new potential correspondences. We can use these new matches to update the quality of existing correspondences, and to initialize or refine the depth estimates of each affected 2D line segment.

5.2.1.1 Line Segment Matching

To match the line segments L_t from image I_t with existing images, we first need to determine which images are potential candidates (i.e. visual neighbors). As described in Section 3.4.1, we use the visibility information of the 3D point cloud, as well as the spatial camera distributions, to determine visual neighbors among the images. Analogue to the offline case, we compute the visual neighbor subsets V_t and $\hat{V}_t$ for image I_t (Equations 3.21 and 3.24). The final visual neighbor set V_t^M is then again defined by the $M/2$ nearest neighbors from $\hat{V}_t$, and the $M/2$ nearest neighbors from V_t (skipping duplicates).

We now match L_t with all L_x ($x \in V_t^M$), by using our purely geometric matching constraints from Section 3.4.2, with the k-nn matching refinement (Section 3.4.3). This is especially crucial for online *SfM* applications, since it limits the amount of potential correspondences that need to be processed, while it does not negatively influence the reconstruction results (as we have demonstrated in Section 4.5.2.2).

After the matching, all remaining correspondences $\ell^t_m \to \ell^x_{\bar{m}}$ are converted to 3D line segment hypotheses, as described in Section 3.4.4. The resulting 3D line segments $h^{t,x}_{m,\bar{m}}$ are stored in the hypotheses set Ψ^t_m belonging to the 2D segment ℓ^t_m.

5.2.1.2 Depth Estimation

After the matching procedure, we end up with an initial set of potential correspondences for all new segments $\ell^t_m \in L_t$, and an increased correspondence set for all existing segments $\ell^x_{\bar{m}} \in L_x$ (with $x \in V^M_t$). As in the offline approach, we now want to evaluate all correspondences based on their mutual support among neighboring images (see Section 3.5). The obtained scores can then be used to remove outlier correspondences, and to select (or update) the most probable 3D location for each 2D segment.

By default we assume that we do not know the reconstruction scale at runtime, which is often the case when a user starts to take pictures of an object from a probably random starting point, and with no proper initialization. Hence, we use the scale invariant scoring formulation as described in Section 3.5.1. However, if scale information is available it is of course straightforward to use our scale-aware scoring formulation as well.

The (non-symmetric) similarity between two arbitrary 3D hypotheses can be expressed as

$$S(h^{x,y}_{a,b}, h^{z,w}_{c,d}) = \min_{Z \in h^{x,y}_{a,b}} \exp\left(-\frac{d_\perp(Z, h^{z,w}_{c,d})^2}{2 \cdot \sigma^x_p(d_x(Z))^2}\right), \tag{5.1}$$

where $d_\perp(Z, h^{z,w}_{c,d})$ is the Euclidean distance between the 3D point Z and the infinite line passing through the segment $h^{z,w}_{c,d}$, and $d_x(Z)$ is the Euclidean distance between Z and the camera center of the image I_x. This equation is very similar to our positional similarity S^p (Equation 3.29), with the difference that we only have one regularizer (σ^x_p) defined by the reference image I_x (hence, the similarity function is non-symmetric). In addition, we now directly use the adapted positional similarity from Equation 5.1 as the total affinity between two 3D hypotheses, by omitting the angular similarity S^a (Equation 3.28) altogether. We do this for efficiency reasons, since the more important part of the similarity computation is the positional similarity, with the angular similarity being more or less only a safety term to prevent degenerate configurations (which are unlikely to occur).

For each hypothetical match $h^{t,x}_{m,\bar{m}}$, we now compute a truncated score (confidence)

$$c(h^{t,x}_{m,\bar{m}}) = \sum_{z \in V^M_t \setminus \{x\}} \begin{cases} \max_{h^{t,z}_{m,\cdot} \in \Psi^t_m} \left\{ S(h^{t,x}_{m,\bar{m}}, h^{t,z}_{m,\cdot}) \right\} & \text{if } \max_{\dots} \{\dots\} > \frac{1}{2} \\ 0 & \text{else} \end{cases} \tag{5.2}$$

analogue to Equation 3.26 in the offline version.

This scoring procedure has a relatively high computational complexity, since for each 3D hypothesis $h^{t,x}_{m,\bar{m}}$ we have to compute the similarities to all other hypotheses in Ψ^t_m. For a new image I_t, this is necessary since we do not have any prior information, and all correspondences between L_t and its visual neighbors L_x ($x \in V_t^M$) are initialized at the same time. However, for the existing images we can use an update formulation with a lower complexity. Given new correspondences between an existing image I_x and a new image I_t, we update the scores for already existing correspondences $h^{x;y}_{\cdot,\cdot}$ between I_x and another (older) image $I_y \in \{I_1, \cdots I_{t-1}\}$ by computing

$$c(h^{x;y}_{\cdot,\cdot}) = c(h^{x;y}_{\cdot,\cdot}) + \max_{h^{x;t}_{\cdot,\cdot} \in \Psi^x_{\cdot}} \left\{ S(h^{x;y}_{\cdot,\cdot}, h^{x;t}_{\cdot,\cdot}) \right\}. \tag{5.3}$$

In a similar fashion, we initialize the scores of the new hypotheses $h^{x,t}_{m,\cdot}$ between I_x and the new I_t as

$$c(h^{x,t}_{m,\cdot}) = \max_{h \in \Psi^x_m} \{c(h) \cdot S(h^{x,t}_{m,\cdot}, h)\}. \tag{5.4}$$

As explained in Section 3.6, we use these (updated) scores to assign a new depth estimate to each affected 2D segment ℓ^x_m ($x \in V_t^M \cup \{t\}$), which is defined as its most likely 3D hypothesis

$$\hat{h}^x_m = \operatorname*{argmax}_{h \in \Psi^x_m} \{c(h)\}, \tag{5.5}$$

with respect to the scoring function. We only accept $\hat{h}^x_m$ if its score $c(\hat{h}^x_m)$ is bigger than zero (i.e. at least one term of the sum in Equation 5.3 must be bigger than 1/2). In addition, we remove all hypotheses $h^{x;\cdot}_{\cdot,\cdot}$ for which $c(h^{x;\cdot}_{\cdot,\cdot}) = 0$, if the underlying image I_x has already been matched with a certain number different images (i.e. such that we are fairly confident that this match is truly an outlier), or if a certain number of new images have been added to the system since the hypothesis was created. This on the one hand ensures that the performance stays high, even when many images are being processed, but on the other hand it prevents correct hypotheses from being removed, only because the scene coverage might still be low (which in return causes lower confidence values).

5.2.2 3D Model Update

Every time a new image I_t is integrated into the system, we update the line-based 3D model with the new information. We therefore perform an incremental clustering procedure, which only considers all new or updated 2D segments from L_t and L_x ($x \in V_t^M$). This means that in each clustering process we always have a fairly constant number of affected segments, which is beneficial especially for large image sets.

5.2.2.1 Segment Clustering

As in Section 3.7, a line cluster Π consists of a set of 2D residuals $L_\Pi = \{\ell_1, \ell_2, \cdots\}$, and a corresponding 3D line segment h_Π. The score $c(\Pi)$ of a cluster is defined as the average score over the best hypotheses ($\hat{h}$) of its residuals. Additionally, we define $C(\ell)$ to be a function that returns the line cluster to which the 2D segment ℓ belongs (initially, and for all unclustered segments, we set $C(\ell) = \emptyset$).

We now compute an affinity matrix W between all segments from L_t and L_x ($x \in V_t^M$), by making use of their estimated depths. The affinity between two segments ℓ_m^x and $\ell_{\bar{m}}^y$ is defined as

$$W(\ell_m^x, \ell_{\bar{m}}^y) = \begin{cases} \frac{1}{2}\left(\bar{S}(\eta_m^x, \eta_{\bar{m}}^y) + \bar{S}(\eta_{\bar{m}}^y, \eta_m^x)\right) & \text{if } \frac{1}{2}(\cdots) > 0.5 \\ 0 & \text{else} \end{cases} \tag{5.6}$$

where $\bar{S}$ is a truncated version of the similarity function (Equation 5.1), with the modified regularizer from Equation 3.36. The input 3D segments η_m^x and $\eta_{\bar{m}}^y$ in Equation 3.27 are defined as

$$\eta_m^x = \begin{cases} \hat{h}_m^x & \text{if } C(\ell_m^x) = \emptyset \\ h_{C(\ell_m^x)} & \text{else} \end{cases}, \tag{5.7}$$

which basically maps a 2D segment ℓ_m^x to its containing cluster (if it exists), or to its estimated 3D hypothesis. Please note that we only consider 2D segments for clustering for which a valid 3D estimate exists.

When the affinity matrix is computed, we segment it using the basic graph clustering algorithm [44], since it is faster and less memory consuming than the diffusion-based approach [35] (the differences are explained in Section 3.7). All found clusters which only contain 2D line segments ℓ for which $C(\ell) = \emptyset$ form a new 3D line cluster Π. The residual set L_Π is initialised with the clustered 2D segments, and the initial position of the 3D line h_Π is simply the maximum score 3D hypothesis of the segments in L_Π.

In all other cases, i.e. when at least one of the clustered 2D segment is already part of a 3D line cluster, we just merge these clusters and their residual sets to one new and bigger cluster Π. 2D segments ℓ for which $C(\ell) = \emptyset$ are simply added to the residual set of the new cluster, and the associated 3D line h_Π is derived from one of the merged clusters (the one with the largest residual set).

We only consider clusters Π to be valid, if their score $c(\Pi) > 0$. Since the 3D line segment h_Π associated with Π does not necessarily need to be an optimal estimate for visualization, we compute the line visibility by projecting the endpoints of all residing 2D segments $\ell \in L_\Pi$ onto the infinite line through h_Π, sorting the resulting set of projected collinear points, and computing all non overlapping 3D line segments which are visible in at least 3 images (as introduced in Section 3.7.1). These estimated 3D lines can then be displayed along with the 3D points from the online *SfM*.

5.2.2.2 Cluster Verification

The quality of the computed 3D line clusters $\{\Pi\}$ depends on correct depth estimates for the 2D segments ℓ. If the scene coverage is low, incorrect depth estimates are possible, which can result in outlier clusters on rare occasions. To correct potential errors, we check how well the estimated 3D hypothesis $\hat{h}_m^x$ for a 2D segment ℓ_m^x fits to its associated cluster $C(\ell_m^x)$, whenever its depth estimate changes (i.e. whenever image I_x is matched with another image). We compute the similarity $S(h^x_m, h_{C(\ell^x_m)})$, and remove ℓ^x_m from the cluster if the similarity is zero, and the score of its current 3D hypothesis $c(h^x_m)$ is larger than the score of its 3D line cluster $c(C(l^x_m))$. We also remove clusters Π for which $c(\Pi) = 0$. In contrast to our previous methods [61, 65], where clusters are always kept once they were created, this procedure allows to correct errors when better depth estimates are available.

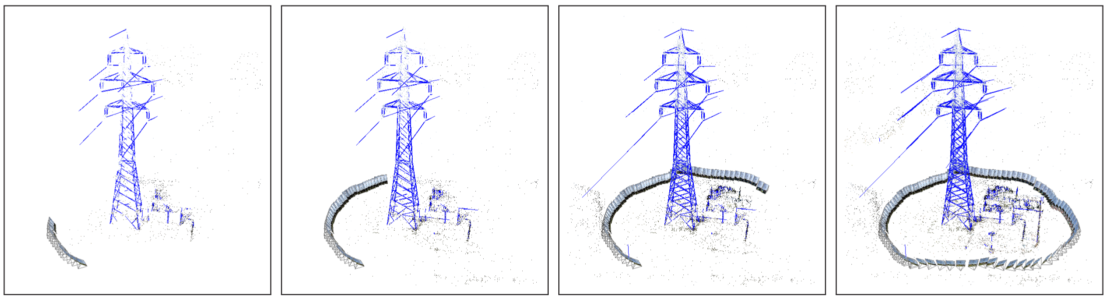
5.2.3 Bundle Adjustment

To create a consistent reconstruction with closed loops, and to ensure that the reconstructed points and lines fit together, bundle adjustment [156] has to be performed periodically. In the online *SfM* pipeline [70] this is done in a parallel thread, in a permanent loop. This means that at the beginning bundle adjustment is performed quasi after each new image, since the looped optimization procedure finishes very quickly. As the reconstruction grows, the optimization takes longer and it can happen that more than just one image have been added to the model before the bundling procedure can start again. However, this is not really a problem, since when the reconstruction already contains a high amount of images the model is usually already very consistent, and additional images do not change the reconstruction as a whole by a large margin.

The combined bundling procedure is performed using the Ceres solver [1], and in the same way as for the offline approach (see Section 3.8). Since the camera poses change very often, especially at the beginning when only few images are available or when loops are closed, all 3D line hypotheses (and not just the 3D line clusters) have to be adapted as well. We could incorporate all pairwise matches into the bundle adjustment to ensure consistency, but this would significantly slow down the optimization procedure. We therefore decided to recompute the 3D positions of all hypotheses h whenever necessary, with the current camera poses. This is only the case when a new image is added to the system, and only for those images that have been affected. We recompute the 3D positions of all remaining pairwise matches in parallel on the *GPU*, which only requires the intersection of two lines with a 3D plane (see Section 3.4.4). This is a minor drawback compared to the offline algorithm, where the camera poses are not changed during the line-based 3D reconstruction, and hence all 3D hypotheses do not have to be modified after their creation. However, re-estimating the 3D hypotheses does not take a lot of time, which is beneficial for our online *SfM* application.

5.3 Experimental Results

To evaluate accuracy and runtime of the combined online *SfM*, we perform several experiments on our test datasets introduced in Chapter 4 (the test system is the same as before). We basically perform a comparative evaluation between the online *SfM* with points and lines, and our in-house offline *SfM* pipeline [72] (which we have used throughout this the-


(a) *Pylon*

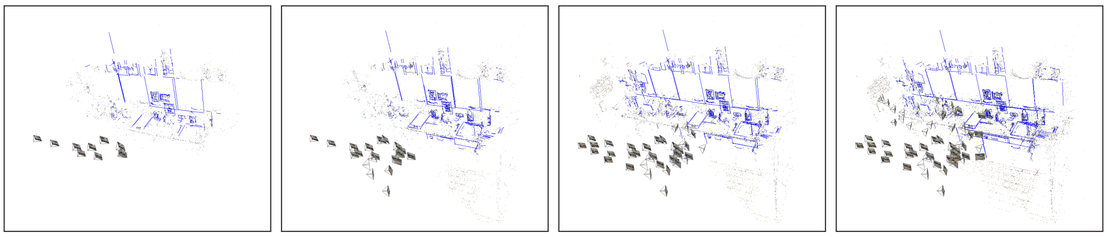

(b) *Kitchen*

Figure 5.4: An illustration of the incremental reconstruction procedure on the *Pylon* and the *Kitchen* dataset.

sis) followed by the 3D line reconstruction afterwards. Both *SfM* pipelines build up on very similar core principles, which guarantees a fair comparison. Figure 5.4 illustrates the incremental reconstruction procedure for the *Pylon* and the *Kitchen* sequence. As we can see, the completeness of the 3D models is significantly improved when more images are available.

5.3.1 Test Setup

As in the offline version [72], the online *SfM* pipeline also makes use of *SIFT* [101] features, and a vocabulary tree [120]. We again use the 5000 largest *SIFT* features per image, but reduce the voctree radius v_r to only 5 images to be more runtime efficient. As stated above, the online *SfM* is designed as two parallel processes. The first one incrementally integrates new images into the *SfM* reconstruction, and the second one performs an infinite bundle adjustment in a loop. We modify the system such that the first (alignment-) process pushes the newly aligned images into our line-based 3D reconstruction framework, and the second (bundle-) process updates all 3D line equations after the bundling procedure. All data access by both threads is synchronized by making use of *MUTEX* structures. For a lower per-image runtime, we reduce the images to approximate FullHD resolution during the line segment detection step. This notably boosts the performance, while it does not change the results by a large margin (as discussed in Section 4.5.1). The average increase in runtime between the pure point-based online *SfM*, and the combined version (points and lines) is approximately 0.2 seconds per image.

For a fair and meaningful comparison to the offline method (offline *SfM* [72] + *Line3D++*), we also set the voctree radius to 5 images here, and further reduce the image sizes to FullHD resolution for line segment detection as well. However, we have seen that for several datasets (*Timberframe, Building, Pylon, Façade, and Kitchen*) the offline *SfM* is not able to successfully reconstruct the scene with such a low voctree radius, while this was not a problem for the online version. Hence, we had to increase the radius to 10 or 15 images for these datasets.

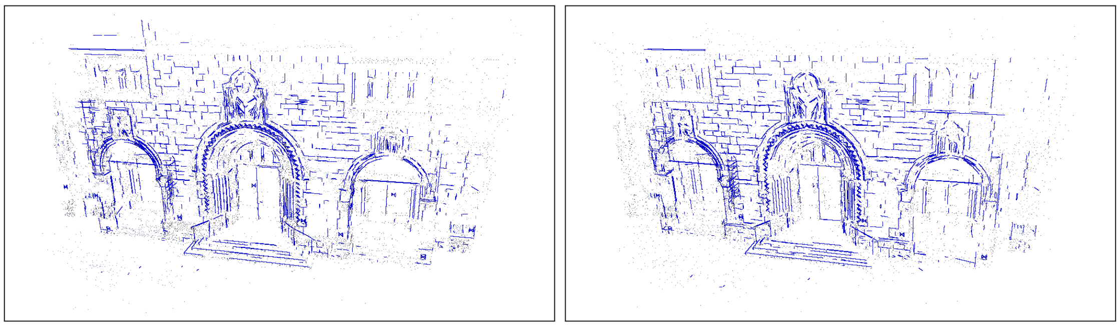
This behaviour is observable especially for larger datasets, and for datasets with a high amount of repetitive structures. Here, when using a low voctree radius it happens quite frequently that the selection of the images for matching is not optimal, since the pool of images to draw from might be very large, while the distinctiveness of the image features of the query image is often rather low. This problem is not native to the *ICG3D* pipeline [72] alone, but also observable in e.g. *COLMAP* [137]. An online *SfM* pipeline on the other hand has the inherent benefit that the images for matching are only selected among the already processed images, and not the complete and final image set. In many cases, this helps to enable a correct image alignment, even for highly repetitive scenes. Technically, the same issue can of course happen with the online *SfM* as well, especially when the image set grows very large. In our experiments however this did not happen.

5.3.2 Quantitative Evaluation

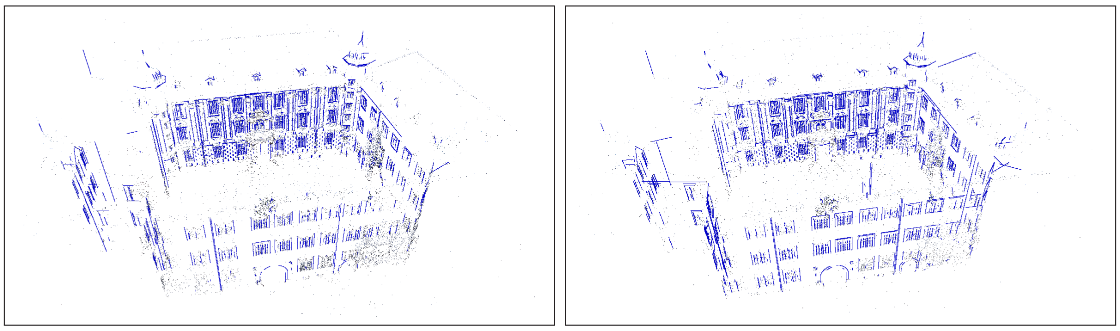
Table 5.1 shows a comparison between the offline- and the online algorithm on all test sequences. As we can see, the online version is approximately twice as fast as the offline version (even for those datasets where we did not have to increase the voctree radius). The average runtime is approximately two seconds per image for the online method, which means that a quasi real-time performance is achieved, given that the operator (or more general: *the camera*) has to change its position between shots. The number of reconstructed lines and points is fairly similar for both methods.

	Offline (*SfM* [72] → Line3D++)					**Online** (online *SfM* [70] ↔ inc. Line3D++)				
Dataset	time t [s]	$\varnothing t$ [s]	# lines	# points	v_r	time t [s]	$\varnothing t$ [s]	# lines	# points	σ [px]
Herz-Jesu-P25 [146]	75	3.00	2,042	12,473	5	45	1.80	1,971	10,216	4.0
Herz-Jesu-P8 [146]	28	3.54	801	4,863	5	10	1.25	964	3,582	4.0
Castle-P30 [146]	72	2.41	2,411	17,448	5	45	1.50	2,854	11,661	2.5
Fountain-P11 [146]	35	3.22	788	7,396	5	17	1.55	1,062	5,210	4.0
Timberframe [73]	455	1.90	5,691	46,135	10	417	1.74	5,644	53,344	2.5
Building	2,168	6.91	6,309	90,437	10	1,110	3.54	4,621	95,960	2.5
Pylon	722	6.81	3,484	18,322	15	308	2.91	3,400	21,884	2.5
Façade	1,774	5.70	6,251	77,602	15	564	1.81	4,497	70,730	4.0
Kitchen	346	5.77	1,336	23,720	10	184	3.07	822	14,029	15.0
Mean		**4.36**					**2.13**			

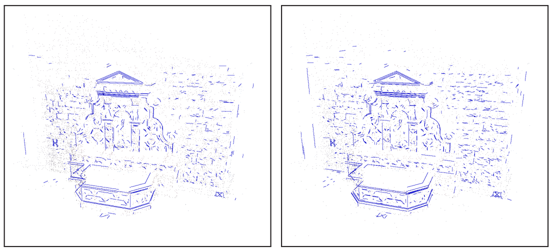
Table 5.1: A comparison between the offline-, and the online combined reconstruction pipeline. More information about the datasets can be found in Section 4.3.



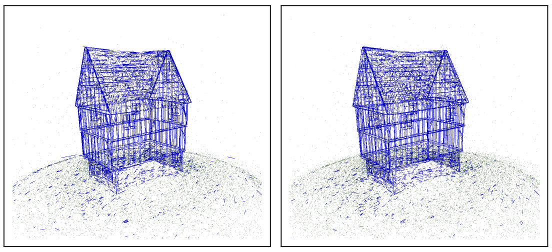
(a) *Herz-Jesu-P25* [146]



(b) *Castle-P30* [146]



(c) *Fountain-P11* [146]

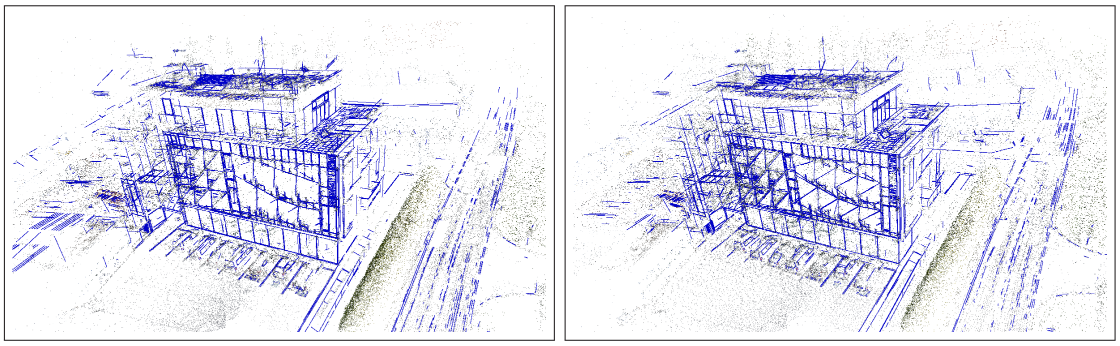


(d) *Timberframe* [73]

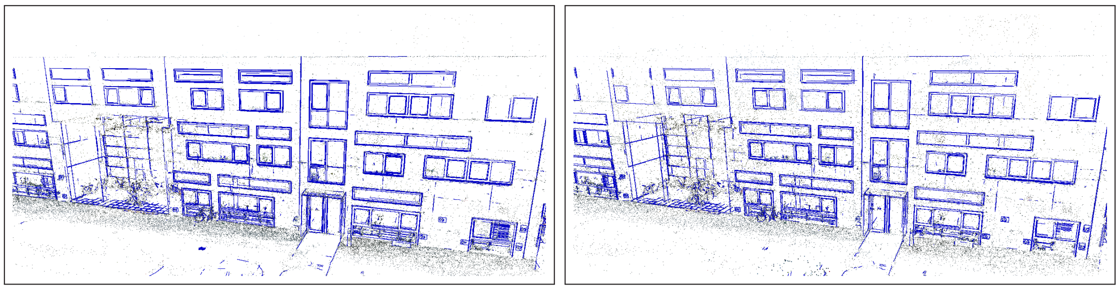
Figure 5.5: A visual comparison between the offline- (*left*) versus the online method (*right*), on the groundtruth datasets. The corresponding numbers can be found in Table 4.4.

A visual comparison can be seen in Figure 5.5 for the groundtruth-, and in Figure 5.6 for the real-world datasets. In both cases, most reconstruction results are fairly similar, and it is hard to make a general statement about which method is to be preferred.

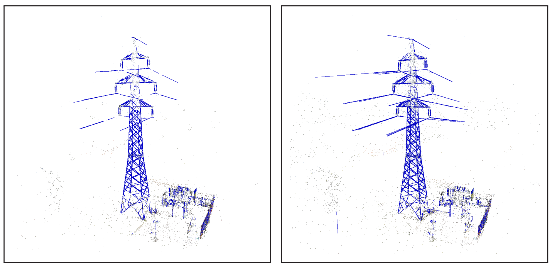
Table 5.2 shows an evaluation of the camera pose accuracies between the offline- and the online method, on the groundtruth datasets. For the offline case, the values relate to the final bundled camera poses (with points and lines). As we can see, the angular error is in general lower when the online version is used. However, the differences are very small and negligible in practice. The more interesting case is the positional error. Here, the offline *SfM* is more accurate on all datasets, apart from *Castle-P30* and *Fountain-P11*. For *Castle-P30* the significantly higher maximum- and mean positional errors of the



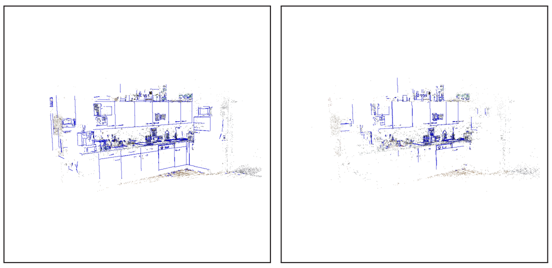
(a) *Building*



(b) *Façade*



(c) *Pylon*



(d) *Kitchen*

Figure 5.6: A visual comparison between the offline- (*left*) versus the online method (*right*), on the real-world datasets. The corresponding numbers can be found in Table 4.4.

offline version are the result of one single displaced camera, which is off by more than one meter (hence, the maximum error of 65.95 cm). Interestingly, this did not happen with the online version, which is most likely due to a different initialization of the two competing incremental *SfM* procedures. For *Fountain-P11* it is less conclusive why the online algorithm is more accurate, but it is again likely that the reason lies within different choices for the initial camera pair. However, in contrast to *Castle-P30* the differences are way less severe and hardly noticeable.

A comparison of the accuracy of the obtained line models can be seen in Table 5.3. Again, the differences are quite small on these datasets. Both experiments show that the online *SfM* pipeline is in practice not less accurate than the offline version, but with the

	Offline				**Online**			
	Pos. Err. [cm]		Ang. Err. [deg]		Pos. Err. [cm]		Ang. Err. [deg]	
Dataset	max	mean	max	mean	max	mean	max	mean
Herz-Jesu-P25 [146]	**1.278**	**0.520**	0.200	0.162	1.301	0.570	**0.108**	**0.053**
Herz-Jesu-P8 [146]	**0.995**	**0.536**	**0.171**	0.155	1.189	0.565	0.181	**0.152**
Castle-P30 [146]	65.949	8.008	1.281	0.265	**10.750**	**4.139**	**0.273**	**0.110**
Fountain-P11 [146]	0.472	0.309	0.102	0.088	**0.365**	**0.236**	**0.065**	**0.045**
Timberframe [73]	**7.703**	**2.222**	**0.211**	0.070	15.803	5.541	0.233	**0.062**

Table 5.2: Camera pose accuracies of the offline- versus the online 3D reconstruction method, on the groundtruth datasets.

additional benefit of interactivity. However, both pipelines follow the same core principles, which probably best explains the similarity of the reconstruction outcome between them. In practice, the best way to obtain an accurate 3D model is to use the online *SfM* during image acquisition, to guarantee that the recorded image set is suitable for reconstruction. Here, one should use a small voctree radius, a smaller number of feature points, and potentially also downsampled images, to ensure real-time computation, such that the camera operator does not have to wait long periods of time for the SfM to query each new image. Then, one can easily re-compute the *SfM* result using e.g. a state-of-the-art offline pipeline on full resolution, before a real dense (or also line-based) 3D model is computed.

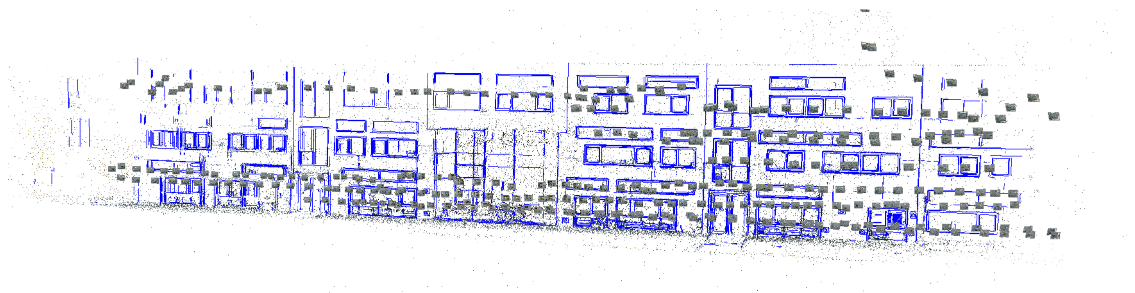
	Offline		**Online**	
Dataset	*RMSE* [cm]	mean err. [cm]	*RMSE* [cm]	mean err. [cm]
Herz-Jesu-P8	5.80	**2.77**	**5.51**	3.13
Fountain-P11	2.31	**0.76**	**2.15**	0.98
Timberframe	**5.08**	**3.26**	5.73	3.51

Table 5.3: Line model accuracies of the offline- versus the online 3D reconstruction method, on the three groundtruth datasets with surface model.

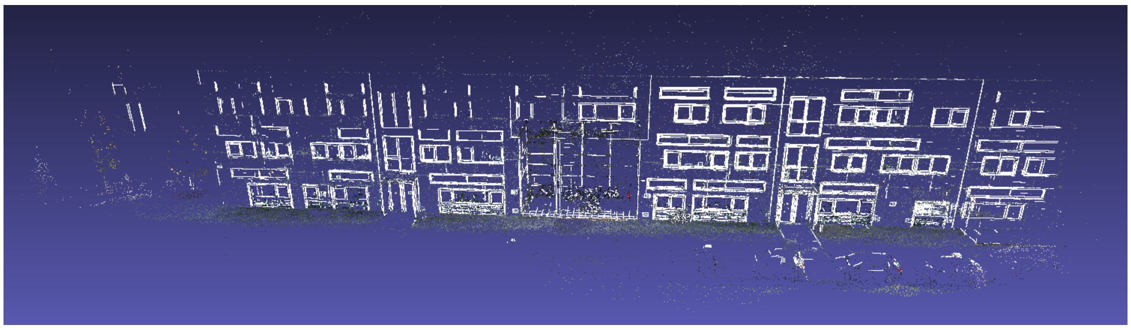
Figure 5.7 shows an example for this workflow. As we can see, the online *SfM* enables a very fast image alignment when the image dimensions and the number of feature points are reduced, with an average per image runtime of less than one second. Hence, one can consider this approach real-time capable, given that a moving human camera operator is involved. Afterwards, the offline (or also the online-) *SfM* pipeline can be executed on the full-resolution images, and with a higher number of image features, to ensure a more accurate and a more complete result. Since all images are properly aligned by the online *SfM*, we can be fairly certain that all images can be aligned by the offline version as well. As a final step, the *SfM* result can e.g. be used to compute a dense 3D point cloud (if desired). What we can clearly observe in this example, is that the scene coverage is much lower on the left part of the scene, than on the right. This is especially noticeable for the 3D lines, where several parts of the windows are missing on the top left part of the building. This is already the case in the online result, which means that the user would have had the opportunity to react to this situation, by taking some additional images in this part of the scene. Hence, missing parts in the final 3D models (both the line model as well as the dense point cloud) could have been easily avoided by the user. This shows once more the potential benefits of an online *SfM* pipeline, especially when dealing with inexperienced users, and for highly complex scenes.

5.4 Summary

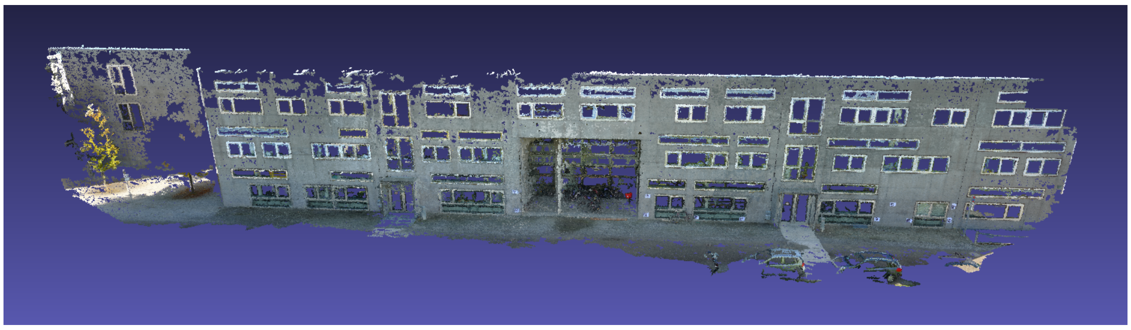
We have discussed the principle of online *SfM*, and the potential benefits compared to offline methods. As stated above, the main benefit is the interactivity, which allows to directly see if an acquired image is suitable for 3D reconstruction or not. Hence, the user can always correct his acquisition strategy in case that new images cannot be aligned with the existing ones (e.g. by too severe camera rotations). We have shown how our method can be efficiently integrated into an online *SfM* pipeline [70], which allows the direct computation of a 3D point- and line model on-the-fly. This additional information further aids the user in his judgement about the quality of the obtained image set, and about the completeness of the observed scene. In the next chapter we conclude our work in line-based 3D reconstruction, and go on to discuss some ideas for future projects.



(a) Online SfM [70] $\leftrightarrow$ incremental *Line3D++* ($\varnothing t = 0.82$ s)



(b) Offline SfM [72] $\rightarrow$ *Line3D++* ($\varnothing t = 6.78$ s)



(c) *PMVS* [49]

Figure 5.7: An exemplary result of the online/offline 3D reconstruction workflow for the *Façade* dataset. (a) First, the online *SfM* pipeline [70] is run on downsampled images (50%), with only 2,500 SIFT [101] features per image, and with a voctree radius of just 5 images. (b) Then, the offline *SfM* pipeline [72] is executed on the full-resolution images, with 5,000 features and a voctree radius of 20 images. (c) As a final step, any point-based *MVS* pipeline can be used to create a semi-dense point cloud, based on the offline *SfM* result (e.g. *PMVS* [49]).

6 Conclusion

Contents

6.1 Summary 101

6.2 Future Work: Beyond Lines 102

In this thesis, we motivated the use of line-based 3D reconstruction (or line-based *MVS*) for the task of 3D scene abstraction, and introduced our proposed method in full detail. Our approach is targeted on man-made environments, in which piece-wise planar- and linear objects frequently occur. In Section 6.1 we sum up our core findings, and in Section 6.2 we discuss some ideas for future work, which aim at going beyond sparse line-based 3D models.

6.1 Summary

Our approach is best described as an *SfM* post-processing step, which takes an already oriented image sequence as input. This has the benefit that we can already use more information than just the raw image sequence, since we can make use of epipolar-geometric relations between the images. This gives us the opportunity to focus on the task of generating an accurate and complete 3D line model, rather than dealing with camera pose estimation from line correspondences. Pure line-based *SfM* pipelines (see Section 2.3) are often motivated by the fact that especially in monotone man-made environments reliable feature point matching is hard to achieve, which would render traditional *SfM* pipelines ineffective (and hence, our method could not be used). In practice this is usually not the case. Our numerous experiments in urban indoor- and outdoor environments have seen that state-of-the-art general purpose *SfM* pipelines (e.g. VisualSfM [172] or colmap [137]) work very well in the majority of the cases, and - at the very least - deliver the correct camera poses even when the resulting point cloud is sparse. In addition, point-based *SfM* pipelines are more straightforward to use than line-based ones, since underlying tasks like relative pose estimation can be easily solved by a set of almost arbitrary point correspondences, while one either needs matched lines in a special spatial configuration (e.g. [37]), or a trifocal tensor [55], when using lines as sole image features.

From the reconstruction point of view, having the camera poses available has one core benefit: we can use highly invariant geometric relations between images for the task of line segment matching, rather than unstable appearance constraints. This enables us to match line segments under changing illumination conditions, and when dealing with thin wiry structures for which meaningful patch-based line descriptors are virtually impossible to compute. We have shown in numerous experiments, and on several challenging datasets, that geometric constraints are in fact enough to robustly reconstruct accurate and outlier free 3D line models, for indoor as well as outdoor scenes. In addition, our algorithm is very runtime efficient and allows for a much faster 3D reconstruction than e.g. dense point-based *MVS* algorithms (e.g. *PMVS* [49]). This makes our algorithm also suitable for real-time computation, in conjunction with an online *SfM* pipeline.

The biggest benefit of an algorithm like ours is the inherent abstraction procedure. In contrast to a potentially huge dense point-cloud, our method generates a highly compressed amount of 3D data, which still offers a high degree of semantic information. As seen in our experiments, just a few thousand 3D lines are enough to successfully represent a 3D scene from hundreds of images, while a dense point-cloud obtained from the same images easily consists of millions of points. However, the world in which we live is of course not made of lines. For many tasks it is necessary to generate more realistic 3D models, which closely resemble our own visual impression of our surroundings. In these cases, it is unavoidable to use a more memory- and runtime consuming (and less abstract) 3D representation, such as a dense point-cloud or a subsequently generated photo-textured 3D mesh. Nevertheless, there are lots of scenarios in which the efficiently computable and highly abstract 3D information provided by a line-based 3D reconstruction method is more than sufficient, or even preferred over a large unordered set of 3D points. In addition, 3D lines can also be used to refine 3D surface models, since lines are often located on the intersection of two physical planes, which naturally helps to estimate more accurate object boundaries in triangle meshes (e.g. as recently shown in [148]). In the following section, we want to give an outlook of what can be done with the obtained 3D line models, and how we can potentially go beyond lines without sacrificing the higher level of abstraction.

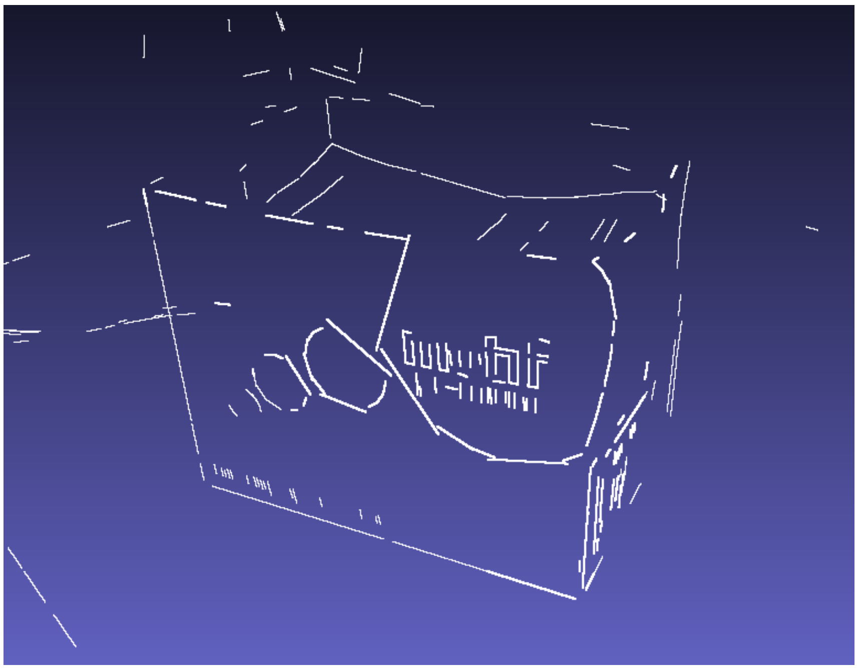
6.2 Future Work: Beyond Lines

With our open-source framework we enable the broad public to easily obtain 3D line models from arbitrary *SfM* results. Despite their high sparsity, these models contain a high degree of semantic information about the scene structure (e.g. walls, windows, doors,...). While there are certainly a lot of tasks for which these line models are sufficient, there are also numerous scenarios which require more than that (e.g. a dense model of some kind). However, this does not necessarily mean that we need to replace line-based 3D reconstruction with something else, but that it makes sense to think of potential post-processing tasks that build up on the reconstructed 3D lines.

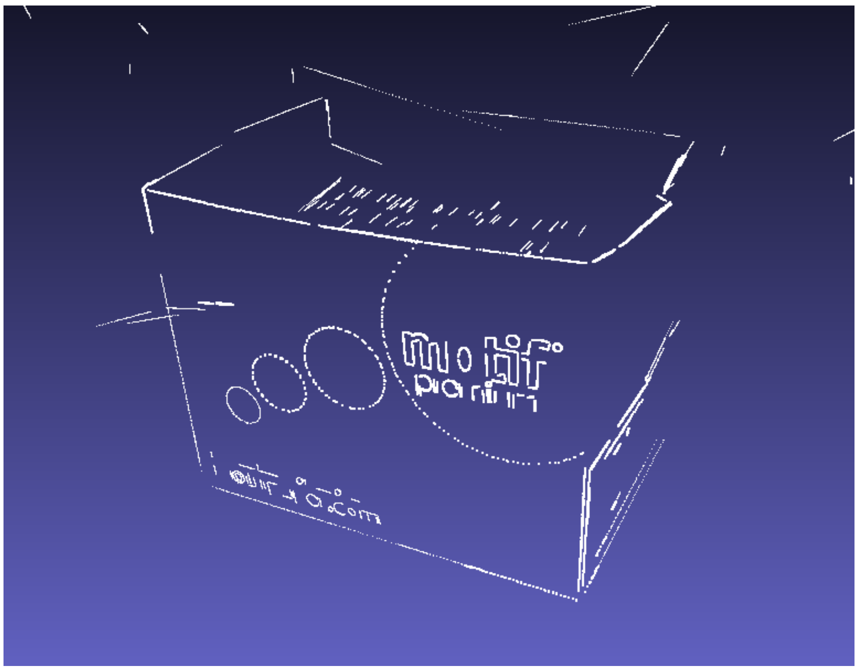
What inevitably comes in mind when thinking about 3D line models, is the reconstruction of a piecewise planar 3D model. In contrast to a traditionally generated 3D mesh from a dense point-cloud, such a piecewise planar model is usually much more memory efficient, since arbitrarily large 3D plane-like structures (e.g. polygonal structures) can be represented by just a few large triangles. If these triangles are then colorized using a proper phototexture, the resulting 3D models are in general highly realistic and visually pleasing (e.g. see [141, 168]). Since all reconstructed structural lines are commonly caused by the intersection of two 3D planes, it makes sense to use the 3D lines as an indicator for the existence of actual physically meaningful planes. This idea is of course not new. As discussed in Section 2.5, many line-based *MVS* algorithms presented so far only used the 3D lines as an intermediate step towards computing a piecewise planar 3D model. However, most of these methods only focus on relatively simple cases, such as rooftops seen from aerial images [19, 113, 139], or one single building or scene [141, 166, 168]. Our method supports the line-based 3D reconstruction of more complex and significantly larger scenes out-of-the-box, which raises the question whether this can be extended to the task of piecewise planar 3D modeling. We believe that it would be quite straightforward to extract potential plane hypotheses, by analyzing the spatial relationships between reconstructed 3D lines (e.g. intersections or parallelism), and to further verify and cluster these hypotheses in a similar way as we do for 3D lines in our approach. The only part that is less straightforward is the visibility check of actual regions on these clustered (infinite) planes (e.g. to find out which regions on the planes are actually physically existing, and therefore visible in the 3D model). However, for this task numerous approaches exist in the literature (e.g. [168]), from which we could draw inspiration.

A second extension that comes in mind is the incorporation of *curved* image features into the reconstruction process. When using line segments alone, it frequently happens that curved structures are piecewise linearly approximated by straight line segments. Figure 6.1a illustrates this behaviour for a toy example dataset of a paper box. As we can see, the circular structure on the box is visually quite unpleasantly approximated by a set of lines. This is in general unavoidable, since available 2D line segment detectors all have this behaviour of representing curved edges as a set of lines, which are then subsequently treated like any other (regular) lines by the matching and reconstruction procedure. If the approximation is consistent enough between neighboring images, the respective segments will get matched and reconstructed as a consequence. However, it would be highly desirable to distinguish between straight- and curved edges, and to reconstruct them properly in 3D.

In the past, some but few methods for the task of curve-based 3D reconstruction have been presented (e.g. [40, 77, 94, 154]). While most of them show some promising results, they are not generally applicable for arbitrarily complex datasets, due to the much higher complexity of reconstructing 3D curves compared to 3D lines. For instance, corresponding



(a) 3D lines



(b) 3D curves

Figure 6.1: A comparison between (a) a line-based-, and (b) a curve-based 3D model of a simple paper box. This toy-example dataset consists of 10 high-resolution images, obtained using a hand-held compact camera.

points between two matched lines from different images can be easily obtained by intersecting the epipolar line of a point on the first line with the corresponding line in the other image. Since we are dealing with lines, there can be at most one unique intersection point. However, if we are dealing with curves, it is possible that the epipolar line intersects the respective curve more than once. Difficulties like this make curve-based 3D reconstruction a very hard task, and several of the aforementioned methods simplify the task by considering the 2D and 3D curves as connected chains of pixels (or 3D points), rather than parametric curves. This however results in a potentially large (connected) point-cloud, which somehow diminishes the benefits of having an abstract 3D model of a low complexity.

Very recently, Nurutdinova and Fitzgibbon [122] revisited curve-based 3D reconstruction, by proposing an *SfM* refinement algorithm based on matched curves. They use splines with a fixed number of control points as 3D entities, and show how reconstructed 3D curves can be used to improve the underlying camera parameters from the *SfM*, and to enhance the visual impression of the sparse 3D model. Sadly, they assume that matched curves between all images are already given, which omits two of the most challenging tasks: curve detection and matching. However, their findings are still very valuable, given that the matching problem can be efficiently solved. To that end, we made some first experiments with a combination of line- and curve features, in the form of elliptical arcs [126]. These features can be quite efficiently detected in large-scale images, and easily encoded in parametric form. In addition, most outlines of man-made structures can be accurately approximated by a chain of such elliptical arcs, which are basically quadratic curves. Figure 6.1b shows a first reconstruction result on a small dataset depicting a paper box, where we have straightforwardly applied our line-based 3D reconstruction concepts on curve matching and reconstruction. As we can see, the circular structures on the box are correctly reconstructed as 3D circles, which is much more accurate than a piecewise linear approximation, while it still preserves a similar level of abstraction as the 3D lines.

To conclude, we strongly believe that one key research aspect for image-based 3D reconstruction in the future will be to develop novel and even more efficient ways to create a highly abstract, but also semantically rich, 3D model of any given scene. While huge dense point clouds can be computed quite easily today, it is necessary to think about new ways of 3D scene abstraction, such that for any given task only the amount of 3D information is extracted that is actually really needed. This will help to free computational power for more important tasks, like actually using the generated 3D data e.g. for autonomous driving or flying, to just name two popular examples.

A

List of Acronyms

CAD	Computer Aided Design
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
DEM	Digital Elevation Model
DSLR	Digital Single-Lens Reflex
EKF	Extended Kalman Filter
GB	Gigabyte
GPU	Graphics Processing Unit
HDD	Hard Disk Drive
IMU	Inertial Measurement Unit
LBD	Line Band Descriptor
LiDAR	Light Detection and Ranging
LS	Line Signature
LSD	Line Segment Detector
MRF	Markov Random Field
MSHS	Mean-Standard Deviation of the Hue and Saturation
MSLD	Mean-Standard Line Deviation
MUTEX	Mutual Exclusion
MVS	Multi-View Stereo
NCC	Normalized Cross Correlation
PCA	Principle Component Analysis
PMVS	Patch-Based Multi-View Stereo
PnL	Perspective-n-Line
PoP	Pencil-of-Points
PTAM	Parallel Tracking and Mapping
RANSAC	Random Sample Consesus

RMSE	Root-Mean-Square Error
SfM	Structure-from-Motion
SIFT	Scale-Invariant Feature Transform
SILT	Scale Invariant Line Transform
SLAM	Simultaneous Localization and Mapping
SURF	Speeded Up Robust Features
TB	Terabyte
UAV	Unmanned Aerial Vehicle

B List of Publications

My work at the Institute for Computer Graphics and Vision led to the following peer-reviewed publications, listed in chronological order along with the respective abstracts.

B.1 2013

Line-based 3D Reconstruction of Wiry Objects

Manuel Hofer, Andreas Wendel, and Horst Bischof
In: *Proceedings of the 18th Computer Vision Winter Workshop (CVWW)*
pp. 78 - 85
February 2013, Hernstein, Austria
(Accepted for oral presentation)
Best Student Paper Award

Abstract: Man-made environments contain many weakly textured surfaces which are typically poorly modeled in sparse point reconstructions. Most notable, wiry structures such as fences, scaffolds, or power pylons are not contained at all. This paper presents a novel approach for generating line-based 3D models from image sequences. Initially, camera positions are obtained using conventional Structure-from-Motion techniques. In order to avoid explicit matching of 2D line segments in the various views we exploit the epipolar constraints and generate a series of 3D line hypotheses, which are then verified and clustered to obtain the final result. We show that this approach can be used to densify various sparse occupied point clouds of urban scenes in order to obtain a meaningful model of the underlying structure.

Related chapter(s): 3

Incremental Line-based 3D Reconstruction using Geometric Constraints

Manuel Hofer, Andreas Wendel, and Horst Bischof
In: *Proceedings of the 24th British Machine Vision Conference (BMVC)*
pp. 92.1 - 92.11
September 2013, Bristol, UK
(Accepted for oral presentation)

Abstract: Generating accurate 3D models for man-made environments can be a challenging task due to the presence of texture-less objects or wiry structures. Since traditional point-based 3D reconstruction approaches may fail to integrate these structures into the resulting point cloud, a different feature representation is necessary. We present a novel approach which uses point features for camera estimation and additional line segments for 3D reconstruction. To avoid appearance-based line matching, we use purely geometric constraints for hypothesis generation and verification. Therefore, the proposed method is able to reconstruct both wiry structures as well as solid objects. The algorithm is designed to generate incremental results using online Structure-from-Motion and line-based 3D modelling in parallel. We show that the proposed method outperforms previous descriptor-less line matching approaches in terms of run-time while delivering accurate results.

Related chapter(s): 5

B.2 2014

Semi-Global 3D Line Modeling for Incremental Structure-from-Motion

Manuel Hofer, Michael Donoser, and Horst Bischof
In: *Proceedings of the 25th British Machine Vision Conference (BMVC)*
September 2014, Nottingham, UK
(Accepted for poster presentation)

Abstract: Structure-from-Motion (SfM) approaches, which are conventionally based on local interest point matches, tend to work well for richly textured indoor- and outdoor environments. However, in less textured scene areas the density of the resulting point cloud suffers from the lower number of matchable interest points. This significantly affects subsequent computer vision tasks like image based localization, surface extraction or visual navigation. In this paper, we propose a novel 3D reconstruction approach that increases the amount of 3D information in the reconstruction by exploiting line segments as complementary features. We introduce an efficient and effective semi-global approach, which takes into account local (per 2D line segment) as well as global (graph clustering) 3D line hypotheses constellations. Our approach outperforms the state-of-the-art in terms of accuracy, with comparable runtime.

Related chapter(s): 5

Improving Sparse 3D Models for Man-Made Environments Using Line-Based 3D Reconstruction

Manuel Hofer, Michael Maurer, and Horst Bischof
In: *Proceedings of the 2nd International Conference on 3D Vision (3DV)*
pp. 535 - 542, *IEEE*
December 2014, Tokyo, Japan
(Accepted for oral presentation)

Abstract: Traditional Structure-from-Motion (SfM) approaches work well for richly textured scenes with a high number of distinctive feature points. Since man-made environments often contain textureless objects, the resulting point cloud suffers from a low density in corresponding scene parts. The missing 3D information heavily affects all kinds of subsequent post-processing tasks (e.g. meshing), and significantly decreases the visual appearance of the resulting 3D model. We propose a novel 3D reconstruction approach, which uses the output of conventional SfM pipelines to generate additional complementary 3D information, by exploiting line segments. We use appearance-less epipolar guided line matching to create a potentially large set of 3D line hypotheses, which are then verified using a global graph clustering procedure. We show that our proposed method outperforms the current state-of-the-art in terms of runtime and accuracy, as well as visual appearance of the resulting reconstructions.

Related chapter(s): 3

B.3 2015

Line3D: Efficient 3D Scene Abstraction for the Built Environment

Manuel Hofer, Michael Maurer, and Horst Bischof
In: *Proceedings of the 37th German Conference on Pattern Recognition (GCPR)*
pp. 237 - 248, *Springer*
October 2015, Aachen, Germany
(Accepted for poster presentation)

Abstract: Extracting 3D information from a moving camera is traditionally based on interest point detection and matching. This is especially challenging in the built environment, where the number of distinctive interest points is naturally limited. While common Structure-from-Motion (SfM) approaches usually manage to obtain the correct camera poses, the number of accurate 3D points is very small due to the low number of matchable features. Subsequent Multi-view Stereo approaches may help to overcome this problem, but suffer from a high computational complexity. We propose a novel approach for the task of 3D scene abstraction, which uses straight line segments as underlying features. We use purely geometric constraints to match 2D line segments from different images, and formulate the reconstruction procedure as a graph-clustering problem. We show that our method generates accurate 3D models, with a low computational overhead compared to SfM alone.

Related chapter(s): 3

B.4 2016

UAVs rather than planes in the Neolithic Tavoliere: understanding site structure using UAV-based NIR imaging and photogrammetric mapping, magnetometry and field survey

Craig Alexander, Keri Brown, Kyle Freund, Manuel Hofer, Tommaso Mattioli, Andrea Di Miceli, Italo Muntoni and Robert Tykot
In: *2nd International Conference of Aerial Archaeology*
February 2016, Rome, Italy

Abstract: Aerial photography has played an extremely important role in Tavoliere Neolithic archaeology, the earliest sites being reported by Bradford (1949) on the basis of WWII reconnaissance photography. Subsequently researchers such as Jones (1987) and, more recently, Keri Brown (2004) have also exploited aerial photographs. Analysis of aerial photography continues to this day through projects involving regional research and cultural heritage institutions (e.g. Caldara et al. 2014).

During summer 2015, the authors used a fixed wing SenseFly eBee UAV to create near-infrared (NIR) imagery (see, e.g., Verhoeven 2012) of 4 Neolithic sites on the Tavoliere in Puglia - 2 in the northwest around Lucera, one nearer to Foggia and one to the southwest near Cerignola. At two of these sites, we also undertook magnetometer survey and field survey.

In this paper, we present a case study of the site FG003663 near Lago Capacciotti in the province of Foggia. The site proved extremely rich in post-harvest surface finds, typically of much larger size than found at other sites. A high-resolution DSM - in the context of post-harvest cereal fields this is essentially also a DTM - was created by photogrammetry from the UAV NIR imagery. A 20m x 160m transect was surveyed by magnetometer, oriented so as to cut the site boundary ditch (as revealed in earlier aerial photographs). Both magnetometry and the NIR imaging showed the boundary ditch, while the magnetometer survey also revealed internal structures.

This work formed part of the third field season of a project co-directed by Craig Alexander, Keri A. Brown (University of Manchester) and Robert H. Tykot (University of South Florida). The project has conducted field survey at 26 Neolithic sites and used a pXRF (portable X-ray fluorescence) spectrometer to analyse the ceramic and obsidian finds. Further pXRF analyses were conducted on materials excavated in recent years by the Soprintendenza. The team has also collected and analysed 75 clay samples from across the Tavoliere. Analysis of the trace element chemical data is in progress to reconstruct the sourcing and exchange networks of the area. The work described herein is intended to provide a basis for a substantial expansion of the project.

Related chapter(s): -

Efficient 3D Scene Abstraction Using Line Segments

Manuel Hofer, Michael Maurer, and Horst Bischof
In: *Computer Vision and Image Understanding (CVIU)*
In press, *Elsevier*
March 2016

Abstract: Extracting 3D information from a moving camera is traditionally based on interest point detection and matching. This is especially challenging in urban indoor- and outdoor environments, where the number of distinctive interest points is naturally limited. While common Structure-from-Motion (SfM) approaches usually manage to obtain the correct camera poses, the number of accurate 3D points is very small due to the low number of matchable features. Subsequent Multi-view Stereo approaches may help to overcome this problem, but suffer from a high computational complexity. We propose a novel approach for the task of 3D scene abstraction, which uses straight line segments as underlying features. We use purely geometric constraints to match 2D line segments from different images, and formulate the reconstruction procedure as a graph-clustering problem. We show that our method generates accurate 3D models with low computational costs, which makes it especially useful for large-scale urban datasets.

Related chapter(s): 3, 4

Bibliography

[1] Agarwal, S., Mierle, K., Others: Ceres Solver. `http://ceres-solver.org` (page 59, 62, 93)

[2] Agarwal, S., Snavely, N., Simon, I., Seitz, S.M., Szeliski, R.: Building Rome in a Day. In: IEEE 12th International Conference on Computer Vision (ICCV). pp. 72–79. IEEE (2009) (page 1, 7, 10)

[3] Agisoft: Agisoft PhotoScan. `http://www.agisoft.com/` (page 1)

[4] Aider, O.A., Hoppenot, P., Colle, E.: A model-based method for indoor mobile robot localization using monocular vision and straight-line correspondences. Robotics and Autonomous Systems 52(2), 229–246 (2005) (page 24)

[5] Akinlar, C., Topal, C.: EDLines: Real-Time Line Segment Detection by Edge Drawing. In: 18th IEEE International Conference on Image Processing (ICIP). pp. 2837–2840. IEEE (2011) (page 43)

[6] Atkinson, K.: Close Range Photogrammetry and Machine Vision. Whittles Pub. (1996) (page 9)

[7] Ayache, N., Faugeras, O.: Building, Registrating, and Fusing Noisy Visual Maps. The International Journal of Robotics Research 7(6), 45–65 (1988) (page 7, 24)

[8] Ayache, N., Faverjon, B.: Efficient Registration of Stereo Images by Matching Graph Descriptions of Edge Segments. International Journal of Computer Vision (IJCV) 1(2), 107–131 (1987) (page 7, 13)

[9] Baillard, C., Schmid, C., Zisserman, A., Fitzgibbon, A.: Automatic line matching and 3D reconstruction of buildings from multiple views. In: ISPRS Conference on Automatic Extraction of GIS Objects from Digital Imagery. vol. 32, pp. 69–80 (1999) (page 13, 27)

[10] Ballard, D.H.: Generalizing the Hough Transform to Detect Arbitrary Shapes. Pattern recognition 13(2), 111–122 (1981) (page 42)

[11] Bartoli, A., Coquerelle, M., Sturm, P.: A Framework For Pencil-of-Points Structure-From-Motion. In: European Conference on Computer Vision (ECCV), pp. 28–40. Springer (2004) (page 19)

[12] Bartoli, A., Sturm, P.: Multiple-View Structure and Motion From Line Correspondences. In: Ninth IEEE International Conference on Computer Vision (ICCV). pp. 207–212. IEEE (2003) (page 19)

[13] Bartoli, A., Sturm, P.: Structure-from-motion using lines: Representation, triangulation, and bundle adjustment. Computer Vision and Image Understanding (CVIU) 100(3), 416–441 (2005) (page 19)

[14] Bauer, J.: Feature-Based Reconstruction of 3D Primitives from Multiple Views. Phd thesis, Graz University of Technology (11 2009) (page 16)

[15] Bauer, J., Bischof, H., Klaus, A., Karner, K.: Robust And Fully Automated Image Registration Using Invariant Features. In: International Society for Photogrammetry and Remote Sensing (ISPRS). pp. 12–23 (2004) (page 16)

[16] Bay, H., Ess, A., Neubeck, A., van Gool, L.: 3D from Line Segments in Two Poorly-Textured, Uncalibrated Images. In: Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT). pp. 496–503. IEEE (2006) (page 19, 20)

[17] Bay, H., Ferrari, V., van Gool, L.: Wide-Baseline Stereo Matching with Line Segments. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). vol. 1, pp. 329–336. IEEE (2005) (page 13, 16, 19, 48)

[18] Bay, H., Tuytelaars, T., van Gool, L.: SURF: Speeded Up Robust Features. In: European Conference on Computer Vision (ECCV), pp. 404–417. Springer (2006) (page 2, 7, 9, 14)

[19] Bignone, F., Henricsson, O., Fua, P., Stricker, M.: Automatic extraction of generic house roofs from high resolution aerial imagery. In: European Conference on Computer Vision (ECCV), pp. 83–96. Springer (1996) (page 27, 103)

[20] Bosse, M., Rikoski, R., Leonard, J., Teller, S.: Vanishing points and three-dimensional lines from omni-directional video. The Visual Computer 19(6), 417–430 (2003) (page 24)

[21] Brown, M., Windridge, D., Guillemaut, J.Y.: A generalisable framework for saliency-based line segment detection. Pattern Recognition 48(12), 3993–4011 (2015) (page 43)

[22] Canny, J.: A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 6, 679–698 (1986) (page 42)

[23] Caprile, B., Torre, V.: Using vanishing points for camera calibration. International Journal of Computer Vision (IJCV) 4(2), 127–139 (1990) (page 31)

[24] CapturingReality: CapturingReality. `https://www.capturingreality.com/` (page 1)

[25] Chandraker, M., Lim, J., Kriegman, D.: Moving in Stereo: Efficient Structure and Motion Using Lines. In: IEEE 12th International Conference on Computer Vision (ICCV). pp. 1741–1748. IEEE (2009) (page 25)

[26] Chen, T., Wang, Q.: 3D Line Segment Detection for Unorganized Point Clouds from Multi-View Stereo. In: Asian Conference on Computer Vision (ACCV), pp. 400–411. Springer (2010) (page 28, 29)

[27] Cohen, A., Sattler, T., Pollefeys, M.: Merging the Unmatchable: Stitching Visually Disconnected SfM Models. In: IEEE International Conference on Computer Vision (ICCV). pp. 2129–2137. IEEE (2015) (page 12)

[28] Crandall, D., Owens, A., Snavely, N., Huttenlocher, D.P.: Discrete-Continuous Optimization for Large-Scale Structure-from-Motion. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3001–3008. IEEE (2011) (page 9, 12)

[29] Daftry, S., Maurer, M., Wendel, A., Bischof, H.: Flexible and User-Centric Camera Calibration using Planar Fiducial Markers. In: British Machine Vision Conference (BMVC) (2013) (page 34, 37, 62)

[30] Dailey, M.N., Parnichkun, M.: Landmark-based simultaneous localization and mapping with stereo vision. In: Proceedings of the Asian Conference on Industrial Automation and Robotics (2005) (page 24)

[31] Davison, A.J.: Real-time simultaneous localisation and mapping with a single camera. In: Ninth IEEE International Conference on Computer Vision (ICCV). pp. 1403–1410. IEEE (2003) (page 24)

[32] Deriche, R., Faugeras, O.: Tracking Line Segments. In: European Conference on Computer Vision (ECCV). pp. 259–268. Springer (1990) (page 7, 13)

[33] Desolneux, A., Moisan, L., Morel, J.M.: From Gestalt Theory to Image Analysis, A Probabilistic Approach, vol. 34. Springer Science & Business Media (2007) (page 43)

[34] Devernay, F., Faugeras, O.: Straight Lines Have to Be Straight. Machine Vision and Applications 13(1), 14–24 (2001) (page 31)

[35] Donoser, M.: Replicator Graph Clustering. In: British Machine Vision Conference (BMVC) (2013) (page 56, 58, 64, 92)

[36] Elfes, A.: Sonar-based real-world mapping and navigation. IEEE Journal of Robotics and Automation 3(3), 249–265 (1987) (page 24)

[37] Elqursh, A., Elgammal, A.: Line-Based Relative Pose Estimation. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3049–3056. IEEE (2011) (page 20, 21, 24, 102)

[38] Engel, J., Schoeps, T., Cremers, D.: LSD-SLAM: Large-scale direct monocular SLAM. In: European Conference on Computer Vision (ECCV), pp. 834–849. Springer (2014) (page 26, 88)

[39] Engel, J., Stueckler, J., Cremers, D.: Large-Scale Direct SLAM with Stereo Cameras. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1935–1942. IEEE (2015) (page 26)

[40] Fabbri, R., Kimia, B.: 3D Curve Sketch: Flexible Curve-Based Stereo Reconstruction and Calibration. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1538–1545. IEEE (2010) (page 103)

[41] Fan, B., Wu, F., Hu, Z.: Line Matching Leveraged By Point Correspondences. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 390–397. IEEE (2010) (page 15)

[42] Fan, B., Wu, F., Hu, Z.: Robust line matching through line-point invariants. Pattern Recognition 45(2), 794–805 (2012) (page 15)

[43] Fathi, H., Brilakis, I.: A Scale, Rotation, and Affine Invariant Line Detection and Matching Algorithm for 3D Reconstruction of Infrastructure. In: Computing in Civil and Building Engineering. pp. 942–949. ASCE (2014) (page 16)

[44] Felzenszwalb, P., Huttenlocher, D.: Efficient Graph-Based Image Segmentation. International Journal of Computer Vision (IJCV) 59(2), 167–181 (2004) (page 56, 58, 92)

[45] Fischler, M.A., Bolles, R.C.: Random Sampling Consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM 24(6), 381–395 (1981) (page 9, 10, 19)

[46] Flint, A., Mei, C., Reid, I., Murray, D.: Growing semantically meaningful models for visual SLAM. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 467–474. IEEE (2010) (page 24, 25, 26)

[47] Frahm, J.M., Fite-Georgel, P., Gallup, D., Johnson, T., Raguram, R., Wu, C., Jen, Y.H., Dunn, E., Clipp, B., Lazebnik, S., Pollefeys, M.: Building Rome on a Cloudless Day. In: European Conference on Computer Vision (ECCV), pp. 368–381. Springer (2010) (page 1, 3, 7, 9, 10)

[48] Fu, K.P., Shen, S.H., Hu, Z.Y.: Line Matching Across Views Based on Multiple View Stereo. Acta Automatica Sinica 40(8), 1680–1689 (2014) (page 28)

[49] Furukawa, Y., Curless, B., Seitz, S., Szeliski, R.: Towards Internet-Scale Multi-View Stereo. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1434–1441. IEEE (2010) (page 2, 3, 5, 28, 29, 39, 67, 68, 69, 84, 100, 102)

[50] Gee, A.P., Mayol-Cuevas, W.: Real-time model-based SLAM using line segments. In: Advances in Visual Computing, pp. 354–363. Springer (2006) (page 24)

[51] von Gioi, R.G., Jakubowicz, J., Morel, J.M., Randall, G.: LSD: A Fast Fine Segment Detector With a False Detection Control. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 4, 722–732 (2008) (page 43, 70, 89)

[52] Habib, A.: Motion parameter estimation by tracking stationary three-dimensional straight lines in image sequences. ISPRS Journal of Photogrammetry and Remote Sensing 53(3), 174–182 (1998) (page 19)

[53] Harris, C., Stephens, M.: A combined corner and edge detector. In: Alvey vision conference. vol. 15, p. 50. Citeseer (1988) (page 7)

[54] Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision. Cambridge University Press (2003) (page 2)

[55] Hartley, R., Zisserman, A.: Multiview Geometry in Computer Vision. Cambridge University Press (2004) (page 10, 22, 23, 35, 38, 39, 102)

[56] Hartley, R.I.: Projective Reconstruction from Line Correspondences. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). pp. 903–907. IEEE (1994) (page 19)

[57] Hartley, R.I.: A Linear Method for Reconstructing from Lines and Points. In: Fifth International Conference on Computer Vision (ICCV). pp. 882–887. IEEE (1995) (page 7, 13, 19)

[58] Havlena, M., Schindler, K.: VocMatch: Efficient Multiview Correspondence for Structure from Motion. In: European Conference on Computer Vision (ECCV), pp. 46–60. Springer (2014) (page 3, 23)

[59] Heinly, J., Schoenberger, J.L., Dunn, E., Frahm, J.M.: Reconstructing the World* in Six Days *(As Captured by the Yahoo 100 Million Image Dataset). In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3287–3295. IEEE (2015) (page 11)

[60] Heuel, S., Foerstner, W.: Matching, Reconstructing and Grouping 3D Lines From Multiple Views Using Uncertain Projective Geometry. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). vol. 2, pp. 517–524. IEEE (2001) (page 29)

[61] Hofer, M., Donoser, M., Bischof, H.: Semi-Global 3D Line Modeling For Incremental Structure-from-Motion. In: British Machine Vision Conference (BMVC) (2014) (page 24, 31, 47, 52, 54, 56, 70, 85, 89, 93)

[62] Hofer, M., Maurer, M., Bischof, H.: Improving Sparse 3D Models for Man-Made Environments Using Line-Based 3D Reconstruction. In: International Conference on 3D Vision (3DV). vol. 1, pp. 535–542. IEEE (2014) (page 31, 33, 44, 52, 54, 55, 56)

[63] Hofer, M., Maurer, M., Bischof, H.: Line3D: Efficient 3D Scene Abstraction for the Built Environment. In: German Conference on Computer Vision and Pattern Recognition (GCPR), pp. 237–248. Springer (2015) (page 8, 31, 33, 44, 45, 50, 52, 55, 56, 70)

[64] Hofer, M., Maurer, M., Bischof, H.: Efficient 3D Scene Abstraction using Line Segments. Computer Vision and Image Understanding (CVIU) (2016) (page 33, 56)

[65] Hofer, M., Wendel, A., Bischof, H.: Incremental Line-based 3D Reconstruction using Geometric Constraints. In: British Machine Vision Conference (BMVC). pp. 92.1–92.11 (2013) (page 24, 31, 44, 52, 70, 85, 89, 93)

[66] Hofer, M., Wendel, A., Bischof, H.: Line-based 3D Reconstruction of Wiry Objects. In: Computer Vision Winter Workshop (CVWW). pp. 78–85 (2013) (page 31, 33, 44, 49, 50)

[67] Holzmann, T., Fraundorfer, F., Bischof, H.: Direct Stereo Visual Odometry Based on Lines. In: International Conference on Computer Vision Theory and Applications (VISAPP) (2016) (page 25)

[68] Hoppe, C.: Interactive Structure-from-Motion. Phd thesis, Graz University of Technology (5 2014) (page 9, 11, 35, 86, 88)

[69] Hoppe, C., Klopschitz, M., Donoser, M., Bischof, H.: Incremental Surface Extraction from Sparse Structure-from-Motion Point Clouds. In: British Machine Vision Conference (BMVC). pp. 94.1–94.11 (2013) (page 3, 87)

[70] Hoppe, C., Klopschitz, M., Rumpler, M., Wendel, A., Kluckner, S., Bischof, H., Reitmayr, G.: Online Feedback for Structure-from-Motion Image Acquisition. In: British Machine Vision Conference (BMVC). vol. 2, p. 6 (2012) (page 11, 87, 88, 89, 93, 95, 99, 100)

[71] Huang, T.S.: Motion Analysis. Encyclopedia of Artificial Intelligence pp. 620–632 (1987) (page 19)

[72] Irschara, A., Zach, C., Bischof, H.: Towards Wiki-Based Dense City Modeling. In: IEEE 11th International Conference on Computer Vision (ICCV). IEEE (2007) (page 2, 5, 10, 11, 12, 18, 42, 62, 67, 68, 69, 81, 82, 83, 87, 93, 94, 95, 100)

[73] Jain, A., Kurz, C., Thormaehlen, T., Seidel, H.P.: Exploiting Global Connectivity Constraints for Reconstruction of 3D Line Segments from Images. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1586–1593. IEEE (2010) (page 7, 29, 30, 49, 50, 56, 64, 65, 69, 71, 77, 82, 83, 84, 95, 96, 98)

[74] Jeong, W.Y., Lee, K.M.: Visual SLAM with line and corner features. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 2570–2575. IEEE (2006) (page 24)

[75] Jiang, N., Lin, W.Y., Do, M.N., Lu, J.: Direct Structure Estimation for 3D Reconstruction. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2655–2663. IEEE (2015) (page 12)

[76] Jung, F., Paparoditis, N.: Extracting 3D Free-Form Surface Boundaries of Man-Made Objects from Multiple Calibrated Images: A Robust, Accurate and High Resolving Power Edgel Matching and Chaining Approach. International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences 34(3/W8), 39–46 (2003) (page 27)

[77] Kahl, F., August, J.: Multiview Reconstruction of Space Curves. In: Ninth IEEE International Conference on Computer Vision (ICCV). pp. 1017–1024. IEEE (2003) (page 103)

[78] Kahl, F., Heyden, A.: Affine Structure and Motion from Points, Lines and Conics. International Journal of Computer Vision (IJCV) 33(3), 163–180 (1999) (page 19)

[79] Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data: An Introduction to Cluster Analysis. Wiley Online Library (1990) (page 11)

[80] Khaleghi, B., Baklouti, M., Karray, F.O.: SILT: Scale-Invariant Line Transform. In: IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA). pp. 78–83. IEEE (2009) (page 14, 16)

[81] Kim, C., Manduchi, R.: Planar Structures from Line Correspondences in a Manhattan World. In: Asian Conference on Computer Vision (ACCV), pp. 509–524. Springer (2014) (page 20)

[82] Kim, H., Lee, S.: A Novel Line Matching Method Based on Intersection Context. In: IEEE International Conference on Robotics and Automation (ICRA). pp. 1014–1021. IEEE (2010) (page 15, 20)

[83] Kim, H., Lee, S.: Wide-Baseline Image Matching Based on Coplanar Line Intersections. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1157–1164. IEEE (2010) (page 15, 16)

[84] Kim, H., Lee, S.: Simultaneous line matching and epipolar geometry estimation based on the intersection context of colplanar line pairs. Pattern Recognition Letters 33(10), 1349–1363 (2012) (page 19)

[85] Kim, Z., Nevatia, R.: Automatic description of complex buildings from multiple images. Computer Vision and Image Understanding (CVIU) 96(1), 60–95 (2004) (page 27)

[86] Kitanov, A., Bisevac, S., Petrovic, I.: Mobile robot self-localization in complex indoor environments using monocular vision and 3D model. In: IEEE/ASME international conference on Advanced intelligent mechatronics. pp. 1–6. IEEE (2007) (page 24)

[87] Klein, G., Murray, D.: Parallel tracking and mapping for small AR workspaces. In: 6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR). pp. 225–234. IEEE (2007) (page 24, 25, 26, 88)

[88] Klein, G., Murray, D.: Improving the agility of keyframe-based SLAM. In: European Conference on Computer Vision (ECCV), pp. 802–815. Springer (2008) (page 24)

[89] Kneip, L., Scaramuzza, D., Siegwart, R.: A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2969–2976. IEEE (2011) (page 11, 18, 62, 87)

[90] Krantz, S.G.: Handbook of Complex Variables. Springer (1999) (page 21)

[91] Labatut, P., Pons, J.P., Keriven, R.: 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). IEEE (2007) (page 3)

[92] Lee, Y.H., Nam, C., Lee, K.Y., Li, Y.S., Yeon, S.Y., Doh, N.L.: VPass: Algorithmic compass using vanishing points in indoor environments. In: International Conference on Intelligent Robots and Systems (IROS). pp. 936–941 (2009) (page 24)

[93] Lemaire, T., Lacroix, S.: Monocular-vision based SLAM using Line Segments. In: IEEE International Conference on Robotics and Automation (ICRA). pp. 2791–2796. IEEE (2007) (page 24)

[94] Li, G., Genc, Y., Zucker, S.W.: Multi-View Edge-based Stereo by Incorporating Spatial Coherence. In: Sixth International Conference on 3-D Digital Imaging and Modeling (3DIM). pp. 341–348. IEEE (2007) (page 103)

[95] Li, K., Yao, J., Li, L., Zhang, Z.: Hierarchical line matching based on Line-Junction-Line structure descriptor and local homography estimation. Neurocomputing (2016) (page 15)

[96] Li, K., Yao, J., Lu, X.: Robust Line Matching Based on Ray-Point-Ray Structure Descriptor. In: Asian Conference on Computer Vision (ACCV) Workshops. pp. 554–569. Springer (2014) (page 15)

[97] Li, Y., Snavely, N., Huttenlocher, D.P.: Location Recognition using Prioritized Feature Matching. In: European Conference on Computer Vision (ECCV), pp. 791–804. Springer (2010) (page 66, 67, 69)

[98] Lin, Y., Wang, C., Cheng, J., Chen, B., Jia, F., Chen, Z., Li, J.: Line segment extraction for large scale unorganized point clouds. ISPRS Journal of Photogrammetry and Remote Sensing 102, 172–183 (2015) (page 28)

[99] Liu, Y., Huang, T.S.: Estimation of Rigid Body Motion Using Straight line Correspondences. Computer Vision, Graphics, and Image Processing 43(1), 37–52 (1988) (page 19)

[100] Lopez, J., Santos, R., Fdez-Vidal, X.R., Pardo, X.M.: Two-view line matching algorithm based on context and appearance in low-textured images. Pattern Recognition 48(7), 2164–2184 (2015) (page 15)

[101] Lowe, D.G.: Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision (IJCV) 60(2), 91–110 (2004) (page 2, 5, 7, 9, 10, 14, 18, 21, 42, 62, 68, 81, 86, 87, 88, 94, 100)

[102] Lu, F., Milios, E.: Robot Pose Estimation in Unknown Environments by Matching 2D Range Scans. Journal of Intelligent and Robotic Systems 18(3), 249–275 (1997) (page 24)

[103] Lu, X., Yao, J., Li, K., Li, L.: CannyLines: A parameter-free line segment detector. In: IEEE International Conference on Image Processing (ICIP). pp. 507–511. IEEE (2015) (page 43)

[104] Matas, J., Galambos, C., Kittler, J.: Robust Detection of Lines Using the Progressive Probabilistic Hough Transform. Computer Vision and Image Understanding (CVIU) 78(1), 119–137 (2000) (page 42)

[105] Matinec, D., Pajdla, T.: Line Reconstruction from Many Perspective Images by Factorization. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). vol. 1, pp. 497–502. IEEE (2003) (page 19)

[106] McIntosh, J.H., Mutch, K.M.: Matching Straight Lines. Computer Vision, Graphics, and Image Processing 43(3), 386–408 (1988) (page 7, 13)

[107] McLauchlan, P.F., Murray, D.W.: A unifying framework for structure and motion recovery from image sequences. In: Fifth International Conference on Computer Vision (ICCV). pp. 314–320. IEEE (1995) (page 19)

[108] Medioni, G., Nevatia, R.: Matching Images Using Linear Features. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 6, 675–685 (1984) (page 7, 13)

[109] Medioni, G., Nevatia, R.: Segment-Based Stereo Matching. Computer Vision, Graphics, and Image Processing 31(1), 2–18 (1985) (page 7, 13)

[110] Micusik, B., Wildenauer, H.: Structure from Motion with Line Segments Under Relaxed Endpoint Constraints. In: International Conference on 3D Vision (3DV). vol. 1, pp. 13–19. IEEE (2014) (page 8, 21, 22, 23)

[111] Mikolajczyk, K., Schmid, C.: A Performance Evaluation of Local Descriptors. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 27(10), 1615–1630 (2005) (page 86)

[112] Montiel, J.M.M., Tardos, J.D., Montano, L.: Structure and motion from straight line segments. Pattern Recognition 33(8), 1295–1307 (2000) (page 19)

[113] Moons, T., Frere, D., Vandekerckhove, J., Gool, L.V.: Automatic modelling and 3D reconstruction of urban house roofs from high resolution aerial imagery. In: European Conference on Computer Vision (ECCV), pp. 410–425. Springer (1998) (page 27, 103)

[114] Moulon, P., Monasse, P., Marlet, R., Others: OpenMVG. An Open Multiple View Geometry library. `https://github.com/openMVG/openMVG` (page 1, 12, 18, 34)

[115] Mouragnon, E., Lhuillier, M., Dhome, M., Dekeyser, F., Sayd, P.: Real Time Localisation and 3D Reconstruction. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). vol. 1, pp. 363–370. IEEE (2006) (page 87)

[116] Mouragnon, E., Lhuillier, M., Dhome, M., Dekeyser, F., Sayd, P.: Generic and Real-Time Structure from Motion. In: British Machine Vision Conference (BMVC). vol. 7, p. 6 (2007) (page 87)

[117] Nakayama, Y., Honda, T., Saito, H., Shimizu, M., Yamaguchi, N.: Accurate camera pose estimation for kinectfusion based on line segment matching by LEHF. In: 22nd International Conference on Pattern Recognition (ICPR). pp. 2149–2154. IEEE (2014) (page 26)

[118] Newcombe, R.A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A.J., Kohli, P., Shotton, J., Hodges, S., Fitzgibbon, A.: KinectFusion: Real-time dense

surface mapping and tracking. In: IEEE International Symposium on Mixed and Augmented Reality (ISMAR). pp. 127–136. IEEE (2011) (page 25, 26)

[119] Nister, D.: An efficient solution to the five-point relative pose problem. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 26(6), 756–770 (2004) (page 18, 39)

[120] Nister, D., Stewenius, H.: Scalable Recognition with a Vocabulary Tree. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). vol. 2, pp. 2161–2168. IEEE (2006) (page 3, 10, 11, 23, 44, 62, 94)

[121] Noronha, S., Nevatia, R.: Detection and modeling of buildings from multiple aerial images. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 23(5), 501–518 (2001) (page 27)

[122] Nurutdinova, I., Fitzgibbon, A.: Towards Pointless Structure from Motion: 3D reconstruction and camera parameters from general 3D curves. In: IEEE International Conference on Computer Vision (ICCV). pp. 2363–2371. IEEE (2015) (page 105)

[123] Ok, A.O., Wegner, J.D., Heipke, C., Rottensteiner, F., Soergel, U., Toprak, V.: Accurate matching and reconstruction of line features from ultra high resolution stereo aerial images. In: Proceedings of ISPRS Hannover Workshop, High Resolution Earth Imaging for Geospatial Information, Hannover, Germany (2011) (page 27, 28)

[124] Oliva, A., Torralba, A.: Modeling the shape of the scene: a holistic representation of the spatial envelope. International Journal of Computer Vision (IJCV) 42(3), 145–175 (2001) (page 10)

[125] Park, Y.B., Kim, S.S., Suh, I.H.: Visual Recognition of Types of Structural Corridor Landmarks Using Vanishing Points Detection and Hidden Markov Models. In: International Conference on Pattern Recognition (ICPR). pp. 3292–3295. IEEE (2010) (page 24)

[126] Patraucean, V., Gurdjos, P., von Gioi, R.G.: A parameterless line segment and elliptical arc detector with enhanced ellipse fitting. In: European Conference on Computer Vision (ECCV), pp. 572–585. Springer (2012) (page 105)

[127] Quan, L., Kanade, T.: Affine Structure from Line Correspondences With Uncalibrated Affine Cameras. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 19(8), 834–845 (1997) (page 19)

[128] Radenovic, F., Schoenberger, J.L., Ji, D., Frahm, J.M., Chum, O., Matas, J.: From Dusk Till Dawn: Modeling in the Dark. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE (2016) (page 11)

[129] Ramalingam, S., Antunes, M., Snow, D., Lee, G.H., Pillai, S.: Line-Sweep: Cross-Ratio for Wide-Baseline Matching and 3D Reconstruction. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 1238–1246. IEEE (2015) (page 22, 23)

[130] Rother, C.: Linear Multi-view Reconstruction of Points, Lines, Planes and Cameras using a Reference Plane. In: Ninth IEEE International Conference on Computer Vision (ICCV). pp. 1210–1217. IEEE (2003) (page 19)

[131] Rothermel, M., Wenzel, K., Fritsch, D., Haala, N.: SURE: Photogrammetric Surface Reconstruction from Imagery. In: Proceedings LC3D Workshop, Berlin. vol. 8 (2012) (page 2, 39)

[132] Schindler, G., Krishnamurthy, P., Dellaert, F.: Line-Based Structure from Motion for Urban Environments. In: Third International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT). pp. 846–853. IEEE (2006) (page 7, 20)

[133] Schindler, K., Bauer, J.: Towards feature-based building reconstruction from images. International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG) (2003) (page 27)

[134] Schmid, C., Zisserman, A.: Automatic Line Matching across Views. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). pp. 666–671. IEEE (1997) (page 13, 27, 28)

[135] Schmid, C., Zisserman, A.: The Geometry and Matching of Lines and Curves Over Multiple Views. International Journal of Computer Vision (IJCV) 40(3), 199–233 (2000) (page 13)

[136] Schoenberger, J.L., Fraundorfer, F., Frahm, J.M.: Structure-from-motion for mav image sequence analysis with photogrammetric applications. The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences 40(3), 305 (2014) (page 1, 12)

[137] Schoenberger, J.L., Frahm, J.M.: Structure-from-Motion Revisited. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE (2016) (page 1, 12, 62, 95, 101)

[138] Schoenberger, J.L., Radenovic, F., Chum, O., Frahm, J.M.: From Single Image Query to Detailed 3D Reconstruction. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 5126–5134. IEEE (2015) (page 11)

[139] Scholze, S., Moons, T., Gool, L.V.: A probabilistic approach to roof extraction and reconstruction. International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences 34(3/B), 231–236 (2002) (page 27, 103)

[140] Seo, Y., Hong, K.S.: Sequential Reconstruction of Lines in Projective Space. In: Proceedings of the 13th International Conference on Pattern Recognition (ICPR). vol. 1, pp. 503–507. IEEE (1996) (page 19)

[141] Sinha, S.N., Steedly, D., Szeliski, R.: Piecewise Planar Stereo for Image-based Rendering. In: IEEE 12th International Conference on Computer Vision (ICCV). pp. 1881–1888 (2009) (page 27, 103)

[142] Smith, P., Reid, I.D., Davison, A.J.: Real-Time Monocular SLAM with Straight Lines. In: British Machine Vision Conference (BMVC). vol. 6, pp. 17–26 (2006) (page 24)

[143] Snavely, N., Seitz, S., Szeliski, R.: Photo Tourism: Exploring image collections in 3D. ACM Transactions on Graphics (SIGGRAPH) (2006) (page 1, 7, 9, 10, 12, 18, 28, 62, 67)

[144] Sola, J., Vidal-Calleja, T., Civera, J., Montiel, J.M.M.: Impact of landmark parametrization on monocular EKF-SLAM with points and lines. International Journal of Computer Vision (IJCV) 97(3), 339–368 (2012) (page 24)

[145] Sola, J., Vidal-Calleja, T., Devy, M.: Undelayed initialization of line segments in monocular SLAM. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 1553–1558. IEEE (2009) (page 24)

[146] Strecha, C., von Hansen, W., Gool, L.V., Fua, P., Thoennessen, U.: On Benchmarking Camera Calibration and Multi-View Stereo for High Resolution Imagery. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE (2008) (page 64, 65, 68, 70, 71, 72, 77, 82, 83, 84, 95, 96, 98)

[147] Strecha, C., Kueng, O., Others: Pix4D, UAV mapping software. `https://pix4d.com/` (page 1, 3, 12, 18)

[148] Sugiura, T., Torii, A., Okutomi, M.: 3d surface reconstruction from point-and-line cloud. In: International Conference on 3D Vision (3DV). pp. 264–272. IEEE (2015) (page 102)

[149] Sweeney, C.: Theia multiview geometry library: Tutorial & reference, university of California Santa Barbara. (page 1, 12)

[150] Sweeney, C., Sattler, T., Hoellerer, T., Turk, M., Pollefeys, M.: Optimizing the Viewing Graph for Structure-from-Motion. In: IEEE International Conference on Computer Vision (ICCV). pp. 801–809. IEEE (2015) (page 9, 12)

[151] Tang, A.W.K., Ng, T.P., Hung, Y.S., Leung, C.H.: Projective reconstruction from line-correspondences in multiple uncalibrated images. Pattern Recognition 39(5), 889–896 (2006) (page 19)

[152] Tarrio, J.J., Pedre, S.: Realtime edge-based visual odometry for a monocular camera. In: IEEE International Conference on Computer Vision (ICCV). pp. 702–710 (2015) (page 26, 88)

[153] Taylor, C.J., Kriegman, D.J.: Structure and Motion from Line Segments in Multiple Images. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 17(11), 1021–1032 (1995) (page 7, 19)

[154] Teney, D., Piater, J.: Sampling-based Multiview Reconstruction without Correspondences for 3D Edges. In: Second International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT). pp. 160–167. IEEE (2012) (page 103)

[155] Triggs, B.: Factorization Methods for Projective Structure and Motion. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR). pp. 845–851. IEEE (1996) (page 19)

[156] Triggs, B., McLauchlan, P., Hartley, R.I., Fitzgibbon, A.W.: Bundle Adjustment — A Modern Synthesis. In: Vision Algorithms: Theory and Practice, pp. 298–372. Springer (1999) (page 9, 57, 87, 93)

[157] Verhagen, B., Timofte, R., van Gool, L.: Scale-invariant line descriptors for wide baseline matching. In: IEEE Winter Conference on Applications of Computer Vision (WACV). pp. 493–500. IEEE (2014) (page 16)

[158] Vieville, T., Faugeras, O.: Feed-Forward Recovery of Motion and Structure from a Sequence of 2D-Lines Matches. In: Third International Conference on Computer Vision (ICCV). pp. 517–520. IEEE (1990) (page 19)

[159] Vieville, T., Faugeras, O., Luong, Q.T.: Motion of Points and Lines in the Uncalibrated Case. International Journal of Computer Vision (IJCV) 17(1), 7–41 (1996) (page 19)

[160] Waechter, M., Moehrle, N., Goesele, M.: Let There Be Color! Large-Scale Texturing of 3D Reconstructions. In: European Conference on Computer Vision (ECCV), pp. 836–850. Springer (2014) (page 3)

[161] Wang, L., Neumann, U., You, S.: Wide-Baseline Image Matching Using Line Signatures. In: IEEE 12th International Conference on Computer Vision (ICCV). pp. 1311–1318. IEEE (2009) (page 15, 16)

[162] Wang, Z.H., Zhi, S.S., Liu, H.M.: MSHS: The Mean-Standard Deviation Curve Matching Algorithm in HSV Space. In: International Conference on Machine Learning and Cybernetics (ICMLC). vol. 3, pp. 1064–1069. IEEE (2012) (page 14)

[163] Wang, Z., Wu, F., Hu, Z.: MSLD: A Robust Descriptor for Line Matching. Pattern Recognition 42(5), 941–953 (2009) (page 7, 14, 16, 20, 43)

[164] Wei, S., Yagi, Y., Yachida, M.: Building local floor map by use of ultrasonic and omni-directional vision sensor. In: IEEE International Conference on Robotics and Automation (ICRA). vol. 3, pp. 2548–2553. IEEE (1998) (page 24)

[165] Weng, J., Huang, T.S., Ahuja, N.: Motion and Structure from Line Correspondences: Closed-Form Solution, Uniqueness, and Optimization. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 3, 318–336 (1992) (page 19)

[166] Werner, T., Zisserman, A.: New Techniques for Automated Architectural Reconstruction from Photographs. In: European Conference on Computer Vision (ECCV), pp. 541–555. Springer (2002) (page 27, 103)

[167] Wilson, K., Snavely, N.: Robust Global Translations with 1DSfM. In: European Conference on Computer Vision (ECCV), pp. 61–75. Springer (2014) (page 12)

[168] Witt, J., Mentges, G.: Maximally Informative Surface Reconstruction from Lines. In: International Conference on Robotics and Automation (ICRA). pp. 2029–2036. IEEE (2014) (page 103)

[169] Woo, D.M., Han, S.S., Park, D.C., Nguyen, Q.D.: Extraction of 3D line segment using digital elevation data. In: Congress on Image and Signal Processing (CISP). vol. 2, pp. 734–738. IEEE (2008) (page 27)

[170] Woo, D.M., Park, D.C.: 3D line segment detection based on disparity data of area-based stereo. In: WRI Global Congress on Intelligent Systems (GCIS). vol. 4, pp. 219–223. IEEE (2009) (page 27)

[171] Woo, D.M., Park, D.C., Han, S.S.: Extraction of 3D line segment using disparity map. In: International Conference on Digital Image Processing. pp. 127–131. IEEE (2009) (page 27)

[172] Wu, C.: Towards linear-time Incremental Structure-from-Motion. In: International Conference on 3D Vision (3DV). pp. 127–134. IEEE (2013) (page 1, 9, 11, 12, 18, 34, 101)

[173] Wu, C., Agarwal, S., Curless, B., Seitz, S.M.: Multicore Bundle Adjustment. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 3057–3064. IEEE (2011) (page 11)

[174] Yen, B.L., Huang, T.S.: Determining 3-D Motion and Structure of a Rigid Body Using Straight Line Correspondences. In: Image sequence processing and dynamic scene analysis, pp. 365–394. Springer (1983) (page 19)

[175] Zhang, G., Kang, D.H., Suh, I.H.: Loop closure through vanishing points in a line-based monocular SLAM. In: IEEE International Conference on Robotics and Automation (ICRA). pp. 4565–4570. IEEE (2012) (page 24, 25)

[176] Zhang, G., Suh, I.H.: Building a partial 3D line-based map using a monocular SLAM. In: IEEE International Conference on Robotics and Automation (ICRA). pp. 1497–1502. IEEE (2011) (page 24, 25)

[177] Zhang, L., Ghosh, B.K.: Line Segment Based Map Building and Localization Using 2D Laser Rangefinder. In: IEEE International Conference on Robotics and Automation (ICRA). vol. 3, pp. 2538–2543. IEEE (2000) (page 24)

[178] Zhang, L., Koch, R.: Hand-held Monocular SLAM Based on Line Segments. In: Irish Machine Vision and Image Processing Conference (IMVIP). pp. 7–14. IEEE (2011) (page 24)

[179] Zhang, L., Koch, R.: An efficient and robust line segment matching approach based on LBD descriptor and pairwise geometric consistency. Journal of Visual Communication and Image Representation 24(7), 794–805 (2013) (page 7, 14, 15, 21)

[180] Zhang, L., Koch, R.: Structure from motion from line correspondences: Representation, projection, initialization and sparse bundle adjustment. Journal of Visual Communication and Image Representation 25(5), 904–915 (2014) (page 8, 21, 22, 23, 56, 59)

[181] Zhang, L., Xu, C., Lee, K.M., Koch, R.: Robust and Efficient Pose Estimation from Line Correspondences. In: Asian Conference on Computer Vision (ACCV), pp. 217–230. Springer (2012) (page 13, 21, 43)

[182] Zhang, Y., Yang, H., Liu, X.: A Line Matching Method based on Local and Global Appearance. In: 4th International Congress on Image and Signal Processing (CISP). vol. 3, pp. 1381–1385. IEEE (2011) (page 7, 14, 43)

[183] Zhang, Z.: Estimating Motion and Structure from Correspondences of Line Segments between Two Perspective Views. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI) 17(12), 1129–1139 (1995) (page 19)

[184] Zhou, H., Zou, D., Pei, L., Ying, R., Liu, P., Yu, W.: StructSLAM: Visual SLAM With Building Structure Lines. IEEE Transactions on Vehicular Technology 64(4), 1364–1375 (2015) (page 24, 25)