University of Alberta

Incremental Free-Space Carving for Real-Time 3D Reconstruction

by

David Israel Lovi

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Fall 2011
Edmonton, Alberta

Abstract

This thesis addresses the problem of automatic real-time 3D reconstruction of general scenes from monocular video. Whereas many impressively accurate reconstruction techniques exist in the multi-view stereo literature, most are slow offline batch methods designed to work in highly calibrated settings. Real-time reconstruction opens doors for real-time applications. This thesis presents a swift but approximate incremental method based on visibility and free-space constraints. "Free space" refers to the observation that lines of sight between photographed surfaces and their observing cameras must be empty; otherwise the surfaces would be occluded. Our approach begins with a sparse reconstruction from online Structure from Motion and interpolates the resulting points in a free-space aware manner to produce a physically consistent dense 3D model. We validate our algorithms on real and synthetic data, perform complexity analysis proving the real-time quality, and demonstrate the algorithms' usefulness for improving visualization in a tele-robotics context.

Acknowledgements

My friends, teachers, colleagues, and family have all influenced my work as well as my academic and personal self. I am deeply grateful to all of you.

First, I would like to thank my supervisor, Martin Jagersand, as well as his wife and my unwritten co-supervisor Dana Cobzas. You introduced me to computer vision, taught me and guided me through my degree, nurtured my skills, put up with my epic procrastinations, and even traded bread with me. The bread was delicious.

All of my colleagues have shaped the positive experience that I had over the past four years. For this I say thanks. Alejandro Hernandez-Herdocia deserves special mention for his time spent on the IROS tele-robotics project. His expertise with the WAM was essential. I consider Adam Rachmielowski as my predecessor, as his work before mine was toward the same project and goals.

Uncountable thanks go out to Neil Birkbeck, the most driven person I have ever met. He helped me greatly, both through insightful discussion and by sharing code. In fact, the IROS tele-robotic system's software was essentially developed by him. All I had to do was integrate my code-base and algorithms. Needless to say, this software was a giant undertaking and an extraordinary help.

My lab mates and my friends are not without overlap. Thank you, Karteek Popuri, for being awesomely ridiculous and thereby ridiculously awesome. Neil, we had good times both in Paris and back at home, even when just griping at each other. Now that you have moved away, you are missed more than you know.

Azad Shademan, Sheehan Khan, and of course my supervisor Martin Jagersand as well as my defence committee John Bowman and Pierre Boulanger, have all provided invaluable feedback regarding the writing of this document.

I thank all the teachers that have raised me academically from the beginning. I dare not declare this list exhaustive, but Mr. Bratu, Mr. Millar, Mr. Smilanich, and Mr. McNab were particularly influential. Weirding out my high-school computing teacher, Harvey Duff, was extremely fun; I look forward to the next time we bump into each other. Among the first and most important teachers in my life were those in my immediate family. Thank you, my brother Aaron, for giving me a spark and a head start on computing and math.

And thank you to my parents, Ita and Efim Lovi, and my friends, for being the excellent people that you are.

Finally, thanks whoever stole my birthday cake! The memory will forever be hilarious. You even returned the plates.

Table of Contents

1 Introduction **1**
- 1.1 Motivation . . . 3
- 1.2 Contributions and Contents . . . 4

2 Background **5**
- 2.1 Camera Model . . . 5
- 2.2 Triangulation and Structure from Motion . . . 7
- 2.3 Delaunay Triangulation . . . 8

3 The Literature **11**
- 3.1 3D Shape Reconstruction From Images . . . 11
  - 3.1.1 Offline Reconstruction . . . 11
  - 3.1.2 Free-Space Methods . . . 15
  - 3.1.3 Real-time / Near Real-time Reconstruction . . . 16
  - 3.1.4 Shape From Points . . . 18
- 3.2 SLAM / Online Structure from Motion . . . 22
  - 3.2.1 Recursive Filtering . . . 22
  - 3.2.2 Online Bundle Adjustment . . . 25
- 3.3 Tele-robotics . . . 25
  - 3.3.1 Photo-Realistic Predictive Display . . . 26

4 Method and Algorithms: Incremental Free-Space Carving **28**
- 4.1 Free Space . . . 29
- 4.2 Inputs and Representation . . . 30
- 4.3 Algorithms . . . 31
  - 4.3.1 Commonalities . . . 31
  - 4.3.2 Keyframe Insertion . . . 32
  - 4.3.3 Data Association and Dissociation . . . 33
  - 4.3.4 Outlier Deletion . . . 34
  - 4.3.5 Refinement . . . 35
  - 4.3.6 Forgetting Heuristic . . . 36
- 4.4 Isosurface Extraction and Regularization . . . 36
- 4.5 Computational Complexity . . . 38
- 4.6 Software System . . . 39

5 Experiments **41**
- 5.1 Reconstruction Results . . . 41
- 5.2 Timings and Heuristic Evaluation . . . 44
- 5.3 Synthetic Data Evaluation . . . 45
- 5.4 Application: Predictive Display for Tele-Robotics . . . 51
  - 5.4.1 System . . . 52
  - 5.4.2 Alignment Experiment . . . 54
  - 5.4.3 Inspection Experiment . . . 55

6 Conclusions **57**
- 6.1 Limitations and Future Work . . . 58

Bibliography **60**

A Complexity Proofs **66**

List of Figures

2.1 Pinhole camera . . . 6
2.2 Triangulation problem . . . 8
2.3 3D Delaunay triangulation . . . 9
2.4 2D Delaunay triangulation . . . 9

4.1 Free space: the general concept and our discrete representation . . . 29
4.2 The exact discrete free-space problem . . . 30
4.3 Reconstruction system . . . 32
4.4 A 2D illustration of Algorithm 1, Keyframe Insertion . . . 33
4.5 The traversal algorithm for processing a free-space constraint . . . 34
4.6 Free-space constraint similarity measure . . . 36
4.7 Illustration of the method's graph-cut regularization . . . 37

5.1 Three models reconstructed in real-time from video without regularization . . . 42
5.2 A model reconstructed in real-time from video with regularization . . . 43
5.3 Efficiency for $K = 1, 5$, and ∞ on a representative dataset . . . 44
5.4 Forgetting heuristic comparison . . . 45
5.5 Synthetic-data point cloud generation . . . 46
5.6 Ground truth meshes . . . 46
5.7 View sampling conditions for all the synthetic data experiments . . . 47
5.8 Noiseless reconstructions . . . 48
5.9 Noisy reconstructions . . . 48
5.10 Accuracy error as a function of sampling noise . . . 49
5.11 Accuracy error as a function of outlier ratio . . . 50
5.12 Dense Cup reconstruction for several outlier ratios . . . 51
5.13 Reconstructions with regularization versus without . . . 52
5.14 Tele-robotics system . . . 53
5.15 Illustration of predictive display . . . 53
5.16 Tele-robotics alignment task . . . 55
5.17 Mean normalized times to perform the tele-robotic tasks . . . 55
5.18 Tele-robotics inspection task . . . 56

Chapter 1

Introduction

This thesis addresses the central yet difficult problem in computer vision that is the automatic recovery of 3D shape from camera imagery. Specifically, given a collection of photographs or video frames redundantly observing a scene in our three dimensional world, and taken from multiple distinct viewpoints, the problem is to compute and reconstruct some representation of the geometry of that scene such as a conventional 3D graphics mesh.

Many techniques have been devised in an attempt to solve this problem, however all solutions thus far entail limiting assumptions, approximations, and compromises within their respective formulations. No known solution is general enough to truly *solve* the problem for all cases and applications (and our method is no exception). The primary focus of the research community has been on attaining the most detailed accurate and complete geometry possible given the inputs.

The multi-view stereo literature conveys the common approach, which is to use all the texture[1] and color information in the images to contribute to the estimation of shape: the assumption is that each point of the scene geometry should look the same from any imaged viewpoint, *i.e.*, have the same local image color and texture. This notion is called photo-consistency (and *e.g.* it is violated for mirrors or specular objects that do not look the same from distinct viewpoints). Many algorithms centred on this reconstruction cue have been developed that achieve impressively accurate 3D reconstructions from just handfuls of images. However, as a result of optimizing geometry using texture and color information densely, almost all published methods result in slow, offline batch operation, requiring many minutes to hours of processing time for just tens of input images [95]. A few recently developed exceptions exist that can operate in real time, but they require extreme computational horsepower and parallelized (GPU) hardware [77, 80]. This can be a problem *e.g.* for robotics applications where onboard hardware is limited. We review multi-view stereo and other techniques for 3D reconstruction in Chapter 3.

This thesis develops and explores a method for 3D reconstruction with an emphasis on speed over geometric detail. The approach relies on different assumptions from the multi-view stereo

[1] Texture, in this context, refers to the pattern of brightness or luminosity on an imaged surface. This is related to tactile texture, as distinct bumps or gratings on a surface will produce a corresponding distinct contrast when that surface is lit and viewed.

norm, is comparably lightweight and provably real-time, and in this way attempts to broaden the applicability of 3D reconstruction to new domains.

Our approach begins with a Structure from Motion (SFM) starting point. Structure from Motion refers to the problem of estimating *sparse*, usually point-wise, 3D geometry from corresponding image features that are photo-consistently matched across more than one view. Rather than a dense 3D model, the output is a point cloud that samples the scene surfaces. The geometry of imaging is well studied and directly applicable [46], and robust real-time solutions to the SFM problem exist [61].[2] From a single moving camera, real-time SFM computes a set of 3D scene points, a camera-pose track, and visibility information relating which points were visible from each camera vantage point. This problem is fundamentally easier than dense 3D shape estimation because only the most reliably estimable features need be matched and reconstructed. Ambiguous regions such as mirrors or textureless parts of monotone walls need not be considered. We summarize Structure from Motion in Chapter 2 and review real-time solutions in Chapter 3.

We take the points, camera track, and visibility information from SFM as input to our algorithms, thus the method inherits a sparse feature-based approach. The algorithms coherently "connect the dots" to output a dense interpolative 3D surface reconstruction. Interpolation is achieved by using the visibility information to reason about where the scene surface can and cannot be. The concept is called "free space," [35, 49, 103] and it is presented in Section 4.1. Essentially, we know that the space between photographed surfaces and their observing cameras must be empty, because otherwise the surfaces would have been occluded. This geometric constraint is the method's primary reconstruction cue. Combining free space with the right adaptive spatial discretization results in a favourable impact on the viewpoint or visibility sampling requirements as well as the computational complexity of our algorithms. We introduce the Delaunay discretization in Chapter 2 and review its known properties with respect to surface reconstruction in Section 3.1.4.

Since real-time SFM is an online process, as new video frames become available, our algorithms' inputs continuously change online. Therefore, more precisely, our algorithms take small incremental changes to the point cloud, camera track, and visibility as input. These incremental changes can be classified as one of several events, *e.g.*, the SFM system's addition of a new keyframe view[3], or the deletion of an outlier point. The method encompasses a set of five algorithms to handle five general types of changes in an incremental and event-driven fashion. The incremental nature of the algorithms offers the speed necessary for real-time reconstruction from video sequences. The algorithms are presented in Chapter 4, Method and Algorithms.

[2] In the robotics literature, Simultaneous Localization and Mapping, or SLAM for short, effectively reduces to the same real-time problem when the robot's sensor is a single camera.

[3] A keyframe is just a video frame chosen from a video sequence for special use. The selection scheme varies from system to system. Keyframes can be selected periodically, *e.g.* every 30 frames, or at spatially disparate locations, *e.g.* when the camera has moved a threshold distance away from all previous keyframe camera positions.

1.1 Motivation

The motivation for real-time reconstruction can be found in its potential applications. From the perspective of utility, solving even the offline version of the problem is of great interest. Whenever an artist would need to undergo the labour-intensive process of creating a 3D model of a real-world object or scene, such as for the games or cinematography industry, the labour could be saved and automated away. For example, vision-based reconstruction has been used for special effects in films like The Matrix [26], and it has been used for digitally preserving and capturing historical and archaeological sites or artefacts [86, 104]. However, if automatic modeling can be performed quickly enough, real-time applications become possible.

An example is augmented reality, where renderings of virtual objects are mixed with real-world footage [6]. To realistically meld in virtual objects, a 3D model of the real-world scene allows for correct mutual occlusions when rendering them, and it facilitates physical interaction and contact in animation. In the absence of a priori scene models, real-time modeling enables pick-up and go augmented reality as demonstrated in [80]. Other applications of real-time modeling include improving 3D visual modeling itself. Typically multi-view stereo reconstructs the object or scene of interest from images captured separately at an earlier time. If the images taken did not observe the subject's surfaces everywhere sufficiently or did not cover enough variety in viewpoints, one may have to repeat the interleaved processes of image capture and computation until the image sampling suffices. This is laborious, especially when the subject is a far distance away from the computer. Real-time modeling with visualization gives online feedback regarding whether the image sampling is adequate, and images captured with such feedback can be used for more accurate offline reconstruction later, when computational power is less restricted [91, 93].

Real-time 3D modeling is generally useful whenever 3D geometry cannot be known in advance. Robotic operation in unknown environments is one such important case. For example, 3D modeling of the environment's obstacles, and specifically the explicit estimation of where obstacles are not (*i.e.*, free space), can supply constraints for autonomous robot navigation and motion-planning algorithms. We have applied the method developed in this thesis to predictive display for tele-robotics, and experiment with this application in Chapter 5. Tele-robotics refers to the scenario where a human directly operates a distant remote robot. Such remote tele-operation lends itself to many sub-applications where it would be dangerous or impractical for a human to perform the robot's task directly. Space and marine robotics, tele-surgery, and remote bomb defusal are just a few examples. Because of the distance, both operator commands and sensory feedback need to be transmitted, and latency on the order of a fraction of a second in this transmission loop can be detrimental to the fluidity and performance of such a system [96, 47, 33, 36]. Online capture of the robot's environment enables undelayed predictive rendering for the operator's visual feedback. We discuss such predictive display at greater length in Chapter 3, The Literature, as well as in Chapter 5, Experiments.

1.2 Contributions and Contents

The remainder of this thesis is organized as follows, and the contributions of each part are highlighted.

Chapter 2, Background, introduces the very basics of how to obtain three dimensional information from two dimensional imagery. This section does not contain a novel contribution in and of itself, but rather provides a quick primer on the requisite terminology and geometry of shape recovery. In particular, we describe Structure from Motion. We also define the Delaunay triangulation here, a spatial decomposition that is intimately fused with the method developed in this thesis.

Chapter 3, The Literature, surveys what is most relevant to this thesis. Our work relates strongly to those on 3D reconstruction from imagery, or multi-view stereo, as it is essentially this problem that we are concerned with, but with an added constraint on execution time. These works are reviewed to situate our own, and particular attention is paid to those sparse few that consider free space or are real-time. Additionally, this chapter reviews works on the related Shape from Points literature which speaks about the Delaunay triangulation and why one may want to use it for shape estimation. Since the thesis' algorithms rely on online Structure from Motion or Simultaneous Localization and Mapping (SLAM), we cover these as well. Finally, we recount works on online visual modeling as it applies to predictive display and tele-robotics, because this is a setting that we experiment with.

Chapter 4, Method and Algorithms: Incremental Free-Space Carving, presents in full the primary contribution of this thesis that is the shape recovery method and its implementation. We formalize the inputs and the problem and discuss free space as a reconstruction cue in general. We present a regularization scheme, as well as our heuristics and approximations. This chapter additionally analyzes the method's computational complexity. In combination with timings from the next chapter, the complexity results solidify the claim that the method is real-time.

Chapter 5, Experiments, exhibits online reconstructions obtained from our method and system. We present timings, as well as evidence that a heuristic that we employ for speed-up implies minimal sacrifice with respect to reconstruction quality. Using synthetic data and corresponding ground truth, we explore the method's behaviour on inputs of varying quality and sampling properties. Also, we present a prototype tele-robotics system which uses the real-time modeling for predictive display. Here, we document two small-scale human-factors experiments performed with this system.

Chapter 6, Conclusions, closes the thesis by summarizing what was done. This part identifies the important caveats and limitations of this thesis' contributions, and discusses potential routes for future work to address these lacks.

Finally, the Appendix completes the complexity proofs for our algorithms.

Chapter 2

Background

This chapter covers the requisite basics of geometric computer vision that are necessary to understand the remainder of this thesis. After reading this chapter, the reader should have a general understanding and vocabulary regarding basic estimation of 3D scene structure and camera position from 2D photographs. This is a brief and selective primer; algorithmic and algebraic details and alternatives for computing the discussed quantities are largely outside the scope of this thesis. Most content in this chapter is influenced by portions of Hartley and Zisserman's excellent book on this topic [46]. We refer the reader to this resource for more detail on the geometry of computer vision.

Because we use the Delaunay Triangulation heavily in this thesis, this chapter also reviews this geometric entity. The chapter contains the following major sections:

- "Camera Model" explains the linear pinhole camera model used in computer vision.
- "Triangulation and Structure from Motion" describes how to estimate 3D structure from 2D image matches.
- "Delaunay Triangulation" presents the 2D and 3D Delaunay triangulation, which are combinatorial structures connecting sets of points in space.

2.1 Camera Model

To estimate 3D information from 2D photographs, we first need to understand and model the image formation process before we can attempt to invert it. The simplest and most common image formation model is the linear pinhole camera. A pinhole camera is essentially just a box with a small aperture or pin prick on one face, as in Figure 2.1. Scene objects emit and reflect rays of light, and those rays directed at the camera will pass through the aperture and fall onto the inside of the opposite box face. Consider this box face the camera's film. In this way, an image of the scene is projected onto a planar film, or so-called image plane. Modern camera lenses approximate the aperture.

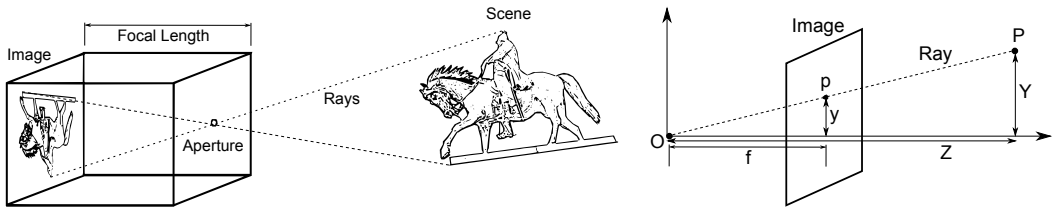


Figure 2.1: Pinhole camera projection. Left: Rays of light reflected off scene objects pass through the pinhole aperture into the camera onto the image face. Right: The geometry of a pinhole camera. The aperture is at optic center O. The image plane can equivalently be considered a focal length f in front of O instead of behind. The intersection of the image plane with rays from any scene point $\mathbf{P}$ through the optic center O defines the position of the image point $\mathbf{p}$.

Figure 2.1 identifies some important geometric features of a pinhole camera. First, the aperture's position in the 3D world is called the optic center, which we denote as O. A coordinate frame is affixed to this camera center, therefore the notation O is also suggestive of the origin. The distance between the camera center and the image plane is called the focal length, f. We can see from the figure that a pinhole camera image is vertically flipped with respect to the world. To abstract away the flip, we can equivalently consider a virtual image plane the same distance f in front of the optic center, as in the right-hand part of the figure.

Let $\mathbf{P} = (X, Y, Z)^T$ be a three dimensional point on a scene surface, and $\mathbf{p} = (x, y)^T$ be its projected image position. Then, by similar triangles in Figure 2.1, the projection process follows the following equation: $\frac{Y}{Z} = \frac{y}{f} \rightarrow y = \frac{fY}{Z}$, and equivalently for the x coordinate: $x = \frac{fX}{Z}$.

It is mathematically convenient to use homogeneous coordinates for projective geometry. Infinity is handled elegantly, and equations can be written in linear form. In homogeneous coordinates, a 3D point $\mathbf{P}$ is represented by a 4-coordinate vector up to an arbitrary scale factor. That is, $\mathbf{P} = (X, Y, Z, 1)^T \sim (\lambda X, \lambda Y, \lambda Z, \lambda)^T$, where $\sim$ denotes equivalence w.r.t. projective scale λ. In homogeneous coordinates, $\mathbf{P} \sim (X, Y, Z, 1)^T$ and $\mathbf{p} \sim (x, y, 1)^T$, and we can write the projection equation as a simple linear matrix multiplication with a projection matrix $\mathbf{C}$.

$$\mathbf{p} \sim \mathbf{CP} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{P}$$

This projection matrix does not consider pixel to world-unit (*e.g.* mm) scaling, nor that the center of the image may not be at the image coordinate frame's origin; *e.g.* considering the top-left pixel of an image to be the image origin is common. Therefore, the projection matrix can be written more generally as:

$$\mathbf{p} \sim \mathbf{CP} = \mathbf{K} \left[\mathbf{I}|\mathbf{0}\right] \mathbf{P},$$
$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}.$$

Here, f_x and f_y are the focal length scaled to account for the pixel size in world coordinates, *i.e.* f_x is the focal length in units of pixel widths, and $(c_x, c_y)^T$ specifies the pixel location of the

principle point which is the intersection of the camera coordinate frame's Z axis with the image plane, ideally at the center of the image. The parameter s represents a skew to account for non-perfectly rectangular pixel grids. Generally, the skew is negligible in real cameras and considered zero. $[\mathbf{I}|\mathbf{0}]$ is the 3 by 4 matrix with its left 3 by 3 part as the identity, and the rest as zeros.

To model cameras that don't have their optic center O at the world's origin, and that don't look down the world's z-axis, we need to add a rotation and translation to the projection equation to align world coordinates with the canonical camera coordinates:

$$\mathbf{p} \sim \mathbf{K}\,[\mathbf{R}|\mathbf{t}]\,\mathbf{P}.$$

Here, $\mathbf{R}$ is a 3 by 3 rotation matrix and $\mathbf{t}$ is a 3 by 1 translation vector. Now we can see the reason why $\mathbf{K}$ was separated out of the projection matrix $\mathbf{C}$, which we redefine to be $\mathbf{C} = \mathbf{K}\,[\mathbf{R}|\mathbf{t}]$. $\mathbf{K}$ speaks only of *intrinsic* camera parameters such as the focal length and principle point, and is therefore called the intrinsic camera matrix. $[\mathbf{R}|\mathbf{t}]$ defines the *extrinsic* parameters which situate the camera in world space, and is called the extrinsic camera matrix. When we speak of intrinsic or extrinsic camera calibration, we refer to estimation or knowledge of $\mathbf{K}$ or $[\mathbf{R}|\mathbf{t}]$ respectively.

2.2 Triangulation and Structure from Motion

Now we have the complete form of the camera projection equation under a linear pinhole camera model. With this we can mathematically simulate projection, but what of inversion? What of recovering a 3D point $\mathbf{P}$ from its projection $\mathbf{p}$ and from camera knowledge $\mathbf{C}$? Note that $\mathbf{C}$ is a non-square matrix and thus it is not directly invertible. Investigating its pseudoinverse or just referring to Figure 2.1 leads us to the conclusion that projection is a depth-destructive process. All points on the ray of projection between the optic center and $\mathbf{P}$ also project to the same image point $\mathbf{p}$. All that we can determine about the position of $\mathbf{P}$ is that it lies on somewhere on this ray from O through $\mathbf{p}$. This is true, unless we consider more than one image.

Projections of same world point $\mathbf{P}$ in two separate images are related. Suppose we can solve the image matching problem that is to identify in two or more images where the same world point projects, given just the images. For example, we might point out where in the images a certain table corner is: $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_M$. Humans are good at this because they are good at understanding the composition and context of visual scenes. Automatic approaches generally forgo image understanding and consider color and texture similarity, *i.e.* photo-consistency, often in small local windows of pixels around hypothesized match points; we refer the reader to [74, 7, 70, 94] for a small starting point on the vast literature related to image matching. Now if we can solve the image matching problem, and if we have calibration for images taken from multiple distinct viewpoints $O_1, O_2, \ldots, O_M$, then we can estimate $\mathbf{P}$ as the intersection point of all the known rays of projection. This process is called triangulation, and estimating $\mathbf{P}$ from image matches and camera calibrations is called the triangulation problem.

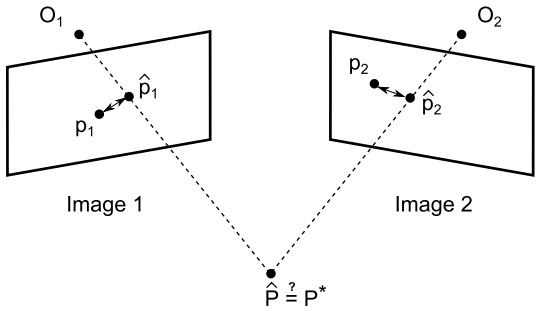


Figure 2.2: The triangulation problem. We wish to find the optimal point $\mathbf{P}^*$ given camera knowledge and image correspondences $\mathbf{p}_1, \mathbf{p}_2, \ldots$. Therefore we find the point $\hat{\mathbf{P}}$ that minimizes the total reprojection error, which is the sum of pixel distances between each $\mathbf{p}_i$ and $\hat{\mathbf{p}}_i$.

Because of imperfect calibration as well as noise in the matching process due to pixel grid discretization, poor image feature localization, mismatches, *etc*., the rays will not perfectly align and intersect in three dimensional space for such simple triangulation. To find the optimal estimate of $\mathbf{P}$ for triangulation under match noise, the best [46] approach is to minimize the total reprojection error and find the 3D point $\mathbf{P}^*$ most consistent with the data:

$$\mathbf{P}^* = \underset{\mathbf{P}}{\operatorname{argmin}} \sum_{i=1}^{M} \|\mathbf{p}_i - \mathbf{C}_i\mathbf{P}\|^2$$

Figure 2.2 illustrates reprojection error and triangulation. Minimizing this error can be done using a variety of numerical optimization techniques such as gradient descent or Newton-like methods. We left implicit in the above equation a detail for computing the norm, $\|\cdot\|$. This norm denotes the Euclidean 2-norm distance with respect to image x, y coordinates between a detected feature match $\mathbf{p}_i$ and the projection of the hypothesized world point $\hat{\mathbf{p}}_i = \mathbf{C}_i\hat{\mathbf{P}}$. Therefore the fact that we use homogeneous coordinates in the equation needs to be taken into account.

Structure from Motion is related to triangulation. It is also the problem of computing 3D structure, but it goes a step further. The problem is to compute *both* the 3D structure of several world points $\mathbf{P}_1, \mathbf{P}_2, \ldots, \mathbf{P}_N$ *and* all the camera matrices $\mathbf{C}_1, \mathbf{C}_2, \ldots, \mathbf{C}_M$ from just the image matches $\{\mathbf{p}_{ij} | i = 1 \ldots M, j = 1 \ldots N\}$. SFM plus image matching entails a complete solution for obtaining sparse 3D scene information from nothing but 2D images. It can be optimally solved by minimizing total reprojection error, just as with triangulation [46]. This algorithm is called bundle adjustment:

$$\{\mathbf{P}^*\}, \{\mathbf{C}^*\} = \underset{\{\mathbf{P}\},\{\mathbf{C}\}}{\operatorname{argmin}} \sum_{i=1}^{M}\sum_{j=1}^{N} \|\mathbf{p}_{ij} - \mathbf{C}_i\mathbf{P}_j\|^2$$

2.3 Delaunay Triangulation

Solutions to the Structure from Motion problem compute only a sparse reconstruction consisting of a set of points, and not a dense 3D model. The difference is illustrated in the two left subfigures

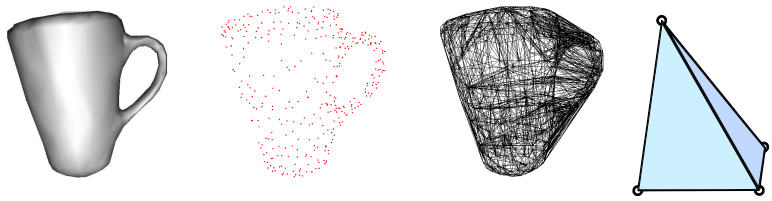


Figure 2.3: 3D Delaunay triangulation. Far Left: A three dimensional object: a cup with a handle. Middle Left: Points sampling the cup's surface (*e.g.* as from SFM). Middle Right: A 3D Delaunay triangulation of those points; the tetrahedra are drawn in wireframe. Far Right: A tetrahedron, the volume element and simplex of the 3D Delaunay triangulation.

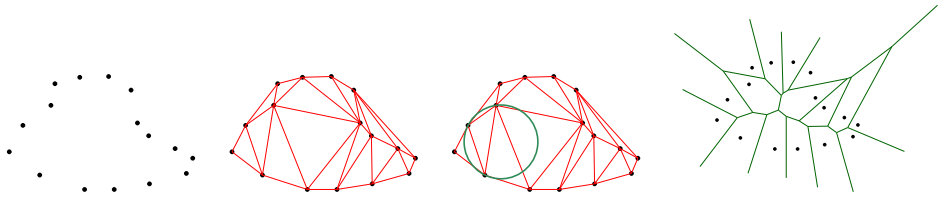


Figure 2.4: 2D Delaunay triangulation. Far Left: Points sampled from a curve. Middle Left: The 2D Delaunay triangulation of the points. Middle Right: The same Delaunay triangulation, with an empty circumcircle shown. Far Right: The Voroni diagram overlaid in green.

of Figure 2.3. This thesis proposes a method that uses SFM point clouds to interpolate a dense 3D surface reconstruction. To interpolate, we take a volumetric approach; we first discretize space using the 3D Delaunay triangulation of the SFM point cloud, and then carve out a 3D model using free-space constraints.

Formally, a triangulation of a set of points $\{P\}$ is a partition of the convex hull of $\{P\}$ into a connected set of simplices, which are triangles in two dimensions and tetrahedra in three. This definition is not to be confused with the triangulation problem of Section 2.2. A Delaunay triangulation is a triangulation that satisfies the empty circumsphere property, *i.e.* the interior of every simplex's circumsphere contains no vertices from $\{P\}$. See Figures 2.4 and 2.3 for an illustration of the 2D and 3D case.

Also depicted in Figure 2.4 is the Voroni diagram which is dual to the Delaunay triangulation. The Voroni diagram is a partition of space into a set of Voroni cells, each associated to its own point in $\{P\}$ called the cell's Voroni vertex. A Voroni cell is a polygon in 2D and a polyhedron in 3D. It is the set of space consisting of all points (not only from $\{P\}$) that are nearer to its associated Voroni vertex than to any other point of $\{P\}$. While not used explicitly in the method of this thesis, we occasionally refer to Voroni diagrams.

Observe from the figures that the Delaunay triangulation of points sampled from a surface, or equivalently from a curve in 2D, is *adaptive*. This is true in more than one sense. First, note that sections of space with fewer points of $\{P\}$ contain fewer but larger simplices. Places denser w.r.t. $\{P\}$ contain more numerous finer simplices. This is important for our discretization choice not

only in terms of computational efficiency, but also in terms of allowing our method to operate with sparser samplings of free space. Second, note that it appears that the Delaunay triangulation contains within it a good approximation of the original surface or curve. This is important for reconstruction. We can see the wireframe of Figure 2.3's cup showing through as a subset of the wireframe of the Delaunay triangulation. We can see a likely curve connecting the points of Figure 2.4 as a subset of the lines of that Delaunay triangulation. This observation is not coincidence, and there is theory in the literature on this point [2, 3, 29, 75]. However, the case is not so simple when there is noise in $\{P\}$ w.r.t. sampling the surface, and there is always noise in SFM point clouds. We refer the reader to the review on Delaunay-based Shape from Points in Section 3.1.4 that highlights this property of the Delaunay triangulation.

Chapter 3

The Literature

This chapter surveys the literature most related to our approach for 3D reconstruction from 2D images. Chapter 2 covered the fundamentals behind sparse point-wise geometric reconstruction from 2D image information. Our method begins with these same fundamentals, but the implementation is real-time; we review online Structure from Motion and SLAM (Simultaneous Localization and Mapping) systems here.

Our method however also connects the sparse 3D structure coherently to form a dense 3D surface estimate by geometrically reasoning about occlusions and free space in the imaged scene. Before covering real-time SFM and SLAM, we discuss dense reconstruction and review the multi-view stereo literature. Our survey has a particular focus on real-time and free-space based methods. In addition, we review works on the Shape from Points problem, where one would like to infer dense 3D geometry only from a sparse 3D point-wise reconstruction without occlusion information or color and texture from images. Much of the Shape from Points literature explicitly or implicitly makes use of the same discretization of space that we do, and therefore some theoretical results from this literature are notably of interest.

A potential application for real-time reconstruction that we have experimented with is improving visualization for remote-controlled or tele-robotics. We conclude by reviewing works on "predictive display" that apply computer vision to this goal.

3.1 3D Shape Reconstruction From Images

3.1.1 Offline Reconstruction

While Structure from Motion addresses the problem of reconstructing sparse 3D scene information from 2D images, two-view and multi-view stereo addresses how to obtain dense 3D from images. In stereo, the pose and calibration of each image's camera is usually assumed known and given as input.

Two-view stereo considers reconstruction from just two calibrated views, one of which is designated as the reference view. The 3D representation used is a discrete per-pixel mapping from

image space to camera-relative scene depth w.r.t. the reference image. This representation is called a "depth map." The two-view problem boils down to finding the optimal depth map via dense image-matching and triangulation. In this way, it is similar to feature-based Structure from Motion but with a match generated for every pixel rather than at sparsely detected feature points. Depth maps are often referred to as a 2.5D representation because they do not encode connectivity between depth estimates. For example, if simply assuming 8-connectivity between neighbouring depth pixels to generate a back-projected surface, foreground objects will incorrectly join with the background of the scene. There are several algorithms and formulations, varying primarily in terms of the texture-based matching cost, spatial regularization, and depth-map optimization scheme. For a good summary, we defer the reader to the two-view Middlebury stereo review and benchmark [94].

This section instead concentrates on multi-view reconstruction from more than 2 images. The multi-view stereo literature is vast. While an exhaustive review would be excessively lengthy and peripheral to this thesis, it is important to situate our work in relation to this body of research. Both our work and stereo have a similar goal: to reconstruct a 3D model from a set of input images. While a multitude of methods with differing properties exist, a common trend is that they compete to find a reconstruction that is as accurate and complete as possible. This emphasis can be seen in the very popular Middlebury multi-view benchmark dataset and survey [95], which collects impressive results from the state of the art. As a result of this focus on quality, most multi-view methods are designed to operate on small sets of images (on the order of ten or a hundred), and they can take up to hours to process such datasets. Real-time reconstruction from video is rare; we cover a selection of these works in more detail in Section 3.1.3. In contrast to the norm, this work attempts to reconstruct more approximate models, but in real time.

Commonly, stereo methods are cast as an optimization problem where a representation of the 3D scene or object to reconstruct is fit to the image data. The approaches vary and are distinct from each other broadly in terms of the optimization framework, 3D parametrization, and cost function or functional to optimize.

The objective function invariably contains some form of a texture-based photo-consistency matching cost. This measures how well the recovered 3D surface's projections match between the input images based on scores derived from color or intensity differences, such as Normalized Cross Correlation (NCC) scores used in [15, 39, 43, 48, 64, 98], or the Sum of Squared Differences (SSD) as used in [44, 45] and others. Photo-consistency is the primary reconstruction cue in stereo. Other common terms in the objective function relate to secondary reconstruction cues such as silhouette constraints (in the case of reconstruction of a single object segmented in image space) [5, 13, 48, 64, 98], visibility (enforcing correctness when optimizing the photo-consistency with respect to occlusions in the images) [27, 40], and spatial regularization or smoothness priors.

Optimization of the chosen objective can be performed using a number of standard techniques, which include iterative derivative-driven numerical optimization (*e.g.* gradient descent for surface

deformation or level-set evolution [27, 99, 40, 48] or conjugate gradient for surface-patch refinement [39]) and discrete combinatorial optimization such as with graph cuts [66, 52, 98, 17]. The choice of optimization procedure influences properties of the method such as convergence, reconstruction quality, initialization, and speed. The applicability of a given optimization procedure also depends on both the form of the objective and the parametrization or representation of the reconstruction. For instance, graph-cut optimization can efficiently find global minima for a restricted class of objective functions in polynomial time [65], and it does not require any specific initialization. On the other hand, local gradient-driven surface evolution supports more general objective functions and regularization schemes, but it can require a close initialization to the optimal reconstruction to prevent convergence to local minima. In the case of object reconstruction via surface evolution, Shape from Silhouettes (SFS) [69] can provide an approximate initial surface by back-projecting the (segmented) object's silhouettes to form generalized cones in 3D space and then computing the volumetric intersection of all such cones [5, 48]. For scene reconstruction where silhouettes cannot directly apply, initialization is a limitation.

Generally, the 3D representation used by a stereo algorithm falls under one of four categories: a volumetric discretization of space, an implicit or explicit surface representation, a collection and fusion of depth maps, or a set of small disjoint surface patch elements called surfels [95].

Volumetric approaches discretize a subset of space in which the object or scene to reconstruct is known to reside. They then label the discretized volume elements as either belonging inside or outside the surface to reconstruct. The reconstruction is implicitly defined by the boundary between inner and outer volume elements. Usually the spatial discretization is a regular grid of cubic volume elements, called voxels, *e.g*. as used in [67, 15, 43, 52, 66, 109, 103]. Simple reconstruction schemes can simply carve away photo-inconsistent voxels one at a time [67]. More complex schemes perform optimization on objective functions including smoothing terms using *e.g*. graph-cuts for global optimization [66, 52, 98, 17]. Voxel-based approaches often lend themselves to a straightforward graph structure, but suffer from a trade-off between resolution of the voxel grid and computational slowdown. Additionally, tight bounds on the location and size of the scene must be known a priori. While more adaptive irregular volumetric discretizations have been used for 3D reconstruction, *e.g*. [98, 68, 82], they are not common. In this thesis, we utilize the 3D Delaunay Triangulation for adaptive discretization and efficient reconstruction. As reviewed in Section 3.1.4, the Delaunay triangulation has properties that make it conducive to shape recovery.

Surface-centric approaches can express the reconstruction either explicitly, *e.g*. as a triangular mesh [48, 27, 5], or implicitly via level sets [40, 99, 64]. They usually employ iterative gradient-decent based surface deformation until an objective is optimized. In the level-set formulation, a scalar function f mapping some region of $R^3 \rightarrow R$ implicitly defines a surface at its zero crossings. Usually, the function is discretely sampled and evolved over a voxel grid, and thus level sets inherit similar limitations to volumetric approaches in terms of resolution versus cost trade-offs. For evolu-

tion of explicit meshes, while optimization drives and displaces the mesh vertices, topology changes can occur *e.g*. when the surface evolves to intersect itself or when the initialization mesh does not have the same genus as the real-world object.[1] Topology change becomes a complex issue to handle correctly [87] and is sometimes ignored or heuristically prevented [48, 5]. In the implicit level-set approach, topology changes during the evolution are handled transparently because the topology is only recovered and output after the level set converges. A practical limitation for these approaches is that they expect a fair initialization surface, which, as previously mentioned, may be hard to obtain for open scenes or for online reconstruction of unknown environments.

Alternative to performing optimization on a single world-space 3D representation, several authors opt to perform consecutive independent depth-map estimations between nearby camera views and then merge the resulting 2.5D depth maps into a single 3D mesh [13, 15, 43, 44, 71, 109, 110]. This approach requires a high degree of overlap in the image sequence so that depth maps may be recovered, and is therefore best applied to video-like sequences. While most methods based on depth-map fusion are not designed for real-time use (and some use a voxel grid for depth-map fusion as only a final step [15, 43, 109, 110]), they trend faster run times, and a few reviewed in Section 3.1.3 have shown real-time or near real-time results on live video.

The surfel approach estimates a set of small oriented surface patches with normals. Surfels can be circular discs as in [45] or rectangular as in [39, 17]. Once an initial set of photo-consistent surfels is recovered, the set is iteratively expanded via local photo-consistent region growing, and the patch position and orientation parameters are refined using photo-consistency optimization, *e.g*. using conjugate gradient as in [39] or other numerical techniques. Visibility and occlusion detection for deletion of erroneous patches is typically handled as a step interleaved with the surface growing. Once the surfel reconstruction is dense enough and complete, the result is still an oriented set of patches instead of a single surface with topology information. While such a reconstruction can be directly rendered, some applications prefer a conversion to a conventional triangle mesh. This is often done in the literature using a Shape-from-Points-and-Normals method, such as Poisson Surface Reconstruction, which is reviewed in Section 3.1.4 [39, 45]. Surfel-based methods are typically computationally intensive, solving multiple photo-consistency optimizations for every patch, the results of which can be thrown out due to deletion or re-estimation of patches occluded by newly grown surfels inducing new visibility information. They however can produce very good results on both objects and open scenes without the requirement of a nearby surface initialization from SFS or otherwise. Furukawa's popular PMVS reconstruction system is exemplary [39].

In the following two subsections on free-space and online reconstruction, we present a selective sample of the most related multi-view methods in more depth.

[1] The genus of an object's surface refers to the number of holes in that object. For example, a donut has a single hole, therefore its surface has genus 1. A coffee mug has a hole at its handle, therefore its surface also has genus 1.

3.1.2 Free-Space Methods

As just reviewed, multi-view stereo for 3D reconstruction is most commonly cast as optimization over dense photo-consistency. The use of dense texture as a primary reconstruction cue necessitates that a *large* amount of pixel data per input image is processed. This choice of dense information cue, while powerful for reconstruction, naturally leads to slow run times. Additionally, methods that rely on a texture-based cue without imposing strong regularization or other constraints on the geometry may fail in areas with low texture information, *e.g*. when modeling a wall painted with a single color. This section concerns itself with methods that exploit the idea of using visibility plus occlusion reasoning, *i.e*. information about the empty free space around and outside of the scene or object to be recovered, as an explicit and central reconstruction cue.

The idea of free space for reconstruction can be summarized as follows. If a 3D feature or surface patch is observed in a camera image, then, assuming opacity, the entire volume comprised of the rays of projection between the 3D feature and the camera's center of projection must be empty. This imposes an explicit constraint on the geometry to be recovered, and a collection of these constraints can be used for reconstruction. This idea is similar to but distinct from Shape from Silhouettes (SFS) [37, 38]. In SFS, silhouettes of a segmented object are back-projected from each input image into generalized cones in 3D space, and the intersection of these cones defines an approximate volume bounding the object to reconstruct. SFS carves everything that projects exterior to a 2D image silhouette, whereas free-space carves everything that projects in front of a detected patch. Unlike SFS, free space is applicable to scenes in addition to objects, and it can model an arbitary level of concavity. See Section 4.1 for illustration and a more in-depth discussion of the free space concept.

In contrast to dense photo-consistency optimization, a free-space cue can make use of sparser visibility information induced by surfels (or even degenerate points), and thus can result in faster reconstruction as realized in this work.

Similar to our work, there exist algorithms that infer a graphics model from a set of sparse or quasi-dense features via the 3D Delaunay triangulation of feature points for volumetric discretization, plus free-space carving. Two such works are that of Faugeras *et al*. [35] and Gargallo [41]. Our proposed method builds on these works. It essentially computes the same result, but in a faster way. More recently, Labatut *et al*. developed methods that combine, via graph cuts, the 3D Delaunay triangulation and free space with photo-consistency and regularization [68, 105]. They achieve impressive reconstructions. Unfortunately, all of these related methods are designed for offline batch use. Our contribution is primarily algorithmic. We achieve real-time performance by exploiting the incremental structure of the Delaunay algorithm to complement it with fully incremental carving.

Independently and recently, Pan *et al*. developed ProFORMA, a system for online reconstruction that is very similar to our own [82]. ProFORMA constructs a 3D Delaunay triangulation, and it carves free-space via a probabilistic voting scheme which obtains a smoother mesh. Our system,

however, has at least two key advantages. First, ProFORMA is only capable of reconstructing isolated objects; we support complex scenes. Second, as discussed in Section 3.1.3, the computational complexity of their system is worse than ours, and our algorithms compare favourably in terms of speed and real-time performance.

Hilton describes a method that incrementally reconstructs a 3D model from sparse features with strong run-time guarantees [49]. His algorithm performs 2D Delaunay triangulations of the sparse feature points in each image, and it back-projects and merges these triangulations into a free-space consistent mesh. This approach is akin to estimating approximate depth maps by interpolating the projections of features via the Delaunay triangulation, and then stitching the resulting back-projected triangular depth meshes together in a free-space aware manner.

Taylor *et al*. similarly consider merging approximate depth maps computed via the 2D Delaunay triangulation of a denser set of feature points [103]. The semi-dense features are obtained in terms of depth w.r.t. each reference view via a stereo system that only reports the depth values for pixels with high-confidence matches. Unlike Hilton, Taylor *et al*. interpret the space that projects in front of each approximate depth map as free space, and they merge depth maps using volumetric free-space carving on a voxel grid. An approximate interpolative depth map however may result in overzealous carving if the base stereo system produces too sparse matches, because in this case the free-space volumes induced by these depth maps are also significantly approximate. (Taylor *et al*. propose a heuristic to attempt to correct for this.) While impressive results were obtained, reconstruction was only demonstrated given inputs from a well-calibrated stereo rig that produced fairly dense matches.

A true stereo system developed by Merrell *et al*. and Pollefeys *et al*. computes photo-consistent depth maps from video and then merges them into a triangular mesh while taking free-space into account [77, 85]. Because their method is geared toward real-time operation, they opt to compute noisy depth maps using a fast method. While depth estimates from overlapping maps are obtained independently, and thus can conflict, free-space constraints, formulated with respect to the depth maps, help to filter out the noise and guide a coherent and accurate reconstruction. We compare this method's computational requirements to our approach in the following section.

3.1.3 Real-time / Near Real-time Reconstruction

Real-time[2] 3D surface reconstruction from video is a challenging problem with relatively few published solutions.

As previously mentioned, Merrell *et al*. and Pollefeys *et al*. published a real-time method based on fusing quick and noisy depth maps together into a free-space consistent surface mesh [77, 85]. Other near real-time approaches based on depth-map fusion via voxel grids have been published,

[2]By "real-time", this thesis considers a practical time-budget definition. *E.g.*, if a system takes video as input to perform computation on each frame, we require that the computation completes at frame rate to consider it real time. We may call a system real time in a soft sense. *E.g.*, a reconstruction system can be called real time if it generally operates at frame rate, but occasionally violates its budget and spends upwards of a second on a single video frame. Frame rate is not the only budget possible; Chapter 4's algorithms operate on inputs that come at an event rate less than typical video frame rates.

but their use is more limited because of the voxel representation, *e.g.* [109, 110]. The results of Merrell *et al.* and Pollefeys *et al.*'s are very impressive, and are competitive in terms of accuracy and completeness with offline stereo methods [95]. They demonstrate large-scale video-based fusion at rates of 20 to 30 fps on sequences as long as $170,000$ frames. Their method, however, assumes extrinsically calibrated images as input, and they obtain this calibration in part from an expensive inertial measurement system. This calibration can be costly to obtain from vision in an online scenario [24, 30, 61, 85], and their depth-map fusion alone exhausts computational resources on fast hardware, both CPU and GPU. In contrast, our approach obtains more approximate reconstructions, but it is lightweight enough to run in parallel with SLAM for camera localization on a modest mobile laptop CPU.

Very recently, Newcombe *et al.* developed a real-time reconstruction system that operates on video [80]. Online Structure from Motion [61] is performed both for camera pose estimation and to reconstruct a sparse point cloud. At every new keyframe, this sparse point cloud is passed off to an efficient Shape from Points method based on implicit-surface fitting using radial basis functions (RBFs) [81]; see Section 3.1.4. Once this approximate surface is computed, it is used as a good initial estimate for photo-consistent refinement, which is implemented parallelized on the GPU. This system currently represents the state of the art in real-time dense stereo from monocular video, and it is capable of producing very impressive reconstructions efficiently. While this system is unlike Merrell *et al.*'s mentioned above, in that it is more complete, and it computes everything from vision in real-time including the camera trajectory, this system's comparison with our method is the same. Our approach is certainly more approximate, but it is also more lightweight. Newcombe *et al.*'s system was shown to run on a very high-end desktop architecture (a quad-core machine circa 2010 with two dedicated graphics cards). These heavy computing requirements limit the system's potential applications, *e.g.* in mobile robotics where computing power is more limited; see Section 3.3.

Pan *et al.*'s ProFORMA system reconstructs real-time 3D meshes via a method very similar to our own [82]. It incorporates an online SFM system as well to generate a sparse point cloud with visibility information, and it carves away tetrahedral volume elements that violate free-space constraints. While their system has been shown to reconstruct isolated objects in real time, their free-space carving algorithm is not incremental: it starts over and processes all $O(NM)$ free-space constraints at every keyframe, where N is the number of feature points and M is the number of keyframe views. As a result, their per-keyframe processing time is $\Omega(NM)$. Unlike our method, ProFORMA's runtime complexity suffers from free-space constraint accumulation and scales poorly in the number of keyframes. This dependence on M shows that ProFORMA must eventually slow below real-time capability as the system continues to run. We show in Section 4.5 that the complexity of all of our algorithms is independent of M, and practically constant-time.

Hilton proves that his free-space consistent local model merging algorithm has a run time complexity that is practically constant per frame [49]. As just stated, we show in Section 4.5 that our

algorithms enjoy a similar run-time guarantee. However, his method is strictly incremental in that it only considers incorporation of new local models from reference frames as they appear. Comparatively, our algorithms handle a superset of events, including the incorporation of information from new frames, as well the deletion of erroneous outliers and the refinement of previously estimated structure and visibility information. In this sense they are incremental and event-based. Chapter 4 describes our algorithms and the different events that they can handle in more detail. But we remark here that our approach is more flexible and can integrate correctly with a larger variety of SLAM and SLAM-like systems (see Section 3.2).

Depending on the application, real-time reconstruction of a single coherent 3D mesh may not be necessary. Rachmielowski *et al*. demonstrate that a fair geometry proxy for image-based rendering and visualization can be obtained by simply back-projecting a view-dependent 2D-Delaunay-approximated depth map from sparse tracked 3D features [93, 91]. This is similar to Hilton's approach, but without any model stitching involved [49]. In this case, because the geometry is view-dependent in a discontinuous way, popping artifacts occur as the visualization's viewpoint changes. While the goal of this thesis is to compute rough approximate models for applications that entail visualization, we opt instead to reconstruct a single coherent 3D mesh that respects the physical free-space constraints of the scene. A coherent representation can be used for more than visualization, *e.g*. to aid robotic path planning, *etc*.

3.1.4 Shape From Points

From a Structure from Motion starting point, as in our approach to 3D reconstruction, we have a set of 3D points sampled from the scene or object to reconstruct. Therefore, although directly employing any approaches from the shape from points literature would ignore and discard useful visibility information, this literature is relevant to our work. We begin with the disclaimer that almost all of the results in this body of literature have been applied to dense low-noise no-outlier point clouds, usually obtained from laser scans. They are designed to work on inputs that differ substantially from SFM point clouds in terms of sampling properties. While other categorizations exist, *e.g*. [76], most methods either fall under the scope of implicit-surface techniques, or combinatorial methods. The primary focus of this section is on combinatorial methods, as they are most related to our work; we use the Delaunay triangulation in our work, and in these methods it is ubiquitous. We are interested in what we can learn about the Delaunay triangulation from this literature.

Implicit Surface Techniques

In the implicit-surface approach, the reconstruction is typically the zero-contour or level set of some scalar function defined on three-dimensional space (mapping $R^3 \rightarrow R$). The function is either fit to or defined relative to the input points such that the zero-set lies on or close to the points. In this sense, the result is not an exact interpolation of the points since it does not have to touch the points,

but rather it is a surface that minimizes distance to the points in some sense while maintaining certain regularization properties.

One idea is to choose the function for the level set to be a signed distance function. For example, an early and famous work is the signed-distance approach by Hoppe *et al*. [51]. At every sample point, they first assign a tangent plane to the surface by finding the K Nearest Neighbours (KNN) in the point set and then fitting a plane to them using least squares. The distance function to any query point in space evaluates as the distance to the closest input sample point's tangent plane via projection onto the plane. The sign of the distance function depends on the side of the plane that the point falls on, and the tangent planes of nearby sample points are given a consistent orientation. (This consistent orientation is achieved via a graph-based algorithm.) A voxel grid is used, and the reconstructed surface is extracted via marching cubes [72] as the zero-set of the signed distance function evaluated in voxel space. This method, while certainly effective for dense regularly sampled point sets as obtained from laser scans, has shortcomings. There is no noise model describing perturbation of the sample points, and the locally estimated planes are heuristically and inexactly computed from the KNN, thus the results are not perfect.

More recently, methods have fit signed distance or indicator functions represented by a set of basis functions, by solving for the basis function weights in a linear system of constraints derived from the point set. Radial basis functions of different types (*e.g*. global, or local and compactly supported) have proven a powerful and popular choice, capable of excellent interpolative reconstructions from points containing some noise and large holes of missing data [81, 16]. Once the function is fit, it is again typically evaluated on a discretized mesh, and then the zero-set isosurface is computed, *e.g*. using voxels and marching cubes, or as in [16] using a marching-cube like marching tetrahedra technique. While these methods were demonstrated to work on a small amount of noise, results with outliers are generally not presented.

A newer technique called Poisson Surface Reconstruction for shape from oriented points, *i.e*. shape from points and normals, exhibits very detailed excellent reconstructions supporting sharp features, smooth areas, some amount of input noise, irregular point sampling, and arbitrary surface topology [57]. This method is popular, and has been implemented as the final stage of several successful stereo algorithms [39, 44, 45], as well as appearing as a standard technique in software like MeshLab and libraries like CGAL [1]. Essentially, the idea is that the indicator function defined as 0 outside the mesh and 1 inside has a gradient field (convolved with a smoothing function to be finite) that is equal to the gradient field represented by and sampled as the oriented points (convolved with same smoothing function). Taking the divergence operator on both sides of this equality results in a differential equation. The indicator function and gradient field are approximated in a function space with sparse compactly supported and approximately Gaussian basis functions. Reconstructing the indicator function becomes an optimization problem of solving a sparse linear least squares system. The resulting isosurface of the indicator is extracted with a modification of marching cubes.

The runtime speed is reasonable, on the order of seconds to minutes on dense data, although not quite real-time.

Combinatorial Methods

The combinatorial approach to shape from points, in contrast to implicit-surface methods, finds a surface reconstruction that consists of exactly the input point samples, connected and interpolated by discrete facets. Essentially, these are *connect-the-dot* methods.

In the 1980s and 1990s, people began to show that for points sampled from a 3D surface, the 3D Delaunay triangulation of the points imposes structure on the point set and can define the shape of a point set [11, 32]. Boissonnat showed that by computing the 3D Delaunay triangulation of the points and carving away tetrahedra from outside the convex hull inward using specific topological rules and a priority ordering, a reasonable but heuristic reconstruction results [11]. The topological rules are meant to enforce the constraint that the carved surface must be polyhedral. The carving order depends on the observation that as point sampling density increases, the relevant Delaunay circumspheres approach tangency to the surface S to be reconstructed. This notion foreshadows a property of every method in this literature: they require sufficient sampling density w.r.t. the curvature of S to produce correct results. However, in this early work, no concrete sampling conditions are given, and no noise model is considered.

The 3D alpha shapes of Edelsbrunner and Mucke consider the ambiguous question of how to define the shape of a point set [32]. Intuitively, alpha shapes can be described by carving. To paraphrase a common description of alpha shapes, imagine that R^3 is comprised of some soft malleable substance such as ice cream, as well as hard bits of chocolate chips that are the points in the sample point set. An eraser sphere or ice cream scoop of size α, where α is a level-of-detail parameter, scoops all of the ice cream wherever it will fit without touching any of the hard chocolate chips. The remaining boundary between carved and uncarved space includes curves and domes: straighten each of these out into lines and discrete planar faces, and you have the resulting alpha shape. Alpha shapes are intimately related to the 3D Delaunay triangulation, as the spherical eraser α is related to the empty circumsphere property. Alpha shapes can be computed efficiently as a subset of the points, edges, facets, and tetrahedra of the Delaunay triangulation. For dense relatively uniformly sampled point sets, results with tuned α show that the alpha shape can remarkably resemble the original surface S. However the topology of an alpha shape is not necessarily a manifold (*i.e.*, a topologically well formed surface), and when point samples are spaced non-uniformly w.r.t. α, the quality of an alpha shape reconstruction degrades with spurious connectivity and incorrect carvings. No noise model is considered in the definition or results.

Extensions of alpha shapes and related algorithms attempt to address the problems of non-uniform sampling, *e.g.* weighted alpha-shapes [31], as well as the problem of topology, *e.g.* alpha solids and the Ball Pivoting algorithm [9, 10]. However, weighted alpha-shapes involve assigning

a weight to each point, and choosing these weights is clumsy manual guesswork. Alpha solids, which essentially carve via the spherical eraser only from outside the convex hull inward, as well as the surface-growing Ball Pivoting algorithm, can guarantee a topological surface. But again, these methods are limited as the samples are assumed noiseless.

In the pivotal paper "Surface Reconstruction by Voroni Filtering," Amenta and Bern proposed an algorithm that is provably correct given certain sampling conditions [2]. The method constructs the 3D Voroni Diagram of the input point set, and defines the two maximally distant Voroni vertices of each point's cell as *poles*. If a Delaunay triangle contains no pole in its smallest bounding sphere, it belongs to the "crust," which is the reconstruction (after normal-filtering and trimming steps). The sampling conditions, called r-sampling, are defined with respect to the medial axis of S, which is the skeletal set of points in R^3 with more than one closest point on S. The distance from S to the medial axis is related to the curvature of S, and r-sampling essentially states that the samples must be denser in areas of higher curvature or detail to capture all the details of the surface. This is intuitively necessary, however there are drawbacks in the specific formulation: r-sampling does not support discontinuities such as sharp corners in the surface, where the medial axis touches the surface so that the curvature and thus sampling density is infinite.

Voroni filtering says something concrete and important about the 3D Delaunay triangulation that the earlier methods hint at. If the points are an r-sample from a surface S, their 3D Delaunay triangulation contains a subset of facets that form a surface which is point-wise and normal-wise convergent to S as $r \to 0$ (*i.e.*, as the sampling density increases w.r.t. curvatures). That is, the 3D Delaunay triangulation is a good adaptive discretization of space given a 3D point set sampled from the surface to reconstruct.

Like Voroni filtering, the more recent Power Crust [3] comes with the same theoretical guarantees and r-sampling conditions, but additionally it is robust to missing data and can fill holes like the RBF and Poisson implicit-function approaches. The central observation is that the Voroni poles form a discrete approximation of the medial axis. The poles are used to reconstruct the medial axis with medial distance or curvature information, and the medial axis is used to compute the surface.

A common negative trend of all of these reviewed works is that they consider no noise model. To my knowledge, the only works in combinatorial shape from points that explicitly consider noise extend r-sampling and Power-Crust medial-axis estimation [29, 75]. They identify certain noise-affected poles for exclusion from the discrete medial axis to robustify the estimate. The algorithms are provably topologically correct and convergent given the extended noisy r-sampling conditions. Note that this is still a "connect-the-dots" approach, and therefore the interpolant will appear bumpy, but with correct connectivity. However, the noise model considered is restrictive and unrealistic; the noise must be bounded in magnitude in proportion to the curvature of S (a curvature-adaptive hard bound). Thus, noise models that include outliers, such as Gaussian noise, are excluded.

Outside of the Shape from Points problem, Labatut *et al*. developed a multi-view stereo algo-

rithm that produces excellent reconstruction results as a subset of the 3D Delaunay triangulation's facets, given a very noisy dense point cloud with a large number of outliers [68]. This empirically hints that outliers and more general noise may not affect the quality of a Delaunay triangulation as a discretization choice, and that proofs and algorithms may exist for noise models more general than noisy r-sampling.

3.2 SLAM / Online Structure from Motion

Simultaneous Localization and Mapping (SLAM) describes the problem in robotics of estimating the robot pose while creating a sparse map of the environment. While SLAM in the more general sense can make use of arbitrary sensors such as sonar, laser, and stereo camera rigs, we consider only the case of monocular visual SLAM, that is SLAM using a single camera as the only sensor (not necessarily in the context of robotics). The SLAM problem then boils down to estimating the 6 degrees of freedom (DOF) camera pose and a representation of world structure consisting of a set of landmarks, in real time. The landmarks generally represent the world sparsely: the SLAM problem does not entail computing a dense 3D surface model of the observed world. While these landmarks can include edge features as in [62] or other types of features, they most commonly represent a simple sparse 3D point cloud. SLAM therefore is synonymous to online Structure from Motion. In this thesis's work on real-time 3D reconstruction, we use SLAM as a starting point and triangulate the point cloud to produce a 3D mesh.

There exists two broad categories of approaches in the literature: recursive filtering methods, and keyframe-optimization bundle-adjustment methods.

3.2.1 Recursive Filtering

In the recursive filtering approach, variants of the Kalman filter are popular [107]. The basic Kalman filter represents and maintains the state of the system in a vector (comprised of the camera pose, landmark positions, and often dynamical entities such as the camera's translational and angular velocity), as well as the uncertainty in the state in the form of a covariance matrix. The filter statistically models the process that predicts the evolution of the state from one video frame to the next using a linear function. This function maps the previous state, current control commands (in the case of an actuated robot), and Gaussian process noise to the next state. The filter also models the measurement process as a linear function mapping the state to observations of the state, which in this case is the projection function on landmarks with a Gaussian image-noise term. A set of update equations based on these two functions predict the current state given the previous state, and update the estimate given the current image frame's landmark measurements. Because the projection or measurement function is nonlinear in monocular SLAM due to perspective division, the Kalman filter is not directly suited to the problem. Therefore, monocular SLAM frequently relies on variants designed to handle non-linearity, such as the widely used Extended Kalman Filter

(EKF) [107, 19, 24, 25, 20, 84], the Unscented Kalman Filter [18], or others.

The first successful implementation of a real-time end-to-end monocular SLAM system was done by Davison *et al*. [24, 25]. It used an EKF as its base machinery, and included provisions for adding and deleting landmarks to and from the state. The state representation included linear and angular camera velocities, and a constant-velocity motion prior was assumed. Feature detection and matching were implemented with the aid of the EKF: the 3σ covariance ellipsoids of the landmarks project into the image using the Kalman-predicted camera pose to restrict the search for image matches or measurements. Experiments with this system demonstrate real-time tracking and mapping in room-sized workspaces, with a map size on the order of 100 landmarks. Unfortunately, a major limitation of this and other Kalman-based approaches is the computational complexity of the filter. Full covariance information is stored and updated in an $O(n^2)$ size matrix, where n denotes the number of landmarks. Such filters quickly drop below real-time performance if too many features are present. This computational restriction is a major drawback for our purpose, where we are most interested in the structure of the environment, rather than in obtaining great accuracy in the camera trajectory; we prefer a denser set of landmarks. We remark that our algorithms have similar bounds on run-time complexity (Section 4.5), but these $O(\cdot)$ bounds hide unspecified multiplicative speed constants. Additionally our computational budget is greater since our methods operate at an event rate which is less than a 30 fps frame rate. We demonstrate that our algorithms are fast enough for real time operation with a large number of landmarks in Section 5.2.

Other Kalman filtering implementations exist that attempt to improve tracking robustness in the case that the motion model is severely violated, *e.g*. under high acceleration jerky camera motion. When the motion model is violated, complete tracking failure can result. Civera *et al*. developed a SLAM system that runs multiple EKF filters in parallel with different motion models, and a Bayesian model selection mechanism determines which models should be active at any given point in time [20]. The switch between filters is continuous in that each model has a probabilistic mixing weight with respect to the others, but in practice the selection mechanism assigns sparse weights. Chekhlov *et al*. use a constant position (plus noise) motion model to handle camera shake [18].

Particle filtering provides another statistical framework for recursive estimation. Instead of maintaining full covariance information in a single matrix, part of the variability of the state is represented by a discrete set of weighted samples or particles, which imply a probability distribution. Typically, the particles each represent hypothesized camera poses plus dynamics and their relative probabilities, and in this way represent the camera's distribution at each iteration of the filter. While landmark positions and uncertainties could also be modeled using their own particle distributions, this has been noted to be computationally too expensive for online performance [30].

It is normal in particle filtering formulations for the update equations to be Kalman-like [30, 89]. *E.g*., in FastSLAM 2.0, a separate EKF filter is run on each particle, taking variance information from the particle distribution into proper account, while updating and resampling the particles from

the distribution correctly based on the variance from every EKF.

Eade and Drummond's use of FastSLAM 2.0 is geared toward improving the computational efficiency of monocular SLAM [30, 78]. FastSLAM 2.0 relies on the observation that if the camera trajectory is known exactly (or hypothesized exactly) then the landmark features are conditionally independent with respect to each other, because they are only related through the geometry of projection and back-projection. Therefore, this portion of the covariance can be dropped per particle, and the footprint of the variance information becomes $O(mn)$, where m is the constant number of particles and n is the number of landmarks. Filter updates are more efficient as a result, and real-time SLAM was demonstrated on sequences containing several hundred landmarks, surpassing pure EKF filters in this regard.

Other particle filtering methods have focused on improving tracking robustness. Pupilli and Calway attempt to improve robustness in comparison to a straight EKF by modeling the trajectory's probability distribution more generally than a Gaussian-assumed mean and covariance [88, 89]. The particle filter supports a multi-modal probability distribution that can handle different competing hypotheses of camera motion under ambiguous circumstances. The computational trade-off here is between trajectory and map estimation. Pupilli and Calway demonstrate real-time performance using 500 camera particles, but only approximately 10 landmarks.

Several works have sought to optimize SLAM via sub-mapping [21, 83, 84] and other techniques such as postponement [63]. Sub-mapping techniques divide the environment into a set of slightly overlapping size-bounded segments, and they construct local maps using their own separate filters for each such sub-map. Because the size of sub-maps are bounded, the per-frame operating cost of the active filter is constant-time. However, with most of these approaches some covariance information is dropped. As a result, the overall map is subject to open-loop drift (and scale-drift [101]) over time. Explicit camera trajectory loop-detection via recognition of previously seen landmarks or image appearance, plus an explicit loop-closure operation is necessary to remedy this problem. Additionally, to represent the map in a single Euclidean coordinate frame, sub-maps must be registered to each other via a set of map-to-map transformations. These transformations are often represented as a graph with links the between sub-maps that overlap, and this graph of transformations is optimized as a final post-processing step or as an online loop-closure step. Such optimization does not have constant-time complexity, but on large-scale outdoor sequences with several-hundred-meter trajectories, it has performed with time-cost on the order of a second, which is acceptable for real-time operation in a background process. In contrast, loop closure is inherent in single-filter Kalman systems, as measurements of one landmark correctly affect the entire state as prescribed by the full covariance matrix. Sub-mapping however provides for impressive large-scale SLAM, mapping hundreds to thousands of points over unprecedented trajectories.

3.2.2 Online Bundle Adjustment

As an alternative to the recursive filter approach, real-time SFM based on bundle-adjustment optimization has seen recent success. Full global bundle adjustment is computationally expensive, and cannot operate at frame-rate on every tracked frame's correspondences in a real-time SLAM scenario. However, as a compromise between accuracy and speed, implementations can make choices on how many iterations to perform, as well as when and what to bundle adjust, be it a set of spatially disparate keyframes as in [61, 62, 79, 101], or a sliding window on a constant number of recent frames [34], or some combination of the two [93, 92].

A recent study by Strasdat *et al*. provides theory, monte-carlo experiments, and discussion comparing recursive filtering methods to keyframe-based bundling in the context of monocular SLAM [100]. The accuracy versus computational cost trade-off between the two approaches was investigated. It was shown that generally, except in low-accuracy tight computation-budget situations, bundling methods have superior accuracy per cost. Additionally, bundling methods were shown to be much more efficient at handling many landmarks (in contrast to many image frames), and the benefit of optimizing more features versus more frames in terms of accuracy is shown to be greater. The implication is that, from a state-of-the-art bundling-based SLAM system, we can expect accurate real-time reconstruction with denser point clouds than from recursive filtering.

Klein and Murray present a system based on keyframe bundling, called Parallel Tracking and Mapping (PTAM) [61, 62]. It performs all bundling and map-management (such as outlier deletion) in a background thread, which prioritizes bundling the recently initialized structure and newer keyframes over global bundle adjustments, which are only performed if there is processing time available. This means that loop-closure is not guaranteed or explicit, but in practice the system produces impressively accurate maps with up to approximately 10,000 landmarks in room-sized spaces.

While accurate SLAM with many landmarks is not unique to PTAM [101], PTAM has the advantages of being both open source and very robust, in the sense that it can reliably relocalize to recover from tracking failure if pointed at familiar landmarks. For these reasons, we choose PTAM as our base SFM system for online surface reconstruction; see Section 4.6. On the other hand, Kalman filtering approaches explicitly model the uncertainty in landmark estimates in terms of covariance. This uncertainty information could be used *e.g*. for soft or conservative probabilistic free-space carving, like in the work by Hilton *et al*. [49]. The disadvantage of PTAM is that it does not provide us with a direct estimate of the noise in the system.

3.3 Tele-robotics

In tele-robotics, a human operator controls a remotely located robot. The operator sends control commands to the robot to perform some task, and receives sensory feedback via a communication

channel.

Communication delays are a fundamental problem. When the operator sends commands to the robot, he expects sensory feedback to reflect his inputs in a natural and transparent way. This is essential for closing the master-slave control loop. However, in practice sensory feedback is delayed by the round-trip latency of the communication channel. Therefore, simple streaming of video and haptic information typically proves insufficient. Visual delays as small as 0.3 seconds negatively impact operator performance and produce a sense of decoupling the human's motions from the robot, resulting in effectively open-loop "move and wait" control [96, 47, 33, 36, 50]. This magnitude of delay is common across the internet, and *e.g.*, ground-to-space tele-operation suffers unavoidably larger delay due to the finite speed of light.

Computer vision can ameliorate the effects of visual delay via predictive display. Predictive display refers to rendering a visualization of the robot site directly in response to the operator's control commands, without waiting for delayed video. This visualization predicts what the camera would see, assuming no delay and given a model of the robot and/or its environment. The model can either be specified offline, in the case of an a priori known environment, or acquired online using computer vision techniques at the robot site. From video, a 3D model can be acquired and updated in real time and sent back to the operator for immediate visualization.

We have applied our 3D reconstruction method to online modeling for predictive display. Additional detail can be found in Section 5.4, as well as in our paper on this topic [73].

3.3.1 Photo-Realistic Predictive Display

Traditionally, predictive display has been accomplished in highly pre-calibrated settings by superimposing hand-modeled wire frame and solid-model overlays of the robot manipulator and scene objects atop delayed video [96, 8, 59]. This approach supposes a known environment and non-moving external camera. More recent work aims to produce photo-realistic predictive display in less calibrated scenarios via modeling of the robot's environment using computer vision [14, 54, 108, 90]. Our work's application to predictive display falls into this category; we do not require a 3D model a priori. We therefore narrow this section's review to this set of work.

Image-based rendering techniques have been applied to photo-realistic predictive display, including pure image-based synthesis [54] and hybrid geometric approaches [22, 108]. To date however, these techniques require an offline reconstruction step or a lengthy online learning phase to get a sufficiently dense image-sampling. In [54], a purely image-based technique synthesizes predictive display for an external fixed camera observing a high-DOF robot arm. From control-command history and delayed images, an image-appearance intensity basis is learned via Principal Component Analysis (PCA), and this basis is modulated to predictively render new robot configurations. The method has the advantage of being completely uncalibrated, and it requires no prior model of the robot. Additionally, learning is performed online, but a long acquisition phase and dense image-

sampling is required to construct an adequate basis for rendering. Yerex *et al*. present an eye-in-hand variant of this approach that reconstructs a coarse stabilizing geometry proxy for rendering using Structure from Motion, and that computes a PCA basis in this model's texture space [108]. The 3D model is however extremely coarse, therefore a dense image-sampling is still required. Cobzas *et al*. present a system that provides a high-fidelity realistic display [22]. They construct a panoramic image-and-depth model of the robot site via a rotating camera and a laser scanner. However, the model is acquired and constructed offline, thus limiting potential applications.

True online vision capture of a single coherent 3D model for predictive display is almost non-existent in the literature. However, some works attempt this or something close. Burkert *et al*. describe an online depth-fusion technique for predictive display that acquires a 3D model using a stereo camera rig [14]. However, their system takes upwards of 30 s to integrate each new depth map, and it exhausts computational resources, both CPU and GPU. Additionally, their implementation lacks online localization of the camera rig. Recently, Kelly *et al*. and Huber *et al*. showed excellent results for capture and display of the environment around a tele-operated mobile vehicle [58, 53]. However, their modeling technique is fairly specific. The reconstruction assumes that the model is comprised primarily of a terrain height-map plus a far-depth billboard approximation for the sky and distant horizon. This assumption is acceptable for the driving task, but is not a general solution. Additionally, the technique uses a laser sensor instead of a passive camera for vision, as well as an inertial navigation system for localization. Nevertheless, Kelly *et al*.'s approach provides very impressive predictive display for driving.

Rachmielowski's predictive display system is closely related to ours [90]; we build upon this work. His system reconstructs a sparse camera set and 3D point structure using online SLAM. It creates a coarse view-dependent geometry by connecting projected 3D points and then back-projecting the 2D mesh into a 3D model. Predictive rendering is achieved by projective texturing from keyframes. This method enjoys the benefits of online performance, support for a moving camera without rigorous precalibration, no need for a 3D model a priori, and applicability to a variety of unknown environments. However, the integration with a real robot was preliminary, and thus experimental validation in [90] was performed primarily in simulation. In contrast, our 3D modeling offers improvements. Instead of using a view-dependent rendering proxy, we infer a single coherent view-consistent proxy that respects physical free-space constraints on the scene. Moreover, we provide experimental evaluation on a real robot in Section 5.4.

In contrast to all these systems, our approach attains real-time tracking, localization, and coherent 3D scene reconstruction with online visualization using just a single camera and the low processing requirements of a mobile laptop CPU. To the best of the author's knowledge, this is a unique combination.

Chapter 4

Method and Algorithms: Incremental Free-Space Carving

As seen in Chapter 3, there exists a wide variety of ideas and techniques for 3D reconstruction, each with their own benefits and weaknesses. A main contribution of this thesis is the development of a set of algorithms that encompass an approach for fast approximate 3D reconstruction. The approach is incremental and event-driven. Unlike most methods, this approach does not expect all of its inputs prior to execution. Instead, it allows for incremental additions and modifications to its inputs, and it reuses previous computations to efficiently update the results. In this way, the method achieves real-time performance. The approach employs the principle of "free space" as the primary reconstruction cue. That is, reasoning about visibility and occlusion yields geometric constraints on the scene that the reconstruction is based on and therefore respects.

This chapter introduces the theory, algorithms, and related ideas. Additionally, it describes a real-time software system that implements them for reconstructing 3D models from live video. The chapter consists of the following primary sections:

- "Free Space" explains the geometric principle that the algorithms operate on.
- "Inputs and Representation" describes the inputs to the algorithms, the related restriction of the reconstruction problem, and the volumetric representation chosen for the 3D model and surrounding space.
- "Algorithms" presents the algorithms in detail, including the overarching event-based approach, as well as a "forgetting" heuristic that enables real-time performance.
- "Isosurface Extraction and Regularization" describes how to compute a conventional 3D mesh from the volumetric representation.
- "Computational Complexity" proves important bounds on the run-time and space requirements of the algorithms.
- "Software System" describes the software architecture of the implementation.

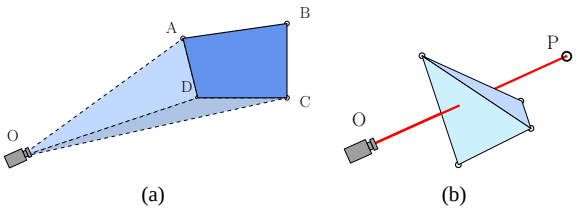


Figure 4.1: Free-space constraints. (a) The general concept. A camera O observes a surface patch, here the quadrilateral $ABCD$. The pyramidal volume $ABCDO$ must be empty; otherwise, the patch would be occluded. (b) Our chosen discrete representation of free-space constraints. The carving method considers only points P instead of generalized patches. Therefore our free-space constraints are infinitesimally thin volumes, the line segments, $\overline{OP}$. These intersect a discretized space of tetrahedra.

4.1 Free Space

The notion of free space is briefly described in Section 3.1.2. We elaborate further here and describe its use. When a camera images a 3D scene surface patch, the volume comprised of the rays of projection from the patch to the camera's optic center must be vacant, and therefore consist of "free space." Otherwise, whatever occupies that space would occlude the patch, assuming surface opacity, and the patch would not be imaged. Figure 4.1(a) illustrates this.

To avoid the computational cost of dense per-pixel depth reconstruction, as well as the difficulties that come with, *e.g.* textureless areas in the scene, the idea is to restrict ourselves to a feature-based approach where only the reliably estimated features in an image are tracked and reconstructed in 3D. Then, construct the union of the 3D free-space volumes induced by these general feature patches, and use this volume to define a dense interpolative mesh.

This approach is distinct from Shape from Silhouettes (see Section 3.1.1) and can be thought of as an interior carving; instead of carving away space that projects outside an observed silhouette (or patch feature), we carve away the known free-space that projects inside and in front of the feature.

While exact methods to compute Shape from Silhouettes exist [37, 38], it turns out that exact methods for interior free-space carving in three dimensions are difficult to construct, because the geometry is hard to reason about. The interpolation induced by the union of free-space volumes from a discretely sampled camera track and patch set is not obviously defined. See Figure 4.2; what are the correct interpolations, and how can we compute them?

Fortunately, if we forgo exactness and utilize discrete approximations, as described in the next section, we still obtain useful interpolative meshes. We have chosen a practical approximation to the free-space problem: we restrict the 3D patch features to infinitesimal point features, and we discretize space based on this point set. In this case, free-space volumes reduce to infinitesimally small line segments; see Figure 4.1(b). The exact union of these constraints is not useful, but we use these line segments to heuristically carve away entire discretized volume elements that intersect. Because of our particular choice of discretization, the boundary between carved and uncarved

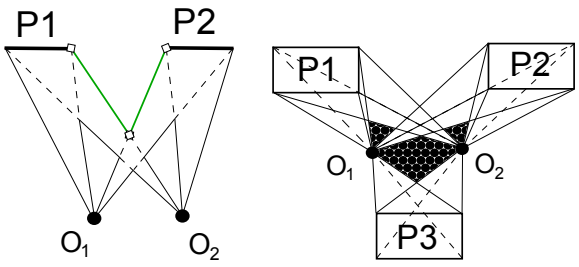


Figure 4.2: The exact discrete free-space problem. Cameras O_n observe patches P_n. (a) The 2D problem. The union of free-space volumes is outlined. Even though the volume can be computed, parts of the boundary (*e.g.* those connecting to the camera centers) project in front of the patches, and are undesirable for rendering from any novel viewpoint that is situated outside of the volume. The interpolative mesh we would ideally compute consists of the patches P_1 and P_2, plus the highlighted part of the boundary in between them (drawn in green with segment endpoints marked). (b) The 3D problem: A frontal view of 3 patches in front of two camera centers. A good interpolation for visualization is not obviously defined. The shaded (blackened) areas contain uncarved space of unknown occupancy, but they are "in between" the patches.

volumes induces an interpolation that is well defined.

4.2 Inputs and Representation

Given as input a set of reconstructed 3D point features $\{P\}$, camera estimates $\{O\}$, and visibility lists $\{\overline{OP}\}$ describing which points P are visible in each view O, our method connects and interpolates the point set to produce a triangulated mesh that approximates the scene. This restriction of the 3D reconstruction problem is not categorized as multi-view stereo (Section 3.1.1) because calibrated images are not the input. The problem also does not align with Shape from Points (Section 3.1.4). Rather, we tackle a Shape-from-Points-and-Visibility problem.

We assume the inputs are readily available from online SLAM [24, 30] or Structure from Motion [61]. Therefore, the estimates of $\{P\}$, $\{O\}$, and the visibility lists are continuously changing.

The method takes a volumetric approach: it discretizes space via the 3D Delaunay triangulation of the input point set $\{P\}$ which yields a set of tetrahedral volume elements that cover the space spanned by $\{P\}$; see Section 2.3 for a description of Delaunay Triangulations. Each tetrahedral 3D volume element has associated with it a labeling: either it is marked as free or occupied space. The 2D boundary surface between the labelings represents our interpolative reconstruction.

The choice of a Delaunay discretization, as opposed to a regular discretization such as voxels, is motivated two-fold. First, it is adaptive. Because we use thin discretely sampled free-space constraints, if every volume element were small like a voxel, we could only carve thin burrow holes into the model. Such a carving would produce useless results with a sparse sampling of free space. The Delaunay triangulation contains many small tetrahedra where denser clusters of surface points are present, and it contains fewer but larger tetrahedra where there are no points in free space.

Therefore, a sparse sampling can carve very effectively.

Second, because the Delaunay discretization is geometrically adaptive, the space-complexity of the triangulation (*i.e.* the number of simplices or volume elements) is also adaptive. The size is worst-case quadratic in the number of points in $\{P\}$ [60, 97], and unlike voxels, the size is not so directly dependent on the desired output 3D model resolution. This relation offers speed to our algorithms: we exploit it to attain worst-case real-time computational complexity. See Section 4.5 for complexity analysis.

Additionally, in Section 3.1.4 we discuss an interesting property of 3D Delaunay triangulations of points sampled from surfaces. Assuming certain sampling conditions are satisfied, a good piecewise-linear approximation of the surface is guaranteed to be contained within the triangular facets of the discretization.

4.3 Algorithms

The fact that our algorithms' inputs come from an online SFM process that continuously modifies and appends to the input data implies two requirements for any real-time reconstruction method that utilizes this information. First, the method itself must produce intermediate outputs. Specifically, the approach should be fully incremental and generate new meshes in real time. Second, the method must support fine-grained event handling. Different types of updates to the input data trigger different algorithms for tailored processing.

In this section, we present a set of algorithms that satisfy these requirements. They handle five main events that are common to SLAM and SLAM-like systems:

1. New-keyframe events, where newly initialized features are also added to $\{P\}$
2. Data-association events, entailing the addition of new visibility information
3. Data-dissociation events, entailing the deletion of erroneous visibility information
4. Deletion events, where outliers are removed from $\{P\}$
5. Refinement events, where subsets of $\{P\}$ and $\{O\}$ are reestimated and moved

Figure 4.3 shows the high-level data flow of the whole method and our implemented system. Because the algorithms all operate on the same data structures, we first discuss their commonalities. What follows is a description of each event handler. Finally, we propose a heuristic that brings the method to real time.

4.3.1 Commonalities

Each event handler's goal is to maintain and update a carving of the Delaunay-discretized space. The algorithms prune entire tetrahedra by marking them as empty if they intersect a free-space constraint, as illustrated in Figure 4.1(b).

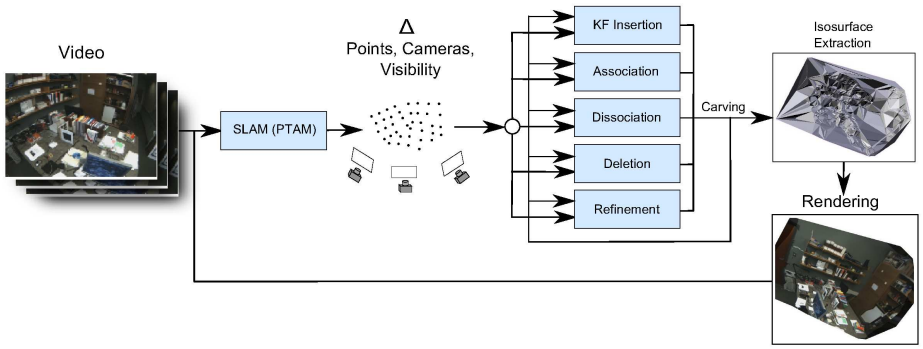


Figure 4.3: Reconstruction system.

When an event updates a subset of the point set $\{P\}$, the Delaunay discretization of a related subset of space changes in turn. Some tetrahedra are deleted, holes are thus created, and new tetrahedra fill the holes and take their place. Therefore, our algorithms must efficiently determine which of the newly created tetrahedra to mark as free space.

For this purpose, we associate with each tetrahedron a set of all the free-space constraints that intersect it. The sets from the old tetrahedra that span the hole(s) determine the minimal constraint set that we must test against. In contrast to a batch method that would redo the triangulation from the beginning and then retest all constraints, our algorithms save time by minimally processing only the new tetrahedra and relevant constraints.

However, this minimal processing is not enough to ensure real-time operation. As many frames are processed, free-space constraints quickly accumulate in the tetrahedra's constraint sets. The size of these sets increase, and so does memory consumption and computation time. Thus, in Section 4.3.6, we propose a heuristic for keeping the size bounded.

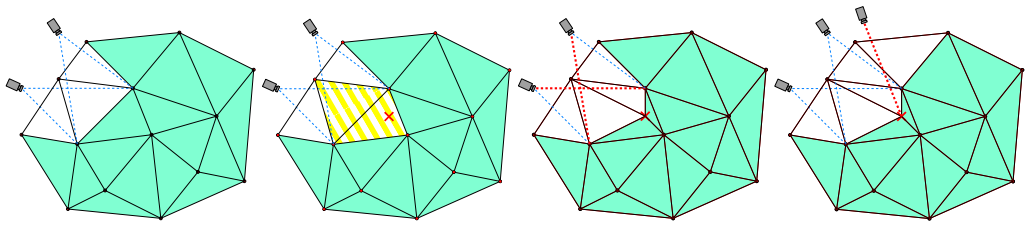
Additionally, we note that some of our algorithms require that all the optic centers $\{O\}$ fall within some tetrahedron in the triangulation. Therefore eight artificial vertices are initially inserted before all events. These vertices represent a loose bounding cube on the features and optic centers.

4.3.2 Keyframe Insertion

Upon addition of a new keyframe, we must handle two changes. First, any 3D point features that are initialized in this keyframe add to $\{P\}$ and thus change the discretization. Second, this keyframe's visibility list carves the resulting discretization.

To preserve the empty circumsphere property, when a new point is inserted, the triangulation algorithm deletes all tetrahedra that contain the point within their circumspheres and then retriangulates the resulting hole. The hole is always a connected set of tetrahedra, and starring[1] off the hole

[1] Starring of a hole with a point means connecting the boundary of the hole to that point. In 2D, each edge of the hole's polygonal boundary connects with the point to form a new triangle. In 3D, each triangular face of the hole's polyhedral boundary connects with the point to form a new tetrahedron.



(a) Initial (b) Conflicts (c) Retriangulation (d) Final Carving

Figure 4.4: A 2D illustration of Algorithm 1, Keyframe Insertion. (a) The initial triangulation. The blue dashed lines are free-space constraints currently in the triangulation. Shaded aquamarine cells have not yet been carved. (b) An incoming point (red cross). The highlighted striped cells are in conflict with the point because it falls inside their circumcircles. (c) The highlighted cells were deleted, and the hole stared off. The red free-space constraints (bolded) belonged to the deleted tetrahedra; they are now used to carve away two of the four new tetrahedra. (d) Finally, the new free-space constraint(s) from the current view are applied.

results in a valid retriangulation [102]; see Figure 4.4.

After retriangulating holes, our algorithm determines which of the new tetrahedra to mark as free space. The minimal constraint set to test against is the union of the constraint sets from the old tetrahedra that spanned the holes.

Finally, the free-space constraints induced by the current view are applied. The entire process is summarized in Figure 4.4 and in Algorithm 1.

Algorithm 1 New-Keyframe Event Handler

$U \leftarrow \emptyset$, the empty constraint set
for all $Q \in \{P\}.NewPoints$ **do**
  $C \leftarrow \{$Cells whose circumspheres conflict with $Q\}$
  for all $T \in C$ **do**
    $U \leftarrow U \cup T.ConstraintSet$
  Insert Q into the triangulation and star off the hole
Apply all constraints $\in U$ as described in § 4.3.3
Apply constraints from the current visibility list

4.3.3 Data Association and Dissociation

Data association and dissociation events only change visibility information. Association events add visibility rays, and dissociation events remove them. For example, if a point is initialized in some keyframe and then later matched also in an earlier keyframe, this raises an association event. If, *e.g.* after a refinement event, a point's reprojection error in some keyframe is large, the corresponding visibility ray is likely erroneous. This may raise a dissociation event: that ray's carving must then be undone.

To handle an association event, we only need to carve. To carve via a given free-space constraint $\overline{OP}$ with optic center O and feature point P, we adopt Gargallo's traversal algorithm [41]. (This is our only event handler that is not novel.)

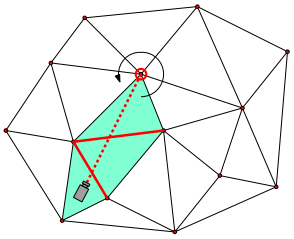


Figure 4.5: The traversal algorithm for processing a free-space constraint (dashed line). First, the cells adjacent to the highlighted vertex are tested for intersection with the constraint. Then, the algorithm hops from cell to adjacent cell via the red facets (bolded) until the camera center is reached. The carved cells are shaded.

The algorithm is depicted in Figure 4.5. The line segment $\overline{OP}$ stabs a connected set of tetrahedra through their shared facets (red and bolded in the figure). This makes traversal possible: begin at vertex P, and perform a series of facet-segment intersection tests to determine which adjacent tetrahedron to traverse to. To find the initial tetrahedron, iterate over all the tetrahedra incident to P, testing only the facet opposite to P for intersection with $\overline{OP}$. If at any time the current tetrahedron contains point O in its interior, the traversal is complete. We add $\overline{OP}$ to all the crossed tetrahedra's constraint sets.

To initially find vertex P, we maintain a vector of vertex pointers as we construct the triangulation. We represent $\overline{OP}$ as a pair of indices, and index the vector to find the vertex (in $O(1)$ time).

For dissociation events, our event handler simply traverses all tetrahedra in the triangulation to erase the selected constraint from their intersection sets. See Algorithm 2. We could apply Gargallo's traversal algorithm instead, but the entailed ray-triangle intersection test is expensive, and Section 4.5 and Appendix A show that the worst case asymptotic run time is the same for both approaches.

Algorithm 2 Dissociation Event Handler

$\overline{OP} \leftarrow$ The free-space constraint to remove
for all Tetrahedra T **do**
  $T.ConstraintSet \leftarrow T.ConstraintSet \setminus \{\overline{OP}\}$

4.3.4 Outlier Deletion

When a point Q is marked for deletion, we remove it from both $\{P\}$ and the triangulation, but we must also undo the carving of all visibility rays induced by it.

Therefore, Algorithm 3 first traverses all tetrahedra to remove any constraints incident to Q from their intersection sets.

Deletion of Q from a Delaunay triangulation rediscretizes space: a hole is formed and retriangulated. The hole is precisely the set of tetrahedra adjacent to Q [28]. This is because point removal is

the inverse operation of point addition; a starred off hole around Q forms the set of tetrahedra adjacent to Q (*cf*. Figure 4.4 and assume that Q was inserted last). A robust procedure for retriangulating the hole is given in [28].

Algorithm 3 therefore collects a set of free-space constraints from incident cells before removing Q, to apply this set afterwards.

Algorithm 3 Outlier-Deletion Event Handler

$U \leftarrow \emptyset$, the empty constraint set
$Q \leftarrow$ Point marked for deletion
// Remove all constraints that reference Q
for all Tetrahedra T **do**
    $T.ConstraintSet \leftarrow T.ConstraintSet \setminus \{\overline{OP} \mid P = Q\}$
// Collect constraints from incident cells
$C \leftarrow \{$Cells incident to $Q\}$
for all $T \in C$ **do**
    $U \leftarrow U \cup T.ConstraintSet$
Delete Q from the triangulation (this retriangulates)
Apply all constraints $\in U$ as described in § 4.3.3

4.3.5 Refinement

The refinement event handler is invoked whenever a set of points and cameras are moved. This happens continuously in SLAM, and it corresponds to partial or full bundle adjustments in online Structure-from-Motion.

Algorithm 4 summarizes. It is essentially a series of point deletions and insertions (akin to Sections 4.3.2 and 4.3.4), with free-space constraints collected and applied only when necessary.

Algorithm 4 Refinement Event Handler

// Collect constraints from incident cells
$U \leftarrow \emptyset$, the empty constraint set
for all $Q \in \{P\}.MovedPoints$ **do**
    $C \leftarrow \{$Cells incident to $Q\}$
    for all $T \in C$ **do**
        $U \leftarrow U \cup T.ConstraintSet$
// Remove the vertices (this retriangulates)
for all $Q \in \{P\}.MovedPoints$ **do**
    Delete Q from the triangulation
// Insert moved vertices while collecting constraints
for all $Q \in \{P\}.MovedPoints$ **do**
    // Q_{moved} refers to Q's new coordinates.
    $C \leftarrow \{$Cells Delaunay-conflicting with $Q_{moved}\}$
    for all $T \in C$ **do**
        $U \leftarrow U \cup T.ConstraintSet$
    Insert Q_{moved} into the triangulation
Remove all constraints from the triangulation that reference moved points, like in Algorithm 3
Apply all constraints $\in U$ as described in § 4.3.3, using moved point and camera locations

For efficiency, we implement this handler with a slight approximation: we partially ignore the

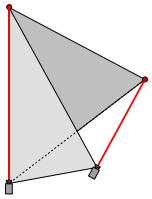


Figure 4.6: The similarity of the left-hand segment w.r.t. the right-hand segment is equal to 1 over the sum of areas spanned by the gray triangles.

movement of keyframes, and thus of some visibility constraints. This works due to two realistic assumptions. First, we assume that keyframes are initialized accurately such that they only move slightly. Second, we rely on future events to re-discretize the remainder of space; constraints then will be reprocessed using the moved optic centers. This implementation choice allows for a better run-time complexity that is independent of the number of keyframes in $\{O\}$; see Section 4.5.

4.3.6 Forgetting Heuristic

The heuristic is simple: retain the K least similar constraints in each tetrahedron's constraint set. As shown in Section 4.5 and Chapter 5, the efficiency greatly improves with insignificant impact on reconstruction quality.

We define the similarity measure by (the inverse of) the sum of the areas spanned between the two constraints; see Figure 4.6. By retaining the K most spatially distinct constraints that intersect each tetrahedron, we hope to cover as much volume as possible so that, when a hole is retriangulated, space is sampled well enough that the new tetrahedra can be carved.

This is not a true similarity measure as it is asymmetric, but it is only a heuristic. The asymmetry arises because the areas depend on the base segment's length $||\overline{OP}||$. This weights longer segments as more different, which usually is desirable.

When a set is full, an incoming constraint is inserted iff it is less similar to all the constraints than their most similar constraint in the set. This knocks out the constraint with the highest (asymmetric) similarity score.

In the case $K = 1$, we simply retain the first inserted constraint and reject all others. For $K = \infty$, the set is never full, so no similarity measures need to be computed.

4.4 Isosurface Extraction and Regularization

Given a current carving, it is straightforward to output the reconstruction as a conventional 3D graphics mesh. Because tetrahedral facets are triangles, this can be computed as the set of facets that border adjacent tetrahedra with differing labels ("carved" or "uncarved"). However, we adopt a more sophisticated scheme for extracting a smoother regularized mesh that can serve as a better geometry proxy for image-based rendering. We remark that surface extraction without regularization also has

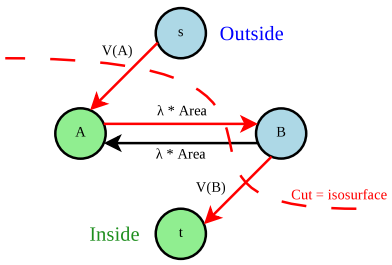


Figure 4.7: A graph with two tetrahedra A and B. Edges between A and B carry the regularization term; edges connecting to s and t represent the data term. (V denotes volume). The cut's cost is the sum of edge weights leading from s's connected component to t's (red edges).

its uses, *e.g.* for non-visual applications like robotic navigation where a more pure and conservative estimate of free space is better.

Let $\mathbf{x}$ denote a 3D point $(x, y, z)^T$, and let $\triangle(\mathbf{x})$ be the tetrahedron containing $\mathbf{x}$. Let $u(\mathbf{x}) \in \{0, 1\}$ be a binary labeling denoting whether $\mathbf{x}$ is carved or uncarved, and let $v(\triangle(\mathbf{x}))$ be the labeling for tetrahedron $\triangle(\mathbf{x})$ provided by our base algorithm. In essence, we minimize the following (continuous) energy functional, but over a discrete carving $u(\triangle)$:

$$\iiint |u(\mathbf{x}) - v(\triangle(\mathbf{x}))| \, d\mathbf{x} + \lambda \iiint H(\|\nabla u\|) \, d\mathbf{x}. \tag{4.1}$$

Here H is the Heaviside step function defined such that $H(0) = 0$, and λ is a scalar regularization parameter.[2] The left-hand integral is the data term, and the right-hand integral evaluates to the surface area of the isosurface of u.

Put another way, we find the optimal carving $u(\triangle)$ that minimizes the volume that disagrees with the original carving $v(\triangle)$ plus a surface-area penalty term. We then extract the isosurface of u.

To optimize, we cast the minimization as a discrete min-cut graph problem. We construct the graph as follows. The vertices correspond one-to-one with each tetrahedron, except for an additional source s and sink t. Node s is associated to the label $0 =$ carved, and t to $1 =$ uncarved. To encode the data term, each node representing a tetrahedron $\triangle$ with original label $v(\triangle) = 0$ connects to s by an edge with weight equal to $\triangle$'s volume. Nodes with the opposite label connect similarly to t. To encode the penalty term, nodes corresponding to adjacent tetrahedra connect to each other by directed edges with weight λA, where A is the area of their shared facet. See Fig. 4.7 for an illustration.

Because the tetrahedra are four-connected, our graph has a topology common to many graph-cut problems in computer vision. To compute the regularized carving u, we use an efficient algorithm optimized for such graphs [12].

[2] Good values of λ were found to be between 0.3 and 0.75. We found these values by manually tuning λ across several datasets, and visually inspecting the outputs.

4.5 Computational Complexity

The run-time bounds derived in this section are not intended to be tight. Instead, in conjunction with the experiments in Chapter 5, they serve to illustrate the real-time quality of our algorithms.

First, we analyze the version that employs the forgetting heuristic ($K < \infty$). Let N be the number of input points, and M the number of views.

Theorem 1. *The worst-case run-time complexity of carving a free-space constraint $\overline{OP}$ (§ 4.3.3) is $O(N^2)$, for $K < \infty$.*

Proof. Because segment $\overline{OP}$ can intersect at most $O(N^2)$ tetrahedra [97], and because the number of tetrahedra incident to P is bounded by $O(N^2)$ [60], the traversal takes $O(CN^2)$ time. Here C refers to the cost of inserting a free-space constraint into a tetrahedron's constraint set. C depends on K, and because K is a constant, $O(C) = O(f(K)) = O(1)$. Thus a single traversal takes $O(N^2)$ time. □

Theorem 2. *The worst-case run-time complexity of Algorithm 1 is $O(N^4)$, for $K < \infty$.*

Proof. First, we cite some relevant properties of the 3D Delaunay triangulation. In the worst case, the number of tetrahedra in the triangulation, as well as the number of tetrahedra that a line can intersect, is of order N^2 [60, 97]. Because we have a structured point set, these bounds are loose. Several papers suggest or prove tighter bounds for point sets sampled from smooth surfaces [4, 35].

The algorithm can be split into two phases: point insertions plus retriangulation with recarving, and carving via the current view's free-space constraints.

For the first phase, there are at most N vertices to insert. For each insertion, at worst all $O(N^2)$ tetrahedra conflict, and thus are deleted and starred off in $O(N^2)$ time. Locating the conflicting cells takes no more than $O(N^2)$ time, since even a naive enumeration of all $O(N^2)$ tetrahedra suffices. Thus, inserting all the vertices takes $O(N^3)$ time.

To recarve the new tetrahedra, the constraints from the old tetrahedra are collected into a single set and then applied. Since $K < \infty$ and since the number of deleted tetrahedra is $O(N^2)$, the number of constraints to reprocess is $O(N^2)$. Therefore, inserting them into a set can be done in $O(N^2 \log(N^2)) = O(N^2 \log(N))$ time using a red-black tree. Theorem 1 shows that applying a single constraint takes $O(N^2)$ time. Since there are $O(N^2)$ constraints, the total time for the first phase of an iteration is $O(N^4)$.

For the second phase of an iteration, because at most N points can be observed in a single view, there are at most N free-space constraints to apply. Therefore, the second phase takes $O(N^3)$ time.

Thus, the complete algorithm takes $O(N^4 + N^3) = O(N^4)$ time per iteration. □

The worst-case complexities of Algorithms 2, 3, and 4 are $O(N^2)$, $O(N^4)$, and $O(N^4)$ respectively, for $K < \infty$. Proofs for these bounds are delegated to Appendix A.

Also, the computational complexity of the graph-cuts isosurface extraction in Section 4.4 is clearly polynomial-time in the number of graph vertices [23], which is $O(N^2)$. Each vertex in the graph has constant bounded degree by construction, and the size of the edge-set is related in this way.

Now for a closed bounded scene, we can assume that the average number of features observed in a given view, N_{avg}, is proportional to N [49]. Therefore, the average case complexity of all our event handlers is $O({N_{\text{avg}}}^4)$. Because N_{avg} depends on system properties such as the camera's field of view and the spatial density of the tracked point set, the per-event time complexity is practically constant [49], which is desirable for an online algorithm.

Without the forgetting heuristic ($K = \infty$), the situation is worse. In this case, there are at most N times M constraints in the triangulation. If, upon a point insertion, all tetrahedra are deleted and retriangulated, then just collecting the constraints into a set takes $O(NM \log(NM))$ time, which is super linear in M.

We do not provide a complete analysis for $K = \infty$. It suffices to show that the complexity is dependent on M, and thus not suitable for online use. Our experimental results support this.

4.6 Software System

By integrating our algorithms with PTAM [61], a real-time Structure-from-Motion system, we have devised a complete system that reconstructs 3D meshes from video. Figure 4.3 shows the system's components and the flow of data. PTAM was chosen as our base tracking system because it is robust and effective; it produces a dense and accurate point cloud, and it reliably relocalizes to recover from tracking failures.

PTAM consists of two main threads: the tracker and the mapper. The mapper is responsible for producing the information that our algorithms operate on. Our routines accept this information in the form of events, and PTAM raises all five types of events.[3] The current integration is far from optimized; event handlers operate in the mapper thread, and they parse argument strings to extract the event type and data. Ideally, the handlers should operate in their own thread to benefit from *e.g.* quad-core processors. The string parsing is a relic from logging and offline testing. In spite of this, we still attain real-time tracking, mapping, modeling, and rendering on a several year old laptop (Intel Core2 Duo CPU T5550 @ 1.83GHz and 3 GB of RAM).

The online visualization is straightforward. This standalone system projects and blends textures from keyframes onto the model for rendering, using GLSL shaders. The texturing scheme is a variant of the view-dependent texture mapping described in [26], where two keyframe images are selected from poses similar to the desired visualization's rendering pose. These keyframes are projected onto the model and blended based on view-ray angle differences. Instead, for simplicity,

[3] PTAM adds new keyframes based on both spatial disparity and time elapsed. *I.e.*, new keyframes are added only when the tracked 3D camera pose is significantly different from all previously recorded keyframe poses *and* when a fixed number of video frames have elapsed since the last keyframe at minimum.

we naively project and blend the four most recent keyframes with equal contributions to the final result. For example reconstructions produced by this modeling system, see the results in Chapter 5. The tele-robotics system described in Section 5.4.1 implements a different visualization mode for experimentation, where only the most recent video frame is projected onto the model for texture.

Our implementation makes use of the CGAL software package for Delaunay triangulations and geometric intersection queries [1]. CGAL is numerically robust and fast, and it provides all the necessary operations, such as Delaunay point insertion and efficient traversals. We remark that naive floating-point implementations of many geometric algorithms and predicates can produce catastrophically wrong results due to the approximate number type. The numerical robustness built into CGAL is an important consideration in practice.

Typically, the full graph-cut regularization and isosurface extraction runs in approximately 0.25 seconds on a single core of the same Intel Core2 Duo CPU T5550 @ 1.83GHz. In practice, to allow other threads and processes to run, we restrict the rate at which we refresh our isosurface model to 1 Hz.

Chapter 5

Experiments

This thesis presented a set of algorithms for real-time 3D reconstruction in Chapter 4. The current chapter documents our experiments with the algorithms and the implemented system. We demonstrate that the system works and that it works in real time, and we also evaluate and characterize the results from the method on inputs with differing properties. Additionally, because the outputs of the method are coarse and approximate, we attempt to show that the method is definitively useful in some specific context. We experiment with the chosen real-time reconstruction application that is improving visual feedback in remote-controlled robotics. In this chapter, we report a small user study involving a tele-robotics system that uses our visual reconstruction method. The chapter has the following major sections:

- "Reconstruction Results" presents several models captured with our real-time system running on live video.
- "Timings and Heuristic Evaluation" validates the algorithms' theoretical run-time complexity results via real data. This section also justifies the use of the algorithms' speed-up heuristic.
- "Synthetic Data Evaluation" explores the reconstruction quality as it relates to sampling and noise. This section seeks more comprehensive insight into the accuracy and behaviour of the proposed method.
- "Predictive Display for Tele-Robotics" details the robotics system and user study, and it shows that this use of our algorithms can positively and uniquely enhance a robot operator's task performance.

5.1 Reconstruction Results

Our algorithms were tested on numerous datasets, and we obtained real-time models of multiple environments. We present three different results in Figure 5.1 that were captured from live video with the PTAM-integrated system of Section 4.6. Sample input images along with shaded and textured output models are shown. We refer to the datasets in the figure, from top to bottom, as

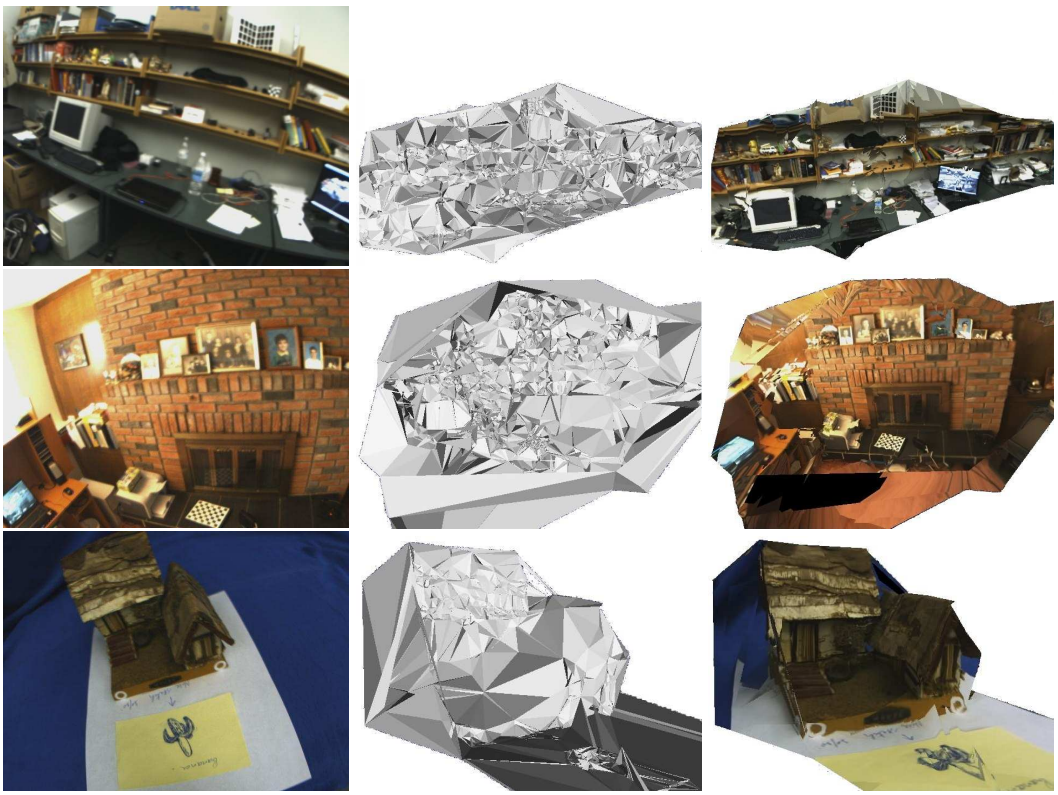


Figure 5.1: Our algorithms' results on three data sets with no smoothing. Top row: "Shelves" dataset. Middle row: "Fireplace." Bottom row: "House." Left: A sample input image from the dataset. Middle: Shaded final models. Right: Offline view-dependent texture rendering.

"Shelves", "Fireplace", and "House" respectively. These results were generated using the forgetting heuristic $K = 1$ (Section 4.3.6) and without the regularization of Section 4.4. In fact, all results in this chapter use $K = 1$, unless stated otherwise.

The results are promising. Even a complex cluttered scene like "Shelves" reconstructs well. "Fireplace" contains specular surfaces to no detriment, *e.g.* the metal fireplace door and glass-framed portraits. This demonstrates the robustness of the feature-based approach. Even though the method carves the convex hull of the features using sparse visibility information, the recovered geometries closely resemble the highly concave scenes, *e.g.* "House".

The textured renderings in Figure 5.1 were generated using a variant of view-dependent texture mapping [26]. This variant was implemented using GLSL shaders, and it is different from the online system's texturing described in Section 4.6. Here, each pixel is rendered by sampling texture from only a single best keyframe, with different keyframes corresponding to each pixel. The keyframe with the camera pose most similar to the desired rendering pose with respect to view-ray angles is selected.

Reconstruction quality is however obviously not perfect. As our method employs no noise model, the meshes are noisy and include some stray uncarved tetrahedra. However, the method

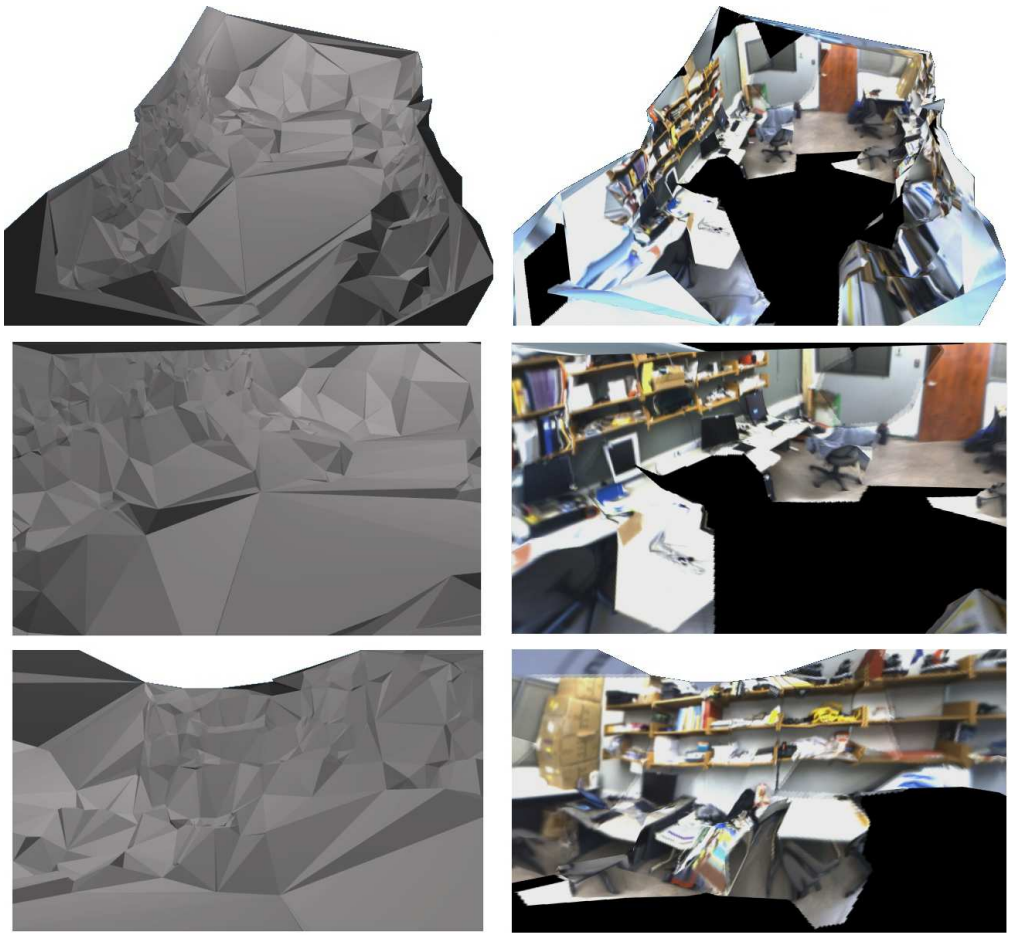


Figure 5.2: A model of the Robotics research laboratory constructed with our system, using smoothing parameter $\lambda = 0.3$. Left: Geometry, shaded. Right: Textured rendering, from the same viewpoints.

is intended for fast, approximate reconstruction. We accomplish this goal: the meshes are adequate geometry proxies for image-based rendering. We explore and attempt to quantify the accuracy of the method more rigorously on synthetic data in Section 5.3.

To combat the effects of noise, we employ the regularization of Section 4.4. Figure 5.2 shows a capture of our lab using the PTAM-integrated system, but this time run with a regularization weight $\lambda = 0.3$. From the shaded model, we can see that the reconstruction produced using regularization is comparatively smooth. It also reprojects well: convincing renderings of the lab are shown using this approximate model and just a single static texture derived from the input video using the technique in [55]. View-dependent image-based rendering was not required for a coherent visualization in this case. This reconstruction was in fact captured using the tele-robotic system that we experiment more with in Section 5.4: the camera was not hand-held but mounted on a robot.

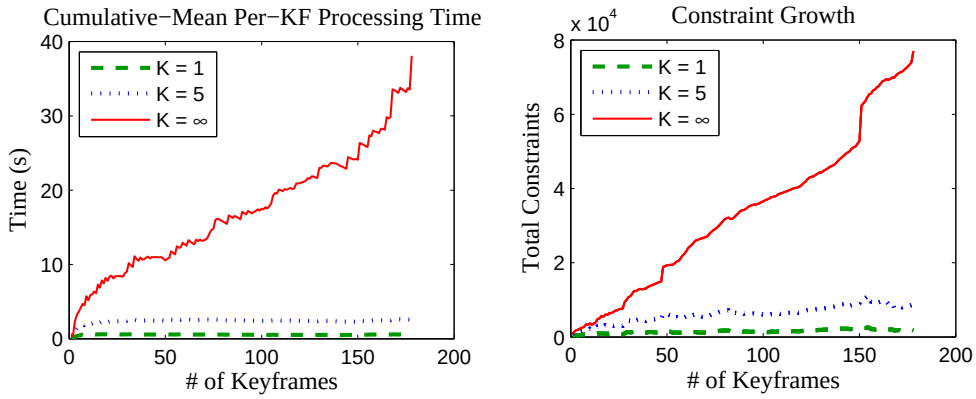


Figure 5.3: Efficiency for $K = 1, 5,$ and ∞ on a representative dataset. Left: Mean per-keyframe processing time as a function of the number of views processed. Right: Number of free-space constraints in the triangulation (a function of the same). Our heuristic effectively bounds computation time and memory usage.

5.2 Timings and Heuristic Evaluation

In Section 4.5, we showed that the computational complexity of the algorithms' are effectively constant per real-time event. Here we provide experimental evidence to reinforce that result and show that our implementation is guaranteed real-time as well.

Figure 5.3 compares timings, and the number of free-space constraints retained, for $K = 1$, 5, and ∞ on a typical dataset from our system. For $K = \infty$, as the number of views increases, the per-keyframe processing time and memory consumption grow. For finite K however (*i.e.* the forgetting heuristic), they are tightly bounded and quickly level off, which supports our complexity results. We conclude that with finite K, our algorithms do operate in real time on image sequences of arbitrary length.

All timings were collected by running our algorithms offline on an event log. The per-keyframe times count the time spent handling all events raised between adjacent keyframes. We average over 30 independent runs before computing the means, except $K = \infty$ was averaged over only 10 due to lengthy run-time. The challenging dataset contained 2887 points in 178 keyframes. This is large compared to typical output of SLAM-type systems [24, 30], and on par with PTAM [61]. The fact that we have run the PTAM-integrated system many times without slowdown shows that the constant per-keyframe run time reported is a small enough constant for online operation.

Finally, Figure 5.4 shows that the outputs for $K = 1$ and $K = \infty$ are almost identical (without regularization). Thus the difference in carving-quality when using the forgetting heuristic is almost negligible. Note that this result compares old datasets produced by offline Structure-from-Motion: we artificially simulated a set of new-keyframe events to feed to our method. Online results, however, support the conclusion: carving is effective with $K = 1$ for all event-types, see Figures 5.1 and 5.2.

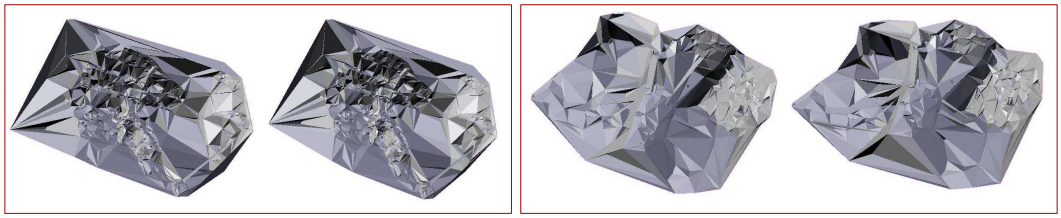


Figure 5.4: Results for $K = \infty$ (left) and $K = 1$ (right) on different data sets. The meshes are similar for extremely different K.

5.3 Synthetic Data Evaluation

To exploratively investigate how Chapter 4's algorithms behave on input data of varying quality, we have created a framework for generating synthetic test data with ground truth. Synthetic data allows us to control the quality in terms of sampling and noise, while ground truth allows us to evaluate performance more concretely and quantitatively than possible with just inspection of the reconstructions from our PTAM-integrated system.

We generate the test data as follows. Given an input ground-truth surface in the form of a triangular mesh, and given a set of user-specified camera viewpoints, we sample a point cloud and compute its visibility information. We perform point sampling by first selecting a random set of triangles from the ground truth mesh and then generating random barycentric coordinates in each selected triangle. To compute visibility, we render the ground truth mesh from each camera's viewpoint and use the OpenGL depth buffer for occlusion queries. The point cloud's sampling density is tunable from sparse to dense. We control this density by specifying parameters for the probability of a triangle being selected, as well as the upper and lower bounds on the random number of points drawn from any selected triangle.

To experiment with varying data quality, after a point cloud is sampled, we perturb it. Each point's x, y, and z coordinate is noised with a Gaussian if the point is not selected to be an outlier. We experiment by varying the probability of outlier selection as well as the standard deviation of the Gaussian noise. Our outlier model is also Gaussian, but with a mean at the center of the ground truth model's bounding box and a standard deviation equal to the bounding box's scale or radius, defined as the distance from the center to a corner. The outlier distribution was chosen to be fairly arbitrary but with a spread matching the bounds of the mesh to reconstruct. Figure 5.5 depicts point samples drawn with different parameters. We remark that the isotropic noise model chosen does not match the noise we expect from visual SLAM or Structure from Motion; see for example the anisotropic covariance ellipsoids in any monocular SLAM paper's results, like in [24]. We assume that the results we present with this noise model can generalize.

We ran experiments using data derived from three ground truth surface models, which we refer to as "Cup," "House2" (not to be confused with "House" from Section 5.1) and "Dog;" see Figure 5.6.

Figure 5.5: Synthetic-data point cloud generation. Far left: A ground truth mesh. Middle left: A noiseless dense point cloud with 50% of ground truth triangles selected for 1 to 3 point samples each. Middle right: A noiseless sparser point cloud with 10% of ground truth triangles selected for 1 point sample each. Far right: A point cloud noised with a standard deviation equal to 2% of the bounding box size and a 1% outlier ratio.

Figure 5.6: Ground truth meshes. Left: Cup. Middle: House2. Right: Dog.

All of our experiments were run with identical view-sampling conditions: fourteen views with optic centers approximately distributed over two rings around the object, one ring in the horizontal x-y plane, and one in the vertical y-z plane, as in Figure 5.7. This view sampling is very sparse, yet all parts of the object are visible from at least one viewpoint, and we have found this to be adequate. Future work includes investigating how view-sampling conditions in relation to point cloud density impacts the quality of our algorithms' reconstructions. This is important since view sampling affects the observation of free-space directly.

First, the algorithms were run without the smoothing regularization of Section 4.4 on noiseless dense and sparse samples drawn from our three datasets. For this experiment and all the subsequent experiments in this section, dense sampling parameters refer to 1 to 3 points sampled per ground truth triangle from approximately 50% of triangles. Sparse sampling corresponds to 1 point sampled from about 10% of triangles. The reconstructions are shown in Figure 5.8. They demonstrate the correctness of our algorithms as well as the ability, given noiseless inputs, to reconstruct isolated objects. (In Section 5.1 we have only demonstrated reconstruction on open scenes). However, while the reconstructions are close to the ground truth, we observe that while our method does essentially produce volume-correct carvings, it does not guarantee topologically well-behaved surfaces as out-

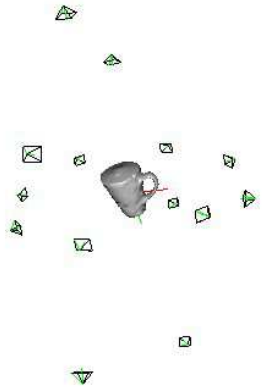


Figure 5.7: View sampling conditions for all the synthetic data experiments. Fourteen camera views are arranged approximately over two rings around our object, one horizontal and one vertical.

put. For the sparser samples, especially as shown on the cup data set which is our sparsest sample and our coarsest ground truth model, the results convey the method's interpolative power and show generally smooth good approximate reconstructions. This observation is consistent with our experience on the comparatively sparse data that we obtain with the PTAM-integrated system: sparse samples reconstruct well.

Next, we evaluate the accuracy of the method as a function of the noise variance, and also as a function of the outlier ratio. To quantify accuracy, we use a measure borrowed from the Middlebury multi-view stereo benchmark [95]. First, project points from the reconstruction onto the ground truth mesh and determine the points' distances to the ground truth. Then compute the histogram of distances and define the accuracy measure to be the threshold distance that exactly 90% of this sample falls within. We sample the reconstructions randomly just as for point cloud generation. Unlike the Middlebury benchmark, which uses the vertices of the reconstruction for the distance sample, we draw points from the interior of triangles, because otherwise the accuracy measure would be meaningless; in our case the free-space carving algorithms take the vertices as input and do not modify them for the output. The method essentially "connects the dots."

For the Cup dataset, we vary the noise's standard deviation between zero and ten percent of the ground truth's bounding box scale (ten percent being very large) and plot the resulting accuracy. Again, we do not use regularization for this experiment. The outlier ratio is fixed at a realistic 1%. To check that the results are not specific to this ground truth geometry, we run the same experiment on another dataset, House2, and obtain effectively identical results. Ten trials per noise level are performed, and the mean observed accuracy threshold is reported in Figure 5.10 for both Cup and House2 under sparse and dense sampling conditions. Examples of reconstructions from highly noisy data (std. dev. 2% of bounding box scale) are shown in Figure 5.9

The accuracy error is roughly a linear function of the noise magnitude. Notice that for dense

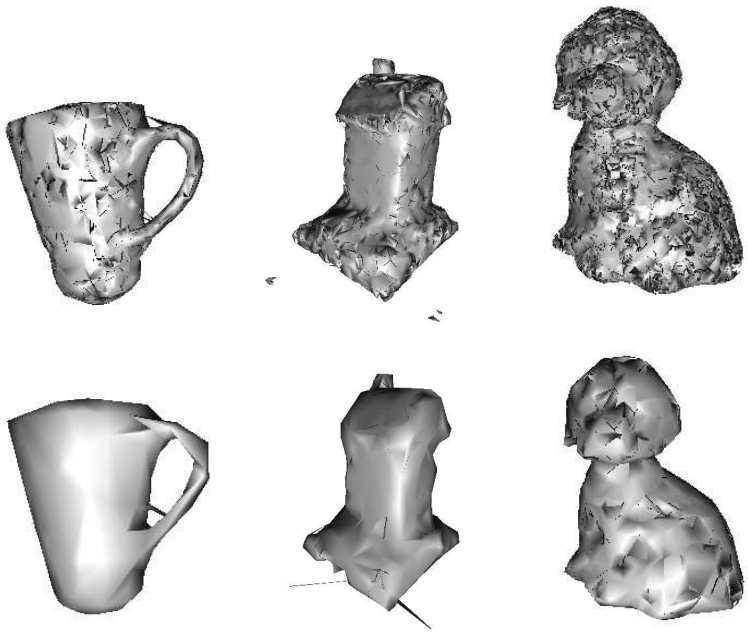


Figure 5.8: Noiseless reconstructions. From left to right: Cup, House2, Dog. From top to bottom: Dense point-cloud sampling; sparse point-cloud sampling.

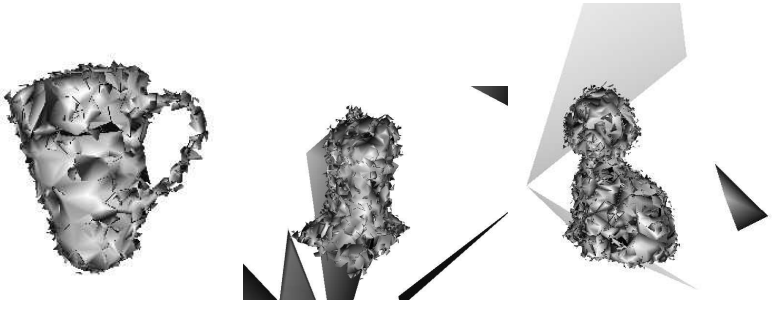


Figure 5.9: Noisy reconstructions. The point-cloud sampling standard deviation is 2% of the bounding box scale, with a 1% outlier rate. Sampling was dense. From left to right: Cup, House2, Dog.

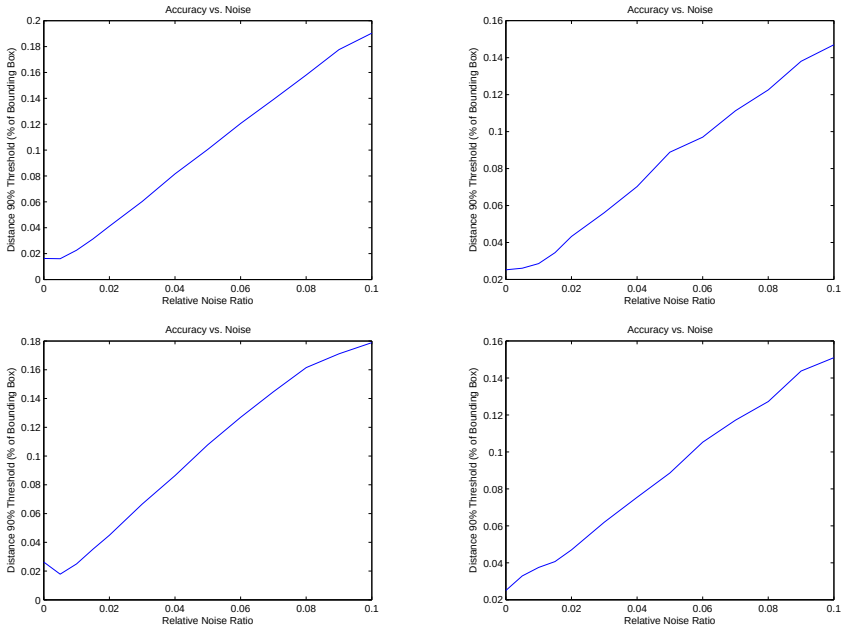


Figure 5.10: Accuracy error as a function of sampling noise. From left to right: Dense point-cloud sampling, Sparse point-cloud sampling. From top to bottom: Cup, House2.

sampling, the slope of these graphs is almost 2, while the slope is about 20% less than that for sparse sampling. Recall that for a Gaussian distribution, generally 90% of samples will fall within 1.645 standard deviations of the mean. This means that the accuracy error of our algorithms is roughly equal to the signal noise in sparser samples, and barely worse than that for dense samples. Intuitively, one should not be able to do better than this (without a biased algorithm that assumes smoothness or regularization, for instance).

We ran a very similar experiment to measure the accuracy as a function of the outlier ratio, this time keeping the noise level fixed to a standard deviation of 0.01% of the bounding box scale. The findings are reported in Figure 5.11. We see that for both the Cup and House2 models under sparse and dense sampling, the accuracy appears practically constant independent of the noise level, up to a breaking point of about 35% to 45% outliers. This breaking point is however not a meaningful number for assessing the accuracy of the method, because it is directly related to the choice of accuracy measure and the *completeness* of the reconstruction, as we discuss next. By inspecting the reconstructions, we observe no special change at this breaking point, but rather a gradual decline in completeness up to this point where the accuracy measure is finally influenced.

The accuracy measure, while it does reflect the general closeness of the result to the ground truth mesh, is forgiving of up to 10% outlying extraneous triangles. Additionally, it does not reflect the notion of completeness in the reconstruction. The Middlebury benchmark uses a separate completeness score, computed by projecting a point sample from the ground truth mesh onto the reconstruction, and determining the proportion of samples that have their nearest point on the topo-

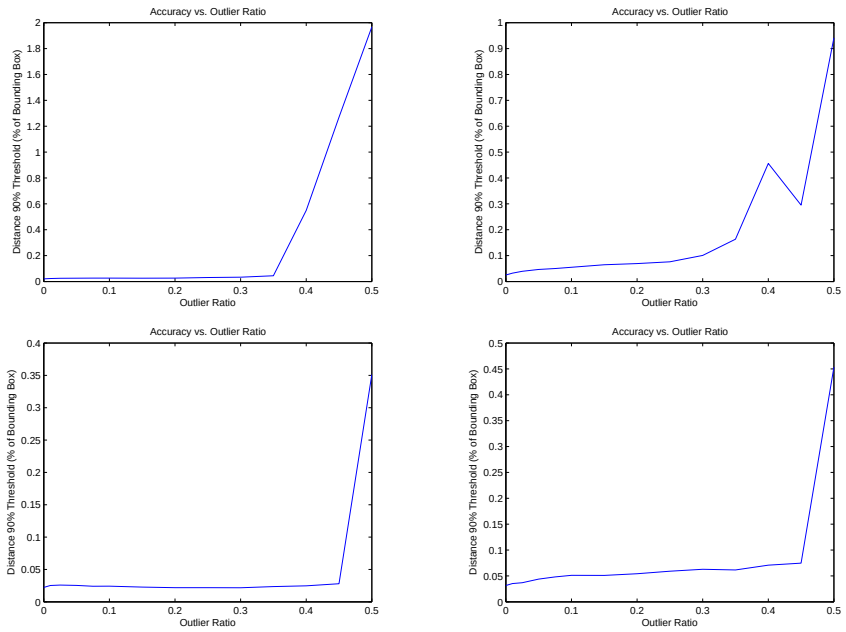


Figure 5.11: Accuracy error as a function of outlier ratio. From left to right: Dense point-cloud sampling, Sparse point-cloud sampling. From top to bottom: Cup, House2.

logical boundary of the reconstruction (*e.g.* at holes). Because our method can produce topologically misbehaved meshes with stray tetrahedra and other non-manifold elements, the boundary of our reconstruction is not well defined, and computation of this completeness score is infeasible. Therefore we assess completeness and accuracy qualitatively by inspecting the results. Figure 5.12 shows several reconstructions of the cup model for various outlier ratios. We observe that while the *accuracy* of the reconstruction where it is near the ground truth appears constant w.r.t. the outlier ratio like the accuracy plots suggest, the *completeness* dwindles as the outlier count increases. This is no surprise, since outliers induce arbitrary free-space constraints that can drill holes through the model. We also observe extraneous uncarved structure consisting of large triangles that connect some outliers around the mesh. While this structure has substantial surface area, because there are few outlier vertices at low outlier ratios, there are few outlier triangles, and therefore the barycentric sampling used in the computation of the accuracy score effectively down-weights this error and hides it beneath the 90% threshold.

If the algorithms are used for applications involving visualization, such uncarved outlying structure can detrimentally project in front of the reconstruction. Mesh incompleteness obviously degrades visual quality as well. We identify outliers as a significant problem for our algorithms, which is consistent with our experience from the PTAM-integrated system. We conclude that the method would benefit from some form of outlier filtering and removal, or from integration with a SLAM system that provides robust inputs in this sense.

For the previous experiments, no regularization was ever employed so that the accuracy of the

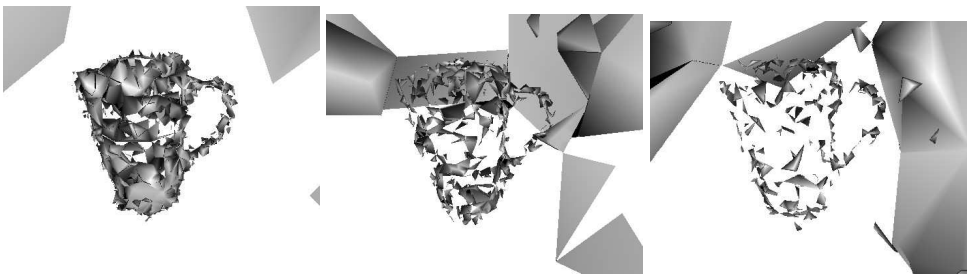


Figure 5.12: Dense Cup reconstruction for several outlier ratios. Left: 5% outliers. Middle: 25% outliers. Right: 45% outliers.

raw carving was assessed. This avoids polluting the results with the minimal surface bias induced by the surface area penalty term. Results with regularization hinge on the raw carving's quality, since the regularization computes the closest-to-raw carving that simultaneously optimizes closeness with the penalty term. If the base carving is significantly wrong, the regularized result will be as well.

We have experimented with regularization by manually finding the optimal weight for the penalty term on each data set with zero noise, and then running the algorithm with that smoothing parameter on samples noised with a standard deviation of 1% of the bounding box scale and a 1% outlier ratio. We present the results in Figure 5.13. Visual inspection indicates that the quality gain from regularization on this data is marginal at best: while some stray tetrahedra are removed, the reconstructions are not much smoother, and sometimes the minimal surface bias can delete structure, *e.g.* the cup's handle. However, in practice we have found that the regularization scheme is often beneficial for the data from the PTAM-integrated system. (See Section 5.1). A more thorough investigation of how the sampling properties impact the difference that this regularization makes is left as future work.

5.4 Application: Predictive Display for Tele-Robotics

In remote-controlled or tele-robotics, a human operator commands a robot that exists at some different and potentially distant location. Because the robot can be far away, perhaps in space or at the opposite end of the earth, transmission delays of both operator commands to the robot and sensory feedback from the robot become an important consideration. Delayed video for visual feedback can be detrimental to operator task-performance, but studies have shown that the situation can improve when augmenting or replacing delayed video with a predictive rendering, or so-called "predictive display" of the robot and/or its environment [96, 50, 90]. While a real-time reconstructed 3D environment model can obviously be used for "display," the "prediction" refers to computing what the pose of the robot and its camera would be with respect to its environment, directly from the operator's control commands under the assumption that no communication delay is present. This predicted pose is used for rendering, and facilitates responsive undelayed visual feedback. We review visual modeling for predictive display in Chapter 3.

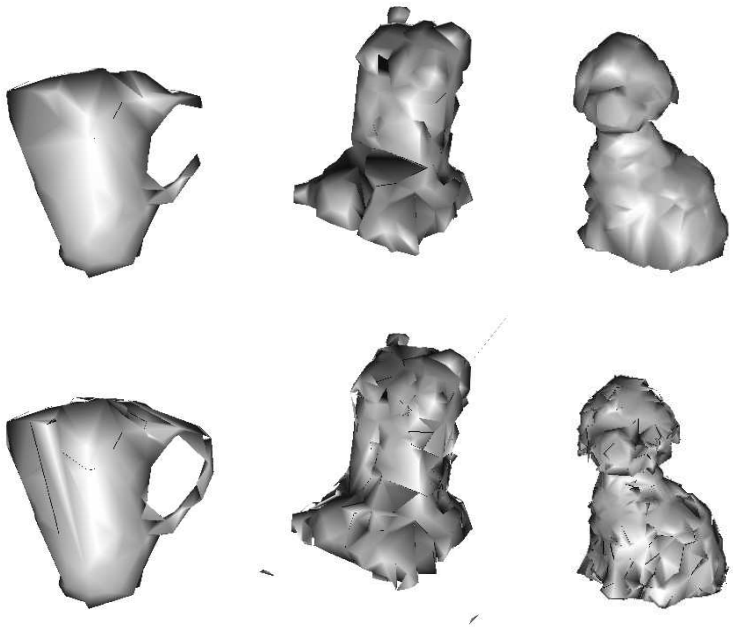


Figure 5.13: Reconstructions with regularization versus without. Top row: with regularization. Bottom row: without regularization.

In this section we present two task-based experiments performed with a prototype tele-robotics system that uses our free-space carving for model acquisition and predictive display. The system is described in more detail in our conference paper on this topic [73]. It consists of a video camera mounted on a robotic arm that is controlled with a joystick-like device, and we inject simulated delays into the system for controlled experimentation. One experiment's task is centred around visual alignment of the arm with a physical target in the environment, and the other focuses on visual inspection. The purpose is to test whether the free-space carving technique is viable and helpful for predictive display in a real robotics set-up.

5.4.1 System

The physical system consists of a Barrett WAM robotic arm mounted on a Segway RMP mobile base. A camera is attached to the forearm of the WAM. We use a PHANTOM Omni haptic device as the operator's joystick, and it controls the WAM via a direct joint angle mapping; see Fig. 5.14. Currently, the Segway is not actuated; the operator has only direct control of the kinematic arm. The communication channel between the master Omni and slave WAM is encapsulated using the PVM software framework [42] for message passing over a wired LAN.

For the vision module, we run incremental free-space carving and PTAM at the robot site. They interface together as in Section 4.6, and they operate on the video from the robot's camera feed. The 3D models that we obtain, the video frames, as well as PTAM's camera pose track defining the

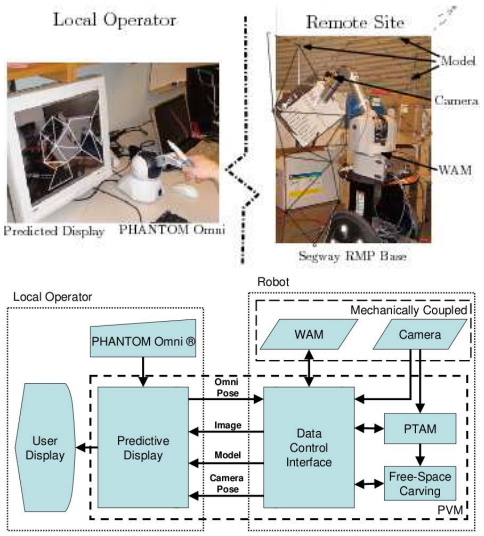


Figure 5.14: Top: the operator tele-operates the robot from the local site; the model is computed at the remote site and transferred to the operator for predictive display. Bottom: system components and data flow.

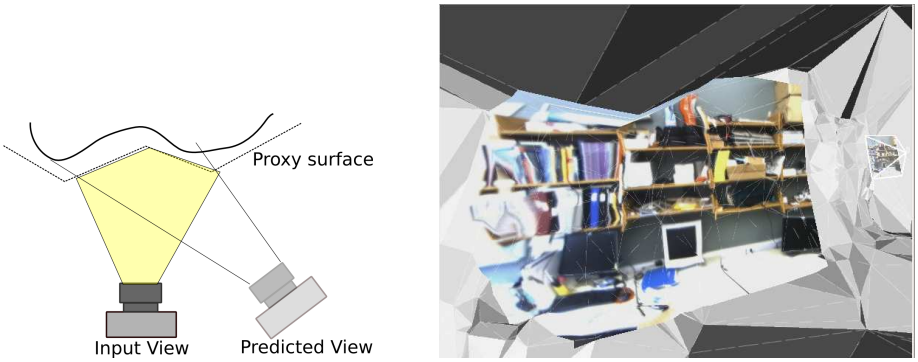


Figure 5.15: Left: Predictive display needs input images, a geometry proxy model, and a predicted camera pose to render from. Right: An example of an input view being back-projected onto a proxy geometry and rendered from a distant predicted pose.

projective texture mapping function between the model and the video frames, are all sent back to the operator and used for predictive display.

To predict what the robot camera will see when the operator moves the Omni, the display module requires: images with known camera pose for texture, a 3D environment model, and a predicted camera pose to render from. See Figure 5.15. We have two of the three requirements sent to us from the vision process. For the third requirement, the correct rendering pose can be determined locally at the operator site with forward-prediction of the robot's motion based on the operator's control commands.

Although more elaborate schemes are possible, we have used only the most recent image and its camera pose to texture the most recent geometric model.

Because the Omni does not have an identical joint configuration to the WAM, the robot control

scheme as well as the predictive display's pose prediction is defined by the specific mapping between the Omni's joints and the WAM's. For a fixed base, pose prediction can be achieved easily through a simple forward-kinematic mapping on the robot's joints. However, because the prototype system only actuates the WAM while the Segway base can still be moved by external forces, the relative base motion must be determined online and used in the prediction. We achieve this via visual registration of the robot-centric coordinate frame and the world or model-centric coordinate frame by making use of PTAM's visual tracking. Details on this visual registration and the joint mapping between the input device and the WAM can be found in the conference paper [73].

5.4.2 Alignment Experiment

Alignment tasks are common in manipulation, *e.g.* in operations like putting a wrench on a bolt. For this experiment, we used visual targets in the scene (the letters A, B, C, and D) that the user had to align with a rendered reference in the video display. When the rendered target matches the real one then the alignment is satisfied. This experiment evaluates and tests alignment performance under our predictive display. We compare three modes of visual feedback: non-delayed video, 0.3 s delayed video, and predictive display using the free-space carved model with a 0.3 s delay for video-derived texture information to render with. (This texture information consists of only the latest available video frame.)

This experiment is largely inspired by and similar to the one presented by Rachmielowski [90], yet ours is conducted on a real-world robot instead of a simulated graphics environment. To be able to compare the timings of several subjects, in this experiment the 3D model was acquired once by the vision system, and the same model was used for all subjects. As just mentioned, texturing used video from the robot camera, and this varied for each trial.

The experiment was conducted first by running a warm-up where the user familiarizes himself with the input device and kinematics of the robot, as well as the three visual modes. After the warm-up session, the timed experiment starts. Each user performs a total of three trials in each of the three modes. For each trial, the display mode and scene configuration are drawn randomly without replacement. The scene configuration is comprised of four targets each placed in one out of six possible calibrated positions. The user has to first align target A, then B, then C and finally D. An alignment is satisfied when the user places the robot in a position which is close enough[1] to the desired position.

The user is only allowed to look at the display and not at the scene where the robot is operating. Fig. 5.16 shows an alignment task where the user is controlling the robot to align the rendered A with the actual A in the scene.

Due to the time needed to arrange the physical scene configuration, the experiments involved

[1] A threshold is set on the distance from the exact alignment to the actual alignment, as well as a 500 ms dwelling time. The threshold was tuned to be reasonably challenging: satisfiable, but near the limit of human precision using the operator's control input device and the 640×480 video / texture feed resolution.

Figure 5.16: Alignment task. Left: image of the setup. Right: the camera view illustrating the overlay (white A) which should be aligned to the real A.

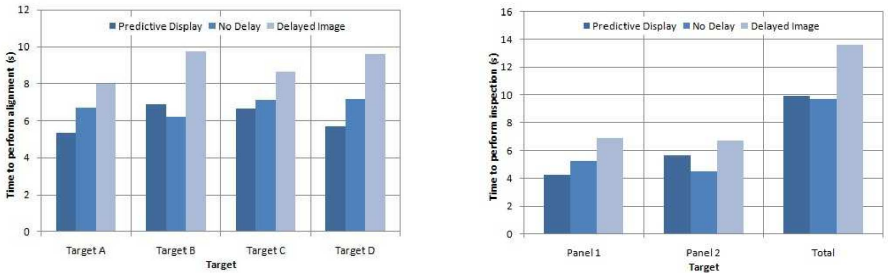


Figure 5.17: Mean normalized times to perform the tele-robotic tasks. Left: Alignment task for each target. Right: Inspection task.

only 36 alignments by each of five lab colleagues for a total of 180 alignments. Despite the small number of test subjects, the results were consistent with Rachmielowski's user study where 1200 alignments were performed on a simulated graphics scene with a virtual robot [90]. This experiment shows that predictive display with our free-space model is beneficial to the operator. The completion time for the alignment task was improved by the use of this predictive display. It helped the users cope with both the transmission delays and the velocity and acceleration limits of the robot (which were more strict than the kinematic limits of the control input device). Fig. 5.17 reports the mean normalized times to perform each of the alignments. Our statistics are normalized the same as in [90] so that each subject contributes to the results equally.

It is important to note that even though the model is rough and the overlaid texture does not match in detail with the scene's geometry, this did not seem to handicap the user. The users were oblivious to the model not being perfect.

5.4.3 Inspection Experiment

Inspection is useful when evaluating systems' functionality in remote environments (*e.g.*, determining if an electronic board is burnt or evaluating a mechanism's operability after some damage). This last experiment's purpose is to test the predictive display in a different task where the user does not perform an action on the environment but instead assesses the situation from information in the scene.

In this experiment, the robot's environment contained two panels with a 3x3 LED matrix on each. In most cases, the panel would have one "damaged" unlit LED whose location is to be identified by

Figure 5.18: An operator performing inspection. Left: Target locations are seen from far away. Right: But identification of the burnt LED requires a close view.

the user (*e.g.* "top left," "center," *etc*.); in other cases there was no damage, and all the LEDs were lit. These panels were placed in two randomly selected positions out of five possible locations for each trial. Two users were asked to do two trials of the inspection task for each visual mode (*i.e.*, six trials per user in total). Again, the mode order was selected at random. The unlit LED in each panel was also random in each trial. In this experiment, the user was allowed to read the panels in any order. Fig. 5.18 shows how the task looks from the operator's point of view.

The experiment started with the robot in a pre-specified home position. The task was to move the camera close enough in front of each panel to identify the unlit LED, if any. From afar, one can locate the lit panels, but it is practically impossible to make an assessment of which LED is unlit.

Fig. 5.17 shows the mean normalized times for the inspection task. The results indicate that predictive display improved the ability of the operator to cope with delay. Although our user study was limited by its small scale and rudimentary robotic system's capabilities (and thus task domain), we can conclude from these experiments that our algorithms and 3D reconstructions, while approximate, have merit and are very likely useful for this real-world application.

Chapter 6

Conclusions

This thesis addressed the difficult open problem of recovering 3D geometric shape from 2D video images in real time. While much work has been done on shape recovery from images, most research has focused on algorithms that compute the best 3D structure possible using image-texture information as the primary reconstruction cue. Because there is a great deal of texture information to process, the literature almost invariably consists of slow methods that are incapable of real-time operation. In contrast to this trend, we have presented an alternative approach centred on the less explored reconstruction cue that is free space, which states that visibility rays impose constraints on the recovered geometry. The approach begins with performing SLAM or real-time Structure from Motion, then discretizes space using the 3D Delaunay triangulation of the obtained point cloud reconstruction, and finally interpolates this sparse reconstruction by carving a surface in a free-space consistent manner. The end result is a lightweight method for real-time 3D reconstruction from video. The method is completely incremental and well suited for online operation; we proved it is real-time efficient, and this was experimentally verified.

We implemented the method and tested it on both real and synthetic data to obtain 3D reconstructions of several scenes and objects. The results show that the method is capable of producing geometries that facilitate convincing renderings and visualizations. We have applied the method to the specific goal of improving visualization in remote-operated or tele-robotics, where transmission delays in visual sensory feedback can degrade the operator's performance, and a virtualized [56, 58] visualization can help to cope with delay. Our experiments show that the 3D models our method computes provide for renderings that serve the operator better in completing his tasks than the alternative of delayed video. We conclude that the method is useful for applications that can benefit from online modeling of general environments, and it is online modeling that the reconstruction literature sorely lacks.

6.1 Limitations and Future Work

While we have developed and implemented a useful set of algorithms that reconstructs 3D geometry from video, several points remain for future work. They include:

- Improving the reconstructed geometry
- Improving SLAM's tracking by making use of the geometry
- Extending the method to dynamic environments
- Investigating the impact of view-sampling
- Applying the method in new contexts

The reconstructed geometries from our PTAM-integrated system are not of the same quality as from dense offline stereo or the real-time approach of Newcombe [80]. In Chapter 5, we have shown that our reconstructions suffer primarily from the presence of outliers in the SFM point cloud, and to a lesser degree noise. The algorithms ignore noise, and this is a violated assumption in practice. We can improve model quality and robustness by integrating our algorithms with a SLAM system that keeps an explicit covariance representation of landmark uncertainty while incorporating this covariance information in the algorithms. Faithful and promising geometric reconstructions have been previously obtained from Delaunay-discretized free-space carving using slower algorithms [82, 68]. The system in [82] handles noise by probabilistic carving via a model of SLAM's landmark variance, and both of these methods filter outliers by aggregating free-space rays as soft votes on whether tetrahedra should be carved rather than as hard constraints [82, 68]. Our algorithms achieve their real-time speed in part due to the forgetting heuristic of Section 4.3.6 which discards redundant visibility rays. Therefore, retaining real-time speed while implementing voting or some voting-like solution remains a challenge. The regularization in Section 4.4 attempts to mitigate the effects of noise via explicit smoothing using a surface area penalty term, but it was not thoroughly effective. For improving surface geometry, an orthogonal approach to incorporating noise models or voting would be to impose more sophisticated regularization.

The PTAM-integrated system was shown to be capable of modeling open scenes, and in Section 5.3, our algorithms were shown to be capable of modeling isolated objects on inputs independent of PTAM. We found that PTAM could not track landmarks on objects when the camera underwent a full 360° trajectory around them. This is because landmarks on one side of an object are occluded by the reverse side from approximately half of the camera viewpoints. This problem results in failed landmark measurements which classify the points as outliers and mark them for removal from the point cloud. We identify that our real-time 3D models can be used within PTAM or other feature-based SLAM systems to predict occlusions and inform whether or not failed landmark measurements should indicate an outlier, especially if future work results in more accurate geometries.

We note that a limitation of the modeling system is that it only works for static environments. PTAM is robust to some moving objects in the scene. It classifies their features as outliers and removes them, but ideally we would like to track the motions over time and construct a time-evolving 3D model. Fundamentally, our method supports moving structure via the Refinement event handler, presented in Section 4.3.5. The static-environment restriction is inherited from SLAM, but SLAM-style systems that track moving scenes are under research [106].

Additionally, sufficient-sampling theory exists for Delaunay-based methods that approach the Shape from Points problem, discussed in Section 3.1.4. The theory speaks on requirements from the input points in terms of sampling density to guarantee correct surface reconstruction. Similarly, formalizing the requirements on point density and noise, camera trajectory density and route, and particularly the relationship between the implied free-space and the Delaunay discretization is important and remains undone. We have found from experience that very sparse keyframes from intuitive camera trajectories can effectively carve models using our method, *e.g*. see the view sampling depicted in Section 5.3, and we owe this probable fact to the adaptiveness of the Delaunay triangulation, but this result is not concrete.

Finally, what's left for future work is to apply this method. We have used it for predictive display, and performed a small user study to show its merit. Extending the manipulation capabilities of the tele-robotic system is important to test its merit in more interesting tasks. Expanding the study to a much greater sample of users also should be done to obtain stronger results. However, predictive display is only one possibility. It remains to be seen what the method can offer for augmented reality, robotic obstacle avoidance and path planning, online view-sampling feedback for offline visual model acquisition (as in [91, 93]), and any other settings that might benefit from real-time 3D modeling and free-space estimation.

Bibliography

[1] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[2] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discrete and Computational Geometry*, 22(4):481–504, 1999.

[3] N. Amenta, S. Choi, and R.K. Kolluri. The power crust. In *Proceedings of the sixth ACM symposium on Solid modeling and applications*, pages 249–266, 2001.

[4] D. Attali, J.D. Boissonnat, and A. Lieutier. Complexity of the delaunay triangulation of points on surfaces the smooth case. In *ACM Symposium on Computational Geometry*, pages 201–210, 2003.

[5] A. Auclair, N. Vincent, and L.D. Cohen. Using point correspondences without projective deformation for multi-view stereo reconstruction. In *ICIP*, pages 193–196, 2008.

[6] R.T. Azuma. A survey of augmented reality. *Presence-Teleoperators and Virtual Environments*, 6(4):355–385, 1997.

[7] S. Baker and I. Matthews. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, 2004.

[8] A.K. Bejczy, W.S. Kim, and S.C. Venema. The Phantom Robot: Predictive Displays for Teleoperation with Time Delay. In *ICRA*, volume 1, pages 546–551, 1990.

[9] F. Bernardini, C.L. Bajaj, J. Chen, and D. Schikore. Automatic reconstruction of 3D CAD models from digital scans. *International Journal of Computational Geometry and Applications*, 9(4/5):327–369, 1999.

[10] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin. The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 5(4):349–359, 1999.

[11] J.D. Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266–286, 1984.

[12] Y. Boykov and V. Kolmogorov. An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision. *PAMI*, 26(9):1124–1137, 2004.

[13] D. Bradley, T. Boubekeur, and W. Heidrich. Accurate multi-view reconstruction using robust binocular stereo and surface meshing. In *CVPR*, pages 1–8, 2008.

[14] T. Burkert, J. Leupold, and G. Passig. A Photorealistic Predictive Display. *Presence: Teleoperators & Virtual Environments*, 13(1):22–43, 2004.

[15] N. Campbell, G. Vogiatzis, C. Hernández, and R. Cipolla. Using multiple hypotheses to improve depth-maps for multiview stereo. In *ECCV*, pages 766–779, 2008.

[16] J.C. Carr, R.K. Beatson, J.B. Cherrie, T.J. Mitchell, W.R. Fright, B.C. McCallum, and T.R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76, 2001.

[17] J.Y. Chang, H. Park, I.K. Park, K.M. Lee, and S.U. Lee. GPU-friendly multi-view stereo reconstruction using surfel representation and graph cuts. *Computer Vision and Image Understanding*, 2010.

[18] D. Chekhlov, M. Pupilli, W. Mayol-Cuevas, and A. Calway. Real-time and robust monocular SLAM using predictive multi-resolution descriptors. *Advances in Visual Computing*, pages 276–285, 2006.

[19] A. Chiuso, P. Favaro, H. Jin, and S. Soatto. 3-d motion and structure from 2-d motion causally integrated over time: Implementation. In *ECCV*, pages 734–750, 2000.

[20] J. Civera, A.J. Davison, and J.M.M. Montiel. Interacting multiple model monocular SLAM. In *ICRA*, pages 3704–3709, 2008.

[21] L.A. Clemente, A.J. Davison, I. Reid, J. Neira, and J.D. Tardós. Mapping large loops with a single hand-held camera. In *Robotics: Science and Systems*, 2007.

[22] D. Cobzas, M. Jägersand, and H. Zhang. A Panoramic Model for Remote Robot Environment Mapping and Predictive Display. *International Journal of Robotics and Automation*, 20(1):25–34, 2005.

[23] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT press and McGraw-Hill, second edition, 2001.

[24] A.J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *ICCV*, pages 1403–1410, 2003.

[25] A.J. Davison, I.D. Reid, N.D. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1052–1067, 2007.

[26] P.E. Debevec, C.J. Taylor, and Malik J. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. In *SIGGRAPH*, pages 11–20, 1996.

[27] A. Delaunoy, E. Prados, G.I. Piracés, J.P. Pons, and P. Sturm. Minimizing the multi-view stereo reprojection error for triangular surface meshes. In *BMVC*, 2008.

[28] O. Devillers and M. Teillaud. Perturbations and vertex removal in a 3D Delaunay triangulation. In *ACM-SIAM Symposium on Discrete algorithms*, pages 313–319, 2003.

[29] T.K. Dey and S. Goswami. Provable surface reconstruction from noisy samples. In *Proceedings of the twentieth annual symposium on Computational Geometry*, pages 330–339, 2004.

[30] E. Eade and T. Drummond. Scalable Monocular SLAM. In *CVPR*, volume 1, pages 469–476, 2006.

[31] H. Edelsbrunner. *Weighted alpha shapes*. University of Illinois at Urbana-Champaign, Dept. of Computer Science, 1992.

[32] H. Edelsbrunner and E.P. Mücke. Three-dimensional alpha shapes. In *Proceedings of the 1992 workshop on Volume visualization*, pages 75–82, 1992.

[33] S.R. Ellis, M.J. Young, B.D. Adelstein, and S.M. Ehrlich. Discrimination of changes in latency during head movement. In *International Conference on Human-Computer Interaction*, pages 1129–1133, 1999.

[34] C. Engels, H. Stewénius, and D. Nistér. Bundle adjustment rules. *Photogrammetric Computer Vision*, 2, 2006.

[35] O.D. Faugeras, E. Le Bras-Mehlman, and J.D. Boissonnat. Representing Stereo Data with the Delaunay Triangulation. *Artificial Intelligence*, 44(1-2):41–87, July 1990.

[36] W.R. Ferrell. Remote manipulation with transmission delay. *IEEE Transactions on Human Factors in Electronics*, 6:24–32, 1965.

[37] Jean-Sébastien Franco and Edmond Boyer. Exact polyhedral visual hulls. In *BMVC*, volume 1, pages 329–338, 2003.

[38] Jean-Sébastien Franco and Edmond Boyer. Efficient Polyhedral Modeling from Silhouettes. *PAMI*, 31(3):414–427, 2009.

[39] Y. Furukawa and J. Ponce. Accurate, dense, and robust multi-view stereopsis. In *PAMI*, 2010.

[40] P. Gargallo, E. Prados, and P. Sturm. Minimizing the reprojection error in surface reconstruction from images. In *ICCV*, pages 1–8, 2007.

[41] Pau Gargallo. Modélisation de Surfaces en Vision 3D. Master's thesis, INRIA Grenoble Rhône-Alpes, 2003.

[42] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing*. MIT Press, 1995.

[43] M. Goesele, B. Curless, and S.M. Seitz. Multi-view stereo revisited. In *CVPR*, volume 2, pages 2402–2409, 2006.

[44] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S.M. Seitz. Multi-view stereo for community photo collections. In *ICCV*, pages 1–8, 2007.

[45] M. Habbecke and L. Kobbelt. A surface-growing approach to multi-view stereo reconstruction. In *CVPR*, pages 1–8, 2007.

[46] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge University Press, 2003.

[47] R. Held, A. Efstathiou, and M. Greene. Adaptation to Displaced and Delayed Visual Feedback From the Hand. *Journal of Experimental Psychology*, 72(6):887–891, 1966.

[48] C. Hernández Esteban and F. Schmitt. Silhouette and stereo fusion for 3D object modeling. *Computer Vision and Image Understanding*, 96(3):367–392, 2004.

[49] A. Hilton. Scene modelling from sparse 3D data. *Image and Vision Computing*, 23(10):900–920, 2005.

[50] P.F. Hokayem and M.W. Spong. Bilateral teleoperation: An historical survey. *Automatica*, 42(12):2035–2057, 2006.

[51] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *COMPUTER GRAPHICS-NEW YORK-ASSOCIATION FOR COMPUTING MACHINERY*, 26:71–71, 1992.

[52] A. Hornung and L. Kobbelt. Hierarchical volumetric multi-view stereo reconstruction of manifold surfaces based on dual graph embedding. In *CVPR*, 2006.

[53] D. Huber, H. Herman, A. Kelly, P. Rander, and J. Ziglar. Real-time photo-realistic visualization of 3D environments for enhanced tele-operation of vehicles. In *Computer Vision Workshops (ICCV Workshops)*, pages 1518–1525, 2009.

[54] M. Jägersand. Image based predictive display for tele-manipulation. In *ICRA*, pages 550–556, 1999.

[55] Z. Jankó and J.P. Pons. Spatio-temporal image-based texture atlases for dynamic 3-D models. In *3DIM*, pages 1646–1653, 2009.

[56] T. Kanade, P. Rander, and P.J. Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE Multimedia*, 4(1):34–47, 1997.

[57] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 61–70, 2006.

[58] A. Kelly, E. Capstick, D. Huber, H. Herman, P. Rander, and R. Warner. Real-time photorealistic virtualized reality interface for remote mobile robot control. *Robotics Research*, pages 211–226, 2011.

[59] W.S. Kim and A.K. Bejczy. Demonstration of A High-Fidelity Predictive/Preview Display Technique for Telerobotic Servicing in Space. *IEEE Transactions on Robotics and Automation*, 9(5):698–708, 1993.

[60] V. Klee. On the complexity of d-dimensional Voronoi diagrams. *Archiv der Mathematik*, 34(1):75–80, 1980.

[61] G. Klein and D. Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *ISMAR*, pages 1–10, 2007.

[62] G. Klein and D. Murray. Improving the agility of keyframe-based SLAM. In *ECCV*, pages 802–815, 2008.

[63] J. Knight, A. Davison, and I. Reid. Towards constant time SLAM using postponement. In *IROS*, volume 1, pages 405–413, 2001.

[64] K. Kolev, M. Klodt, T. Brox, S. Esedoglu, and D. Cremers. Continuous global optimization in multiview 3d reconstruction. In *Energy Minimization Methods in Computer Vision and Pattern Recognition*, pages 441–452, 2007.

[65] V. Kolmogorov and Zabih R. What energy functions can be minimized via graph cuts? *PAMI*, 26(2):147–159, 2004.

[66] V. Kolmogorov and R. Zabih. Multi-camera scene reconstruction via graph cuts. *ECCV*, pages 8–40, 2002.

[67] K.N. Kutulakos and S.M. Seitz. A theory of shape by space carving. *IJCV*, 38(3):199–218, 2000.

[68] P. Labatut, J.P. Pons, and R. Keriven. Efficient Multi-View Reconstruction of Large-Scale Scenes using Interest Points, Delaunay Triangulation and Graph Cuts. In *ICCV*, 2007.

[69] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 150–162, 1994.

[70] J.P. Lewis. Fast normalized cross-correlation. In *Vision Interface*, volume 10, pages 120–123, 1995.

[71] J. Li, E. Li, Y. Chen, L. Xu, and Y. Zhang. Bundled depth-map merging for multi-view stereo. In *CVPR*, 2010.

[72] W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH*, 21(4):163–169, 1987.

[73] D. Lovi, N. Birkbeck, Hernandez-Herdocia A., A. Rachmielowski, M. Jägersand, and D. Cobzas. Predictive Display for Mobile Manipulators in Unknown Environments Using Online Vision-Based Monocular Modeling and Localization. In *IROS*, 2010.

[74] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[75] B. Mederos, N. Amenta, L. Velho, and L.H. de Figueiredo. Surface reconstruction from noisy point clouds. In *Proceedings of the third Eurographics symposium on Geometry processing*, 2005.

[76] R. Mencl and H. Müller. Interpolation and Approximation of Surfaces from Three-dimensional Scattered Data Points. In *Dagstuhl'97, Scientific Visualization*, pages 223–232, 1997.

[77] P. Merrell, A. Akbarzadeh, L. Wang, P. Mordohai, J.M. Frahm, R. Yang, D. Nistér, and M. Pollefeys. Real-Time Visibility-Based Fusion of Depth Maps. In *ICCV*, 2007.

[78] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *IJCAI*, volume 18, pages 1151–1156, 2003.

[79] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd. Real Time Localization and 3D Reconstruction. In *CVPR*, volume 1, pages 363–370, 2006.

[80] R.A. Newcombe and A.J. Davison. Live dense reconstruction with a single moving camera. In *CVPR*, pages 1498–1505, 2010.

[81] Y. Ohtake, A. Belyaev, and H.P. Seidel. A multi-scale approach to 3D scattered data interpolation with compactly supported basis functions. In *Shape Modeling International*, 2003.

[82] Q. Pan, G. Reitmayr, and T. Drummond. ProFORMA: Probabilistic Feature-based On-line Rapid Model Acquisition. In *BMVC*, London, September 2009.

[83] L.M. Paz, P. Jensfelt, J.D. Tardós, and J. Neira. EKF SLAM updates in $O(n)$ with Divide and Conquer SLAM. In *ICRA*, pages 1657–1663, 2007.

[84] P. Piniés and J.D. Tardós. Large-scale slam building conditionally independent local maps: Application to monocular vision. *IEEE Transactions on Robotics*, 24(5):1094–1106, 2008.

[85] M. Pollefeys, D. Nistér, J.M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S.J. Kim, P. Merrell, et al. Detailed real-time urban 3d reconstruction from video. *IJCV*, 78(2):143–167, 2008.

[86] M. Pollefeys, L. Van Gool, M. Vergauwen, F. Verbiest, K. Cornelis, J. Tops, and R. Koch. Visual modeling with a hand-held camera. *IJCV*, 59(3):207–232, 2004.

[87] J.P. Pons and J.D. Boissonnat. Delaunay deformable models: Topology-adaptive meshes based on the restricted Delaunay triangulation. In *CVPR*, pages 1–8, 2007.

[88] M. Pupilli and A. Calway. Real-time camera tracking using a particle filter. In *BMVC*, pages 519–528, 2005.

[89] M. Pupilli and A. Calway. Real-time visual slam with resilience to erratic motion. In *CVPR*, volume 1, pages 1244–1249, 2006.

[90] A. Rachmielowski, N. Birkbeck, and M. Jägersand. Performance Evaluation of Monocular Predictive Display. In *ICRA*, pages 5309–5314, 2010.

[91] A. Rachmielowski, N. Birkbeck, M. Jagersand, and D. Cobzas. Realtime visualization of monocular data for 3D reconstruction. In *CRV*, pages 196–202, 2008.

[92] A. Rachmielowski, D. Cobzas, and M. Jägersand. Robust SSD tracking with incremental 3D structure estimation. In *CRV*, pages 1–8, 2006.

[93] Adam Rachmielowski. Concurrent acquisition, reconstruction, and visualization with monocular video. Master's thesis, University of Alberta, 2009.

[94] D. Scharstein and R. Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *IJCV*, 47(1/2/3):7–42, 2002.

[95] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms. In *CVPR*, volume 1, pages 519–526, 2006.

[96] T.B. Sheridan. Space teleoperation through time delay: Review and prognosis. *IEEE Transactions on Robotics and Automation*, 9(5):592–606, 1993.

[97] J.R. Shewchuk. Stabbing Delaunay Tetrahedralizations. *Discrete and Computational Geometry*, 32(3):339–343, 2004.

[98] S.N. Sinha, P. Mordohai, and M. Pollefeys. Multi-view stereo via graph cuts on the dual of an adaptive tetrahedral mesh. In *ICCV*, pages 1–8, 2007.

[99] J.E. Solem, F. Kahl, and A. Heyden. Visibility Constrained Surface Evolution. In *CVPR*, volume 2, pages 892–899, 2005.

[100] H. Strasdat, J.M.M. Montiel, and A.J. Davison. Real-time monocular SLAM: Why filter? In *ICRA*, pages 2657–2664, 2010.

[101] H. Strasdat, J.M.M. Montiel, and A.J. Davison. Scale drift-aware large scale monocular SLAM. In *RSS*, 2010.

[102] K. Sugihara and H. Inagaki. Why is the 3D Delaunay triangulation difficult to construct? *Information Processing Letters*, 54(5):275–280, 1995.

[103] C.J. Taylor. Surface Reconstruction from Feature Based Stereo. In *ICCV*, pages 184–190, 2003.

[104] Maarten Vergauwen and Luc Van Gool. Web-based 3D Reconstruction Service. *Machine Vision and Applications*, 17(6):411–426, 2006.

[105] H. Vu, R. Keriven, P. Labatut, and J.-P Pons. Towards high-resolution large-scale multi-view stereo. In *CVPR*, 2009.

[106] C.C. Wang, C. Thorpe, S. Thrun, M. Hebert, and H. Durrant-Whyte. Simultaneous localization, mapping and moving object tracking. *The International Journal of Robotics Research*, 26(9):889–916, 2007.

[107] G. Welch and G. Bishop. An introduction to the Kalman filter. *University of North Carolina at Chapel Hill, Chapel Hill, NC*, 7(1), 1995.

[108] K. Yerex, D. Cobzas, and M. Jägersand. Predictive Display Models for Tele-Manipulation from Uncalibrated Camera-capture of Scene Geometry and Appearance. In *ICRA*, volume 2, pages 2812–2817, 2003.

[109] C. Zach. Fast and High Quality Fusion of Depth Maps. In *3DPVT*, 2008.

[110] C. Zach, M. Sormann, and K. Karner. High-Performance Multi-View Reconstruction. In *3DPVT*, pages 113–120, 2006.

Appendix A

Complexity Proofs

This appendix completes the set of proofs of our algorithms' complexity claims made in Section 4.5. As in that section, the bounds are sufficient in that they say enough about the speed of the algorithms, but the bounds are not intended to be tight. Let K be the value of the forgetting heuristic, *i.e.* the maximum number of free-space constraints retained in each tetrahedron. Let N be the number of input points, and M the number of views.

Theorem 3. *The worst-case run-time complexity of Algorithm 2 is $O(N^2)$, for $K < \infty$.*

Proof. Because there are at most $O(N^2)$ tetrahedra in the triangulation [60], looping over all the tetrahedra and removing the constraint $\overline{OP}$ from their constraint sets takes $O(CN^2)$ time. Here C refers to the cost of deleting a free-space constraint from a tetrahedron's constraint set. C depends on K, and because K is a constant, $O(C) = O(f(K)) = O(1)$. Thus Algorithm 2 takes $O(CN^2) = O(N^2)$ time. □

Theorem 4. *The worst-case run-time complexity of Algorithm 3 is $O(N^4)$, for $K < \infty$.*

Proof. Let Q be the outlier point marked for removal. Algorithm 3 can be split into four steps:

1. Iterate over all cells, and remove all the constraints that reference Q.
2. Collect free-space constraints from cells incident to Q into a unioned set U.
3. Delete Q from the triangulation (and retriangulate the hole left by the cells that were incident to Q.)
4. Apply the free-space constraints in U (*i.e.* use them to carve, just as in an association event; see § 4.3.3).

For step 1, to simply loop over all the cells takes $O(N^2)$ time, because there are at most $O(N^2)$ tetrahedra in the triangulation [60]. Because $K < \infty$, the number of constraints in each cell's constraint set is bounded by $K = O(1)$, and therefore there are a total of $O(N^2)$ constraints in the triangulation to iterate over, determine if removal is necessary, and potentially remove. Testing a

constraint for removal is an $O(1)$ operation, since it entails checking if the constraint (represented by a pair of indices) references Q. Removing a constraint from a constraint set is an $O(f(K)) = O(1)$ operation, because $K < \infty$ is constant. Therefore the total cost of step 1 is $O(N^2)$.

For step 2, determining the set of cells incident to Q from adjacency information takes $O(N^2)$ time, since the number of cells in the triangulation, and therefore potentially incident to Q, is bounded by $O(N^2)$ [60]. Using this fact, and because $K < \infty$, the number of constraints to collect from these cells and insert into U is $O(N^2)$. Using red-black trees to implement sets, this bound on the number of constraints to insert into U implies that inserting them all takes $O(N^2 log(N^2)) = O(N^2 log(N))$ time. Therefore step 2 takes $O(N^2 log(N))$ time.

For step 3, deleting vertex Q from a 3D Delaunay triangulation and retriangulating the hole takes $O(fd)$ time, where f is the number of tetrahedra that retriangulate the hole, and d is the degree of Q [28]. In practice, for most point sets, f and d are typically small and roughly constant numbers. However, as loose bounds, we can consider $O(f)$ to be $O(N^2)$, since there are not more than $O(N^2)$ tetrahedra in the triangulation [60], and $O(d)$ to be $O(N)$, since there are at most $N - 1$ vertices that Q can be incident to. Step 3 then takes $O(N^3)$ time.

For step 4, as mentioned in the discussion above regarding step 1, there are at most $O(N^2)$ constraints in U. By Theorem 1, applying a single constraint takes $O(N^2)$ time. Therefore, step 4 takes $O(N^4)$ time to apply all the constraints.

The total time complexity for Algorithm 3 is then $O(N^2+N^2 log(N)+N^3+N^4) = O(N^4)$. □

Theorem 5. *The worst-case run-time complexity of Algorithm 4 is $O(N^4)$, for $K < \infty$.*

Proof. Algorithm 4 can be split into five steps:

1. Collect free-space constraints from cells incident to each point that is to be moved, Q_i, into a unioned set U.

2. Successively delete the to-be-moved vertices Q_i from the triangulation while retriangulating the holes.

3. Add the moved vertices Q'_i to the triangulation (this retriangulates) while collecting the constraints from Delaunay-conflicting cells into U.

4. Iterate over all cells, and remove all the constraints that reference any of the Q_i.

5. Apply the free-space constraints in U (*i.e.* use them to carve, just as in an association event; see § 4.3.3).

Because of the similarity between Algorithms 3 and 4, proving the complexity of these steps is, in places, similar to the proof in Theorem 4

For step 1, determining the set of cells incident to at least one vertex Q_i from adjacency information takes no more than $O(N^3)$ time. This is because the number of cells in the triangulation, and

therefore potentially incident to each Q_i, where i can range from 1 to N, is bounded by $O(N^2)$ [60]. Using this fact, and the fact that $K < \infty$, the number of constraints (potentially with repetition from repeated cells) to collect from these cells and insert into U is $O(N^3)$. Using red-black trees to implement sets, this bound on the number of constraints to insert into U implies that inserting them all takes $O(N^3 \log(N^3)) = O(N^3 \log(N))$ time. Therefore step 1 takes $O(N^3 \log(N))$ time.

For step 2, deleting each vertex Q_i from a 3D Delaunay triangulation and retriangulating the hole takes $O(fd)$ time, where f is the number of tetrahedra that retriangulate the hole, and d is the degree of Q [28]. This equates to the loose bound $O(N^3)$ time per vertex Q_i, as in the proof of Theorem 4. Because there may be up to N such vertices, Q_i, step 2 takes $O(N^4)$ time.

For step 3, there are at most N vertices Q_i' to insert. For each insertion, at worst all $O(N^2)$ tetrahedra conflict, and thus are deleted and stared off in $O(N^2)$ time. Locating the conflicting cells takes no more than $O(N^2)$ time, since even a naive enumeration of all $O(N^2)$ tetrahedra suffices. Thus, inserting all the vertices takes $O(N^3)$ time. To recarve the new tetrahedra, the constraints from the old tetrahedra are collected into U. Since $K < \infty$, and since the number of deleted tetrahedra is $O(N^2)$, the number of constraints to reprocess is $O(N^2)$. Therefore, inserting them into U, which will never contain more than the original $O(N^2)$ constraints from the triangulation at the start of this algorithm, can be done in $O(N^2 \log(N^2)) = O(N^2 \log(N))$ time.

For step 4, to simply loop over all the cells and their $O(1)$-bounded constraint sets takes $O(N^2)$ time, because there are at most $O(N^2)$ tetrahedra in the triangulation [60]. Testing a constraint for removal is an $O(N)$ operation, since it entails checking if the constraint (represented by a pair of indices) references any of the up to N Q_i. Removing a constraint from a tetrahedron's constraint set is an $O(f(K)) = O(1)$ operation, because $K < \infty$ is constant. Therefore the total cost of step 4 is $O(N^2 N) = O(N^3)$.

For step 5, Theorem 1 shows that applying a single constraint takes $O(N^2)$ time. Since there are $O(N^2)$ constraints in U, the total time for applying them all in step 5 is $O(N^4)$.

The total time complexity for Algorithm 4 is then $O(N^3 \log(N) + N^4 + N^2 \log(N) + N^3 + N^4) = O(N^4)$. □