Aerial Reconstructions via Probabilistic Data Fusion

by

Randi Cabezas

B.S., Electrical Engineering
Florida International University, 2010

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science
in Electrical Engineering and Computer Science
at the Massachusetts Institute of Technology

August 2013

Signature of Author: _____

Department of Electrical Engineering and Computer Science
August 23, 2013

Certified by: _____

John W. Fisher III
Senior Research Scientist
Thesis Supervisor

Accepted by: _____

Professor Leslie A. Kolodziejski
Chair, Department Committee on Graduate Students

Aerial Reconstructions via Probabilistic Data Fusion

by Randi Cabezas

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Science

Abstract

In this thesis we propose a probabilistic model that incorporates multi-modal noisy measurements: aerial images and Light Detection and Ranging (LiDAR) to recover scene geometry and appearance in order to build a 3D photo-realistic model of a given scene. In urban environments, these reconstructions have many applications, such as surveillance, and urban planning. The proposed probabilistic model can be viewed as a data fusion model, in which the two data sources complement each other and allow for better results than when only a single one is present. Moreover, this modeling approach has the advantages that it can capture uncertainty in reconstructions, and the ability to incorporate additional scene measurements easily when the sensor models are available.

Furthermore, the results obtained with the proposed method are qualitatively comparable to those obtained with traditional structure from motion, despite differences in modeling approach and reconstruction goals. The appearance and geometry trade-off present in the model between the different data sources can be used to obtain a similar (and sometime superior) reconstruction of complex urban scenes with fewer image observations over traditional reconstruction methods. Extending beyond reconstructions, the proposed model has two alluring features: first we are able to determine absolute scale and orientation, and secondly, we are able to detect moving objects.

From an implementation standpoint, this thesis has shown how to leverage the power of graphic processing units (GPUs) and parallel programming to allow fast inference. Achieving real time rendering of scenes with hundreds of thousands of geometric primitives and inferring latent appearance, camera pose and geometry in the order of seconds each.

Thesis Supervisor: John W. Fisher III
Title: Senior Research Scientist

Acknowledgments

I would like to start by thanking John Fisher for providing me with a stimulating research project. More than that, for his patience, encouragements and overall ability to run with whatever topic I happen to walk in with any given week. Without his support this thesis would not have been possible.

I would also like to thank my fellow students in vision and SLI; thanks for all the invaluable help. In particular, my office mates Sue Zheng, Archana Venkataraman, and Jason Chang have always been there to answer any type of questions from the silly and bizarre to the challenging and unexplainable (sorry for that). Sue your patience and ability to distill a problem to its root amazes me. You have taught me to look beyond the obvious and focus on the more important things. Archana your deep understanding of probability and inference inspire me to be better. Jason your sampling and coding skill astound me; thanks to you my coding skills have improved vastly. Many thanks to Duke for the distractions and fun while at the office.

Outside lab, I would like to thank the inhabitants of the Pleasant House: Anna Lessios, Guillaume Lestoquoy, Sam Nicaise, Andy Sweet, for a great atmosphere and plenty of food-centered good times. Anna, your deserts are amazing! Please keep on indulging us with your wonderful creations. Gui, your passion and skills are inspiring. To the vegetarian zombies, I have not giving up in converting you. Sam, you are a lean and mean machine, I don't know how you find time for everything in your plate. Andy, one day I'll match your style and bold color scheme, but until then, shine on! To the rest of my friends, both near and afar, thank you for your support and the memories we have shared.

I've had the pleasure of being part of an extraordinary group of people in the Graduate Student Council. I would like to thank all the other excomm members who helped me play a small role in the shaping of the present and future of this great place we all love dearly.

Most importantly, I thank my parents and sister for their love and encouragements.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

In this thesis we consider the problem of 3D scene analysis inferred from multi-modal data sources. Wide area motion imagery (WAMI) and Light Detection and Ranging (LiDAR) provide rich information about large geographic areas at relatively low-cost. While a great deal of attention has been paid to the idea of 3D scene reconstruction from image sequences, the existing literature on multi-modal scene reconstruction is somewhat lacking. One challenge to principled multi-modal approaches is that one must consider the joint statistical properties across modalities in order to combine measurements within a consistent mathematical framework. Here we investigate a generative Bayesian formulation in which joint properties are encoded implicitly via a latent representation of the scene.

At some level this introduces greater model complexity since in order to obtain the desired geometry we must not only infer latent camera parameters but also the latent parameters associated with each of the additional sensor modalities. Furthermore, the application of interest represents a greater challenge to standard image-based approaches owing to the extreme geometry disparity and wide baseline of the data collection.

In formulating the problem as one of statistical inference within the Bayesian framework, we address several key modeling and computational issues. In particular, we incorporate the use of graphics processing units (GPUs) for fast computation. Furthermore, we demonstrate a variety of advantages of the proposed approach as compared to image-based methods, such as the ability to obtain dense reconstructions, in both geometry and appearance, that can help identify small but crucial scene details.

Empirical comparisons demonstrate that comparable reconstructions (from a qualitative perspective) are achievable using fewer image measurements when exploiting the geometric information provided by LiDAR. In contrast to image-based methods we also demonstrate the ability to measure the physical dimensions of a scene at an absolute versus relative scale, and the ability to detect moving objects against a static background. These last aspects follow directly from the proposed approach.

In the rest of this chapter we discuss the motivation for this work and briefly review the proposed approach; we culminate with an outline for the rest of this thesis.

1.1 Motivation

Aerial imagery provides the means for obtaining a large amount of information for a large scene. In dense urban environments this data has many applications, such as surveillance, object tracking and urban planning among others. The common factor in all these applications is the knowledge of the underlying 3D structure of the scene. Despite recent and major advances in 3D reconstructions, the reconstructions of 3D environment from a collection of aerial views remains a challenging problem. The problem of estimating the 3D geometry of a scene from a collection of images without any prior knowledge is known as Structure from Motion (SfM). That is, SfM attempts to recover the 3D world state (structure) of a scene and the camera parameters (motion) of an ensemble of images.

In order to accomplish this, several techniques have been developed, based on the principles of image formation and multi-view geometry. These techniques can be broadly classified into sequential methods, in which structure and motion are estimated iteratively while adding additional images; or factorization methods in which structure and motion are computed simultaneously for all images. Usually, these two methods are treated as initial estimates and are further refined using a non-linear optimization to minimize a suitable cost function; this step is referred to as bundle adjustment.

In practice, SfM can take advantage of additional information about the camera pose parameters –position (x,y,z), viewing directions (yaw, pitch, roll), and focal length. Digital camera routinely embed camera information within the image representation, consequently it is common to information regarding the intrinsic camera parameters (e.g. focal length) and, on occasion, information regarding the extrinsic information (e.g. GPS measurements). Despite these advantages several challenges still make this problem difficult. Specifically, a large number of images are needed to accurately characterize a scene, this in turn increases computation complexity and running time since the amount of coupling between images drastically increases with each additional image that observes a common 3D point. As a result, large and complex scenes require hundreds or thousands of images and weeks of computation time.

While SfM methods are widely used, the high number of images required for suitable reconstructions reduces their appeal. As an alternative, one can reduce the number of images used, but this typically results in a decrease in accuracy. The issue is primarily one of obtaining a sufficient number of accurate correspondences within wide baseline image set. However, these two components typically tend to work against each other. That is, wide baseline give better estimates, but make it more difficult to find accurate correspondences, since reliably finding correspondences depends on how distinct the appearance is locally within a scene.

A possible alternative to decreasing the number of input images would be to introduce additional geometric information about the scene. As an example, let us consider adding the information that a given scene is composed of a single plane, i.e. that all points lie in the same plane. In this case the planar structural information can reduce computation cost (search a 2D plane instead of 3D space). Additionally, the number of

images needed can be reduced, reducing image coupling and further reducing computation time. Moreover, since we are trading off images for geometric information, the accuracy of the reconstruction should not decrease.

As we will see, the idea of using geometric information in SfM has been examined in previous work. The conclusion is similar: under certain assumptions, introducing geometric constraints into SFM generally leads to more accurate solutions at lower computational cost. While consensus exist that adding geometric constraints is of great use, the conditions under which these constraints are best utilized, and even the form of the constraints is not generally agreed upon. Common constraints include a variety of geometric primitives, such as lines and planes, but higher order primitives, such as cuboids and spheres, can also be used.

In practice, the form of the constraint is limited by its source; for example knowing that images are from an urban scene will probably introduce cuboid constraints due to the high regularity in a man-made environments. However, since no general information is available it is up to the end-user to input these constraints, usually in the form of points or lines for simplicity. However, if additional geometric data was available, automatic methods could be develop to incorporate this information. In this thesis, we propose such a method.

■ 1.2 Approach

We propose a probabilistic model that incorporates multi-modal measurements; optical and geometric; to recover scene geometry and appearance in order to build a 3D photo-realistic model of a given scene. Our model can be viewed as a data fusion model, in which multiple observations are being combined to provided a cohesive and unified representation of a scene.

This framework will allow us to incorporate a noise model in each measurement, which can be used to capture uncertainty in the reconstructions. Furthermore, this approach has the ability to incorporate additional scene measurements easily, allowing future expansions and extra sources of information when available.

We focus on two measurement modalities: aerial images and Light Detection and Ranging (LiDAR). LiDAR is helpful since it provides geometric information. This information can be used to reduce the number of images and the required computation time while maintaining comparable results.

There are several differences between the proposed model and traditional SfM, both in approach and end result. In terms of approach, traditional SfM takes a purely geometric route in that all image measurements are resolved to correspondences across images, once the correspondences have been established a suitable geometric function is minimized and image information is no longer used. In contrast the proposed approach explicitly uses appearance information, and relies heavily on it as a way of computing probabilistic likelihoods.

In terms of results, the most noticeable difference is the choice of geometry primitive,

while SfM reconstructions are comprised of point clouds, the approach in this thesis uses triangular elements as the basis of the representation. Furthermore, the model recovers an appearance image for each triangle while SfM recovers a color for each point.

1.3 Outline

The thesis is structured as follows: chapter 2, discusses necessary background, including the basics of image formations and the details of Structure from Motion (SfM), as well as the extensions to piecewise planar reconstructions; useful mathematical concepts such as the Gaussian Process prior are also discussed.

Chapter 3 presents the reconstruction problem as a probabilistic data fusion model. That chapter discusses the observation models, and prior assumptions use in this work. It also presents inference procedures and update equations for the latent variables in the proposed model. The implementation details and inference algorithms are presented in chapter 4. The chapter details the computation of the likelihoods and how to leverage graphics hardware to obtain substantial computational efficiency.

Experimental results are presented in chapter 5. These results include model validation, comparison to well known structure from motion algorithms, as well as aspects of the model that extend beyond reconstructions. Concluding remarks and future work are discussed in chapter 6.

Appendices provide useful information including an overview of the data used throughout this work (appendix A); derivations of the appearance updates and useful Gaussian marginalization as discussed in appendices B and C respectively. OpenGL implementation details are provided in appendix D.

Chapter 2

Background

In this chapter we develop key background concepts related to Structure from Motion and LiDAR. We begin with a brief review of previous works. We follow with basic concepts such as homogeneous coordinates and image formation. Concepts relating to single-, two-, and multi-view geometry are discussed next. Once the basics have been discussed we circle back to structure from motion and piecewise planar reconstructions.

After the recap of 3D scene reconstructions we introduce a set of useful concepts beginning with a description of LiDAR and Gaussian Process priors. We culminate with a concise introduction to OpenGL and CUDA.

2.1 Related Work

As we will demonstrate this work differs from other aerial reconstruction works in both goal and approach. This section will look at related works in three main categories, general reconstruction works, higher order primitives and aerial reconstruction works.

Traditional reconstruction works fall on the Structure and Motion (SfM) category. These works [19, 49, 50, 52, 56], relay solely on a collection of images and require extracting and matching correspondence points across the images. Recent advancements of SfM by [1, 42, 49, 50] have made reconstructions of urban scenes possible from a large collection of street views. This work differs from classical SfM in that we incorporate scene geometry; we model higher order primitives rather than single points; and most importantly we substitute explicit point correspondence, with dense implicit correspondences.

Previous works that include geometry information include [2, 4–6, 12, 16, 55, 63]. As stated earlier, these works all point out that the introduction of geometric information can help find a better solution faster for the SfM problem. However, most of these methods require the user to input the geometric information, typically in the form of lines and points. This undesirable step can lead to errors and reconstruction quality that vary from user to user and day to day.

There are many methods that recover higher order primitives such as planes, or parallelograms [24, 48] – also known as piecewise planar reconstructions. All of these have SfM as a required initial step. The results of these methods are highly coupled with the results of traditional SfM since most of these works focus on refining and fitting

the SfM output. These methods produced very accurate geometric results when the primitives reconstructed match the underlying geometry.

Aerial reconstruction works solely based on images has not been of much interest [27, 35, 60]. On the other hand Aerial reconstruction with geometry information has achieved much attention in the photometry field over the last decade [9, 10, 17, 22, 26, 31, 33, 51, 53, 58, 61, 62, 64]. These work rely on LiDAR to reconstruct underlying scene geometry and images to produce appearance for that geometry. To our knowledge no work attempts to refine LiDAR geometry with image information. Previous works that include SfM in a probabilistic framework include [11, 20, 43].

To our knowledge no work attempts to combine geometry information and appearance information in a probabilistic model to reconstruct 3D urban scenes.

■ 2.2 Homogeneous Coordinates

Before beginning we should clarify some of the notation that will be used. The distinction between euclidean and homogeneous coordinates is an important one; we denote 3D Euclidean points by $\mathbf{X} = \begin{bmatrix} X & Y & Z \end{bmatrix}^\top$, similarly a 2D Euclidean point is represented by $\mathbf{x} = \begin{bmatrix} x & y \end{bmatrix}^\top$. Homogeneous coordinates are denoted by tildes, such that a 3D homogeneous point is $\tilde{\mathbf{X}} \sim \begin{bmatrix} \tilde{X} & \tilde{Y} & \tilde{Z} & \tilde{W} \end{bmatrix}^\top$ and a 2D point $\tilde{\mathbf{x}} \sim \begin{bmatrix} \tilde{x} & \tilde{y} & \tilde{w} \end{bmatrix}^\top$. Image pixel location, have similar notation to 2D homogeneous coordinates $\tilde{\mathbf{u}} = \begin{bmatrix} u & v \end{bmatrix}^\top$. Notation aside, let us get started.

Homogeneous vectors refer to the equivalent class, under which two vectors are equivalent up to a non-zero scaling. That is, 3D Euclidean vector $\mathbf{X} = \begin{bmatrix} X & Y & Z \end{bmatrix}^\top$ and $k\mathbf{X} = \begin{bmatrix} kX & kY & kZ \end{bmatrix}^\top$ are equivalent $\forall k \neq 0$. Thus any vector $\mathbf{X}$ is representative of the equivalence class. The space of such vectors is $\mathbb{P}^3$. It becomes natural to think of an augmented vector to include the scaling factor, such that $\tilde{\mathbf{X}} \sim \begin{bmatrix} kX & kY & kZ & k \end{bmatrix}^\top = \begin{bmatrix} \tilde{X} & \tilde{Y} & \tilde{Z} & k \end{bmatrix}^\top$ (for clarity we continue to use k as a scaling parameter but a more general notation $\tilde{W}$ can also be used). It becomes clear that $\mathbf{X} = \begin{bmatrix} \tilde{X}/k & \tilde{Y}/k & \tilde{Z}/k \end{bmatrix}^\top$, and $\tilde{\mathbf{X}} \sim \begin{bmatrix} X & Y & Z & 1 \end{bmatrix}^\top$.

Similar reasoning applies to the 2D Euclidean and homogeneous coordinates (in space $\mathbb{P}^2$). A 2D Euclidean point $\mathbf{x} = \begin{bmatrix} x & y \end{bmatrix}^\top$ has homogeneous coordinates given by $\tilde{\mathbf{x}} \sim \begin{bmatrix} \tilde{x} & \tilde{y} & \tilde{w} \end{bmatrix}^\top$. Furthermore, $\tilde{\mathbf{x}} \sim \begin{bmatrix} x & y & 1 \end{bmatrix}^\top$ and $\mathbf{x} = \begin{bmatrix} \tilde{x}/\tilde{w} & \tilde{y}/\tilde{w} \end{bmatrix}^\top$

For more detailed treatment of homogeneous coordinates refer to [28].

■ 2.3 Image Formation

In this section we describe the basics of image formation. This section is based on the projective pinhole camera model. This model is an accurate description of real cameras when non-linear effects can be neglected (or compensated).

The pinhole model assumes that ray traicing can form a 2D image. That is, the

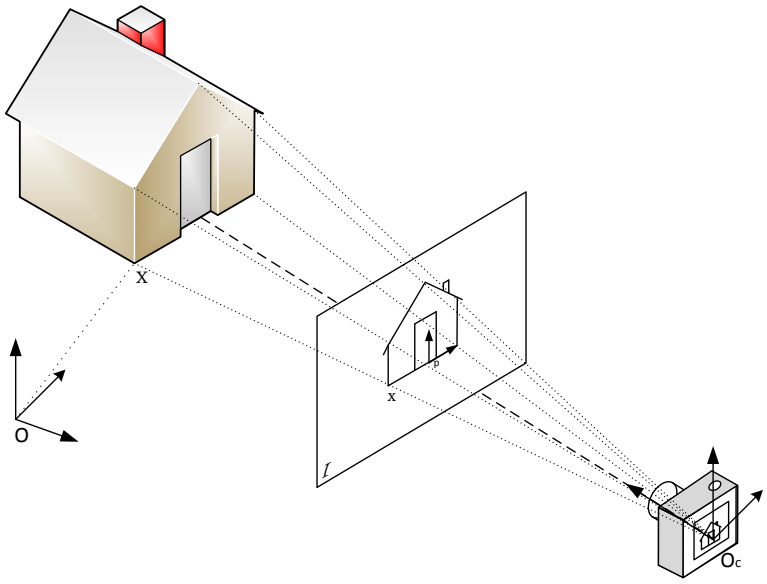


Figure 2.1: Pinhole Camera Diagram. The image is formed by keeping all rays (a few denoted by dotted lines) that intersect plane $\mathcal{I}$ and pass through camera center O_c (dashed line denotes principal axis, point p denotes the principal point, where the principal axis intersects the image plane).

image is formed by keeping every ray that intersects an image plane and passes through the camera center[1], see figure 2.1.

Mathematically, we can express the transformation of a 3D world point to a 2D image pixel by three components, a 3D-to-3D transformation of world coordinates to camera coordinate, a 3D-to-2D transformation of camera coordinates to image plane and a 2D-to-2D image plane to pixel transformation.

The 3D-to-3D transformation can be written in homogeneous coordinates as:

$$\tilde{\mathbf{X}}_c \sim \begin{bmatrix} \mathbf{R} & \mathbf{T} \\ \mathbf{0} & 1 \end{bmatrix} \tilde{\mathbf{X}} \tag{2.1}$$

where $\mathbf{R}$ is 3×3 rotation matrix, $\mathbf{T}$ is a 3×1 translation vector, $\tilde{\mathbf{X}}$ is the homogeneous coordinate of point $\mathbf{X}$ with respect to world coordinate frame, and $\tilde{\mathbf{X}}_c$ is the homogeneous coordinate with respect to camera coordinate frame.

The 3D-to-2D transformation of camera coordinate frame to image plane can be written as:

$$\tilde{\mathbf{x}} \sim diag(f, f, 1) \left[I \mid 0 \right] \tilde{\mathbf{X}}_c \tag{2.2}$$

[1]For simplicity we are adopting the convention that rays pass through the camera center and the image plane is a distance of f (focal length) from the camera center in the direction of the scene, rather than having rays pass through the focal point and the image plane located behind camera center and inversed. These interpretations are equivalent

where $diag(f, f, 1)$ is a 3x3 diagonal matrix with diagonal entries $\{f, f, 1\}$, where f is the camera focal length. I is the identity matrix and $\tilde{\mathbf{x}}$ is the projection of $\mathbf{X}$ onto the image plane.

The remaining transformation is the 2D-to-2D transformation from image plane to pixel coordinates. This transformation can be written as:

$$\tilde{\mathbf{u}} \sim \mathbf{K}\tilde{\mathbf{x}} \tag{2.3}$$

where

$$\mathbf{K} = \begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

and it is known as the calibration matrix. Generally, $\mathbf{K}$ is upper triangular; entries (f_u, f_v) denote the focal length in the horizontal and vertical direction; (u_0, v_0) denote the principal point; and s denotes the skew of pixels (only used when dealing with non-rectangular pixels), collectively these parameters are referred to as intrinsic parameters.

The three transformations (eq: 2.1, 2.2 and 2.3) can be combine to write the projective transformation between world point $\mathbf{X}$ to pixel coordinate $\tilde{\mathbf{u}}$

$$\tilde{\mathbf{u}} \sim \mathbf{P}\tilde{\mathbf{X}} \tag{2.5}$$

where

$$\mathbf{P} \sim \mathbf{K}\,[\mathbf{R}\,\mathbf{T}] \tag{2.6}$$

is a 3×4 projection matrix.

Thus, equation 2.5 allows us to calculate the pixel location of a 3D world point provided we know, the parameters for the rotation matrix $\mathbf{R}$ usually as a function of three angles: yaw, pitch and roll, the translation vector $\mathbf{T}$, and the focal length f. A total of 7 parameters[2]. The parameters $\mathbf{R}$, and $\mathbf{T}$ are referred to as extrinsic parameters.

We can consider the opposite situation in which we know the location of the points $\mathbf{X}$ and $\tilde{\mathbf{u}}$, then we can stack the equations in the form of equation 2.5 and solve for the unknown parameters. This calibration problem can be solved by knowing 6 3D world points and their corresponding pixel coordinates. This will allows to calculate $\mathbf{P}$, we can then obtain $\mathbf{K}$ and $[\mathbf{R}\,\mathbf{T}]$ by RQ decomposition.

For a more detailed discussion of Pinhole cameras and projective transformations refer to [21, 28, 54].

■ 2.4 Two-View Geometry

In this section we cover the basics of two-view geometry. In particular we relate pixels in one image to pixels in another image, provided the pixels correspond to the same

[2]Under the assumption that $f_u = f_v = f$ and $s = 0$, which is quite reasonable in practice, since non-rectangular pixels are not common and focal length tends to be symmetric.

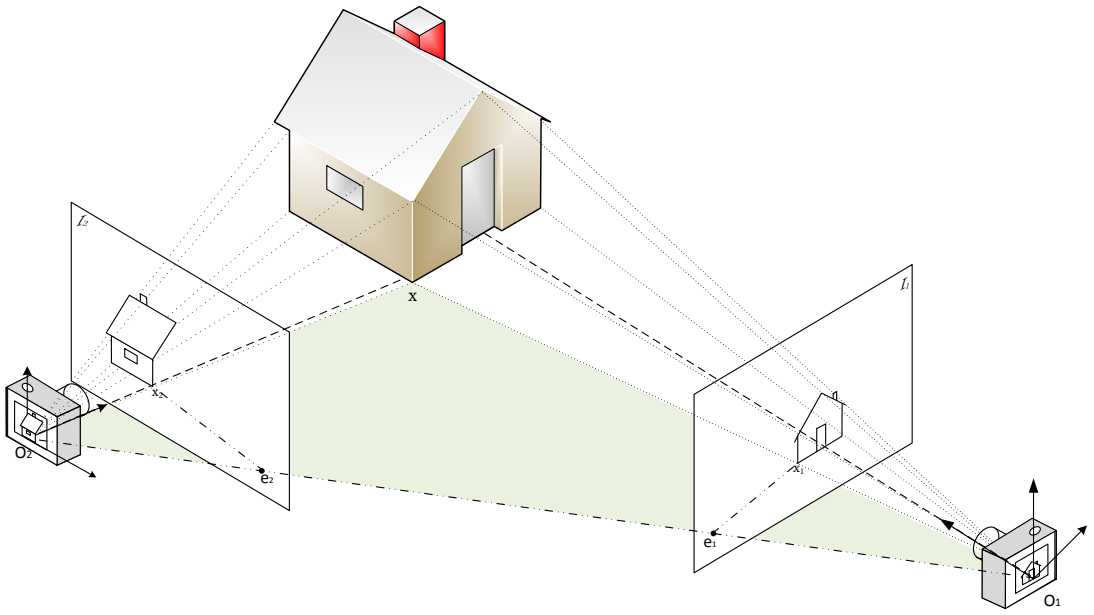


Figure 2.2: Two View geometry with cameras centered at O_1 and O_2, epipoles e_1 and e_2 and epipolar plane shown in green.

3D world point; we recover camera parameters and 3D world points for corresponding pixels.

Let us begin by considering the problem depicted in figure 2.2. We can generalize that camera 1 with center O_1 has projection matrix $\mathbf{P}_1$ (as given by eq 2.6), similarly camera 2 with center O_2 has projection matrix $\mathbf{P}_2$. Furthermore, we can generalize that the relation between both cameras is an unknown rotation $\mathbf{R}$ and translation $\mathbf{T}$.

Given a point $\mathbf{X}_1$ in camera 1, its position in camera 2, $\mathbf{X}_2$, can be written as:

$$\mathbf{X}_2 = \mathbf{R}\mathbf{X}_1 + \mathbf{T} \tag{2.7}$$

pre-multiplying both sides by $\mathbf{X}_2^\top [\mathbf{T}]_\times$ where for $\mathbf{T} = [t_x\ t_y\ t_z]^\top$

$$[\mathbf{T}]_\times = \begin{bmatrix} 0 & -t_x & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix} \tag{2.8}$$

we obtain

$$\mathbf{X}_2^\top [\mathbf{T}]_\times \mathbf{R}\mathbf{X}_1 = \mathbf{X}_2^\top \mathbf{E}\mathbf{X}_1 = 0 \tag{2.9}$$

where $\mathbf{E} \sim [\mathbf{T}]_\times \mathbf{R}$ and it is known as the essential matrix. We can use equation 2.2 to relate $\mathbf{X}_1$ to $\tilde{\mathbf{x}}_1$ and $\mathbf{X}_2$ to $\tilde{\mathbf{x}}_2$, from which we see that equation 2.9 also holds for image plane points, yielding

$$\tilde{\mathbf{x}}_2^\top \mathbf{E}\tilde{\mathbf{x}}_1 = 0 \tag{2.10}$$

which is known as the epipolar constraint.

This constraint can be visualized as the projection of a point in one image to another must lie in a line. In particular, lie in the epipolar line of the second image which is the intersection of the image plane and the epipolar plane, a plane that contains the camera centers and the point $\tilde{\mathbf{x}}_1$ in camera 1 (figure 2.2 shows the epipolar plane in green and epipolar lines as dot dashed). Note that the projection of camera center into the opposite image is known as the epipoles (shown in figure 2.2 as e_1 and e_2).

We can apply equation 2.3 to equation 2.10 to generate the relationship between pixel of image 1 and image 2:

$$\tilde{\mathbf{u}}_2^\top \mathbf{F} \tilde{\mathbf{u}}_1 = 0 \tag{2.11}$$

where

$$\mathbf{F} \sim \mathbf{K}_2^{-1\top} \mathbf{E} \mathbf{K}_1^{-1} \tag{2.12}$$

is a 3×3 matrix of rank 2 called the fundamental matrix. Composed of the intrinsic parameters of the cameras and the extrinsic parameters (via the essential matrix).

Similar to the approach taken in section §2.3 to estimate the camera matrix, the constraints of the fundamental matrix can be stacked up and solved as a system in the form $\mathbf{A}\mathbf{f} = 0$ where $\mathbf{f}$ is stacked up coefficients of $\mathbf{F}$ and $\mathbf{A}$ is a matrix composed of pixel coordinates. Since there are 9 unknowns in $\mathbf{F}$ we need at least 9 pixel correspondences between the images, methods described in [32] can obtain $\mathbf{F}$ from 7 correspondences.

As discussed above, the fundamental matrix can be obtained from pixel correspondences across two images. In the special case where we know $\mathbf{K}_1$ and $\mathbf{K}_2$, then the only unknown in equation 2.12 is $\mathbf{E}$, which is given by $\mathbf{E} \sim [\mathbf{T}]_\times \mathbf{R}$. Since $\mathbf{R}$ and $[\mathbf{T}]_\times$ are orthogonal we can decompose $\mathbf{E}$ by an SVD decomposition, with form $\mathbf{E} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^\top$, where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal and $\mathbf{\Lambda}$ is diagonal with last entry 0. Then we can compute:

$$[\mathbf{T}]_\times = \mathbf{U} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{U}^\top \tag{2.13}$$

and

$$\mathbf{R} = \mathbf{U} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \mathbf{V}^\top \tag{2.14}$$

Once we recover the translation vector and rotation matrix we can compute the individual camera projection matrices. Without any loss of generality we can let one camera be the reference camera and the second camera have translation $\mathbf{T}$ and rotation $\mathbf{R}$, that is:

$$\mathbf{P}_1 = \mathbf{K}_1[\mathbf{I} \mid \mathbf{0}] \tag{2.15}$$

$$\mathbf{P}_2 = \mathbf{K}_2[\mathbf{R} \mid \mathbf{T}] \tag{2.16}$$

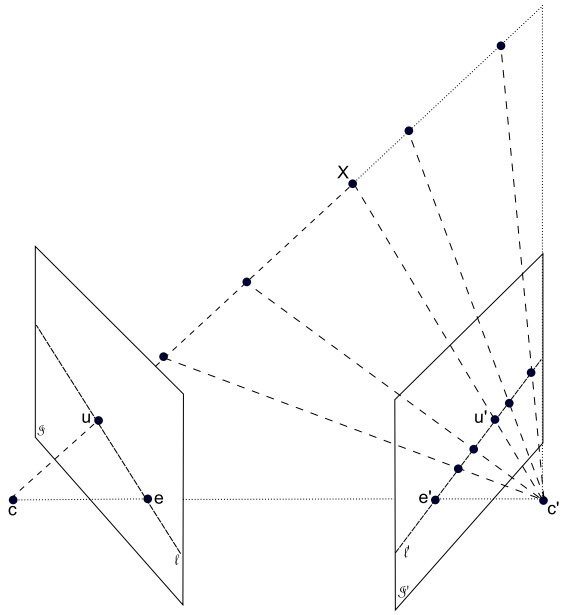


Figure 2.3: 3D world point $\mathbf{X}$ has projection $\tilde{\mathbf{u}}$ in image $\mathcal{I}$ and $\tilde{\mathbf{u}}'$ in image $\mathcal{I}'$. The diagram shows point the location of $\mathbf{X}$ as a search of pixel correspondence $\tilde{\mathbf{u}}'$ over the epipolar line.

The method introduced above assumes that the calibration matrices are known, which is not always the case. However, calibration matrices can be estimated using the Kruppa equations [15] then proceed as above.

To recap, we are now able to describe two views, with their projective matrices based only on pixel correspondences. What we would like to do next is estimate the 3D location of pixel correspondences, this problem is known as Triangulation [29].

The triangulation problem can be visualized in figure 2.3 for two images. We can readily observe, that in a noiseless scenario this problem is trivial. In the noisy case, the search for an optimal 3D point $\mathbf{X}$ can be seen as a correspondence point search along the epipolar line of the second image, which has a non-iterative solution for two views [29].

In a multi-image case for which we know the projection matrix $\mathbf{P}_i$ of each camera i we can cast the problem as a non-linear optimization problem, in the form:

$$\hat{\mathbf{X}} = \underset{\mathbf{X}}{\operatorname{argmin}} \sum_i ||\mathbf{u}_i - \tilde{\mathbf{u}}_i(\mathbf{P}_i, \mathbf{X})||^2 \tag{2.17}$$

where $\mathbf{u}_i$ is the known correspondence pixel in image i, $\tilde{\mathbf{u}}_i$ is the predicted pixel in image i as a function of $\mathbf{X}$ and $\mathbf{P}_i$ and $\hat{\mathbf{X}}$ is the solution found. Equation 2.17, minimizes the square error between the desired pixel and the re-projected pixel based on the estimated value of $\mathbf{X}$ and the projection matrix of image i.

2.5 Multi-View Geometry

In this section we extend the concepts develop in two-view geometry to multiple images. This will be done from the perspective of Structure from Motion. This section will tie together the material from earlier sections. As mentioned in the introduction, SfM methods can be broken down into two main categories: sequential, and factorization methods; we will now briefly explain these.

The most intuitive methods for bundle adjustment are sequential method. They follow from the results develop in section §2.4 and as the name suggest are developed in a sequential manner. That is they start with two images for which a reconstruction based on epipolar constraints is formed, each additional image is reconstructed in a similar manner. Smarter methods take advantage of the 3D points that have already been reconstructed when adding new images as in [7]. As an alternative, merging partial reconstructions of 3D points is also possible [19].

Factorization methods compute camera pose and scene geometry using all images simultaneously. The key assumption behind this method is that the scene does not change between images, and that the number of images is relatively large, so that multiple features are visible across a large number of images [56]. This allows us to decouple the problem of structure and motion; unfortunately this assumption is not always valid.

Bundle adjustment refers to the process of refining the initial estimates of section §2.4 by minimizing an appropriate cost function via an iterative non-linear optimization [8, 57]. Mathematically, the problem can be neatly summarized in a one-line equation:

$$\min_{\hat{\mathbf{P}}^i, \hat{\mathbf{X}}^j} \sum_{ij} d(\hat{\mathbf{P}}^i \mathbf{X}^j, \tilde{\mathbf{x}}^i_j)^2 \tag{2.18}$$

where $\tilde{\mathbf{x}}^i_j = \hat{\mathbf{P}}^i \mathbf{X}_j$ is the projection of world point $\mathbf{X}_j$ to image point $\tilde{\mathbf{x}}^i_j$ of camera i with projection matrix $\hat{\mathbf{P}}^i$ where the image measurement are noisy.

Then bundle adjustment is just finding a projection matrix of camera i and point j that minimize the square of some distance function $d(\cdot, \cdot)$ between the noisy image measurement and the predicted image measurement across all cameras and points. In the case of the noise being modeled as Gaussian, this optimization is equivalent to a Maximum Likelihood (ML) estimate of projection matrix and world point.

The particular distance function $d(\cdot, \cdot)$ must be chosen with some care, since it must be invariant to the transformation of the camera, typically a prediction error is used, $\tilde{\mathbf{x}}^i_j - \hat{\mathbf{P}}^i \mathbf{X}^j$.

Bundle adjustment is powerful tool that in practice can produce satisfactory results. Its main disadvantage is the expensive computation time and its propensity to get stuck in local minima.

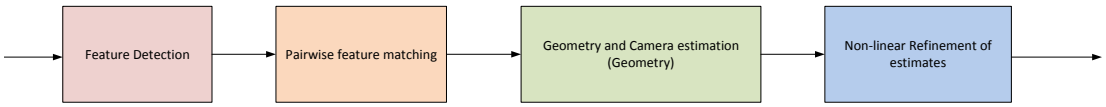


Figure 2.4: Typical Structure from Motion pipeline

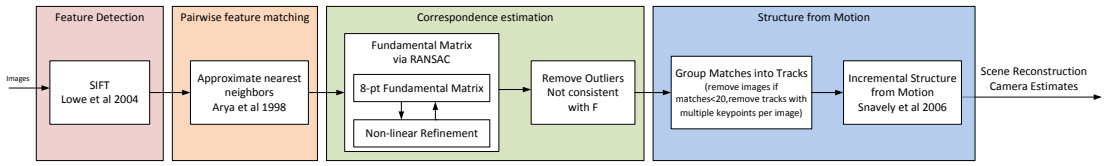


Figure 2.5: Bundler pipeline [50]

■ 2.6 Structure from Motion

The pipeline for sequential structure from motion can be seen in Figure 2.4. As the figure indicates SfM can be thought of as four distinct steps, in the first step interesting image features are extracted for each image, these features are typically chosen to have a set of desired properties such as scale and orientation invariant, and illumination invariant. The next step is to match the features found across the input images, this step relays on the uniqueness of the features and the properties discussed earlier.

Step three of the pipeline involves estimating scene geometry and camera pose from the matched correspondences, using the methods described in §2.4. The main assumption at this step is that matched features correspond to the same 3D scene location. We note that when errors in the matching occur, this step introduces incorrect information. The scene points and camera pose obtained in step three are refined in step four using an iterative refinement, typically bundle adjustment as in §2.5.

Recently, Snavely *et al.* [50], combined the steps of the pipeline in Figure 2.4 with great success into their SfM software package Bundler. Their particular choice of algorithms and methods for each step made the SfM pipeline highly effective. As can be seen from Figure 2.5 their use of SIFT features [37] as well as the approximate nearest neighbor feature matching [3] made the first two steps fast as compared to previous works. The computation of fundamental matrix via the normalized 8-pt algorithm of [29] and RANSAC [18] provided robustness and reduced the chances of including incorrect feature matches.

Their bundle adjust step (cf. Figure 2.6), a modified version of [36] again improved robustness by having a better initialization for the starting match –important since bundle adjustment is known to be initialization dependent and prone to getting stuck in local optima due to initialization. Furthermore, their incremental scheme for adding new images to the solution allows for more stable solutions.

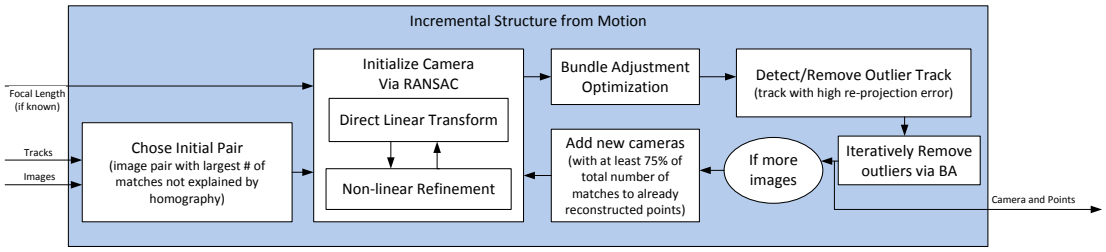


Figure 2.6: Bundler's Bundle Adjustment Details [50]

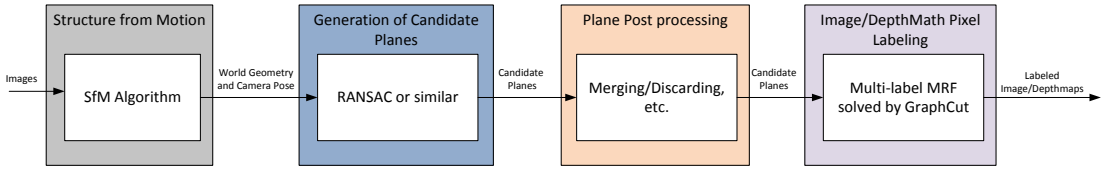


Figure 2.7: Typical piecewise planer reconstruction pipeline

■ 2.7 Piecewise Planar reconstructions

A typical pipeline for piecewise planar reconstructions can be seen in Figure 2.7. Typical piecewise planar algorithms begin by first running structure from motion to obtain candidate camera pose estimates and dense world deapthmaps. The depthmaps are searched to produce a candidate set of planes, the search usually involves RANSAC or other algorithms that are robust to outliers. The next step prunes the list of planes, by merging or discarding planes that are the redundant or incorrect. The final step views the deapthmaps as a labeling problem in a Markov random field and solves for the optimal boundary between planes, typically via GraphCut.

As previously discussed the results of planar reconstructions are highly tided to SfM results, since that is the first step in the pipeline. Furthermore piecewise planar work can be seen as refinement to the results traditional structure from motion.

Variations from this pipeline are common, for example Gallup *et al.* [24], introduce a label to indicate non-planar region, and thus allow for objects, such as trees, to not be reconstructed as planes.

■ 2.8 Light Detection and Ranging

Light Detection and Ranging (LiDAR) refers to the class of optical remote sensing technology that measures distance and/or material properties. In airborne LiDAR, a ranging system is mounted in an aircraft along with a position tracking system such as GPS. The system continuously scans the ground plane with light pulses, usually in the ultraviolet or near infrared spectrum, figure 2.8. By controlling the aircraft's altitude and velocity, and measuring the return delay of a pulse a relative distance to ground

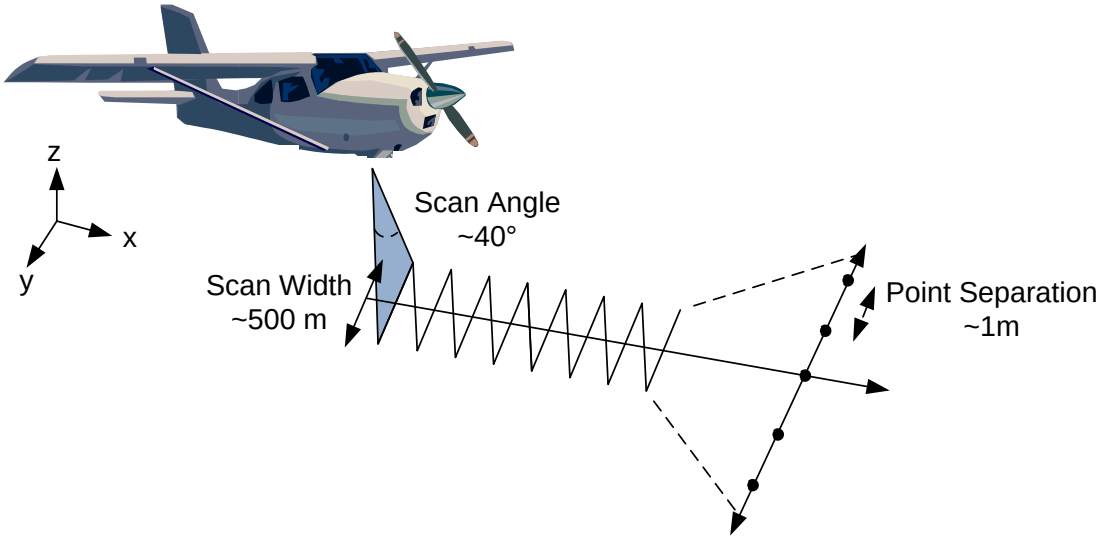


Figure 2.8: Typical LiDAR scan pattern, with common parameters

can be obtained which when combined with horizontal position can give a very accurate 3D point, accuracy in the order of centimeters. By combining a great number of scan lines and pulse returns we can obtain a large point cloud of the area in question, as shown in figure 2.9 for the MIT Dome.

LiDAR is of particular interest because it provides us with a set of points for which we know the geometry with a high degree of accuracy. This makes LiDAR ideal to be introduced as constraints to any model. Furthermore, since the set of points are relatively dense, we can readily extract surface contours from them. As a simple example consider projecting all points to the ground plane, performing a Delaunay triangulation and re-projecting points to their original elevations along with their connections; this produces a triangular mesh that while simple, is still an effective manner of creating a 3D structural model from LiDAR returns.

■ 2.9 Gaussian Process Prior

For most of our applications the data we obtain is from city fly-overs. This type of data has certain characteristics that can be incorporated as prior knowledge. For example, we can expect that that plane collecting the measurements follows a smooth trajectory. This implies that the camera collecting the images will also have a smooth position as a function of time. For similar reasons, we can expect the orientation to vary smoothly. We can incorporate this information as a prior in the form of a Gaussian Process Prior.

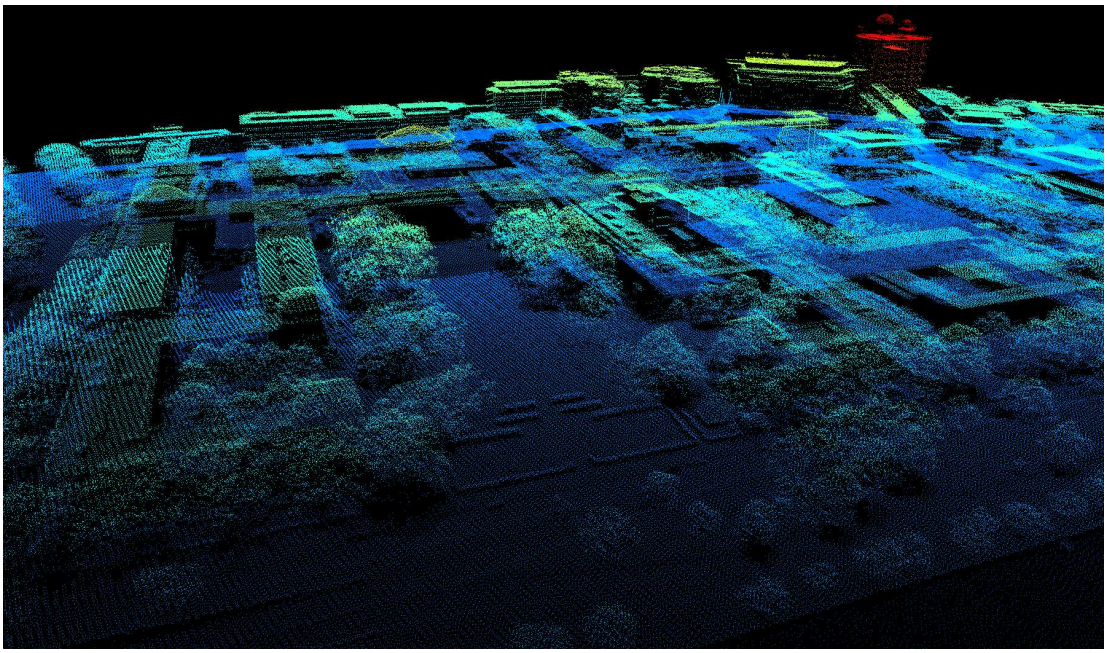


Figure 2.9: LiDAR representation of MIT Dome and Killian Court (color represent height above ground).

■ 2.9.1 Definition

Simply stated a *Gaussian Process* is a collection of infinite random variables, any finite number of which have a Gaussian distributions. A Gaussian process is a generalization of a multivariate Gaussian distribution to infinitely many variables.

We can fully specify a Gaussian process by a mean function $m(x)$ and a covariance kernel $k(x, x')$:

$$f(x) \sim \mathcal{GP}\left(m(x), k(x, x')\right) \tag{2.19}$$

where x is one-dimensional variable (chosen for simplicity of exposition, Gaussian process can easily be extended to multidimensional case).

In practice we do not need to instantiate the infinite collection of random variables, instead we can focus on a finite collection of them $\mathbf{f} = [f(x_1), f(x_2), \ldots, f(x_n)]^\top$, if we let $m(x) = 0$ for all x, then

$$\mathbf{f} \sim \mathcal{N}(0, \Sigma)$$

where $\Sigma_{ij} = k(x_i, x_j)$, $\forall i, j \in [1, n]$.

For many applications $m(x)$ can be made zero and $k(x, x')$ can take the squared exponential function:

$$k(x, x') = v^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) \tag{2.20}$$

The hyper-parameters (υ, ℓ) in the square exponential function (equation 2.20) have specific interpretation. The lengthscale, ℓ, controls the level of variability expected in the input. Larger lengthscales indicate that the function is expected to vary slowly, while shorter lengthscales indicate rapid changes in the function values. The Signal variance, υ, defines the vertical scale of variations of a typical function (see Figure 2.10).

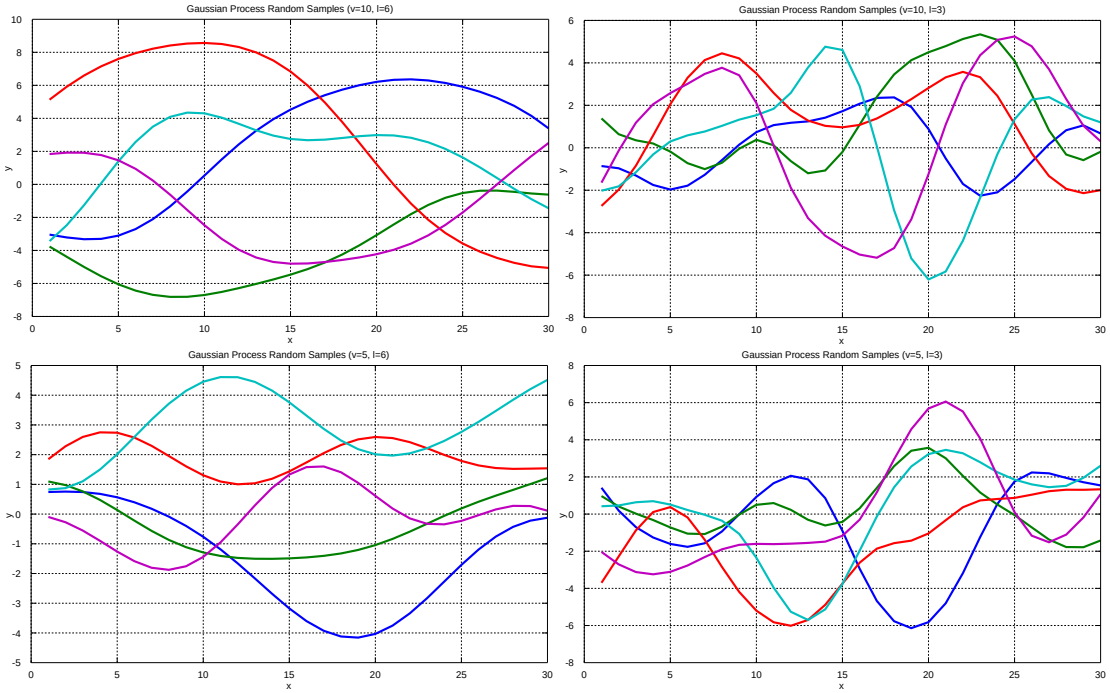


Figure 2.10: Random Samples from a Gaussian process with different hyperparameters. Left column has a large lengthscale, producing very smooth samples; Right column has smaller lengthscale resulting in rapidly varying samples. Top row and bottom row have different signal variances, hence their amplitude ranges vary according.

■ 2.9.2 Inference

Typically, a Gaussian Process is used as a prior. In the case where the likelihood is Gaussian, this results in a close form solution for the posterior (also Gaussian). We will present the prototypical predictive case here. Let us assume we have noisy observations of a smooth function $\mathbf{t}_N = \{x_n, y_n\}_{n=1}^N$, the task is to infer the function $f(x)$.

Assume you have a prior over functions that is a Gaussian Process, such that $f(x) \sim \mathcal{GP}(0, k(x, x'))$, and that the observation model is $p(y|x, f(x)) = \mathcal{N}(\mathbf{f}, \sigma^2 I)$, where I is the identity matrix and σ^2 represent some suitable noise.

The posterior over $f(x)$ becomes:

$$p(f(x)|\mathbf{x},\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x},f(x))p(f(x))}{p(\mathbf{x},\mathbf{y})}$$

The numerator of the right hand side is the multiplication of a Gaussian and a Gaussian Process, which results in another Gaussian process:

$$f(x)|\mathbf{x},\mathbf{y} \sim \mathcal{GP}\left(\mu_{post}(x), k_{post}(x,x')\right)$$

where

$$\mu_{pos}(x) = k(x,\mathbf{x})\left[K(\mathbf{x},\mathbf{x}) + \sigma^2 I\right]^{-1}\mathbf{y}$$

and

$$k_{post}(x,x') = k(x,x') - k(x,\mathbf{x})\left[K(\mathbf{x},\mathbf{x}) + \sigma^2 I\right]^{-1}k(\mathbf{x},x')$$

■ 2.9.3 Hyperparameters

The covariance kernel typically has a few hyper-parameters. For example the square exponential covariance (equation 2.20) has two hyper-parameters, the lengthscale and the scale variance. These parameters can be manually tuned or learned from the data.

An effective way of learning parameters involves looking at the posterior probability of the hyper-parameters $\boldsymbol{\theta}$.

$$p(\boldsymbol{\theta}|\mathbf{x},\mathbf{y},\mathbf{f}) \propto p(\mathbf{y}|\mathbf{x},\mathbf{f},\boldsymbol{\theta})p(\boldsymbol{\theta})$$

where the first term on the right hand side is data likelihood, and the second is a prior over the parameters. Taking the log of the marginal likelihood, we obtain

$$\log p(\mathbf{y}|\mathbf{x},\mathbf{f},\boldsymbol{\theta}) = -\frac{1}{2}\log|K| - \frac{1}{2}\mathbf{y}^\top K^{-1}\mathbf{y} - \frac{n}{2}\log(2\pi)$$

(where K is shorthand for the covariance matrix) which we can optimize over to get the hyper-parameters. Typically the gradient useful in the optimization, we can compute the gradient of the log likelihood with respect to the i^{th} hyper-parameter:

$$\frac{\partial}{\partial\theta_i}\log p(\mathbf{y}|\mathbf{x},\mathbf{f},\boldsymbol{\theta}) = -\frac{1}{2}\text{Trace}\left(K^{-1}\frac{\partial K}{\partial\theta_i}\right) + \frac{1}{2}\mathbf{y}^\top K^{-1}\frac{\partial K}{\partial\theta_i}K^{-1}\mathbf{y}$$

Taking the partial of the matrix K with respect to θ_i is just taking the partials of each entry. Alternatively we can take the partial of the covariance kernel, and build the new ∂K using the modified kernel. For the square exponential covariance, the partial kernels $k_\upsilon(x,x')$ and $k_\ell(x,x')$ are derived below:

$$\begin{aligned}\frac{\partial k(x,x')}{\partial\upsilon} &= \frac{\partial}{\partial\upsilon}\left[\upsilon^2\exp\left(-\frac{(x-x')^2}{2\ell^2}\right)\right] \\ &= \underbrace{2\upsilon\exp\left(-\frac{(x-x')^2}{2\ell^2}\right)}_{k_\upsilon(x,x')}\end{aligned}$$

and

$$\begin{aligned}\frac{\partial k(x,x')}{\partial \ell} &= \frac{\partial}{\partial \ell}\left[\upsilon^2 \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)\right] \\ &= \underbrace{\upsilon^2 \frac{(x-x')^2}{\ell^3} \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)}_{k_\ell(x,x')}\end{aligned}$$

For more on Gaussian Processes see [38, 44].

■ 2.10 OpenGL and CUDA

Our practical implementation rely heavily on OpenGL [34], CUDA [39] (as well as libQGLViewer and Qt [13, 25]).

OpenGL is a low level software interface to graphics hardware, it is specifically built to handle vertex data, typically geometry information, and pixel data, typically image information. These two basic data types allow the construction of sophisticated and complicated objects. These data types (and hence objects they create) are combined in a rendering pipeline to produce 2D representation. These representations allow us to visualize the 3D models with high realistic details such as lighting, occlusion reasoning and perspective effects. For more on OpenGL see [47].

CUDA architecture on the other hand allows us to interact directly with graphic hardware, not for rendering purposes but for general purpose computing. This is desired since using a Graphics Processing Unit (GPU) can lead to high performance implementations due to their parallel nature. Furthermore, combining OpenGL and CUDA is optimal since we can process OpenGL objects while they are in graphics hardware. This leads to a reduction in communication between the Central Processing Unit (CPU) and GPU, which is where most of the performance bottleneck occurs. For more on CUDA see [40, 46].

Chapter 3

Model

Here we describe a probabilistic generative model of LIDAR and motion imagery. Our goal is to construct a latent variable model that is sufficiently expressive to explain multiple measurement modalities while lending itself to efficient inference procedures. The use of graphical models facilitates our goals.

3.1 Model

The implementation presented here follows the directed graphical model shown in Figure 3.1. The implication of the directed graphical model is that the observations are statistically independent *conditioned* on the latent variables. In those terms the model is composed of multiple latent or hidden variables (G, A, K, T, and B), which explain observable variables (L, I, and Z). The variables G and A collectively explain the geometry and appearance of the underlying scene, while K and T explain intrinsic and extrinsic camera parameters. The remaining latent variable, B (denoting background), is incorporated to explain additional variability not captured by the other variables. We explain its role later in this section. Observed variables are shaded gray in Figure 3.1, and denote measured quantities such as a LiDAR point (L), an image (I), or a GPS position (Z).

The model can be conceptually divided into two parts, the first portion encodes the underlying scene structure in the form of geometry and appearance. As seen in Figure 3.1 the world representation consists of N_p independent geometry primitives G_m, with

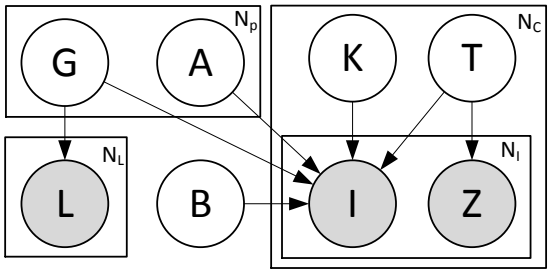


Figure 3.1: Graphical Model Representation of the proposed model.

a canonical appearance A_m, where $m \in [1, N_p]$. The primitives used throughout this work are triangles, parameterized by three 3D points. Appearance variables, A_m, are described over a canonical image coordinate system constrained to the support of a right triangle of specified size. This is then combined with a transformation defined by primitive's parameters to describe the appearance of the primitive in 3-dimensional space.

The second part of the model describes two noisy multi-modal observations: images and LiDAR and their relationship to the underlying world. The LiDAR measurements are expressed in the bottom plate of Figure 3.1 with N_L independent points L_k where $k \in [1, N_L]$. The plate on the right explains image measurements, it consists of N_c independent cameras, each with their own intrinsic parameters K_c and extrinsic camera trajectory T_c where $c \in [1, N_c]$. The intrinsic and extrinsic parameters generate N_i independent images I_n^c and GPS measurements Z_n^c, where $n \in [1, N_i]$.

The joint probability distribution consistent with the graphical model described can be expressed as:

$$p(\mathbf{L}, \mathbf{I}, \mathbf{Z}, \mathbf{G}, \mathbf{A}, \mathbf{T}, \mathbf{K}, \mathbf{B}) = \prod_{c=1}^{N_c} \prod_{i=1}^{N_I} p(I_i^c | \mathbf{G}, \mathbf{A}, \mathbf{B}, K_c, T_c) p(Z_i^c | T_c) \times \prod_{l=1}^{N_l} p(L_l | \mathbf{G}) \prod_{k=1}^{N_p} p(G_k) p(A_k) \prod_{c=1}^{N_c} p(T_c) p(K_c) p(\mathbf{B}) \tag{3.1}$$

The individual terms of equation (3.1) will be discussed in the next sections; for easy reference, Table 3.1 lists the variables used in the model as well as a short description.

■ 3.2 Observation Models

The directed graphical model of Figure 3.1 illustrates independence properties of the variables in the model. However, it is still the case that the parameterization of the model must be provided. In the model presented here, directed edges from latent variable to measurements are derived from a physical sensor model (or approximation there of). Here, we describe the sensor models used in this work and the likelihood equations that follow from those sensors. We emphasize that additional sensors can easily be incorporated to this model by following the approach presented in this section.

■ 3.2.1 LiDAR observation model

We model LiDAR as a point measurement of a 3D world surface plus additive Gaussian noise. That is, the n^{th} LiDAR measurement, L_n, can be expressed as

$$L_n = L_n^{k(n)} + W \tag{3.2}$$

where $L_n^{k(n)}$ is the 3D point on primitive $G_{k(n)}$ that generated the measurement, and $W \sim \mathcal{N}(w; 0, \sigma^2)$. This results in $L_n | L_n^{k(n)} \sim \mathcal{N}(L_n; L_n^{k(n)}; \sigma^2) = \mathcal{N}(d^2(L_n, L_n^{k(n)}); 0, \sigma^2)$, where $d^2(L_n, L_n^{k(n)})$ is the square distance between points L_n and $L_n^{k(n)}$.

Variable	Description
N_p	Number of world primitives.
N_L	Number of LiDAR points.
N_c	Number of individual cameras.
N_I	Number of images (for a given camera).
B	Background Model.
L_k	k^{th} LiDAR measurement, 3D world point, $k \in [1, N_L]$.
G_m	m^{th} world primitive, triangle parameterized by three 3D points, $m \in [1, N_p]$.
A_m	m^{th} world primitive appearance, image of specified size, $m \in [1, N_p]$.
K_c	Intrinsic parameters for the c^{th} camera, consisting of focal length, $c \in [1, N_c]$.
T_c	Extrinsic camera trajectory for the c^{th} camera, 3D position and orientation, $c \in [1, N_c]$.
I_n^c	n^{th} image taken with camera c, $n \in [1, N_I]$.
Z_n^c	n^{th} GPS measurement of camera c, $n \in [1, N_I]$.

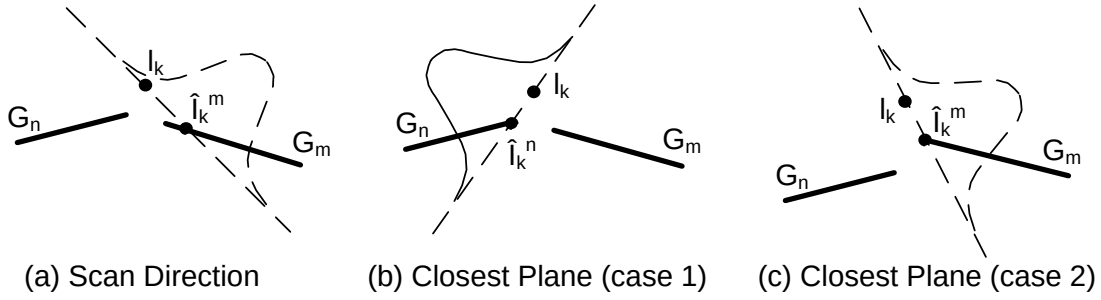
Table 3.1: List of Variables used in Equation (3.1) and Figure 3.1

Equation (3.2) and the likelihood model associated with it uses the distance between the point that generates the measurement and the measurement to assign a probability to the observation. Generally we do not know a priori which point, (or even world primitive) generates which observation, this leads to a data association problem. When thinking about sensor model, the most natural way of associating a LiDAR measurement with a surface is to follow the direction of the ray formed by connecting the collection device (mounted on the airplane as shown in figure 2.8) and the measurement; this yields uncertainty in the direction of the ray as shown in Figure 3.2a. This computation requires knowledge of the aircraft's location at all points during the flight, knowledge of the location of the LiDAR collection device on the aircraft and the scan parameters. Modeling these parameters drastically increases computational complexity and is outside the scope of this thesis.

A simple yet powerful approach would associate the LiDAR measurement to the closest point on a world primitive. (Figure 3.2b, and 3.2c). That is, we can let $L_n|\mathbf{L}_n \sim \mathcal{N}(d^2(L_n, \hat{L}_n^{k(n)}); 0, \sigma^2)$, where $\mathbf{L}_n$ is the collection of all points that L_n can be projected to in all primitives and

$$\hat{L}_n^{k(n)} = \arg\min_{L \in \mathbf{L}_n} d^2(L_n, L). \tag{3.3}$$

Equation (3.3) provides the means to solve the association problem between a measurement and a collection of points that could generate that measurement. The problem is solved using a greedy solution –selecting the closest point in the set to



(a) Scan Direction (b) Closest Plane (case 1) (c) Closest Plane (case 2)

Figure 3.2: LiDAR Observation Models, depicting data association using (a) scan line direction and, (b) and (c) closest plane.

the measurement as the generating point[1]. To obtain the likelihood of a measurement conditioned on all world primitives we can generalize the above procedure as $L_n|\mathbf{G} \sim \mathcal{N}(d^2(L_n, \hat{G}_{k(n)}); 0, \sigma^2)$, where $d^2(L_n, \hat{G}_{k(n)})$ is the distance between point L_n and primitive $\hat{G}_{k(n)}$ and

$$\hat{G}_{k(n)} = \arg\min_{G \in \mathbf{G}} d^2(L_n, G). \tag{3.4}$$

Again we note that the computation of equation (3.4) is computationally intensive since it requires a search over all world primitives. We will discuss implementation details that mitigate this in section 4.3.

■ 3.2.2 Image observation model

The image observation model depends not only on the camera parameters, but also on the world geometry. Pixel (u, v) of the n^{th} image of camera c can be expressed as:

$$I_n^c(u, v) = A_{m^*}(u', v') + Q_n \tag{3.5}$$

where A_{m^*} is the appearance of the m^* primitive, at some coordinate (u', v'), and Q_n is a zero mean Gaussian distribution with variance $r^2_{m^*}$, i.e. $Q_n \sim \mathcal{N}(q; 0, r^2_{m^*})$.

Note that while not shown explicitly in equation (3.5), m^*, u' and v' are functions of u, v, K_c, T_c, and $\mathbf{G}$. We can see this if we interpret equation (3.5) as a mapping of color values from image pixel (u, v) to the appearance of the primitive m^*, this depends on the camera parameter, i.e. where the camera looking, and the visibility of pixels u' and v' of plane m^*, i.e. occlusion reasoning.

We further note that we choose $r^2_{m^*}$, the variance of Q_n to depend on the angle between the triangle in question and the camera's viewing direction, (inverse of the

[1]Data association greatly increases the computational complexity of many inference procedures. For the purpose of this thesis we will utilize a simple greedy method (closest fit). More complex data association methods could be used, but they are beyond the scope of this work.

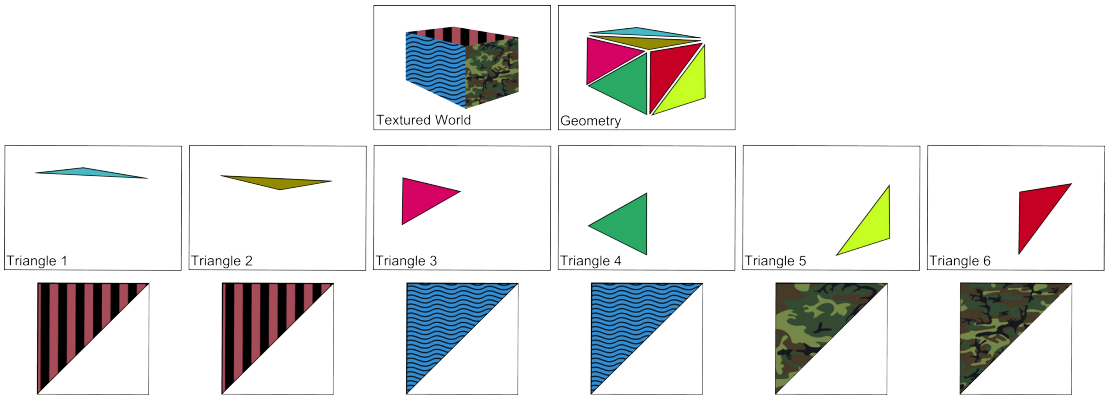


Figure 3.3: Notional representation of the world model. *Top row*: Textured representation and triangles (uniquely color-coded and slightly separated), *Middle and Bottom rows*: 3D triangles and texture representation per triangle.

absolute value of the dot product between viewing direction and triangle normal will be used throughout this work[2]). With these choices, we can write,

$$p(I_n^c|\mathbf{G},\mathbf{A},\mathbf{B},K_c,T_c) = \prod_{k\in\mathcal{S}_n} \mathcal{N}(i_k; a_{m^*(k)}, r_{m^*}^2) \tag{3.6}$$

where $\mathcal{S}_n$ is the set of pixels in image I_n.

Background Model

The image observation model described earlier is not complete. Pictorially, this can be seen in Figure 3.3, where all the white pixels in the textured world image are not currently included in the model representation. Mathematically, equation (3.6) does not explain all observed pixels, only those which project onto existing geometric primitives, all other pixels are undefined.

As such, we include an additional background variable so that all pixel observations are included in the likelihood calculation. That is, allow image pixels to be explained by either a world primitive or a background model. There are a variety of methods by which this could be accomplished. We choose to model background as a ground plane since the camera is typically pointed downwards. We can see the introduction of the background model for the notional example in Figure 3.4, where the background is shown as a stripped cyan plane at ground level that extends across the entire image.

To obtain the new observation model, we need to add the terms that explain the

[2]It is important to point out that this noise parameterization heavily penalizes low grazing angles. Such situations arise when the viewing direction and primitives are almost orthogonal. The dot product should never be zero, since that implies the primitive is not visible

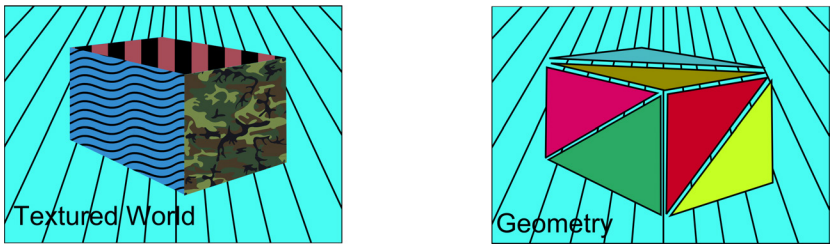


Figure 3.4: Notional representation of the world model including the background (aqua colored)

background:

$$I_n(u,v) = \begin{cases} A_{m^*}(u',v') + Q_n & (u,v) \in \mathcal{V} \\ B_s + W & (u,v) \in \mathcal{B} \end{cases} \tag{3.7}$$

where B_s is one component of the background $\mathbf{B}$, $\mathcal{V}$ is the set of pixels explained by world triangles and $\mathcal{B}$ is the set of background pixels. For simplicity let us assume that the set $\mathcal{B}$ has R unique pixels in it.

With this change the likelihood for the n^{th} image in camera c takes the form:

$$p(I|\mathbf{G},\mathbf{A},\mathbf{B},K_c,T_c) = \prod_{k\in\mathcal{V}} p(I_k^{n,c}|\mathbf{G},\mathbf{A},K_c,T_c) \prod_{k\in\mathcal{B}} p(I_k^{n,c}|\mathbf{G},\mathbf{B},K_c,T_c), \tag{3.8}$$

where we have moved the image index, n, and the camera index, c, to the superscript of I to allow for pixel index k. The background model is not of primary interest and furthermore, owing to the formulation, the model and associated inference procedure can marginalize over this quantity as follows:

$$p(\mathbf{I}|\mathbf{G},\mathbf{A},\mathbf{K},\mathbf{T}) = \int \prod_{c=1}^{N_c} \prod_{n=1}^{N_I} p(I^{n,c}|\mathbf{G},\mathbf{A},\mathbf{B},K_c,T_c)p(\mathbf{B})d\mathbf{B} \tag{3.9}$$

$$= \prod_{c=1}^{N_c} \prod_{n=1}^{N_I} \prod_{k\in\mathcal{V}_c^n} p(I_k^{n,c}|\mathbf{G},\mathbf{A},K_c,T_c) \int \prod_{c=1}^{N_c} \prod_{n=1}^{N_I} \prod_{k\in\mathcal{B}^n} p(I_k^{n,c}|\mathbf{G},\mathbf{B},K_c,T_c)p(\mathbf{B})d\mathbf{B} \tag{3.10}$$

Where equation (3.8) was substituted in equation (3.9) to get (3.10). Note that the image pixels described by existing primitives can be moved outside of the integral. We can perform the integration in (3.10) component-wise for each of the R pixels in the background. If we let

$$p(I_k^{n,c}|\mathbf{G},B,K_c,T_c) = \mathcal{N}(x_k; b_s, \sigma_w^2) \tag{3.11}$$

and

$$p(b_s) = \mathcal{N}(b_s; \mu_b, \sigma_b^2) \tag{3.12}$$

correspond to a single background pixel, then the integral for that component, say b_s, can be evaluated to be:

$$\begin{aligned}\varphi\left(\mathbf{x};\mu_b,\sigma_b^2,\sigma_w^2\right) &= \int \prod_{k=1}^{N} p(I_k|\mathbf{G},B,K_c,T_c)p(B)dB \\ &= \frac{\sigma_w}{(\sqrt{2\pi}\sigma_w)^n\sqrt{N\sigma_b^2+\sigma_w^2}}\exp\left\{-\frac{1}{2}\left(\frac{1}{\sigma_w^2}\sum_{k=1}^{N}x_k^2+\frac{\mu_b^2}{\sigma_b^2}\right)\right\} \\ &\times \exp\left\{\frac{\left(\sigma_b^2\sum_{k=1}^{N}x_k+\sigma_w^2\mu_b\right)^2}{2\sigma_w^2\sigma_b^2\left(N\sigma_b^2+\sigma_w^2\right)}\right\} \end{aligned} \tag{3.13}$$

For proof of this see §C.1.

The background marginalizing of equation (3.13) has the expected consequence of coupling pixels across different images that observe the same background location. This can be easily seen by noticing that the product inside the integral does not distinguish which image the observations come from, simply that they belong to the same background pixel.

Following equation (3.13), the image likelihood can be expressed as:

$$p(\mathbf{I}|\mathbf{G},\mathbf{A},\mathbf{K},\mathbf{T}) = \prod_{r\in\mathbf{B}}\varphi\left(\mathbf{x}_r;\mu_b,\sigma_b^2,\sigma_w^2\right)\prod_{c=1}^{N_c}\prod_{n=1}^{N_I}\prod_{k\in\mathcal{V}_c^n}p(I_k^{n,c}|\mathbf{G},\mathbf{A},K_c,T_c) \tag{3.14}$$

where x_r are the image values that correspond to background pixel r across all the images.

Equation (3.14) has two components, the first couples the background across the images and computes the marginalization. The second, computes the likelihood of a given image pixel under the current camera and world parameters.

■ 3.2.3 GPS observation model

The GPS observation model is a simple additive Gaussian model, $Z_n^c = T_c + W$ where $W \sim \mathcal{N}(w;0,\sigma_w^2)$. This leads a Gaussian likelihood,

$$Z_n^c|T_c \sim \mathcal{N}(z;T_c,\sigma_w^2). \tag{3.15}$$

■ 3.3 Prior Probability Models

In Bayesian modeling, information that is known a priori regarding a parameter of interest is encoded via what is referred to as the prior probabilistic model. These models are typically tailor-made for applications and highlight some desired characteristics by placing higher or lower probability on certain regions of the parameter space. In this section we discuss the prior models used in this work. We further note that in our case,

the posterior model is dominated by the likelihood probability in that the influence of the prior model quickly diminishes.

3.3.1 Appearance Model

As we will discuss later, a natural prior probability model to use in the appearance model is a Normal distribution. The use of this model will lead to a close form inference computation, greatly reducing computational cost (c.f. §3.4) .

3.3.2 Geometry Model

The geometry prior model is not as easily determined as the appearance prior model. In order to select a model to use we must carefully consider the type of behavior that we want to encourage (or discourage) in the model. Furthermore, we should choose prior models that reflect the geometrical properties present in man-made urban environments.

We suggest two such prior geometric models. The first is a prior model on the normal of a triangle. The motivation behind this model is that in man-made environments, surfaces tend to be either horizontal or vertical. In aerial images, the majority of surfaces encountered are either ground or rooftops, or building sides. Thus placing a prior model that encourages horizontal or vertical directions seems to be in-line with both our modeling goals and the underlying world geometry. A third order symmetric Dirichlet distribution with small concentration parameter can achieve this purpose.

The second model is related to the size of each triangle rather than the orientation. We would like world primitives to have roughly the same size, this has two key advantages, first: it discourages triangles from shrinking or growing with out bounds; second, it allows the user to pick texture sizes independent of triangle size variability. Mathematically, we can place a normal distribution on the triangle area or perimeter, where the mean is the desired property value and the variance controls how much deviation we would allow.

3.3.3 Extrinsic Parameter

As we mentioned in the model discussion, T_c is the trajectory of the c^{th} camera. That is, we are not only modeling the camera position and orientation for a given image, but the joint position and orientation for a given sequence of images. To this end we use a Gaussian Process Prior (GP prior) over the camera extrinsic parameters. The Gaussian Process prior incorporates the knowledge that aerial images are typically obtained in a regularly spaced temporal sequence, and that the camera extrinsic parameters vary smoothly across the images. This is typically the case when a single moving platform collects images.

Mathematically, we can motivate the use of the prior model as follows in one dimension. Let T_n be the camera parameters of the n^{th} image, and $C \sim \mathcal{GP}(0, \Sigma)$ be Gaussian Process from which T_n is an observation. That is, $T_n = C + W$, where $W \sim \mathcal{N}(w; 0, \sigma^2 I)$. It follows that, $T_n|C \sim \mathcal{N}(C, \sigma^2 I)$, where I is the identity matrix.

If we marginalize over C, this leads to $T_n \sim \mathcal{N}(0, \Sigma + \sigma^2 I)$. This can be interpreted as T_n following a Gaussian Process with covariance kernel that is the sum of the Gaussian Process prior kernel, Σ, and a constant term, $\sigma^2 I$. Letting the kernel be the squared exponential (eq. 2.20), the kernel of T_n becomes:

$$k(x, x') = \upsilon^2 \exp\left(-\frac{(x - x')^2}{2\ell^2}\right) + \sigma^2 \delta_{x=x'} \tag{3.16}$$

this kernel is a small deviation from the one presented in §2.9.

As we will see shortly, it will be useful to consider looking at the distribution of $T_n | T_{\setminus n}$, this can be interpreted as predicting a new camera parameter given $N-1$ other parameters from a Gaussian process, or as obtaining the conditional of a multivariate Gaussian with N dimensions. Let us take the second point of view for simplicity. Then if $T \sim \mathcal{N}(0, \Sigma)$, we can obtain $T_n | T_{\setminus n} \sim \mathcal{N}(\mu_n, \Sigma_n)$ via Shur's complement of T, such that

$$\mu_n = \mathbf{x}^\top C^{-1} \mathbf{x} \tag{3.17}$$

and

$$\Sigma_n = k - \mathbf{x}^\top C^{-1} \mathbf{x} \tag{3.18}$$

where $\mathbf{x}$ is the n^{th} column of Σ without the n^{th} entry, C is the submatrix obtained by removing the n^{th} column and row from Σ, and k is the (n, n) entry of Σ.

We note that while we found the GP prior useful in our implementation it can be easily replaced with a more general (non-informative) prior model, such as uniform. This would be needed if the image order is not known apriori or if a small UAV was used as an imaging device (the flight path of small UAVs are highly susceptible to changes in wind direction and hence not smooth).

Throughout this work we assume that images collected by the same camera are obtained at a regular time interval, i.e. $x_j - x_i \in \mathbb{Z}, \forall (i, j) \in N_i$ (for use in equation (3.16)). This constant frame rate assumption is enforced to simplify the bookkeeping, and work with the image position in the sequence rather than the image collection time.

■ 3.3.4 Intrinsic Parameter

We opted to use a uniform prior model over the camera focal length for simplicity of computation. While one could exploit the knowledge that aerial images generally have longer focal lengths, the complications of incorporating such a model are unlikely to impact algorithmic performance in our case. An example of such a prior model might be a Gamma distribution with a suitable shape and scale parameter.

■ 3.4 Inference

This section describes the inference algorithms used. For simplicity of exposition we discuss the case where only one camera is used, ($N_c = 1$), and drop the subscript. As we will see, computational efficiency is the main guiding factor in what follows.

3.4.1 Appearance

In the model presented inference over appearance would take the form

$$p(\mathbf{A}|\mathbf{I},\mathbf{G},K,T) \propto \prod_{i=1}^{N_I} p(I_i|\mathbf{G},\mathbf{A},K,T) \prod_{k=1}^{N_p} p(A_k) \tag{3.19}$$

Using the observation model described in section 3.2.2 implies that the first term of the rhs of equation (3.19) is normally distributed. If the prior probability model on A_k were also normal, we could solve equation (3.19) in closed form for observed pixels. We note that ultimately A_k are pixels on an image with support $[0, 255]$ for 32 bit images, a normal distribution has infinite support; this implies we are placing probability mass on non-realizable values. We emphasize that this amount is negligible for a suitable set of means and variances, $\mu = 128$ and $\sigma = 10$ for example, furthermore we assume that the underlying surfaces are continuous, allowing any resolution and we only observe integer values due to physical limitations in the imaging device.

Since both terms in the rhs of equation (3.19) are Gaussian, the lhs would also be a Gaussian, that is

$$p(\mathbf{A}|\mathbf{I},\mathbf{G},K,T) = \mathcal{N}(a;\hat{\mu},\hat{\sigma}^2) \tag{3.20}$$

where

$$\hat{\mu} = \frac{\mu\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 z_i}{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2} \tag{3.21}$$

$$\hat{\sigma} = \frac{\dot{r}\sigma}{\sqrt{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2}}, \tag{3.22}$$

$\dot{r} = \prod_{j=0}^{n-1} r_j$, and $\dot{r}_{\setminus i} = \prod_{j=0, j\neq i}^{n-1} r_j$. For proof see appendix B.

Equations (3.21) and (3.22) provide the update for each pixel of the appearance images. Note that while the form of eq. (3.21) and (3.22) presented here are general, they can easily be simplified to the case where image noise is constant by letting $\dot{r} = r^n$, and $\dot{r}_{\setminus i}^2 = r^{n-1}$, and to the single observation case, by removing the summations (for proof of this see B.4).

3.4.2 Camera Parameters

Inference over camera parameters can be decomposed as inference over intrinsic and extrinsic parameters respectively.

Intrinsic Parameters

The joint model of equation (3.1) leads to the following intrinsic inference equation:

$$p(K|\mathbf{I},\mathbf{G},\mathbf{A},T) \propto p(K) \prod_{i=1}^{N_I} p(I_i|\mathbf{G},\mathbf{A},K,T) \tag{3.23}$$

Unlike the appearance estimation we cannot solve (3.23) in closed form due to the intricate dependence of the image likelihood in the intrinsic parameters. Recall that the likelihood dependence on K of equation (3.14) is hidden into the computation of A_{m^*} and (u', v'), as well as the assignment of pixels to the background. This computation requires a nonlinear projective transformation from 3D world coordinates to 2D image coordinates. Instead we can either optimize equation (3.23) to obtain a MAP estimate or use an MCMC methods to sample from the full distribution as an approximation. This choice is highly dependent on our modeling goals. For computational reason, we focus on optimization methods (e.g. MAP estimates).

Regardless of optimization or sampling, one still needs to evaluate equation (3.23). In our implementation we rely on the OpenGL pipeline to generate virtual images with parameters K and T and scene $\mathbf{G}$ and $\mathbf{A}$ to evaluate the likelihood as discussed in section 3.2.2, implementation details will be discussed in chapter 4.

Extrinsic Parameters

The joint model leads to the following inference expression over extrinsic parameters for the n^{th} image:

$$p(T_n|\mathbf{G}, \mathbf{A}, I_n, K, \mathbf{T}_{\setminus n}, Z_n) \propto p(I_n|\mathbf{G}, \mathbf{A}, T_n, K)p(Z_n|T_n)p(T_n|\mathbf{T}_{\setminus n}) \tag{3.24}$$

Equation (3.24) contains not only the image likelihood, and the GPS likelihood, but also the likelihood of the current camera parameters conditioned on all the other configurations. This term, is a result of the Gaussian Process prior placed on T, and can be evaluate as discussed in section 3.3.3.

We note that $p(T_n|\mathbf{T}_{\setminus n})$ encourages extrinsic parameters configuration that fit well with the sequence of camera parameters not just the current observation. Furthermore, the hyperparameters of the GP prior enforce a set of characteristics that can be learned from the data as discussed in section 2.9. These characteristics are data specific; in our application it typically translates to spatial smoothness over the image sequence. As with the intrinsic parameters, extrinsic parameters can be optimized or sampled. For computational reasons we will focus on optimization methods.

■ 3.4.3 Geometry

Inference over geometric variables in the model presented takes the form:

$$p(\mathbf{G}|\mathbf{L}, \mathbf{I}, \mathbf{A}, \mathbf{T}, \mathbf{K}, \mathbf{B}) \propto \prod_{i=1}^{N_I} p(I_i|\mathbf{G}, \mathbf{A}, \mathbf{B}, K, T) \prod_{l=1}^{N_l} p(L_l|\mathbf{G}) \prod_{k=1}^{N_p} p(G_k) \tag{3.25}$$

Similar to camera pose inference, the complicated form of the image likelihood precludes exact inference procedures for equation (3.25) in close form. Furthermore, the LiDAR likelihood is computationally intensive since the data association problem discussed in section 3.2.1 requires doing a distance minimization for each LiDAR observation over the entire collection of world primitives. As before we will optimize for

configuration of world geometry that maximizes equation (3.25). As a consequence of this, the results of geometry estimation will be initialization dependent. Implementation details that attempt to reduce the computational complexity of this problem will be discussed in chapter 4.

Chapter 4

Implementation

The idea of computational efficiency was presented in chapter 3 when discussing inference techniques for the proposed model. And, while efficient algorithms were presented, the crucial implementation details needed to achieved the desired level of efficiency were not discussed. As such, the goal of this chapter is to present and examine those implementation details.

This chapter will first analyze the appearance computation; once this key component is well understood, we will examine the image and LiDAR likelihood equations and discuss efficient procedures to evaluate them. Following the likelihood computation we will discuss a parallel implementation of the geometry updates. The chapter will culminate with the introduction of texture atlases to speed-up the rendering of large and complex scenes. Throughout this chapter we will provide pseudo-code algorithms for CPU and GPU implementations.

4.1 Appearance Computation

In the model presented in chapter 3, c.f. 3.1, A_m the appearance of the m^{th} primitive is an image of a user specified size, $T \times T$. We are interested in learning the values for each pixel according to the update equations (3.21) and (3.22). For a set of camera parameters and world structure, the appearance equations require collecting all of the observation pixels (image pixels) that correspond to a given pixel in the primitive appearance and weighting them appropriately to produce the desired texture. We note that only visible triangles need to go through this procedure, since we do not have any information for occluded or out of view primitives.

The key piece of information needed for the computation above, is knowing the forward mapping from image pixels to triangle appearance pixel. This mapping can be decomposed into two parts; first, we need to associate an observed image pixel to a triangle, and secondly, we need to determine which pixel in the triangle's appearance map generated the observation. One can certainly think of many ways of obtaining the information above, from explicitly searching over all appearance and observation pixels (reverse mapping), to projecting each image onto the world and search for appearance matches (hybrid forward and reverse mapping).

For our purposes we use a forward mapping from image pixels to world appearance as

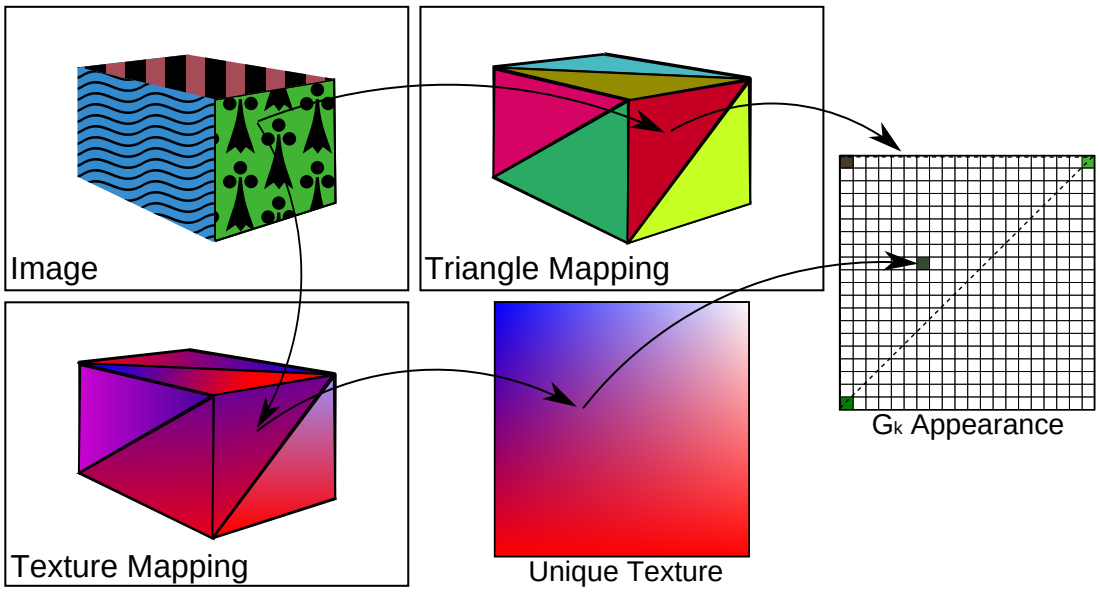


Figure 4.1: Appearance computation procedure. First determine which primitive generated the observation using a triangle mapping, secondly using the texture mapping determine which pixel generated the observation.

described earlier. The forward mapping has several advantages. First, the complexity is bounded by the number of images and the size of each image; we point out that the reverse mapping depends on the number of world triangles and the size of the appearance image, and thus scales poorly to scenes with a large number of primitives[1]. Secondly, using the OpenGL rendering pipeline the data association can be easily and quickly determined as demonstrated next.

The mapping of image observation to triangle appearance can been seen in Figure 4.1. The mapping can be explained in two parts. The first part, shown in the top row of the figure, maps every observation pixel to a triangle (or background) via a unique color mapping of each primitive. This data association problem can be performed efficiently by using the GPU rendering pipeline to encode the index of the primitive as an RGB triple. As a consequence, observing a particular RGB triplet correspond to observing a particular primitive. In this work, we used the mapping:

$$i = \frac{r + 256g + 256^2 b}{\delta} \tag{4.1}$$

[1]Forward mapping has the same dependency. However, the update computation is only performed over primitives which have observations associated to them.

and

$$\begin{aligned} r &= \delta * i \ \& \ 0x0000FF, \\ g &= (\delta * i \ \& \ 0x00FF00) >> 8, \\ b &= (\delta * i \ \& \ 0xFF0000) >> 16, \end{aligned} \tag{4.2}$$

where $\delta = \frac{256^3}{N_p}$, & is the logical and operator, and $>>$ is the right shift operator.

Furthermore, we use the alpha channel to differentiate between background and world triangle, (background is given an alpha value of 0, while primitives are set to the max allowed, 255), this allows for up to 256^3 unique primitives. We note that other encoding schemes are possible, for example background can always be encoded as black and allow the use of the alpha channel for additional primitives (allowing $256^4 - 1$ unique primitives). We found our simple scheme to be sufficient.

The second part of the mapping involves determining which pixel in the appearance map generated the observation. This mapping can be seen in the bottom row of Figure 4.1. For this mapping we create a unique texture, the same size as the appearance map where each RGB triple encode the pixel location, similar to the mapping for primitive number but using $\delta = 1$. For textures smaller than 256×256, this mapping can be achieved with a single render that encodes both x and y location (red channel would be x and blue channel would be y). If the texture is larger, separate renders is required for each dimension.

We can put these two pieces together and obtain the appearance estimation method shown in Algorithm 4.1 for a CPU computation, or Algorithm 4.2 for a GPU computation. The main distinction between CPU and GPU implementation is that on CPU all the rendering is done ahead of time and stored, then the textures are computed; in the GPU implementation due to memory constraints, renders are done one at a time and aggregated to the texture, once all images are accounted for the texture is normalized. Details for the GPU kernels were not provided since their function is the same as the pixel loop in the CPU algorithm.

Intuitively the forward mapping makes sense since it only consider observation pixels, i.e. it assigns observation pixels to appearance pixels that are visible and does not try to fill in unobserved pixels. Stated differently, it produces holes in the appearance maps. These holes are not visible when the textured world is seen from an observation view, but when new views are considered, pixels that were not previously seen, now become visible.

A possible solution for minimizing this rendering artifact is to allow each observation to influence its neighbors as well. We can think of the neighbor mapping as letting an observation pixel map to a region in the appearance image rather than a single pixel. Empirically we have found that using neighbor-hole-filling on the four connected neighbors works well, and does not over-smooth appearances. We further note, that selecting a smaller appearance image per primitive can also reduce the number of empty pixels.

Algorithm 4.1 Appearance Estimation pseudo-code - CPU

1: Compute unique world projection for all images I_{tri}^{map} (associates each pixel with a triangle or background)
2: Compute unique texture projection into all images I_{tex}^{map} (associates each pixel with a texture coordinate)
3: **for** k = 1:1:N_p **do** ▷ compute image weights
4: $w_k = 1$
5: Compute plane normal n_k
6: **for** i = 0:1:numImages **do**
7: Compute image view direction v_i
8: Compute triangle-image weight $w_k^i = \frac{1}{|v_i^\top n_k|}$
9: Compute triangle weight $w_k = w_k * w_k^i$
10: **end for**
11: **end for**
12: Allocate memory for A_k^{tex} and A_k^{weight} (the texture values and weights), set to zero
13: **for** i = 1:1:N_i **do** ▷ Loop over images
14: **for** p=0:1:size(I_i) **do** ▷ Loop over pixels for i^{th} image
15: **if** $I_i(p)$ ==background **then**
16: Continue pixel loop
17: **end if**
18: Compute triangle source of $I_i(p)$, by looking at $I_{tri}^{map}(p)$, call it T
19: Compute texture location of $I_i(p)$, by looking at $I_{tex}^{map}(p)$, call it P
20: Compute triangle-image weight $w_T^i = \frac{1}{|v_i^\top n_T|}$
21: Compute iteration weight $w = \frac{w_T}{w_T^i}$
22: Set $A_T^{tex}(p) = A_T^{tex}(p) + wI_i(p)$
23: Set $A_T^{weight}(p) = A_T^{weight}(p) + w$
24: *Optional* Fill in neighbors of p, as in lines 22 and 23.
25: **end for**
26: **end for**
27: Compute and save weighted texture for A_k by A_k^{tex}/A_k^{weight}
28: *Optional* Fill in empty pixels in G_k, with four neighbor average.
29: De-allocate memory for A_k^{tex} and A_k^{weight}

In terms of computational complexity both CPU and GPU algorithms are bounded by the number of images and the size of each image. The main limitation of the CPU algorithm is the data transfer between main memory and device memory, since it requires transferring two images from device to main memory for each observation, and then transferring all the appearance textures from main memory to device memory. The GPU implementation does not suffer from this problem since the texture maps are never transferred, i.e. they remain in device memory and are accessed via the tex-

Algorithm 4.2 Appearance Estimation pseudo-code - GPU

1: Compute Weights as before.
2: Create Triangle Map and Texture Map framebuffers (assign 2 textures pointers, depth and color to each)
3: Allocate device memory for A^{tex},A^{weight}, I^{map}_{tri}, and I^{map}_{tex}.
4: **for** i = 1:1:N_i **do** ▷ Loop over images
5: Bind Triangle Map framebuffer, render triangle map as viewed from image i.
6: Bind Texture Map framebuffer, render texture map as viewed from image i.
7: Call texture Accumulation Kernel. ▷ Same as pixel loop in algorithm 4.1
8: **end for**
9: Call texture normalize kernel. ▷ Computation of line 27 in algorithm 4.1
10: De-allocate memory for A^{tex}_k and A^{weight}_k

ture accumulation kernel; similarly, the texture data always remains in device memory. Hence the main limitation of the GPU computation is the render itself.

Table 4.1 shows the computation time and speed up factors for several datasets. We can see from the table that at worse the GPU implementation is three times faster than CPU.

Data	**Toy Problem**	**Lubbock**		**CLIF (subset)**	
# Triangles	24	80,000	80,000	227,000	227,000
# Images	15	3	3	24	24
Texture /Atlas Size	256/1024	8/512	16/1024	8/1024	16/1024
CPU Time (s)	1.747	1.361	2.435	8.145	11.304
GPU Time (s)	0.564	0.201	0.193	1.826	1.9404
Speed up	3.1	6.7	12.6	4.5	5.8

Table 4.1: CPU/GPU speedup (times are the average of 5 estimations), all test done on an NVIDIA GTX-580 graphics card

■ 4.2 Image Likelihood Computation

Estimation of camera pose and world geometry rely heavily on the evaluation of the image likelihood. Thus fast and efficient computation of this likelihood is crucial. In this section we describe how we implement the image likelihood and how we leverage the power of graphics hardware to obtain high performance.

The image likelihood as given in section 3.2.2 is:

$$p(\mathbf{I}|\mathbf{G},\mathbf{A},\mathbf{K},\mathbf{T}) = \prod_{r\in\mathbf{B}} \varphi\left(\mathbf{x}_r;\mu_b,\sigma_b^2,\sigma_w^2\right) \prod_{c=1}^{N_c}\prod_{n=1}^{N_I}\prod_{k\in\mathcal{V}_c^n} p(I_k^{n,c}|\mathbf{G},\mathbf{A},K_c,T_c), \tag{4.3}$$

reproduced here for convenience, where the first term in the rhs performs the background marginalization and the second term computes the observation likelihood. As discussed previously, the background marginalization couples all the observation pixels that belong to a given background pixel as given by equation (3.13). This computation involves identifying which observation pixels should be grouped together, and computing some statistics over them.

Recall, that the background in the model is a plane at ground level. For computational simplicity of the background data association we can attach an image of some user specified size, say $B_s \times B_s$ to this plane. With this image in place, we can perform the observation data association in a similar manner as the texture location was done in the previous section. That is, we can let the background image have an alpha value of zeros as previously mentioned, but also encode each background pixel's RGB triple with its location using equations (4.1) and (4.2), with $\delta = 1$.

The evaluation of the data term in the rhs of equation (4.3) requires identifying the association between an observation pixel and a world appearance pixel, then evaluating a Gaussian distribution. This computation can be done efficiently in the OpenGL pipeline with two renders, the first would map observation pixels to triangles, and the second would map appearance pixels to observation pixels. With the information provided in these two renders we can turn equation (4.3) into:

$$p(I_n|\mathbf{G}, \mathbf{A}, K_c, T_c) = \prod_{k \in \mathcal{S}_n} \mathcal{N}(i_k - \hat{i}_k; 0, r^2_{m^*}). \tag{4.4}$$

where $\hat{i}$ is the textured mapped world as viewed from image n.

The joint evaluation of the background and image can be seen in algorithm 4.3. Several observations can be made about algorithm 4.3. First, we point out that the background pixel mapping can be done in either the texture mapping or the triangle mapping render, since it will not interfere with either evaluation. Secondly, notice that the purpose of the triangle mapping render is to identify the noise level that should be associated with the Gaussian evaluation, since we are not interested in knowing which primitive generated the observation, only its appearance value which we obtain from the texture mapping.

The above discussion suggests that we can improve performance by rendering only once, for the texture mapping, and not for the triangle mapping. Mathematically, this change implies changing the noise characteristics for the image observation model from image and primitive dependent to a fixed value. As a preview of results to come, we highlight that this change is possible.

The pseudo-code version of the image likelihood computation in GPU is identical that of CPU. We note that while the pseudo-code is identical some implementation details do change, for example we must be careful about memory usage since it is limited in graphics hardware. Furthermore, some operations such as variable increments must be dealt with great care, since in a multi-thread environment, multiple threads trying to write to the same variable at the same time will yield undefined results. For this

Algorithm 4.3 Image Likelihood Evaluation pseudo-code - CPU/GPU

Compute image weights for all triangles in all images, call it w_n^m, where n indexes image, and m indexes planes
Set $ll = 0$;
Create array of structure bgTrack, size B_s^2, with elements: count, x, xsquared (initialize all elements to zero).
for i = 0:1:N_i **do**
    Render texture map world as viewed from image i, transfer framebuffer image to main memory, call it $\hat{I}$.
    Render Triangle map as viewed from image i, transfer framebuffer image to main memory, call it I_{tri}. ▷ note that $\hat{I}, I_{tri}$, and I_i have the same size
    for p=0:1:size(I_i) **do**
        if $I_{tri}(p) \neq$ background **then**
            Map I_{tri} to plane number using eq. (4.1), call it m.
            Set $\sigma = w_i^m$
            Increment log likelihood $ll = ll + -\frac{(I_i(p)-\hat{I}(p))^2}{2\sigma^2}$
        else
            Map I_{tri} to background pixel, call it b
            Increment counts: bgTrack[b].count +=1
            Increment observations: bgTrack[b].x += $I_i(p)$.
            Increment squared observations: bgTrack[b].xsquared += $I_i^2(p)$
        end if
    end for
end for
for i = 0:1:B_s^2 **do** ▷ computes background contribution
    if bgTrac[i].count>0 **then**
        Compute background marginalization of equation (3.13) using the elements of bgTrack[i], call it φ
        Increment log likelihood $ll = ll + \varphi$.
    end if
end for

particular issue, shared memory can be instantiated and allow thread block elements to sync their values, then add the local value to the global one, typically using atomic operators.

Table 4.2 shows the computation time and speed up factors for the image likelihood computation for several datasets. We can see from the table that at worse the GPU implementation is three times faster than CPU.

Data	Toy Problem	Lubbock	CLIF (subset)	CLIF (all)
# Triangles	24	80,000	227,000	227,000
# Images	15	3	24	50
Image Size	1200x900	1336x891	822x1326	822x1326
Texture/Atlas Size	256/1024	8/512	16/1024	16/1024
CPU Time (s)	2.252	0.295	2.865	5.858
GPU Time (s)	0.754	0.066	0.424	0.873
Speed up	2.99	4.47	6.76	6.71

Table 4.2: CPU/GPU Image Likelihood evaluation speed-up (times are the average of 5 estimations), all test done on an NVIDIA GTX-580 graphics card

■ 4.3 LiDAR Likelihood Computation

As discussed in §3.2.1 the LiDAR observation model contains a data association problem[2], for which the model selects the closest world primitive as the one that generated the LiDAR measurement. Mathematically, the association can be solved by:

$$\hat{G}_{k(n)} = \arg\min_{G \in \mathbf{G}} d^2(L_n, G). \tag{4.5}$$

as presented earlier. This simple equation has some computational issues associated with it since it involves for every LiDAR point searching for the closest triangle from the collection of all world triangles, i.e. $\mathrm{O}(N_i N_p)$, where N_i is typically in the orders of high hundreds of thousands and N_p in the low-mid hundreds of thousands.

Intuitively, we can reduced the number of primitives considered by equation (4.5) since it is very unlikely that a primitive very far from the measurement generated it. Again, this computation would require computing distances between the LiDAR point in question and all the primitives, the same computation we want to avoid.

One approach that we can take is to use data structure that facilitates this search, in particular space partitioning data structures [45]. By using such data structures we can quickly remove a great quantity of search primitives. Out of the data structures considered, point-based k-d trees was chosen due to its simplicity and expressibility. A K-D tree is a binary tree where the underlying k dimensional space is partitioned on just one attribute at each level (and thus attributes are being cycled through as a function of levels); at each level a test is made that determines which branch of the tree to take.

Ideally, we would like to build the tree over the set of primitives and for each input LiDAR measurement identify the closest primitive. This idea, however, is prohibitive since the set of primitives is changing (i.e. we are learning the geometry parameters)

[2]Unlike the appearance likelihood this association problem cannot be solved via renders since the LiDAR point visibility depends on viewing direction.

which would require re-learning the tree every time a primitive changes. Instead we can learn the tree over the static LiDAR observations. Once we learned the tree, we can input a primitive and obtain a list of observation close to the plane. We note that since the k-d tree implementation only works with points, and query term consist of the midpoint of the triangle and a maximum search radius. The kd-tree search returns all tree points that are within the specified radius of the query. We select this radius to be about twice the distance from midpoint to endpoint of the triangles. While this might seem very broad, we found that this works well in practice.

This search allows multiple LiDAR measurements to be associated with a given plane and multiple planes to be associated with a given measurement. Furthermore, there could be planes with no LiDAR associated to them, however, the converse is not possible; every LiDAR measurement must be associated with a plane. To ensure this is the case, we must identify non-associated points and attribute a plane via exhaustive search. For observations with multiple planes, the closest plane is selected.

Figure 4.2 shows four association examples, where we have color-coded the LiDAR observations according to the distance from observation to plane. Figures 4.2a and 4.2b clearly shows the search radius allowed when using the k-d tree. We can see the association interaction between the selected plane and the other world planes in Figures 4.2c and 4.2d, where the cyan and yellow points, can be associated with the selected plane however this is unlikely since other planes are closer.

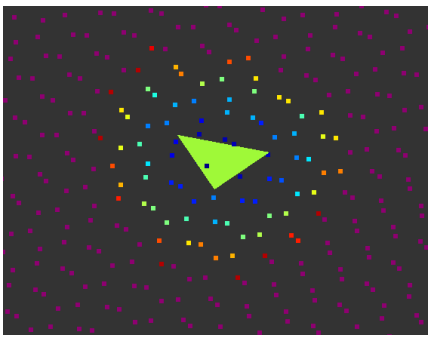
In terms of computation, building a tree for the CLIF Stadium dataset with 705,798 LiDAR points and 227,000 triangles takes 900 milliseconds. The k-d search takes on average 0.98 milliseconds, with a minimum of 0.5ms and maximum of 3ms. The k-d search is able to associate 99.99% of all LiDAR points to primitives; 37 points were associated via exhaustive search, each taking on average 90ms. The above times suggest close to two order of magnitude speedup by introducing the k-d tree search.

■ 4.4 Geometry Computation

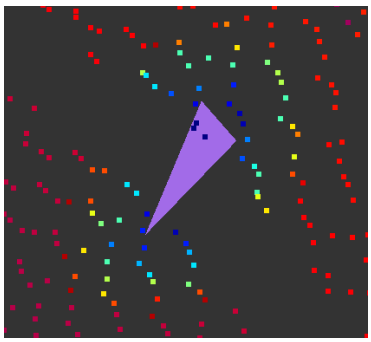
In this section we discuss the implementation details for inferring world geometry parameters. Namely, we are concerned with evaluating equation (3.25), replicated here for primitive G_k for convenience,

$$p(G_k|\mathbf{L},\mathbf{I},\mathbf{A},\mathbf{T},\mathbf{K},\mathbf{B},\mathbf{G}_{\setminus k}) \propto p(G_k)\prod_{i=1}^{N_I} p(I_i|\mathbf{G},\mathbf{A},\mathbf{B},K,T)\prod_{l\in\mathcal{A}_k} p(L_l|G_k) \tag{4.6}$$

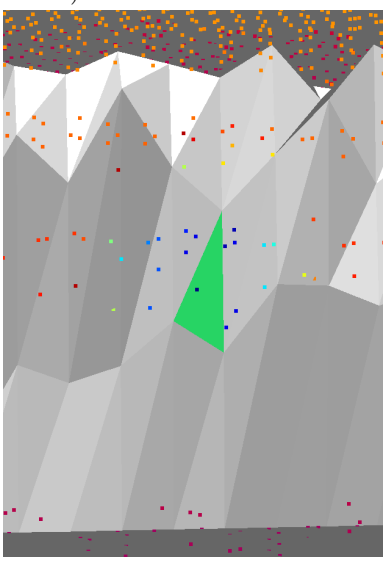
where $\mathcal{A}_k$ is the set of LiDAR measurements associated with G_k. As we saw in the previous sections, the evaluation of the likelihoods in this equation can be computationally intensive; while the savings established in sections 4.2 and 4.3 help reduce the overall compute time of one primitive, the sheer number of primitives for which equation (4.6) needs to be evaluated quickly increases the computation time. This is particularly true for sampling or optimization where equation (4.6) becomes the target distribution



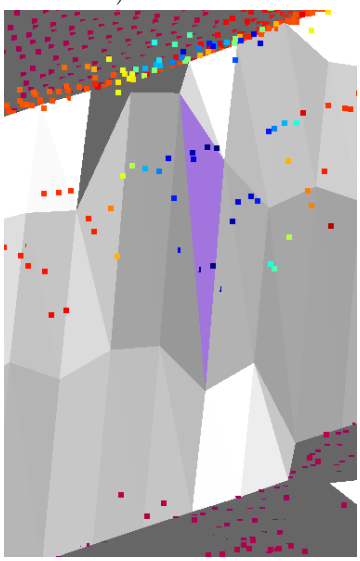
(a) Association (points colored by distance)



(b) Association (points colored distance)



(c) Association (other planes shown in gray)



(d) Association (other planes shown in gray)

Figure 4.2: LiDAR and world primitive association. (Color coding: planes in question color-coded, points associated with planes color-coded according to distance, other planes shown in gray, other points shown in orange-magenta)

or the objective function, and thus requires being evaluated hundreds or thousands of times per primitives.

Several observations can be made from equation (4.6). First, we can decouple primitive G_k from all other primitives in the LiDAR likelihood if we do not change the associations while evaluating the expression, i.e. fix $\mathcal{A}_k$ for all evaluations of primitive G_k. The association would be updated after the current primitive or all primitives have been updated. Second, when evaluating the image likelihood as a function of G_k only a very small subset of pixels change. These two observations along with the desired to reduce computation time motivate evaluations of equation (4.6) for multiple primitives at a time.

The evaluation of equation (4.6) for multiple primitives at a time has the advantage of reducing the number of draws required to compute the image likelihood, which is the computational bottleneck as discussed previously. However, in order to take advantage of this savings the sampling or optimization routines used must propose independent perturbations for triangles in question; these perturbations must be combined to produce a single render and the likelihood for each primitive must be computed separately, then the sampling/optimization scheme must make independent decisions based on each likelihood evaluation. Assuming that this is the case, let us investigate the necessary changes to compute image likelihoods in the multi-triangle case.

■ 4.4.1 Image Likelihood evaluation under multiple primitives

In order to extend the approach to multiple primitives we must determine a way of computing the image likelihoods for a given set of primitives. As stated in equation (4.6) the image likelihood is computed over all pixels despite not being associated with primitive G_k. However, the only change in the likelihood should come from the pixels affected by the change of G_k, since it is the only element varying. This however would not be the case when multiple primitives are considered.

To see how we can properly account for pixels associated with multiple planes let us take a closer look at the image likelihood computation. As described in section 3.2.2, the image likelihood is

$$p(\mathbf{I}|\mathbf{G},\mathbf{A},\mathbf{K},\mathbf{T}) = \prod_{r\in\mathbf{B}} \varphi\left(\mathbf{x}_r;\mu_b,\sigma_b^2,\sigma_w^2\right) \prod_{n=1}^{N_I} \prod_{k\in\mathcal{V}_n} p(I_k^n|\mathbf{G},\mathbf{A},K,T), \tag{4.7}$$

We can split set of visible pixels in the n^{th} image, $\mathcal{V}_n$, into those generated by primitives which we are optimizing over, $\mathcal{P}$, and those that are not, then the image likelihood takes the form:

$$p(\mathbf{I}|\mathbf{G},\mathbf{A},\mathbf{K},\mathbf{T}) = \prod_{r\in\mathbf{B}} \varphi\left(\mathbf{x}_r;\mu_b,\sigma_b^2,\sigma_w^2\right) \prod_{n=1}^{N_I} \left[\prod_{k\in\mathcal{P}} p(I_k^n|\mathbf{G},\mathbf{A},K,T) \prod_{k\in\mathcal{V}_n\setminus\mathcal{P}} p(I_k^n|\mathbf{G},\mathbf{A},K,T)\right] \tag{4.8}$$

We are interested in the set of pixels generated by the planes we are optimizing over, $\mathcal{P}$, and the interactions between this set and the background and other pixels. To see this last part, consider the effect of shrinking a plane as part of an optimization or sampling algorithm, if we were to only compute the likelihood over pixels generated by that plane for this step only, the number of pixels over which this calculation takes place would have strictly decreased from the previous evaluation, yielding an artificially lower likelihood. Expressed differently, a degenerate solution would be to shrink planes so that they are not visible. In order to avoid degeneracies as the one pointed above, we must carefully account for pixel changes when evaluating likelihoods.

To better understand the bookkeeping necessary to achieve the correct calculation, let us concentrate on calculating the partial likelihood for one triangle. We begin by noticing that between any two likelihood evaluations, there are two possible changes: first, the set of pixels over which the evaluation takes place can change; and second, the value of the underlying rendered image can change. These two changes can be visualized for a cartoon region and values in Figures 4.3a and 4.3b as the change in the evaluation mask, and the color change in the rendered image. Let us denote the value of the initial rendered image by $f_0(\cdot)$, and the value of the new image by $f_1(\cdot)$; furthermore, lets denote the evaluation masks by M_0 and M_1 respectively.

With this notation we can calculate the change between the image likelihood as:

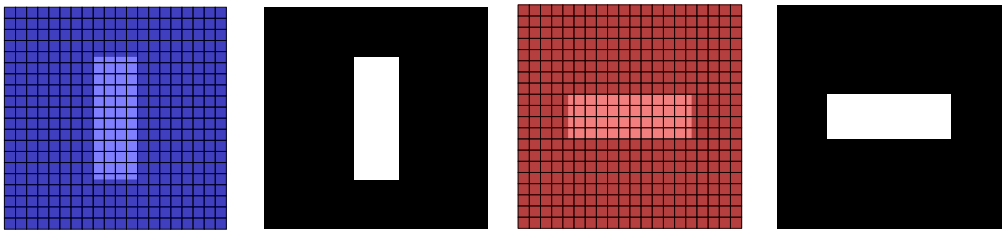
$$\delta_{01}(M_0, M_1) = f_1(M_0 \cup M_1) - f_0(M_0 \cup M_1). \tag{4.9}$$

We note that $\delta_{01}(M_0, M_1)$, properly accounts for changes in the evaluation mask and the function values. If we wanted to obtain the new absolute likelihood, as apposed to the change, we simply add the initial total likelihood and the change,

$$\begin{aligned} f_1(M) &= f_0(M) + \delta_{01}(M_0, M_1) \\ &= f_0(M) + f_1(M_0 \cup M_1) - f_0(M_0 \cup M_1) \end{aligned} \tag{4.10}$$

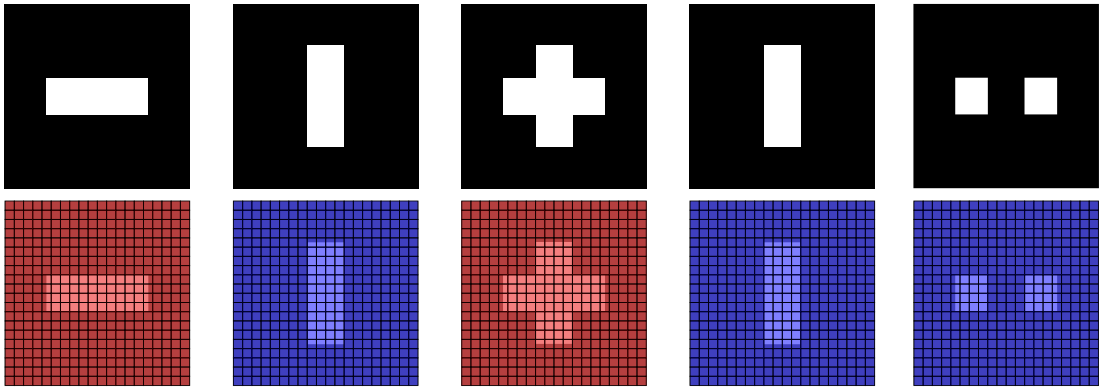
To better understand, equations (4.9) and (4.10), let us analyze the notional computation depicted in figure 4.3. Parts (a) and (b) of Figure 4.3 show the initial and subsequent computation, where we denote changes in the image values by the change in color from blue to red, and the change in pixels by the changes in the evaluation masks (white denotes pixels to include, and black pixels to ignore). Figure 4.3c shows the necessary steps in terms of masks and functions to evaluate to compute the relative image likelihood. We can easily see special cases of this general computation from both figure 4.3 and equation (4.9). For example, if the evaluation masks remain the same, $M_0 = M_1$, then we are simply taking the difference between the image values, that presumably have not changed, leading to a delta of zero and no change in the likelihood function.

Now that we understand how to compute the incremental likelihood for a single triangle, the extension to the multi-triangle case follows trivially, we simply let each triangle have its own evaluation mask. With this minor adjustment we can use all of the previous derivations.



(a) Initial render values and evaluation mask (M_0)

(b) New rendered values and evaluation mask (M_1)



(c) Relative Image Likelihood computation

Figure 4.3: Relative image likelihood computation. Suppose changing a primitive's parameter produces the images shown in in blue and red (with corresponding mask) in (a) and (b). (c) shows how to combine the renders to produce the desired likelihood computation.

Computationally we can evaluate the multi-plane image likelihood as shown in algorithms 4.4 and 4.5. The computation is divided in an initialization step and a subsequent computation since the initial step is much simpler; it only requires being able to evaluate current values. On the other hand, subsequent evaluations are slightly more complicated since for every pixel that we encounter we have to not only compute its current triangle association but its past association as well.

Several observations can be made from the algorithms, first we notice that memory usage has dramatically increased, (more than doubled if using 4 byte floats for ll), since we must now maintain all the previous pixel likelihood values for each image, as well as the evaluation mask. This large memory consumption is prohibitive for graphic hardware, where memory is limited. Furthermore, the use of structures in graphics hardware is cumbersome, since memory management is difficult. As a result, graphics implementation will require unrolling the structures into one continuous array, creating extra bookkeeping details or creating extra kernels to handle memory operations.

Algorithm 4.4 Multi-Triangle Image Likelihood evaluation - Initialization - CPU/GPU

1: Select N planes compute the likelihood over. ($N < 256$)
2: Initialize structure bgTrack (see algorithm 4.3), set all elements to zeros.
3: Initialize an array of N structure planeLL, with members imgLL, imgPxCount, bgPixelsToAdd (set all to zeros) ▷ bgPixelsToAdd variable in size, use containers of variable size (e.g. std::vector)
4: Initialize array of N_i structure imEvalMask, with members mask, ll, and llpixelToAdd, backgroundIndex. Set mask and ll to the size of the corresponding i^{th} image. ▷ as above: llpixelToAdd, backgroundIndex are variable in size, use appropriate container.
5: **for** i = 1:1:N_i **do** ▷ image loop
6: **for** $p \in \mathcal{S}_i$ **do** ▷ pixel loop
7: **if** p visible **then**
8: Compute pixel log likelihood, call it ll.
9: Set imEvalMask[i].ll[p]=ll
10: Identify world triangle associated with current pixel, call it T.
11: **if** optimizing over T **then**
12: imEvalMask[i].mask[p]=T
13: planeLL[k].imgLL += ll; ▷ the k^{th} entry is associated with triangle T
14: **end if**
15: **else**
16: Add p to bgTrack ▷ see algorithm 4.3.
17: **end if**
18: **end for**
19: **end for**
20: Compute background values. ▷ see algorithm 4.3.
21: Fill background values in imEvalMask using llpixelToAdd, and backgroundIndex.
22: maintain imEvalMask and planeLL in memory.

Both algorithms call for identifying the primitives associated with each pixel. This however is computationally expensive and not necessary, since we are only interested in knowing if we are optimizing over this triangle, and its index in the planeLL stack. We can achieve this mapping by using the alpha channel of the triangle's texture, then visibility would be given by determining if alpha channel is greater than zero (previously established as background) and different than ordinary triangles, 255. The position in the planeLL structure can be determine by looking at the actual alpha value, once we ensure we are not in one of the two former cases.

The results for the multi-plane geometry evaluation will be discussed in §5.2.2; as a preview: no difference exists between single and joint evaluations (provided that primitives do not overlap). Computationally, joint geometry estimation is over two order of magnitude faster than its single counterpart.

■ 4.5 Texture Atlas

As we saw in §4.2, the ability to render a textured mapped scene is crucial for the image likelihood computation. Furthermore as hinted above the limiting factor of the GPU implementation is the speed of the rendering itself, rather than the computation. As a result, we are interested in minimizing the drawing time for the scene.

A naive rendering routine would require a texture binding for every world primitive, since each primitive has its own texture. Furthermore, the primitive's parameter (vertex coordinates, texture coordinates, normal, etc.) would be transferred from main memory to device on every draw. Both of these components are very inefficient.

In order to improve performance, we can first push all world primitive parameters to device memory, so that the communication penalty is suffered only when parameters are changed, i.e. once, rather than at every draw. Secondly, we can group multiple appearances into a texture atlas, and minimize the number of bindings required at drawing time. With these modifications, at drawing time, we simply need to bind to the texture atlas and primitive parameters, and specify how many elements to draw (corresponding to the number of primitives per atlas).

In this section we discuss the implementation details for achieving this goal.

■ 4.5.1 Static Texture Coordinates

According to the proposed model, each triangle has a canonical texture, corresponding to a right triangle of user-specified size (Figure 3.3). This modeling choice, has several consequences:

- Texture Coordinates are static.
- Computed texture is subject to different scaling for each direction.
- Rendered texture pixel need not be square.

Algorithm 4.5 Multi-Triangle Image Likelihood evaluation - Subsequent computations - CPU/GPU

Set all members of bgTrack and planeLL to zero, except imgLL.
Set llpixelToAdd and backgroundIndex to zero for all elements of imEvalMask.
for i = 1:1:N_i **do** ▷ image loop
  for $p \in \mathcal{S}_i$ **do** ▷ pixel loop
    if p visible **then** ▷ likelihood computation or background addition
      Compute pixel log likelihood, call it ll.
    else
      Add p to bgTrack ▷ see algorithm 4.3.
    end if
    Identify world triangle associated with pixel p, call it T.
    $\hat{T}$ = imEval[i].mask[p]. ▷ Identify previous triangle associated with pixel p.
    if optimizing over T or $\hat{T}$ **then** ▷ multi-plane bookkeeping
      if optimizing over T **then**
        Set index = k; ▷ where k is the position of triangle T
        Set planeLL[index].imgLL += ll
        Increment planeLL[index].imgPxCount by one.
      else if optimizing over $\hat{T}$ **then**
        if p visible **then**
          Set index = k;
          Set planeLL[index].imgLL += ll
          Increment planeLL[index].imgPxCount by one.
        else
          Set index = l; ▷ where l is the position of triangle $\hat{T}$
          Add p to planeLL[index].backgroundPxToAdd
        end if
      end if
      Subtract previous ll, planeLL[index].imgLL -=imEvalMask[i].ll[p]
    end if
    if p visible **then** ▷ update visibility and ll for next iteration
      Set imEvaMask[i].ll[p] = ll;
      Set imEvaMask[i].mask[p] = T;
    else
      Set imEvaMask[i].mask[p] = -1;
      Add p to imEvalMask[i].llPixelToadd
      Add appropriate background pixel to imEvalMask[i].backgroundIndex
    end if
  end for
end for
Compute background values. ▷ see algorithm 4.3.
Fill background values in imEvalMask using llpixelToAdd, and backgroundIndex.
Add background values in planeLL using bgPixelsToAdd.

The first of these points is essential in practice, since the texture coordinates do not need to be re-calculated when the geometry of a triangle is changed, yielding substantial computational savings; however, these savings comes at a cost - the last two points above.

A fixed texture coordinate defines a transformation of pixels from appearance map (texture image), and the world primitive (triangle being textured mapped). This transformation is implicitly done via OpenGL's texture mapping, where the forward transformation is achieved using the unique texture mapping (Figure 4.1) and the reverse transformation is achieved using the static coordinates.

We note that having appearances be right triangles has the added benefit of allowing two canonical textures to be stored using a single square image (the only type OpenGL allows). Thus providing us the ability of drawing twice as many triangles per atlas binding.

■ 4.5.2 Implementation

In this section we go through the necessary bookkeeping for creating, updating and drawing the texture atlas.

Note that there are two user parameters:

- T is the texture size (texture will be $T \times T$)
- A is the atlas size (atlas will be $A \times A$)

where $A \geq T$, and both are powers of two (A needs to be power of two to satisfy OpenGL standard for textures, T need not be a power of two, but in order to use the fast texture calculation developed earlier we need it to be a power of two as well).

The general idea it to have a texture atlas that looks like Figure 4.4, where even textures are on the top portion of the texture rectangle, and odd textures are on the bottom portion of the texture rectangle, below we describe the initialization, update and drawing procedure for the texture atlas.

Initializing

Assuming that the number of triangles is known (call it M, and triangles are index by k, where $k \in [0, M-1]$). Then the initial set up of the texture atlas can be seen in Algorithm 4.6. The algorithm can be broken down into three main parts, first allocating main memory to contain the list of triangle properties, including the atlas associated with each primitive as well as its texture neighbor. A texture neighbor, is the triangle that shares the texture square in the texture atlas, i.e. 0 and 1, or 14 and 15 in figure 4.4, this information will be needed for the texture updates later.

The second part of the algorithm computes the texture coordinate in the atlas, this computation is dependent on the triangle's ordering. The last step of the algorithm is to transfer all parameters to the graphics card.

Algorithm 4.6 Texture Atlas Initialization

1: Set the values of A and T.
2: Compute $N = \lfloor \frac{A}{T} \rfloor$, the number of textures per dimension.
3: Compute $N_a = \lceil \frac{M}{2N^2} \rceil$, the number of atlases needed to represent all the triangles.
4: Create an array of N_a atlases in device memory, each of size $A \times A$, initialize them to zero, call it $pAtlas$.
5: Create $pTex$ an array that contains the atlas index for each triangle.
6: Create $texNeigh$ an array that contains the texture neighbor (either up or down for each triangle.
7: Compute $\delta = \frac{1}{A}$, the one pixel offset in the diagonal between the top and bottom texture.
8: Compute Texture Coordinates using the counter-clockwise order shown in Figure 4.5.
9: **for** k = 0:1:M-1 **do**
10: $pTex[k] = pAtlas[k \mod 2N^2]$. ▷ assigns atlas to triangle
11: **if** $k \mod 2 == 0$ **then**
12: Compute: $x = \frac{\frac{k}{2} \mod N^2}{N}$; and $y = \frac{\frac{k}{2} \mod N^2 - x}{N}$
13: $T_1^k = (x, y + \delta)$
14: $T_2^k = (x + \frac{1}{N} - \delta, y + \frac{1}{N})$
15: $T_3^k = (x, y + \frac{1}{N})$
16: Set $texNeigh = k + 1$;
17: **else**
18: Compute: $x = \frac{\frac{k-1}{2} \mod N^2}{N}$; and $y = \frac{\frac{k-1}{2} \mod N^2 - x}{N}$
19: $T_1^k = (x + \delta, y)$
20: $T_2^k = (x + \frac{1}{N}, y)$
21: $T_3^k = (x + \frac{1}{N}, y + \frac{1}{N} - \delta)$
22: Set $texNeigh = k - 1$;
23: **end if**
24: **end for**
25: Create device vertex array, normal array, texture coordinate array, and index array (0 to $M - 1$) and transfer primitive information.

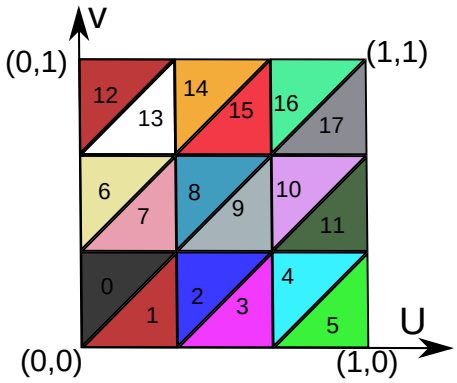


Figure 4.4: Sample Atlas, where the numbers indicate which triangle the texture belongs to, the coordinates are expressed in terms of atlas UV.

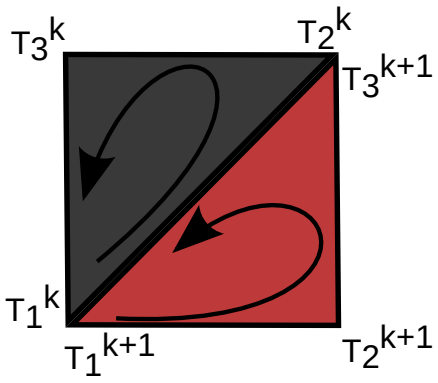


Figure 4.5: Sample Atlas drawing direction (1,2,3) for both the k^{th} and $k^{th}+1$ triangles.

Update

Let us say we want to update the texture of the k^{th} triangle, with some texture image, *data*. The update procedure can be seen in Algorithm 4.7. The update procedure consists of first identifying the orientation of the k^{th} triangle and its neighbor; then building the replacement image *texData* accordingly. Once the replacement image has been created, it can be uploaded to the texture atlas using glTexSubImage2D with the proper starting and ending coordinates.

Draw

The draw routine for texture atlases consist of Algorithm 4.8. It is important to point out that the point draw order is counter-clockwise as shown in Figure 4.5.

Algorithm 4.7 Texture update for the k^{th} primitive

if $texNeigh$ is valid **then** ▷ check status of neighbor triangle
  if $k \mod 2 == 0$ **then** ▷ Orient the texture data
    $topTex = data$, $bottomTex = texNeigh$.
  else
    $topTex = texNeigh$, $bottomTex = data$.
  end if
  Construct $texData$ using the correct portions of $topTex$ and $bottomTex$.
else
  $texData = data$.
end if
if $k \mod 2 == 0$ **then**
  $x = (\frac{k}{2} \mod \frac{N^2}{A^2}) \mod \frac{N}{A}$
  $y = \frac{[(\frac{k}{2} \mod \frac{N^2}{A^2}) - x]A}{N}$
else
  $x = (\frac{k-1}{2} \mod \frac{N^2}{A^2}) \mod \frac{N}{A}$
  $y = \frac{[(\frac{k-1}{2} \mod \frac{N^2}{A^2}) - x]A}{N}$
end if
Replace part of the atlas image (easiest way is to use glTexSubImage2D, with x and y as offsets, and $texData$ as texture data).

Algorithm 4.8 Draw routine using texture atlases

Bind vertex, normal, texture coordinate, and index arrays.
Compute $lastDrawCount = M \mod 2N^2$. ▷ this is the number of triangles to draw on the last atlas (it might not be full)
for $i = 0 : 2N^2 : M - lastDrawCount$ **do**
  Bind $pTex[i]$ Texture.
  Draw $2N^2$ primitives in order. (easiest way is to use glDrawRangeElements
end for
if $lastDrawCount \neq 0$ **then**
  Bind $pTex[i+1]$ Texture.
  Draw $lastDrawCount$ primitives in order.
end if

4.5.3 Speed up

Render times using individual textures and texture atlases are shown in Table 4.3. From the render times we can see that texture atlases provide five orders of magnitude speed up. Furthermore, the marginal gains of having larger and larger atlases can be seen in the table below; up to the point where all triangles can be drawn on a single atlas. Further increase of the atlas size can only slow down the rendering time since we risk the texture being larger than what graphics hardware can cache.

It should be noted that for a scene with a moderate number of triangles, greater than a hundred thousand, the texture atlas implementations achieves real-time rendering (60Hz) while the individual textures do not (approximately 40Hz).

Method	**Texture Size**	**Atlas Size**	**Number of Atlases**	**Render Time** (micro seconds)
Single Textures	8	-	-	110,000 - 115,000
Texture Atlas	8	256	40	9.4 - 11.5
Texture Atlas	8	512	10	6.9 - 8.5
Texture Atlas	8	1024	3	5.5 - 7.3
Texture Atlas	8	2048	1	5.1 - 6.9
Texture Atlas	8	4096	1	5.7 - 7.2
Texture Atlas	8	8192	1	5.7 - 7.2

Table 4.3: Render time for different texture methods on 80k triangles. (tested: on NVIDIA GTX-580)

Chapter 5

Results

In this chapter we discuss experimental results. The goal of these experiments is first: to validate the model; second, to compare it to traditional reconstruction approach; and third to demonstrate advantages of the model discussed in Chapter 3. We begin by discussing some of the parameter s used throughout this work. We then analyze the estimation of each latent variable of the proposed model, namely the geometry and appearance of the primitives and the intrinsic and extrinsic camera parameters. We continue by comparing reconstruction results with a publicly available implementation of structure from motion (SfM). We conclude by highlighting some key aspects of the model not shared by traditional approaches. The results presented here are based on three main datasets, for a description of the data see appendix A.

5.1 Experiment Parameters

As mentioned in chapter 3 we obtain MAP estimates for all random variables in the model. For all results presented here the image noise model is assumed to be independent and identically distributed (iid) Normal with zero mean and a 10 pixel standard deviation, the appearance prior model was iid Normal with mean 128 and standard deviation 15. LiDAR noise measurement has been chosen according to values found in the literature [30] and consistent with empirical observations, iid Normal with zero mean and 12 centimeters standard deviation.

Unless otherwise noted the canonical appearance for each primitive is assumed to be 16×16 image. The size of atlas is 1024×1024. Throughout this work we use a four neighbor appearance fill, as described in §4.1 , and the background appearance consisted of 256×256 pixels. The GPS position noise parameters are modeled as iid Normal with zero mean and with a standard deviation of 20 feet; the orientation is again modeled as iid Normal with zero mean and 5° standard deviation.

5.2 Model Validation

This section present results pertaining to the estimation of latent variables in the model presented in chapter 3. The goal of the experiments is to provide some insight into different aspects of the latent parameter estimation within the described model. In

order to better understand the behavior of the model we will first discuss the appearance, geometry and camera parameter estimation with an emphasis on computational efficiency. Following that we will analyze the trade off between information sources present in the model, particularly the trade off between geometry and appearance information. We will conclude the section with a brief look at geometry initialization and its effect on reconstruction output.

5.2.1 Appearance Estimation

As presented in §3.2.2, the image observation model contains a noise term that is dependent on the camera's viewing angle and the surface's orientation; while this model is intuitive it is computationally expensive. This section analyzes the appearance estimation in the context of this noise model and provides a less computationally intense model while maintaining the desirable results. This section also briefly examines texture atlases.

Image Noise Model: Noise function of viewing angle

We begin by validating our choice to parameterize image noise by the angle of incidence between the world plane normal and the image viewing direction, c.f. Figure 5.1. From the figure we can see that the quality of the texture obtained from a given image degrades as the normal of the triangle and the viewing direction become increasingly orthogonal (left to right in the figure). We note that the poor quality is a result of the reverse projection implementation (c.f. §4.1) and not of the observation. The high variability in the quality of these textures prohibits choosing a single noise level for which a suitable reconstruction would be possible. In order to maintain the simplicity of an additive noise level, while modeling these artifacts we use a noise level that varies as the viewing conditions between the image and the world geometry change.

Consequently, we choose the following standard deviation for the additive noise term:

$$r_i = \frac{1}{|n_m^\top v|}$$

where n_m is the normal of the m^{th} triangle, v is the viewing direction of the current image and i indexes pixels in the current image[1]. We can substitute this standard deviation in the appearance estimation equations of sections 3.4.1.

One interpretation of the appearance update equations, (3.21) and (3.22), is that of averaging the observed pixels and inversely weighting each observation by the associated noise term. The weighted appearance contribution for the triangle in Figure 5.1, can be seen in Figure 5.2. From the figure we can see that the main contribution comes

[1]We not that this standard deviation becomes infinite when the viewing angle and the primitive normal are orthogonal. In our implementation when set the noise to a large value, 10^4, when this happens

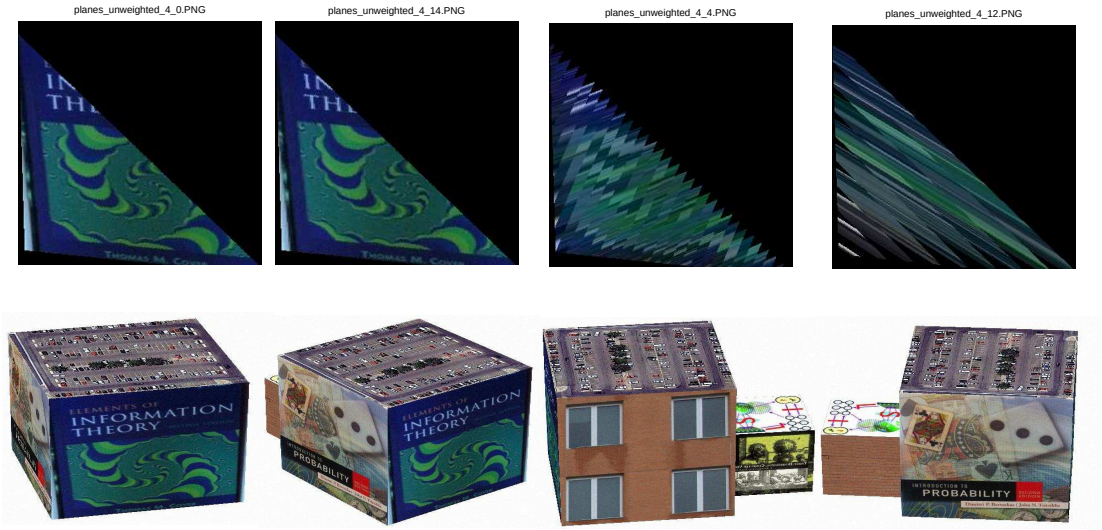


Figure 5.1: Appearance as a function of viewing angles. *Top row:* Texture for a given plane obtained from the rendered image shown in the *bottom row.*. As viewing angle increases, texture quality rapidly degrades. This artifact is a result of the reverse projection scheme, c.f. §4.1.

from the images that view the triangle almost head on; images 4 and 12, those in the right side of Figure 5.1, receive almost no weight due to their oblique viewing angles. Figures 5.3 and 5.4 show a similar behavior for a different triangle.

We emphasize that the weights are evaluated on a per-pixel basis. This is due to the fact that over the set of observations, not all pixels for a given primitive are observed an equal number of times (eg. due to occlusions). This has the expected consequences of having variable weights for each pixel (rather than each image) as can be seen in Figure 5.5. From the figure we can see that despite having a poor viewing angle some pixels, e.g. the strips shown in the bottom center, get a high weight since they are only visible in a small set of images, possibly only one. However, this has the disadvantage of having to compute the weights for each image and world triangle combination; which can be computationally expensive for scenes with a large collection of triangles, such as the ones we are interested in.

Image Noise Model: Constant Noise

While the variable noise model presented in the prior section achieves the desired reconstruction quality, its computational complexity makes it unappealing. As previously mentioned the main drawback of the model is the computation of pixel contributions (weights) for each image and latent primitive pair. Alternatively it would require maintaining a variable stack of weights for each pixel, which can be equally cumbersome and undesirable in GPUs. The motivation for introducing an angle-dependent noise

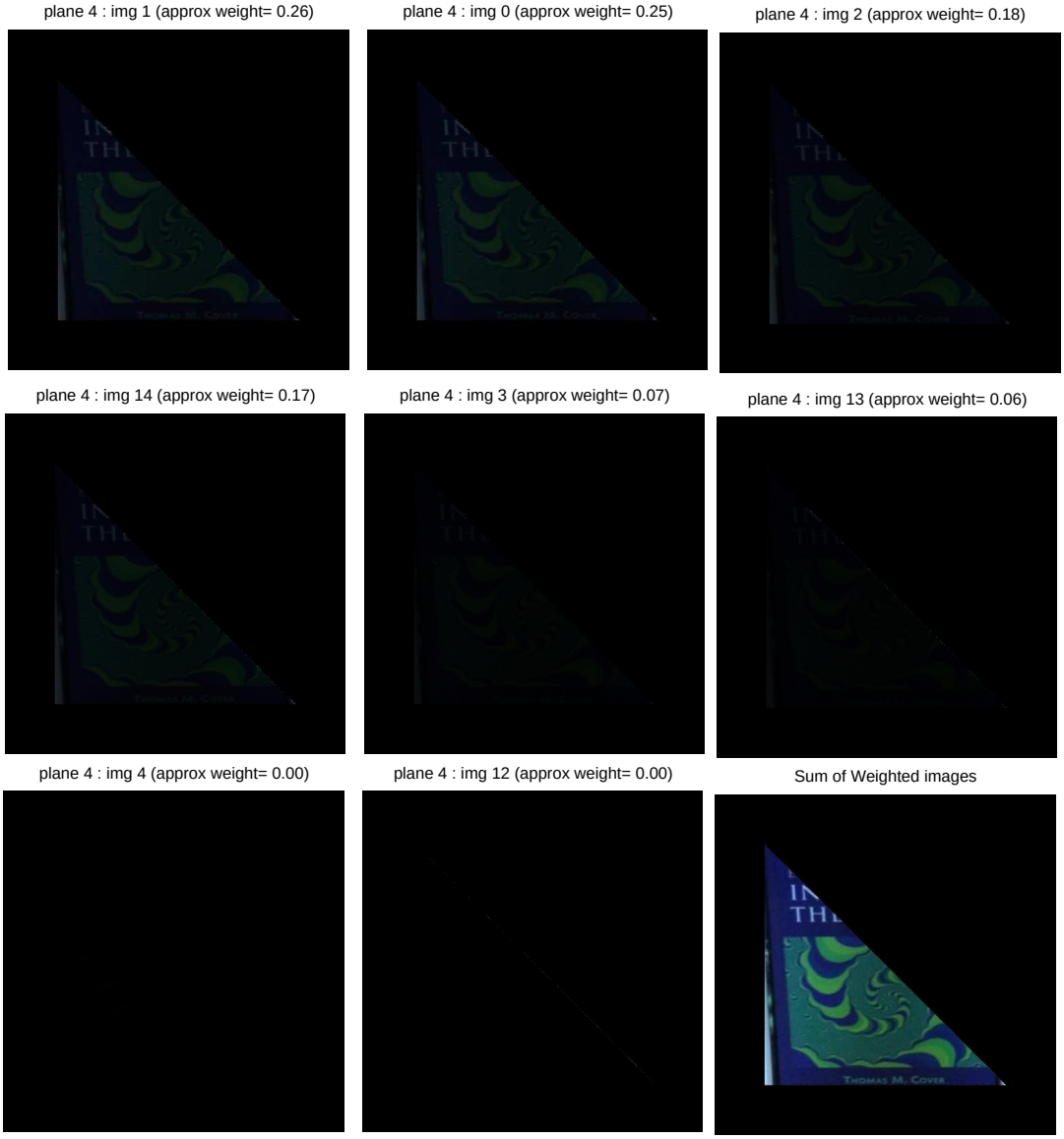


Figure 5.2: Weighted Appearance for a given triangle (the brighter the image the higher the weight). Bottom-right tile is sum of the weighted sources. Sources of Fig. 5.1 seen here in the middle of the top row, left in the center row, and left and center in the bottom row.

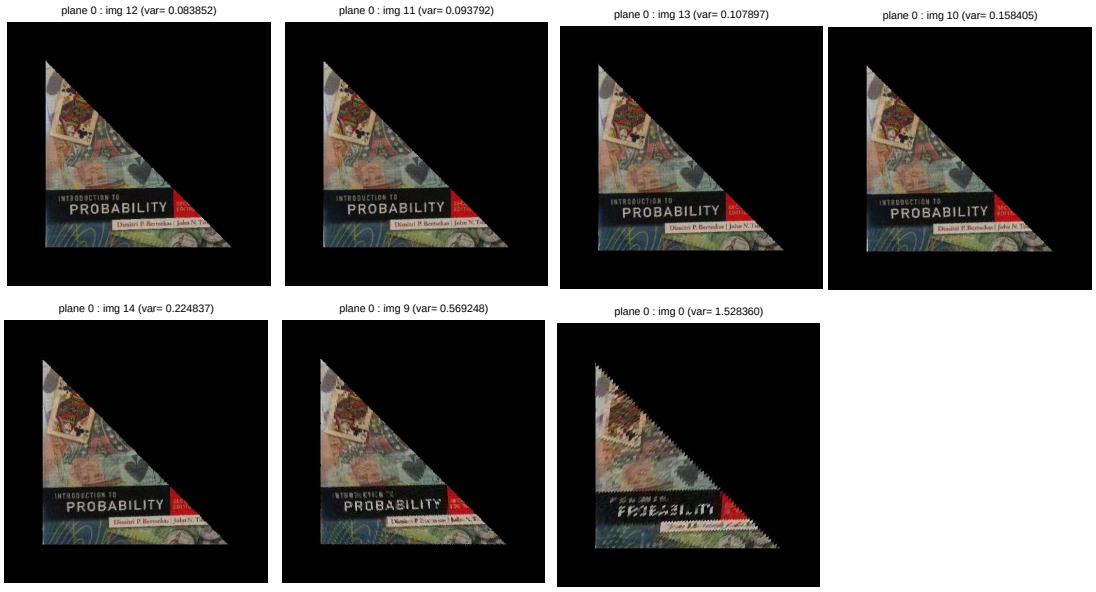


Figure 5.3: Un-weighted Appearance for a given triangle (ordered in increasing angle between primitive normal and source image viewing direction). Notice how the quality of the texture degrades with increasing viewing angle.

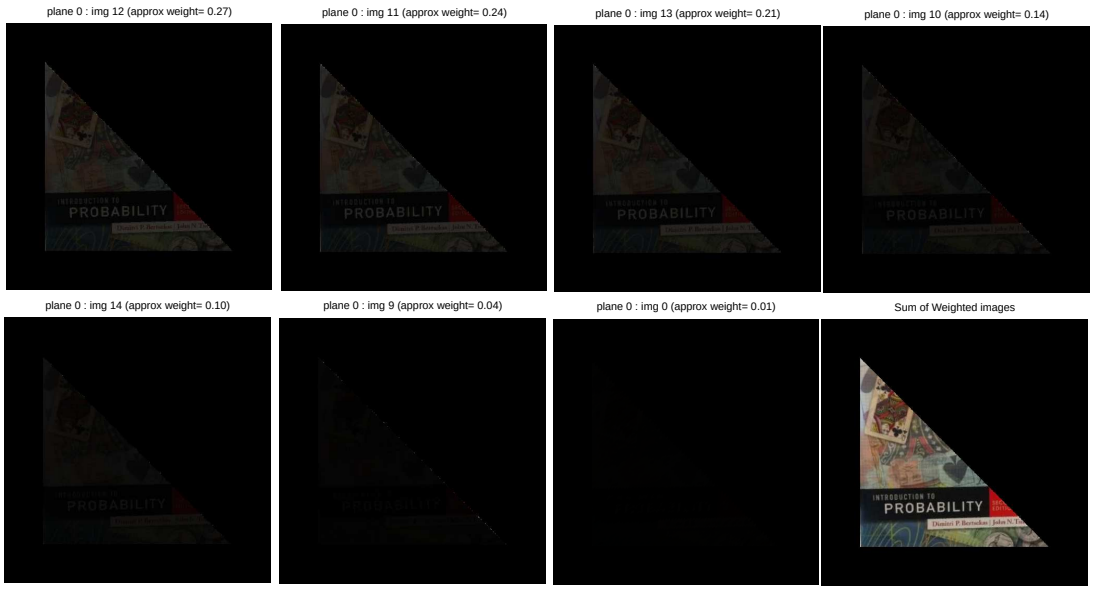


Figure 5.4: Weighted Appearance for a given triangle (the brighter the image the higher the weight). Bottom-right tile is sum of the weighted sources. Source images can be seen in Figure 5.3

Figure 5.5: Weighted Appearance for a given triangle. (images ordered in increasing viewing direction, the brighter a pixel, the higher the weight). Bottom right image is the weighted sum of the sources.

model was based on a reverse projection implementation where the source image was projected into the primitive and the result rendered from a head-on view to generate the canonical texture. This projection techniques requires that the source image explains the entire primitive, despite the size of the texture source. As a result, the source image is typically interpolated; this interpolation can result in some of the striations seen on Figure 5.1 where the small component of the source image is stretched to fill the primitive.

However if we only look at a forward projection method, as described in algorithm 4.2, then the projections from observations to textures would be at the pixel level, meaning observation pixels would only influence a single texture pixel, and no interpolation would be required. Figure 5.6 shows the textures obtained from the same source images as Figure 5.1 using forward projection. From the figure we can see that the forward projection method does not attempt to fill the entire appearance image, it fills in the visible pixels. We can also see that the number of filled pixels decreases as a function of viewing angle, which should be expected since this is the same quantity as present in the source image.

Figure 5.6 leads us to believe that if we use the forward projection method we use a constant noise level since the artifacts we are trying to deter with the variable-noise model are no longer present. The results of this approach with a standard deviation of 10 is shown in Figure 5.7 for each of the latent primitives discussed in the previous section. Qualitatively the results of both methods are identical, c.f. Figures 5.2, 5.4,

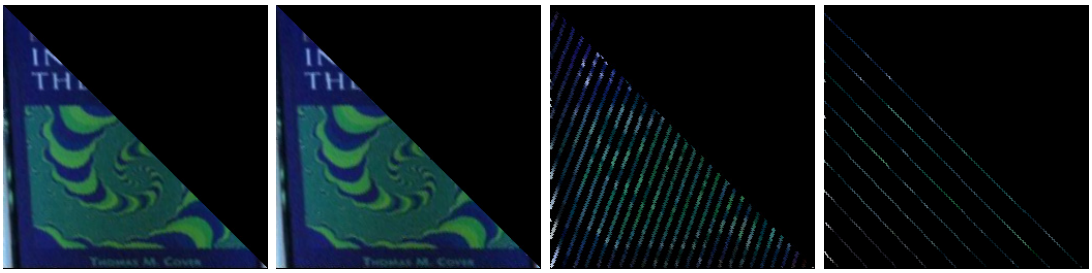


Figure 5.6: Texture generated by forward projection method (source images same as Figure 5.1). Forward projection associates an observation pixel with a single texture pixel, and possibly its neighbors. As a result interpolation is no longer needed, the black pixels in the lower triangular region of the image represent missing data.

(a) plane 0 (b) plane 4 (c) plane 6

Figure 5.7: Learned appearances using a constant noise level and forward projection. Reverse projection results can be seen on the last image of Figures 5.2, 5.4, and 5.5.

5.5 and Figure 5.7. Computationally we no longer need to compute or store weights for all the image and geometric primitives, all we need are pixel counts, resulting in large computation savings, see section 4.1.

Texture Atlases

As discussed previously, the use of texture atlases greatly improves performance. Typical texture atlases can be seen in Figures 5.8 and 5.9. Several observations can be made about the texture atlas, first we can clearly see the effects of fixed texture coordinates, where some textures are scaled unevenly. This occurs when the texture coordinates map the hypotenuse of a primitive to one of the texture edges rather than the texture image hypotenuse; this results in stretching in one image direction and compression in the other (see the second block of Figure 5.8, where the windows on the bottom portion of the texture image are scaled differently than the ones on the top portion). The affine transformation needed to undo this skew is performed by openGL

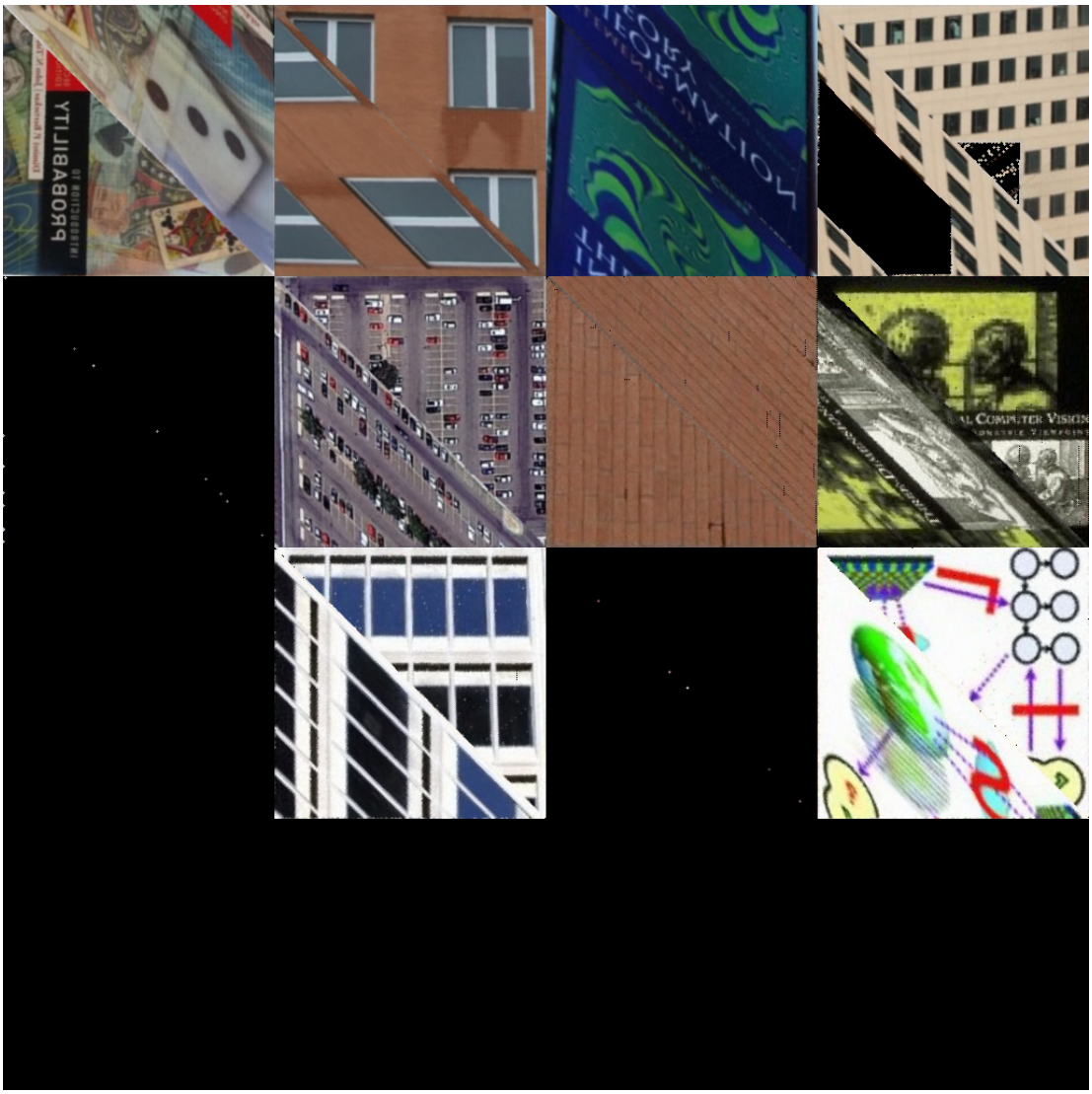


Figure 5.8: Toy problem texture atlas, appearance size 256×256, atlas size 1024×1024 (1 of 1). Notice the effect of fixed texture coordinate.

(in the current implementation) and is transparent to the end-user (unless one looks at the texture atlas directly as the case here).

The appearances of latent primitives that are not visible in any image appear as black (0,0,0) in the texture atlas. Close inspection of Figure 5.8 shows a single pixels along the diagonal for some non-visible triangles. These pixels are a result of openGL rendering artifacts where occasionally due to object culling and inaccuracy of the depth buffer, a single pixel of a non-visible triangle will be visible. We note that this typically happens when primitives with vastly different normal intersect.

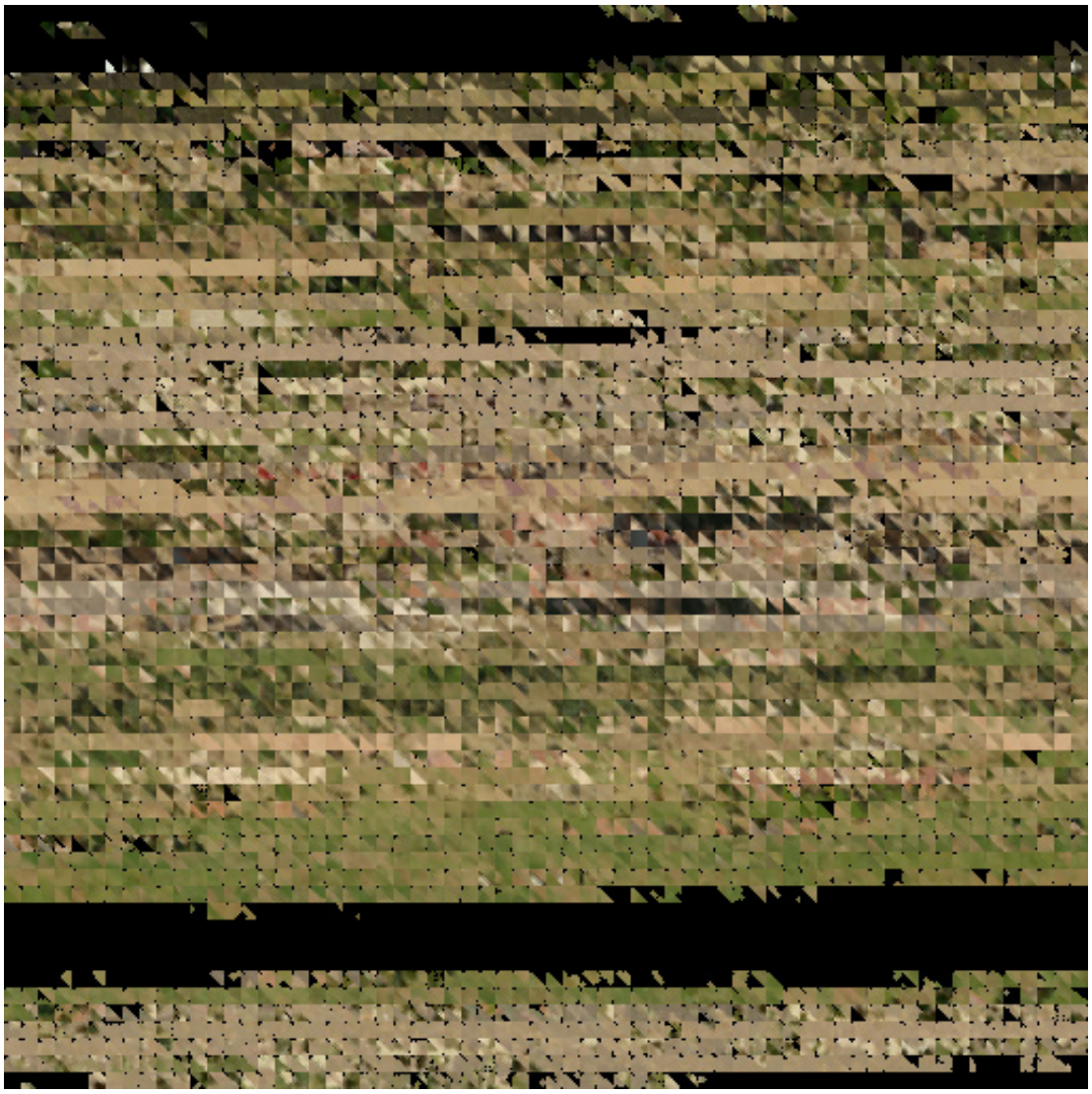


Figure 5.9: Lubbock texture atlas, appearance size 8×8, atlas size 512×512. (1 of 10)

Figure 5.9 shows a texture atlas for the Lubbock dataset. As compared with the toy example the main differences are that texture sizes can be made much smaller since triangle sizes are typically smaller and hence have a smaller number of observation pixels, and that texture atlases might not be entirely populated, as reflected by the entire black rows in the figure. The order of appearance in the texture atlas depends on the order of the world triangles in the stack of primitives; if we wanted to pack the atlases, so that all visible textures came first, we would need to order the primitives according to visibility. This has the advantages of reducing the number of atlases needed, and thus minimizing number of texture bindings at rendering time, and also reducing the memory usage in graphics hardware. However, it has the disadvantage of having to change the atlases whenever visibility changes. We opted not to implement these changes since our renderings already achieve real time performance and we have enough texture memory to hold all atlases for the large scenes that we are interested in analyzing.

■ 5.2.2 Geometry Estimation

In this section we analyze and discuss the estimation of latent geometry variables in the proposed model. Due to the large number of geometric primitives, emphasis is placed on computational efficiency. As a result we begin by comparing the single primitive estimation with the joint estimation over multiple primitives, c.f. §4.4, in terms of accuracy of the results and computational complexity. Once the performance criteria have been met, we present reconstruction examples of typical scene building blocks such as horizontal planes, vertical planes, and trees. These examples are characteristic of the model performance on each building block.

Individual vs. Joint Geometric Optimization

As demonstrated in chapter 4 there are several advantages of looking at a multi-primitive update scheme, the primary being computational complexity. However, before looking at computational complexity we must ensure that the results obtained via both methods are comparable. To this end, Figure 5.10 shows the negative log likelihood of LiDAR as a function of optimization run iteration, image and total for both individual and multi-plane optimization scheme for a given plane. Several observations can be made about the figure.

First we note that the LiDAR likelihood dominates the estimation procedure for most of the initial iterations since the change in image likelihood is not as significant for this portion; at around iteration seventy, the LiDAR likelihood levels off and the image likelihood dominates for the remaining iterations. This is the typical behavior we see in the model, where both terms are competing to explain their respective data. The dominant term[2] depends on a variety of factors such as the number of LiDAR points

[2]We note that the negative log-likelihood terms are additive across modalities, so that the relative scale matters to determine dominance.

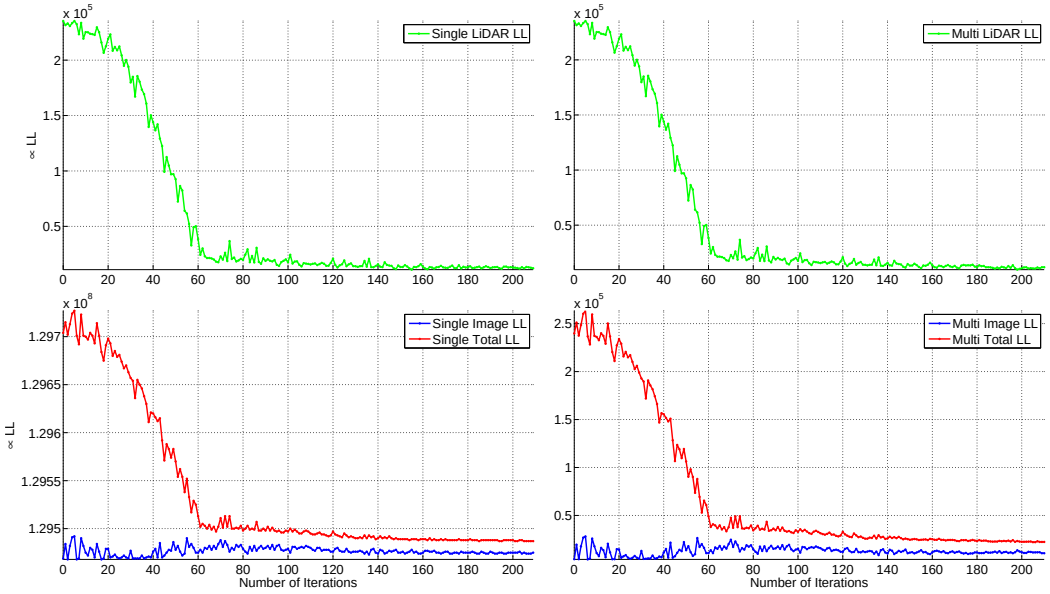


Figure 5.10: Individual and Joint optimization for a given plane (left and right column respectively), divided into LiDAR (top row) and Image likelihood (bottom row). The scale factor changes on the multi-plane image likelihood, this is due to ignoring all primitives not being optimized. Plane 35147, CLIF Image Stack

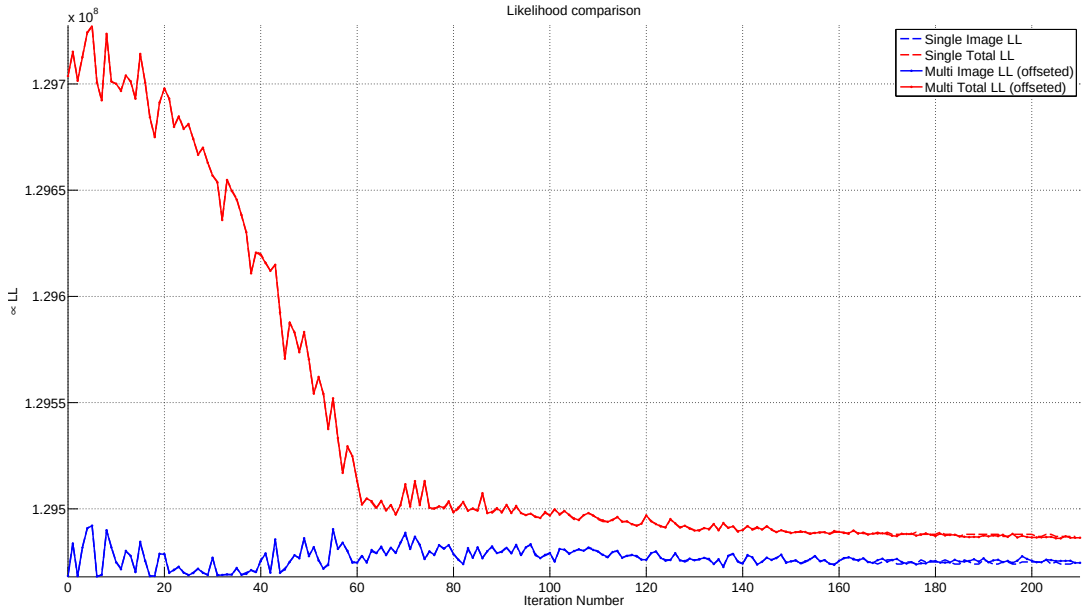


Figure 5.11: Individual and Joint Likelihoods side-by-side for planes of Fig. 5.10. Multi-plane likelihood scaled with constant offset of pixels not associated with current planes

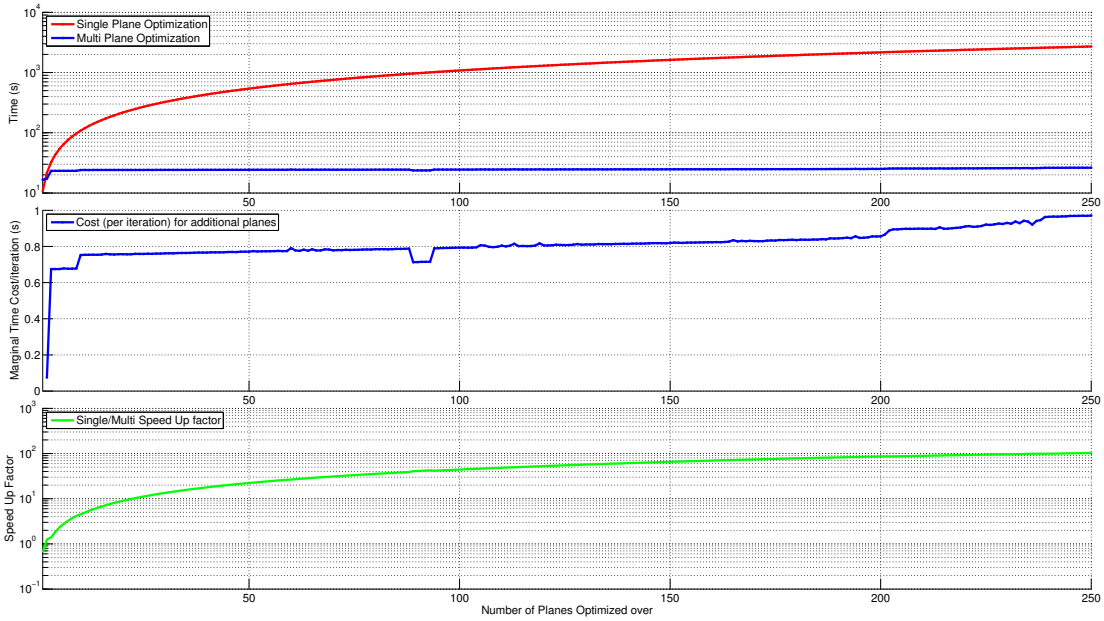


Figure 5.12: Computation time comparison for single and multi-plane optimizations for the CLIF Intersection scene. Top plot: Cost for each method as a function of number of planes being optimized. Middle plot: per iteration additional cost as a function additional planes added (base is 1). Bottom plot: speed up factor

associated with the plane, the number of pixels of the plane that are visible across the image collection, and the noise parameters.

Secondly, the difference between the individual and multi-plane estimation can be seen in the figure if we look at the scale of the ordinate for the bottom row, where we see a difference of three orders of magnitude. This difference corresponds to the likelihood contribution of all the pixels in the image that are not associated with the primitive in question. This quantity is constant since the other pixels are not being changed. We note that there is no change in LiDAR likelihood since the multi-plane scheme only affects image likelihoods.

Figure 5.11 shows the individual and multi-plane likelihood side-by side, (after accounting for the constant offset). From the figure we can see that the likelihoods are identical for most of the iterations shown. We can also see that around iteration 170 the evaluations differ. This difference is negligible and comes as a result of numerical approximation in the computation, which impacts the optimization routines. We point out that, as desired, both individual and multi-plane schemes produced the same result.

In terms of computational complexity, the multi-plane is substantially faster than the single plane. This can be seen in the top plot of figure 5.12, where the computation time for the multiple plane estimation scheme is always lower than single plane. The exception to this is the case when a single plane is being considered; for this case the

overhead associated with the bookkeeping for the multi-plane is more computationally intensive than performing the full likelihood computation.

This result is not surprising since as discussed earlier the main cost of the likelihood computation is rendering the primitives, which for the individual plane scales linearly with the number of planes considered; and is constant for the multiple plane case. As before, we point out that our implementation is limited to a maximum number of 253 planes in parallel since we are encoding plane identity in the alpha channel of the texture maps, which are limited to 255 labels and two are already in use (background and regular planes).

A closer look at the top plot of Figure 5.12 reveals that the computation time of multi-plane is not constant. Which is confirmed when we look at the marginal cost for including additional planes in the optimization (middle plot of figure 5.12). We note that the marginal cost increases with each additional plane included, but this increase is minor when compared with the single optimization. Possible reasons for this cost included the expansion in the required bookkeeping and the graphics hardware memory management (memory usage increases linearly with the number of planes analyzed in the multiple plane case). The overall speed up factor can be seen in the bottom plot of Figure 5.12. From the figure we can see that the gains are anywhere between slightly less than one and over one hundred.

Overall, this section has shown that the multi-primitive optimization scheme is much faster than the single-primitive counterpart. Furthermore, the experiments performed showed that this performance gain does not come at the cost of accuracy since both single- and multi-plane optimization yield similar local optima.

Scene Reconstruction Analysis

In this section we discuss the ability of the model to reconstruct a variety of scene elements. For the purpose of this discussion we will focus on the following scene elements: horizontal planes (such as ground plane or rooftops), buildings, and trees. This small subset of categories were chosen because they have varying geometric and appearance properties as a means of examining different aspects of the proposed model. Throughout this section we will be referring to multiple sites of the CLIF dataset, as shown in the reconstruction overview of Figure 5.13. Along with explicit reference where appropriate, we will color-code the border of each scene detail with the color shown in the overview figure for easy reference.

Horizontal Planes

We begin the scene analysis with the most common type of surface orientation we encounter, horizontal planes, that is extended horizontal regions that are largely planar and typically composed of multiple primitives. Horizontal planes are ubiquitous in aerial imagery, since they account for most of the surfaces seen. The common examples include the rooftops and ground (while not necessarily planar, the ground can be approximated as planar).

Examples of horizontal reconstructions can be seen in Figures 5.14 - 5.17. Figures

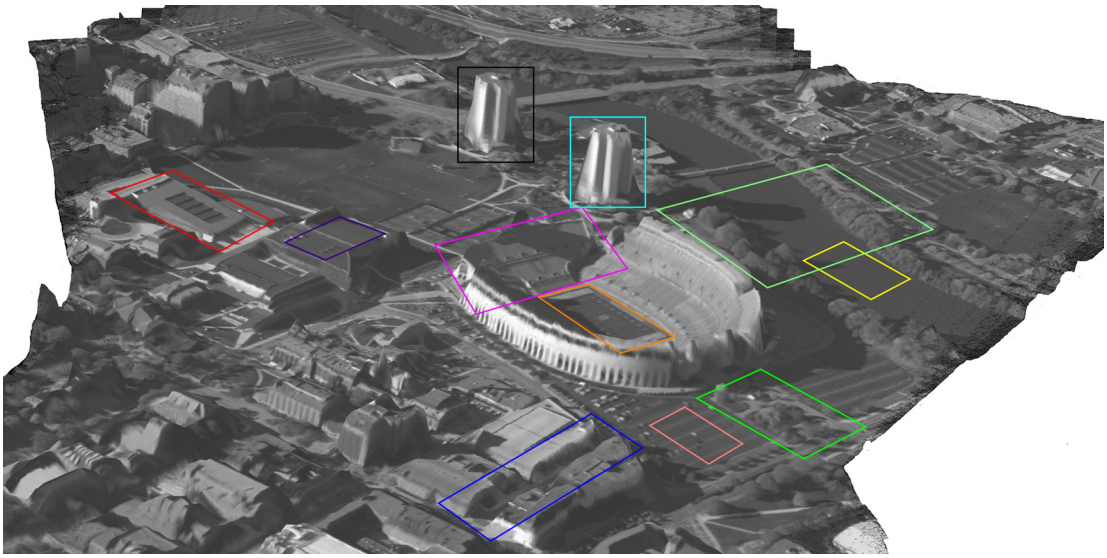
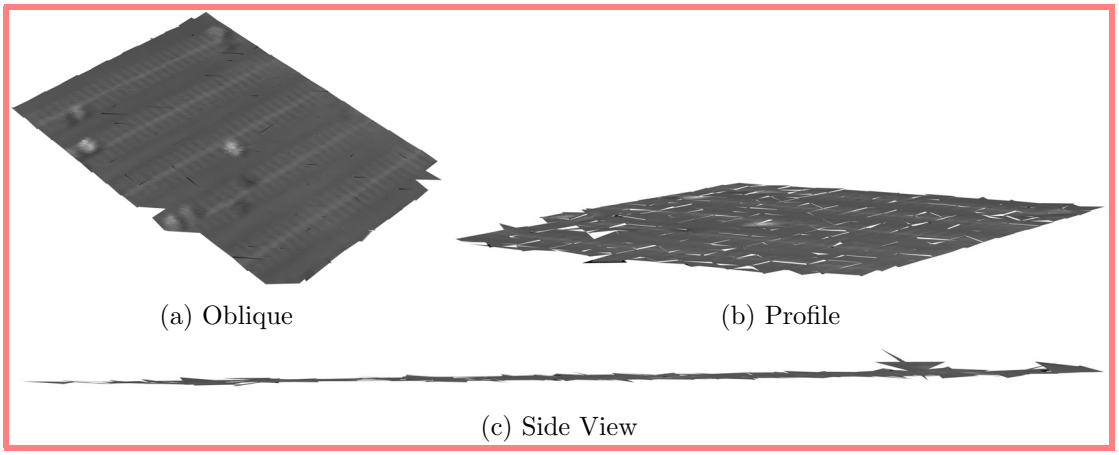


Figure 5.13: Scene overview, regions that will be examined in this section are shown in different colors. Subsequent region details are color-matched.

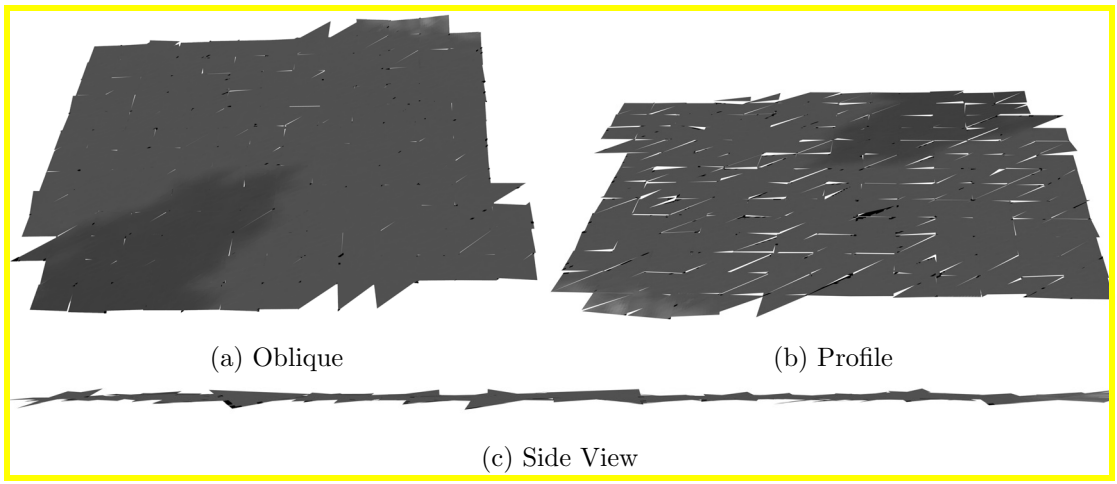


(a) Oblique (b) Profile

(c) Side View

Figure 5.14: Planar Reconstruction Example - Stadium Parking Lot

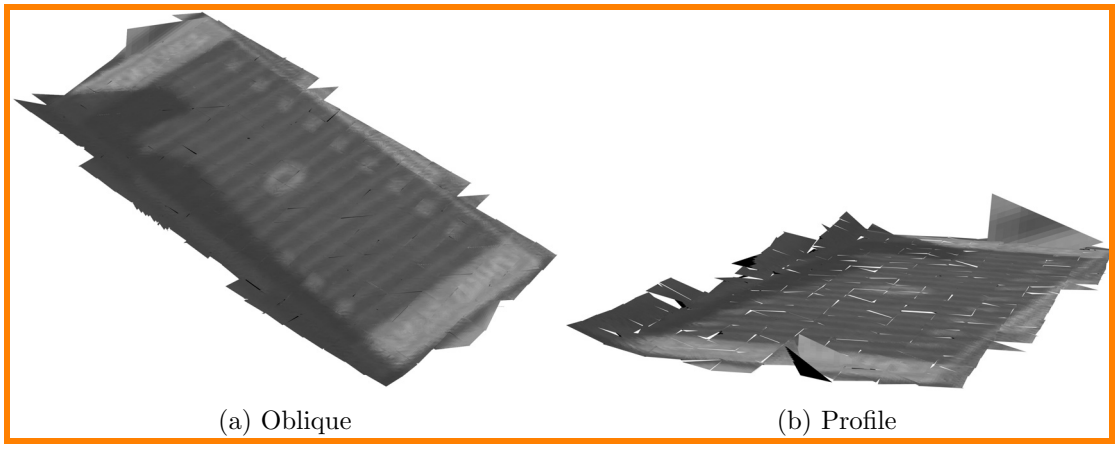
5.14 and 5.15 show reconstructions of strictly planar regions from three different views, an oblique view (roughly aligned with the observations viewing direction), a profile view (typically counter to the viewing direction) and a side-view. We can see from both images that the model estimates the geometry fairly well. This is not surprising since most of the LiDAR returns stem from horizontal surfaces, with the correct association, this would drive ground primitives to explain a large number LiDAR observations. The combination of a large collection of measurements with low noise usually leads to higher quality reconstructions.



(a) Oblique (b) Profile

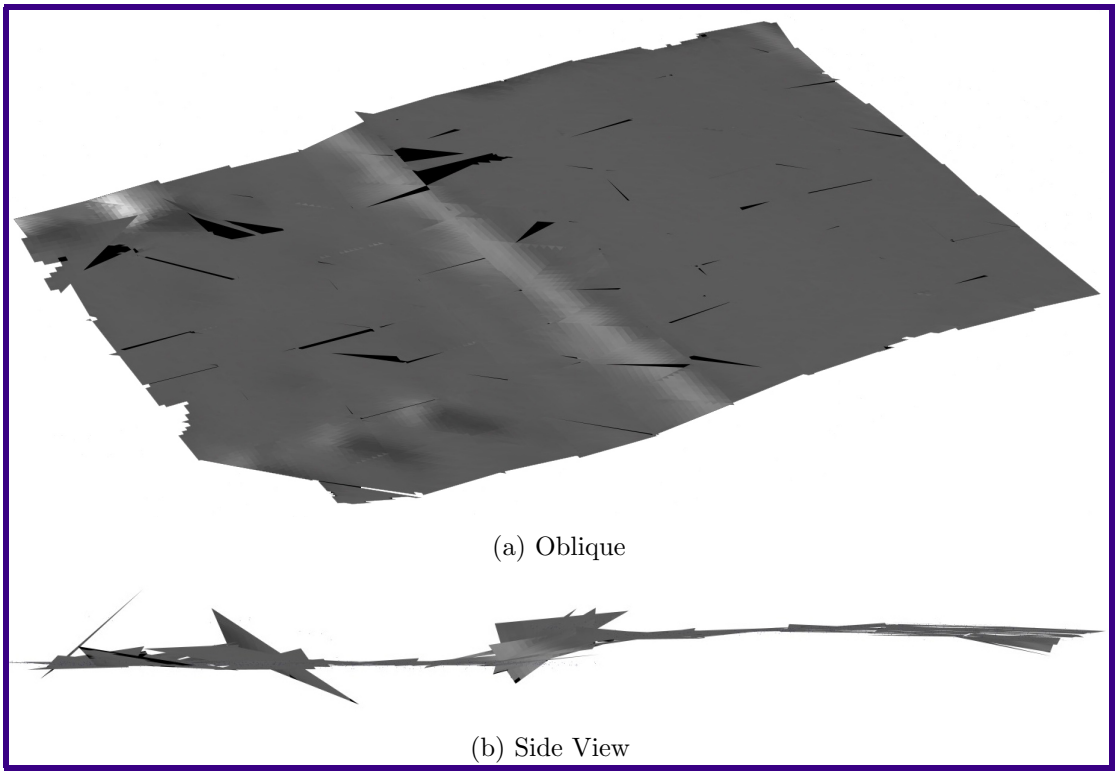
(c) Side View

Figure 5.15: Planar Reconstruction Example - River



(a) Oblique (b) Profile

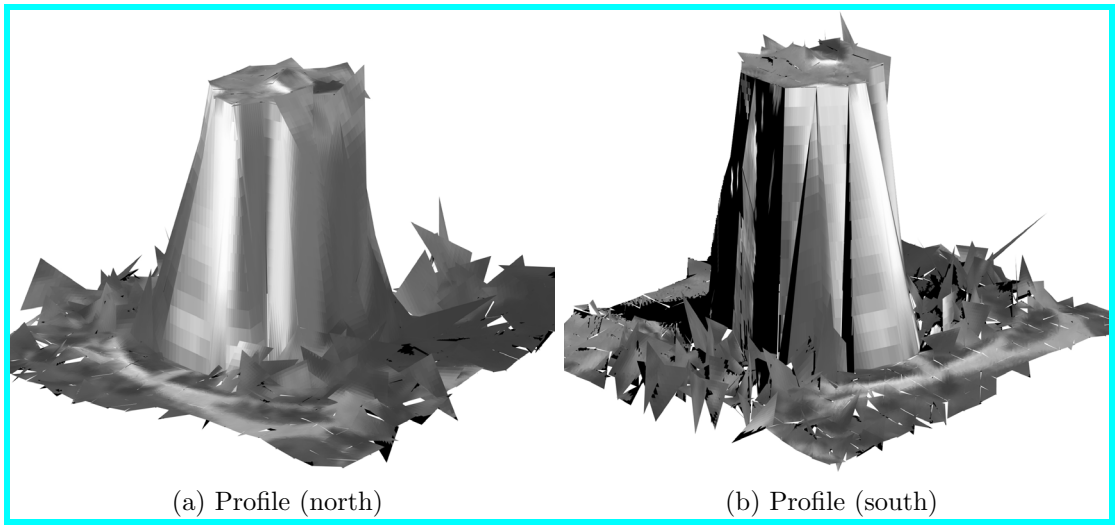
Figure 5.16: Planar Reconstruction Example - Stadium Field



(a) Oblique

(b) Side View

Figure 5.17: Planar Reconstruction Example - Roof (two distinct heights). Heights level smoothed due to incorrect LiDAR association.



(a) Profile (north) (b) Profile (south)

Figure 5.18: Building Reconstruction Example - Tower 1 (northern-most)

Figures 5.16 and 5.17 are interesting cases since they are not strictly planar. For example the goal post in the stadium field generates a small number of LiDAR observations, however, these posts are not captured[3] in the latent representation and as a result the corresponding LiDAR observations are associated with the closest primitive, which is the ground plane. This causes ground primitives to rise to explain these observations as seen in front and back of the stadium as shown in Figure 5.16b.

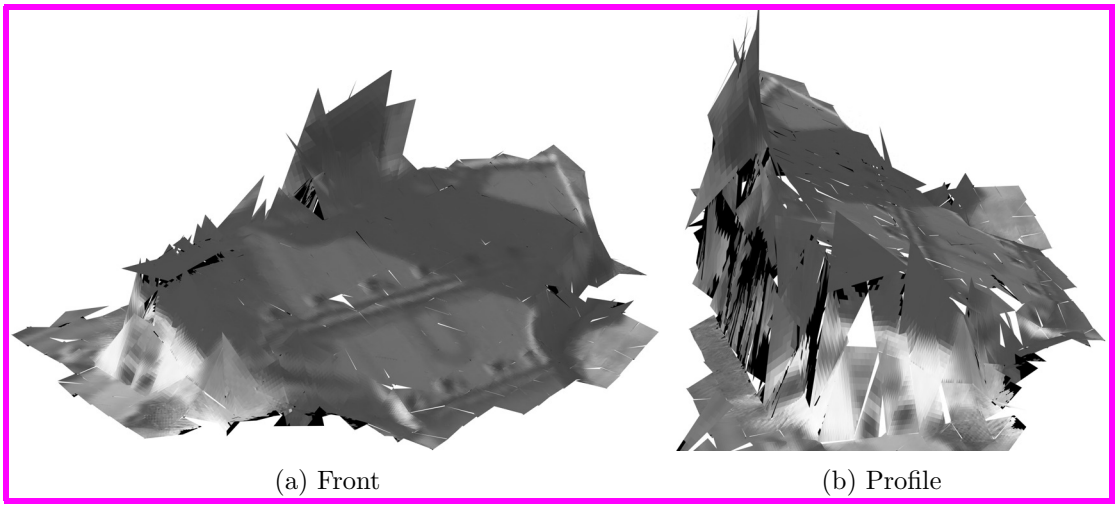
As another example Figure 5.17 contains two different height levels, which can be seen from the side view, however, the reconstructed transition between both levels is smoothed over by latent primitives that cover both levels. This again, can be explained by the LiDAR association, in this case the returns from both levels are being associated with the same plane, since there is no alignment that would satisfy both constraints, the inference algorithms chooses to average the heights and grow the primitives, resulting in the larger triangles seen in figure 5.17b.

Buildings

Buildings are the next set of structures we are interested in. When compared to the ground plane or rooftops, buildings are more complex since they couple the horizontal planes (such as ground and rooftops) with the vertical planes that make up the building walls. The interactions between different planes needed to model buildings makes them particularly challenging to model.

Examples of building reconstructions can be seen in Figures 5.18-5.21. These examples are all interesting and demonstrate the variety of geometry the model can reconstruct. Figure 5.18 shows the coupling between the ground plane and the roof via

[3] A variety of factors contribute to the goal post not being properly reconstructed, some of which are the relative size of them with respect to the image resolutions, and the limited viewing.



(a) Front (b) Profile

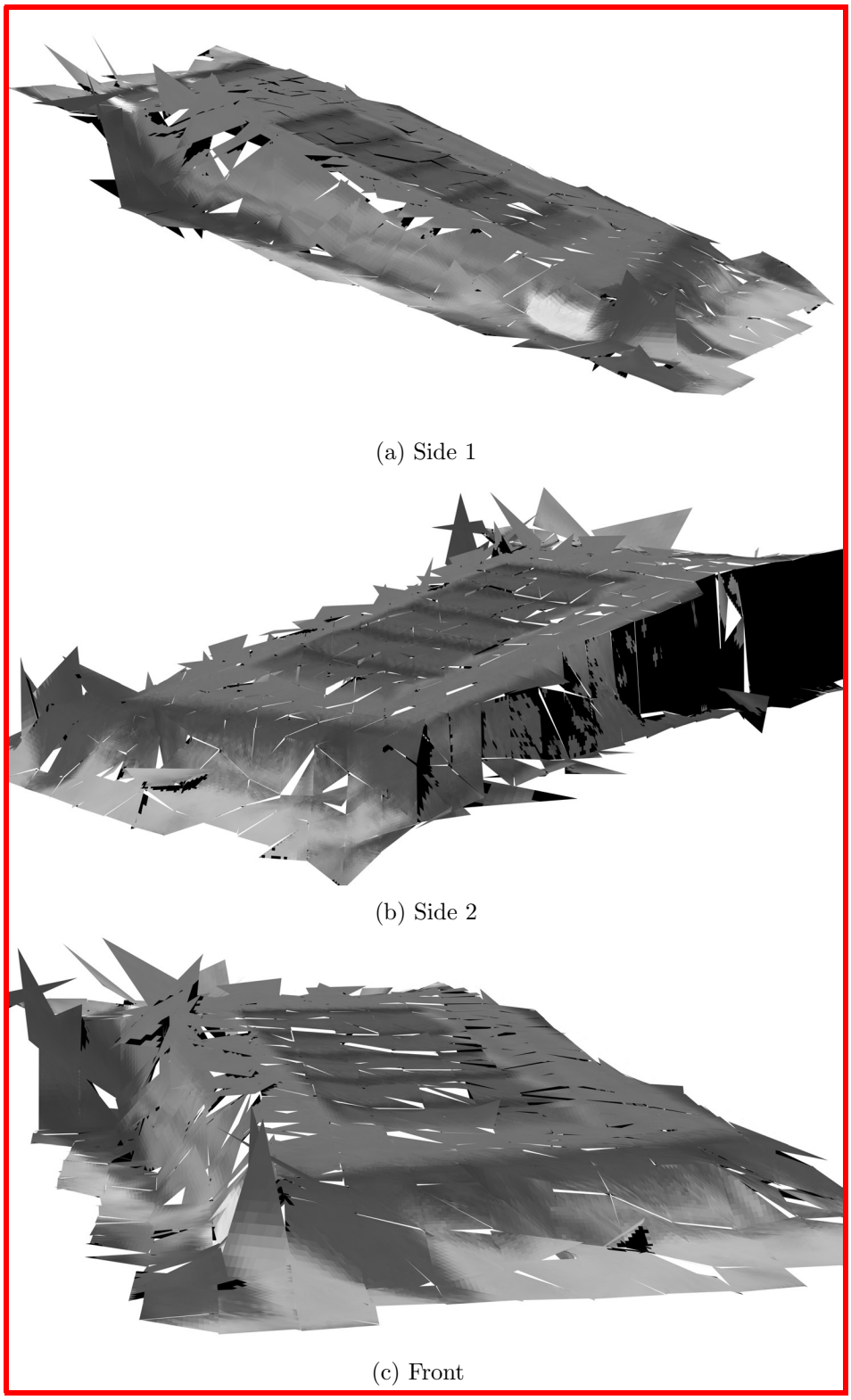
Figure 5.19: Building Reconstruction Example - Part of Stadium

long and narrow primitives that comprise the side of the tower. From the figure we can see that the camera view (a) respects the multi-faceted geometry and appearance of the building. While this reconstruction is pleasing, it is by no means perfect, as can be seen from the incorrect geometry in the roof (should not be planar) and the back portion of the building (not faceted), see appendix A for images of the scene.

Figures 5.19 and 5.20, shows part of the stadium and a slanted building respectively. These reconstructions are particularly interesting since their structure is not that of a traditional building. For example the scoreboard on the stadium is long and thin, and similarly to the goal post of the field, yield a small number of LiDAR returns, which causes primitives to elongate to explain them and produce the shape seen in figure 5.19a. Similarly the incline of the bleachers in the stadium and the roof structure of the slanted building produce diagonal planes that are reconstructed well, while simultaneously keeping the sides of the buildings vertically and shrinking in size to accommodate for the change in height, Figures 5.19b and 5.20c.

Having considered isolated structures, we now consider multiple structures. This case can be best examined in the context of an intersection, cf. Figure 5.21 (where we have intentionally limited the extend of buildings to better show the details between buildings). Figure 5.21 contains two profile views, and two front views; one of each approximately corresponds to an image perspective (bottom row). As before we can see from the figure that the coupling between the ground plane and the roof are achieved by narrow vertical triangles. These triangles as seen in the front view represent the building walls. Furthermore when the intersection is visualized from views different than observation we can see that some of the vertical triangles do not necessary end at the boundary of the roof (or ground), and thus produce a sharp, peaked appearance.

In terms of the interaction between buildings, we can see that the geometry of the

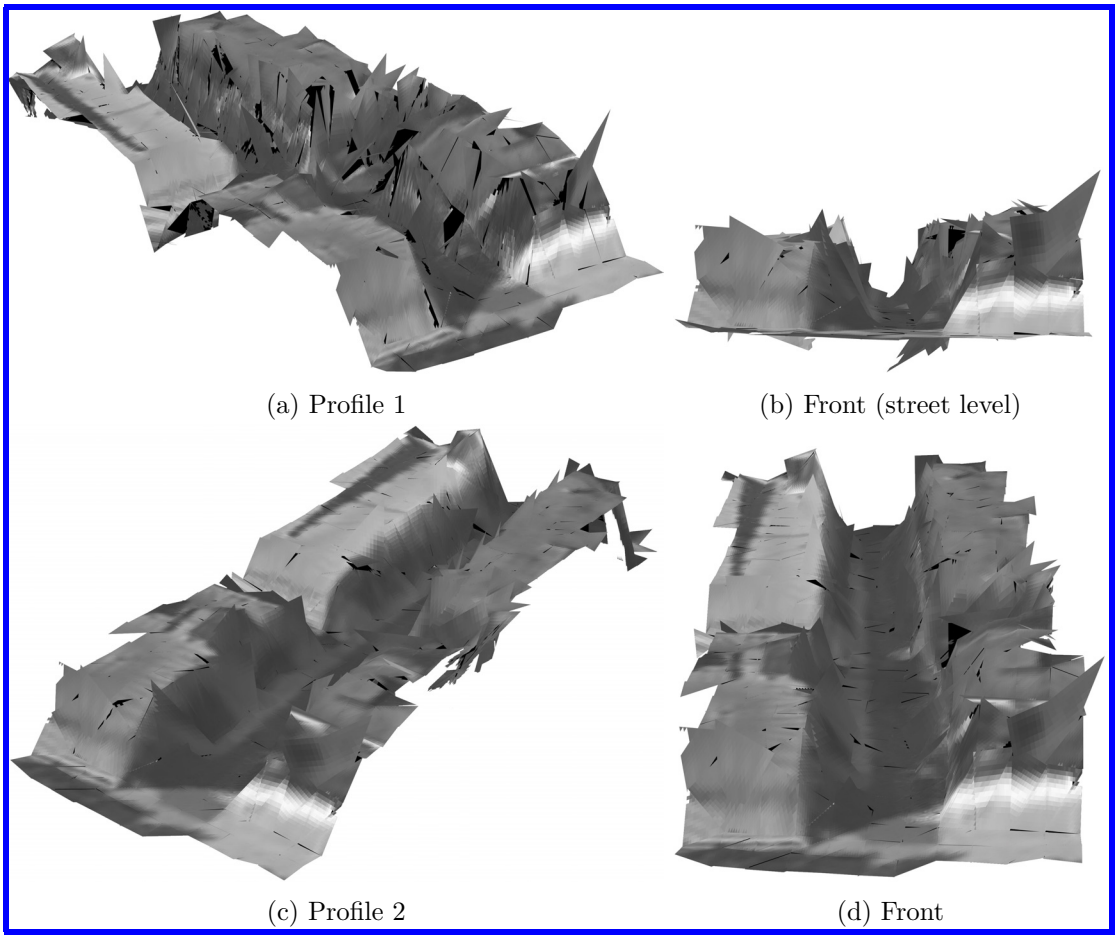


(a) Side 1

(b) Side 2

(c) Front

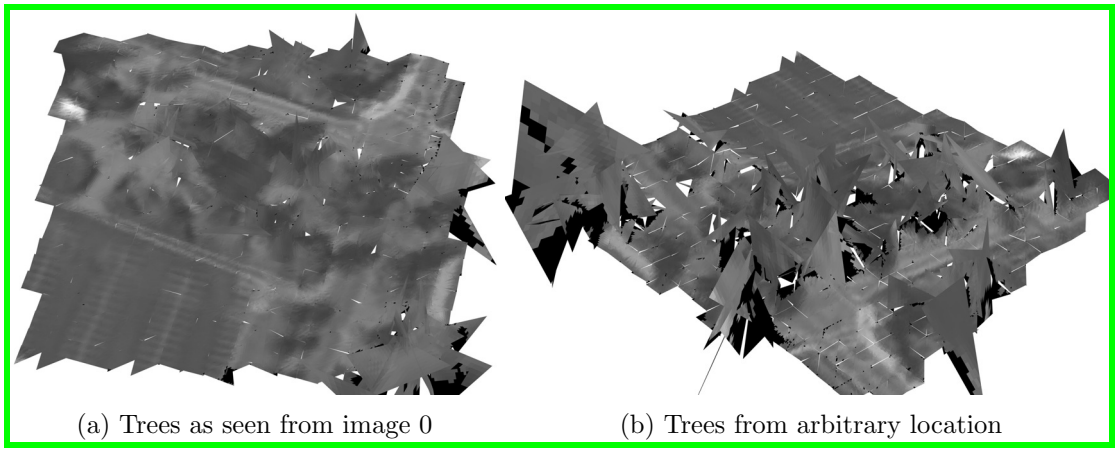
Figure 5.20: Building Reconstruction Example - Slanted Building



(a) Profile 1 (b) Front (street level)

(c) Profile 2 (d) Front

Figure 5.21: Building Reconstruction Example - Building Intersection



(a) Trees as seen from image 0 (b) Trees from arbitrary location

Figure 5.22: Trees Example 1 (near stadium parking lot). Reconstructed trees as viewed from different orientations.

intersection is fairly smooth with no major gaps or holes in it; this is expected since the alternative is to associate image pixels with background, which is discouraged in the model. As a result, it is possible that near primitives would change their position and orientation to fill in some gaps.

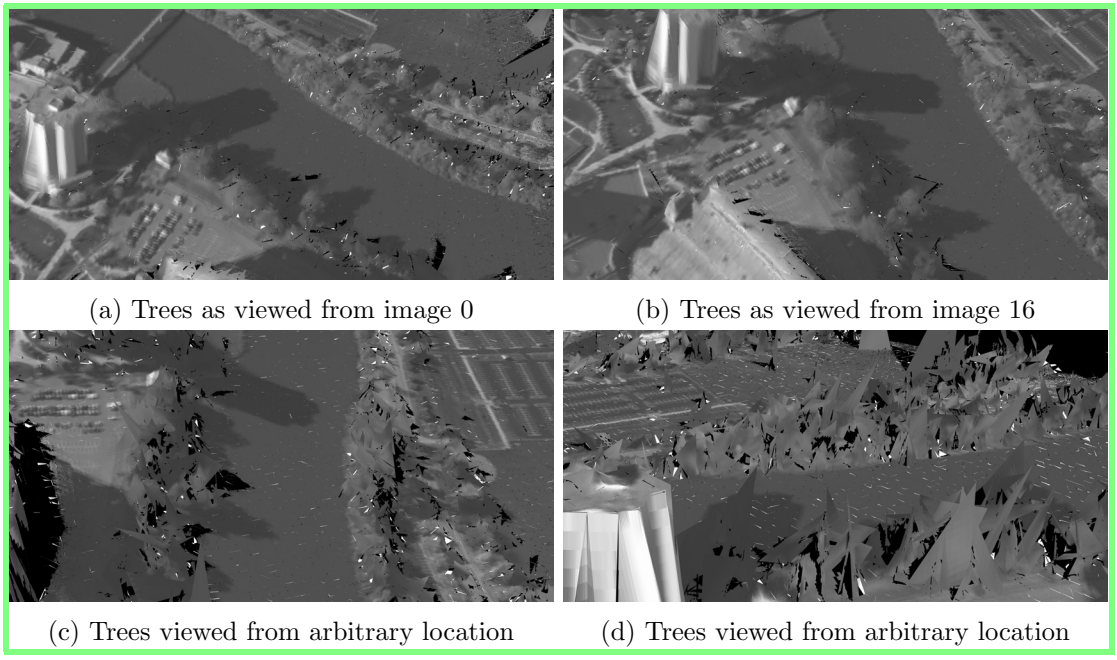
Trees

Trees do not exactly fit within the planar assumptions that the model imposes. Furthermore, LiDAR returns vary significantly near trees, with some returns corresponding to the tree itself (e.g. trunk, branches and leaves), and some others penetrating through leaves and reaching the ground below it. Consequently, association of LiDAR returns in the vicinity of trees is challenging. These two aspects combined yield poor quality reconstruction, cf. Figures 5.22 and 5.23.

The figures demonstrate that when a tree is viewed from positions close to observation views, they appear like trees, since the appearance matches the projection associated with that camera view. However, when the viewing direction is expanded we can see the chaotic nature of the reconstruction. As alluded earlier, the association between latent primitive and LiDAR points is largely responsible for this since the LiDAR likelihood encourages primitives to expand and explain the observations, yielding large primitives which are not suitable for explaining the fine details of foliage.

Failure Cases

Most of the failure cases seen by the model arise from the LiDAR data association, as seen by the stadium field and scoreboard examples, c.f. §4.3 for the challenges of this association problem. Another (extreme) example is presented in Figure 5.24, where we can see that the a primitive on the side of the tower is associated with the LiDAR returns obtained from the light poles in the field (stream of two points seen in the



(a) Trees as viewed from image 0

(b) Trees as viewed from image 16

(c) Trees viewed from arbitrary location

(d) Trees viewed from arbitrary location

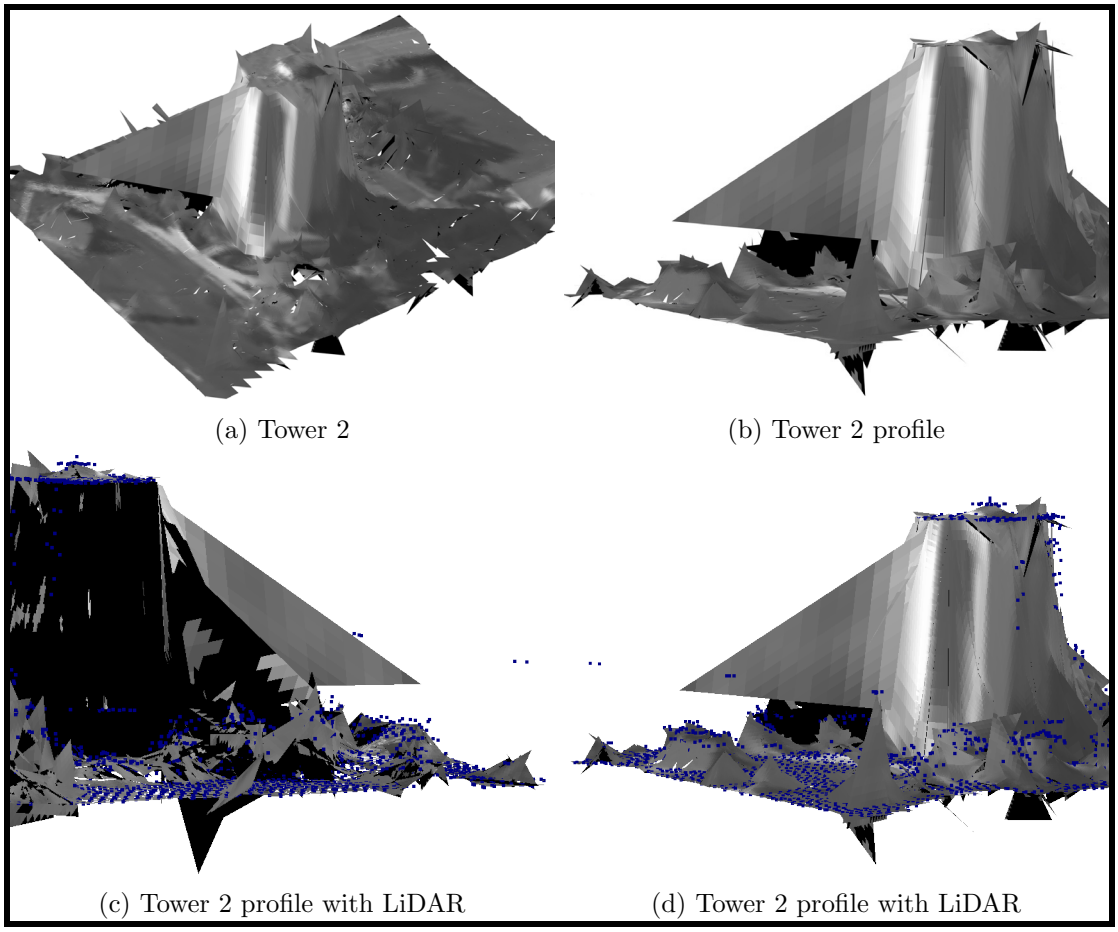
Figure 5.23: Trees Example 2 (near river). Reconstructed trees as viewed from different orientations.

bottom row of the figure). This causes the primitive to grow in the direction of the LiDAR return since this large deviation will incur a very high penalty under the Normal noise model, generally overpowering image likelihood.

We note that while the noise model is the enforcer of the penalty, the real culprit is the approximation of the data association with a static kd tree. The kd tree search for a given primitive associates a larger number of LiDAR measurement with larger primitives. This is the case for this particular triangle. As discussed earlier this choice was made to alleviate computational cost, but it should be re-examined. Despite these failure cases, the model reconstruction quality is adequate.

■ 5.2.3 Camera Parameter Estimation

Camera pose estimation as performed by the proposed model is reviewed in this section. We begin by comparing the model when we include the current image as a texture source for the camera parameter being learned and for the case where we rely on the appearance provided by other observations (cf. §3.4.2). We then proceed to analyze the ability of the model to converge in the form of likelihood capture ranges. We conclude this section with a brief look at the influence of the camera pose prior model.



(a) Tower 2 (b) Tower 2 profile

(c) Tower 2 profile with LiDAR (d) Tower 2 profile with LiDAR

Figure 5.24: Failure Case Example - Tower 2 (southern-most). Incorrect LiDAR association causes primitive to extend to explain measurement.

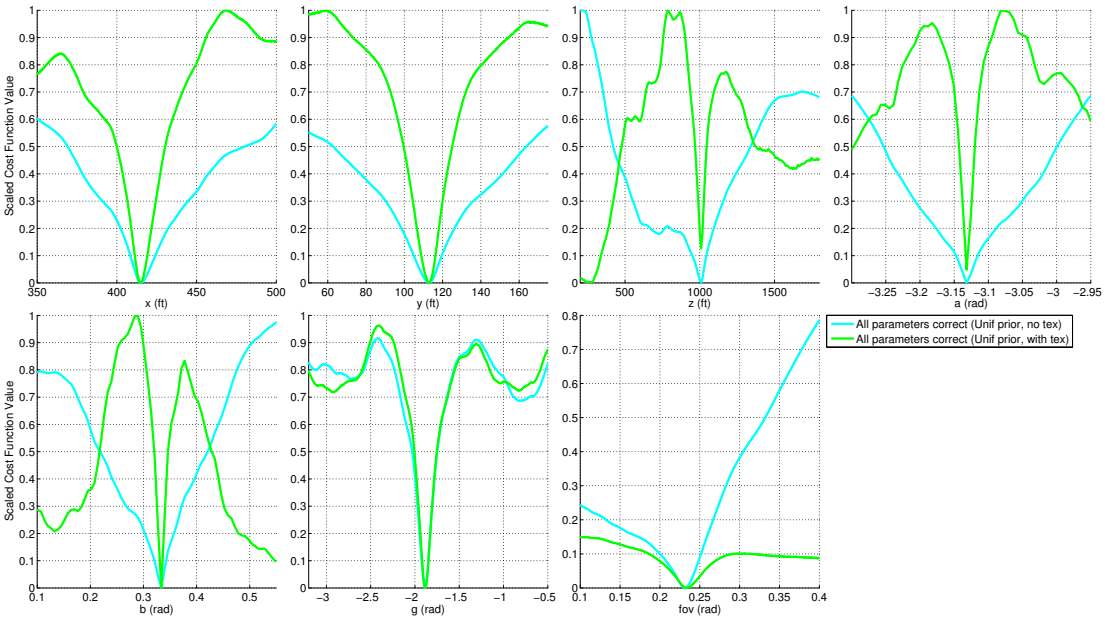


Figure 5.25: Likelihood Comparison with and without current image contribution (green and cyan respectively). Intersection image 0. Both methods have same local optima. Basin of attraction varies for each parameter.

Texture Source Inclusion

As discussed previously, the camera pose computation allows us to either condition on all other observations to generate the latent appearance and use this appearance to learned the current camera pose or simultaneously learn camera pose and latent appearance. This last element has a significant computational cost associated with it, since it requires one to re-compute the appearance for each new camera parameter combination (cf. section 4.1). Due to this cost, we would like to bypass this computation if possible and only condition on the appearance as given by other observations.

To ascertain weather this was possible, we probed the negative log-likelihood equations[4] one parameter at a time for each case (holding the other parameters at their current levels). The results of this parameter sweep can be seen in Figures 5.25 for the Intersection scene. The figure demonstrates that the capture range for all parameters is increased when we do not consider the current image contribution to the appearance. Intuitively this makes sense since by considering the current contribution we are projecting image pixels into world coordinates, mixing them with other image pixels (not necessarily from the same underlying world location) and re-projecting them back to the image frame. This mixing of different pixels cannot improve the capture range and

[4]Conceptually, the likelihood function can be thought of as a cost function that we are trying to optimize, minimize for the results shown in this section.

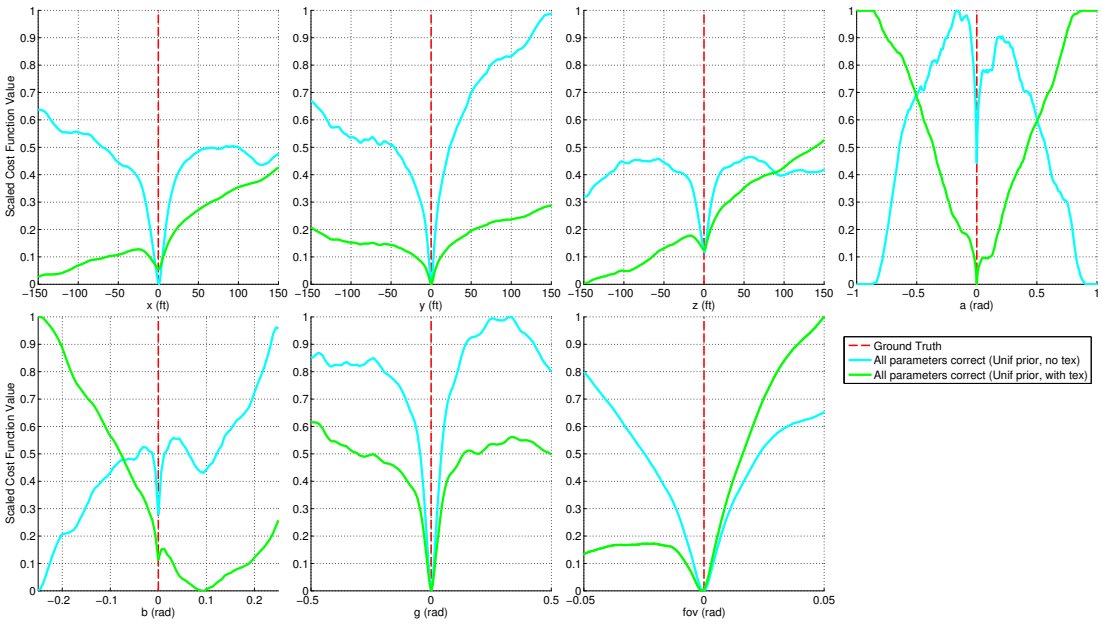


Figure 5.26: Likelihood Comparison with and without current image contribution (green and cyan respectively, ground truth denoted by red vertical line, parameters were translated so that the ground truth appears at zero). Lubbock image 2. Both methods have similar local optima. Capture range varies for each parameter.

causes additional local optima in the parameter space.

On the other hand, the case where we rely on image evidence from the rest of the observations seems to have all the desired characteristics. From the figure we can see that the function is fairly smooth and has the same desired optima as its counterpart. This method seems to work well when there is other image evidence, such as in the intersection case, where 44 other images are imaging the same region. In order to catalog the behavior of the two cases for a limited number of images, we repeated the experiment on the Lubbock scene, which contains only 3 images; the results can be seen in Figure 5.26.

From the figure we see that both methods have local minima at the corresponding ground-truth location. However, we no longer have the expansion of the capture range; in fact, the opposite takes place, the capture range decreases. This somewhat surprising result can be attributed to the wide baseline between images (approximately 90° apart). These large differences coupled with the small number of images available provide little useful information to help align the current image by only considering other images.

Based on the experiments above we can conclude that we can estimate the current camera parameters using appearance contributions from the other images. However, we must be careful in the limiting case where the image baseline is very wide and the number of images is low.

Camera Pose Capture Range

The camera pose capture range, or basin of attraction, refers to the maximum separation between the starting point and the optimum that allows the optimum to be recovered for any given parameter. Informally, the basin of attraction specifies how far away from a true solution one can initialize the model and still reach the desired optima.

We have already seen examples for the basin of attraction in Figures 5.25 and 5.26. By looking at those two figures we can see the significant difference in the capture range between the two scenes, which varies depending on the number of images, the baseline between the images and the scene geometry. As an example, for the intersection in the CLIF data set, the inference procedures converges to the correct altitude (or close to it) if the initialization is within 500 feet as compared to the Lubbock data set for which the initialization must typically be within 50 feet.

Another interesting question that arises, is how does the capture range change as we get closer to the optimum? In order to address this, we looked at the basin of attraction after a given number of updates to all other camera parameters, cf. Figure 5.27. The figure shows the capture range after updating all other observation (referred to as a batch) once, five, and in increments of five until thirty batches was reached. One key observation can be made from the figure: the capture range does not change but the location of the local optimum does. This observation can be decoupled into two parts, first the capture range is static, meaning given the current state of the latent parameters at the time of a batch the local optimum can be reached starting from the same distance each time. This is important and desired since it provides stability to the solution, that is, starting within a given distance is likely to provide the correct minimum.

Furthermore, as seen from the figure, the location of the local minimum changes in systematic manner as the batches increase –e.g. always increasing for the x dimensions, or decreasing the y dimension. This change is the parameter change produced by the last batch (for both the current camera and the other cameras). The systematic changes, along with the decrease in magnitude (as seen from the figure from the curves getting progressively closer), further indicate the presence of a local optima and the ability to reliably converge to it. In addition, we can see that the use of coordinate descent techniques will provide the desired local optima after a sufficient number of batches.

Prior Model Comparison

As mentioned earlier the model is likelihood (or data) dominated. This is primarily the case for camera extrinsic parameters, where the influence of the Gaussian Process Prior quickly diminishes. We compare the effect of the GP prior with an improper uniform prior model in Figure 5.28 for four images of the CLIF Stadium sequence. The Figure shows that the inference algorithms achieve similar parameter values independent of which prior distribution is used after about thirty batches. The exception to

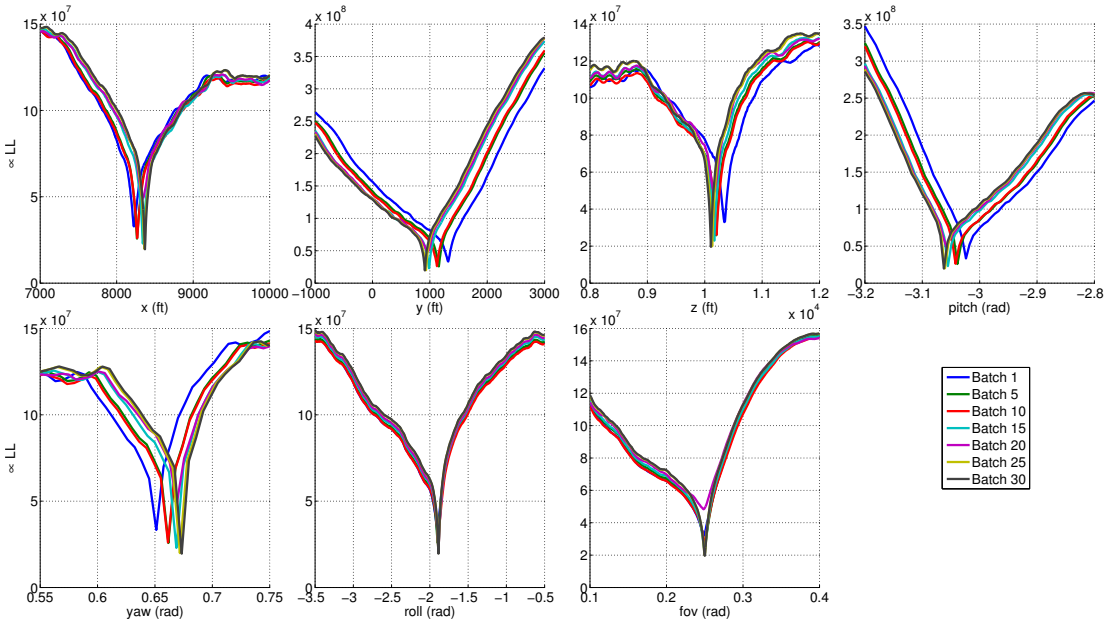


Figure 5.27: Parameter capture range after optimizing all images: once, five, ten, ..., and thirty times. Notice the basin of attraction remains constant across batches. (CLIF Stadium Image 0)

this is the altitude for which after thirty batches the optimization has not converged; convergence occurred after forty-five batches. We note that the smoothness of the Gaussian Process prior cannot be determined from figure 5.28 since the GP prior encourages smoothness across multiple images, not across different batches for the same image. This experiment shows that the local optima found are insensitive to the choice of prior model.

■ 5.2.4 Appearance and Geometry trade-off

In this section we discuss the inherent trade-off between appearance and geometry information present in the proposed model. This trade-off occurs when considering the information that each type of observations, images and LiDAR, provide about the world. In these terms, LiDAR provides information primarily about geometry, while images provide information primarily about appearance. While each type of observation has a dominant information type, the other type is still present, for example material properties, and hence appearance information can be inferred from LiDAR measurements; and geometry can be estimated from image correspondence.

In this section we will qualitatively characterize the model performance as we vary the number of observations of each type of measurement. By varying the number of measurements we will be able to witness the interaction between the two information types. The main goal is to characterize the reconstruction quality as the number of

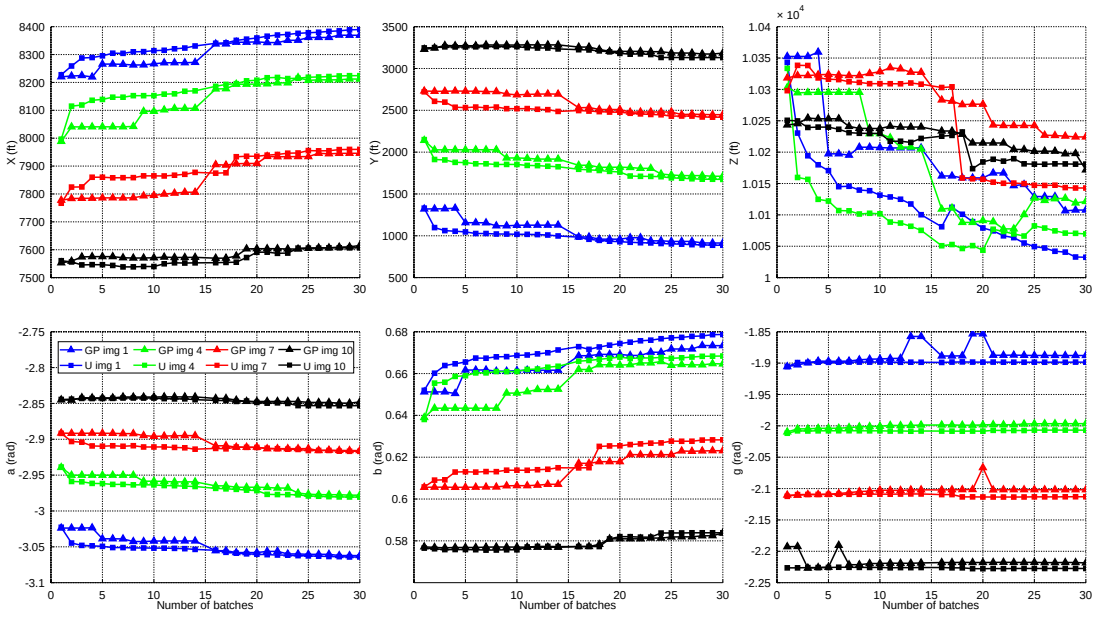


Figure 5.28: Parameter Change for Uniform and GP prior as a function of number of optimization runs. Approximately the same local optima is found after about 30 batches independent of which prior probability model is used. (CLIF Stadium)

LiDAR/image measurements vary. In the remaining of this section we discuss the result of two experiment that try to answer these questions: in the first, we varied the number of images while maintaining the number of LiDAR measurements constant; in the second, we varied the number of LiDAR measurements while maintaining a constant number of images.

Reconstruction as a function of Images

In this experiment, we varied the number of input images from 49, 20, 10, 5 to 2 images for the CLIF Image stack, while maintaining all other parameters the same. In other words, camera initialization (via GPS) and latent geometry was the same for all image configuration (initial configuration, i.e. GPS initialization, can be seen in figure 5.29a for the 49 image case). The images selected for each configuration were chosen to maximize separation between images. As an example, the first and last images in the sequence were chosen for the two-image experiment, while one image was selected every other five in the ten-image experiment.

We optimized over the camera pose for each camera in each set. Each set of images was optimized with the same procedures consisting of four runs:

- Run 1: Standard Update, GP prior, image noise $\sigma = 10$, ten batches.
- Run 2: Standard Update, GP prior, image noise $\sigma = 5$, five batches.

- Run 3: Standard Update, GP prior, image noise $\sigma = 2$, five batches.
- Run 4: Reduced optimization step, GP prior, image noise $\sigma = 5$, five batches.

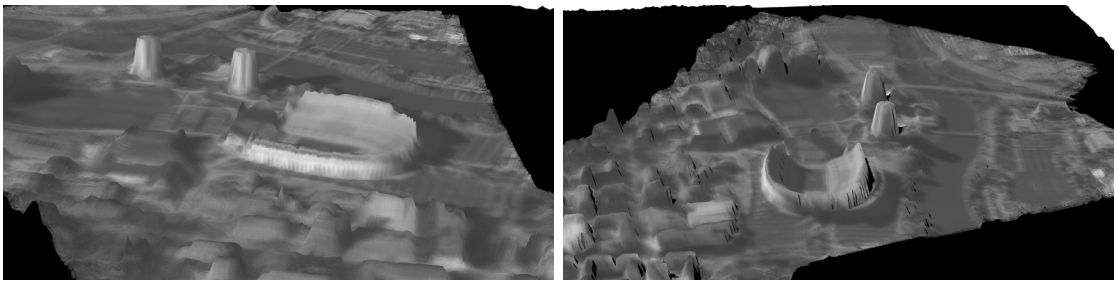
The parameters for each run indicate the number of times we cycled through the images, i.e. ten times in run 1 and the image noise standard deviation used. We note that varying the image noise level does not have any noticeable difference in the reconstruction output. On the other hand decreasing the number of batches could have significant impact, mainly not enough iterations to converge to the optima.

The results of this experiment for the first three sets can be seen in Figure 5.29. The Figure shows two views of the reconstructed world for each set. From the figure we can see that the quality of the reconstruction only marginally degrades when reducing the number of images from 49 to 20. This degradation appears in the form of spotty black pixels in the reconstruction. These black pixels occur as a result of missing information in the latent appearance model, i.e. no observation pixel mapped to that particular appearance location. If we further reduce the number of images from 20 to 10, the degradation further increases; this is expected since we are reducing the number of observation pixels. To see why this is the case consider the top right image section of the first image on the bottom row of figure 5.29, where the change in inferred appearance can clearly be seen as the number of image that observe the region changes from one to three.

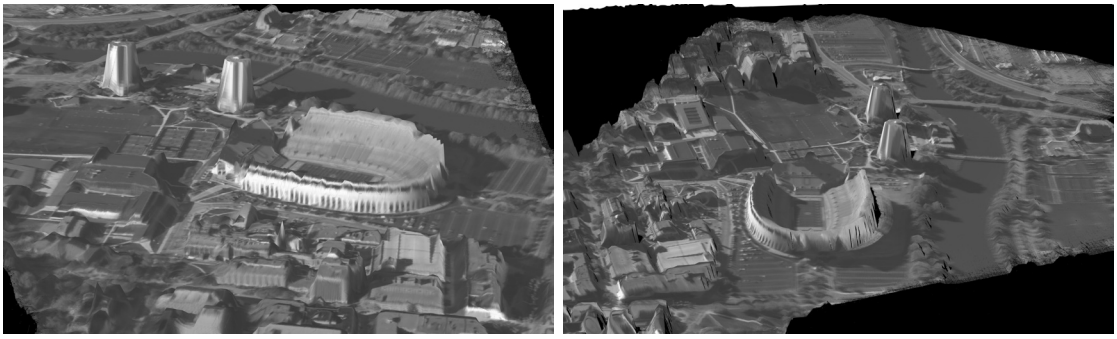
We note that while distracting this deterioration is minor and can be easily reduced by either increasing the number of appearance pixels each observation pixel is allowed to influence (four for these experiments) as discussed in §4.1 or by post-processing the appearance maps to fill in the holes, either with simple neighbor fill or via smoothing filtering operations.

Despite the artifacts mentioned earlier the three sets of images converge to similar configuration of camera parameters, and are aligned well with each other, (as seen by the sharp features in Figure 5.29). If we continue to decrease the number of images to 5- and 2-images, the same does not hold as seen by 5.30a for the 5-image case. The figure shows inconsistent appearance, caused by incorrect camera parameters. The limited image evidence with wide baseline coupled with the high uncertainty in camera pose starting parameter causes the optimization to converge to a local optima. In order to combat this we increased the number of runs performed, from four to eight, but this did not help. We note that this example follows under the category described in §5.2.3, where the capture range for the correct optima is fairly small and is thus hard to find.

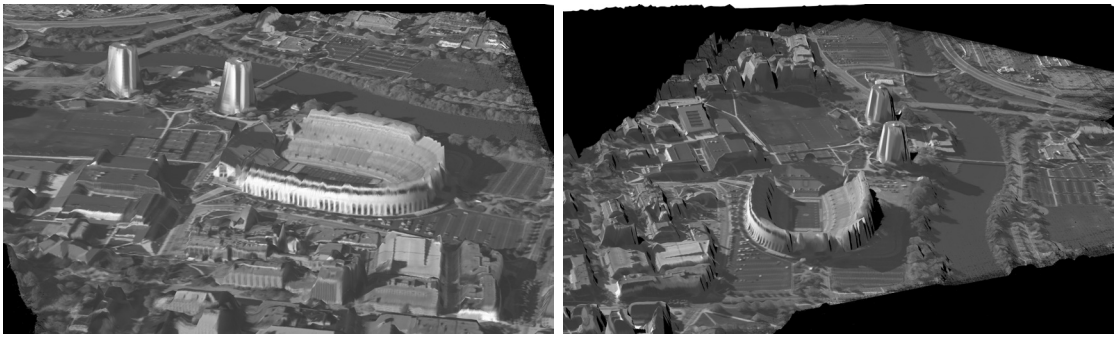
As an interesting result, since we had already obtained the camera pose for the prior sets, which included all the image used in the 5- and 2-image set, we can compute the empirical difference between starting and ending camera poses. This allows us to run experiments in which we can vary the starting position, i.e. obtain a fictitious initial GPS measurement and adjust the GPS noise accordingly. The results of running one of such experiments on the five-image case can be seen in Figure 5.30b. We can estimate that if the noise in the starting position (GPS sensor noise) is cut in half, we are able



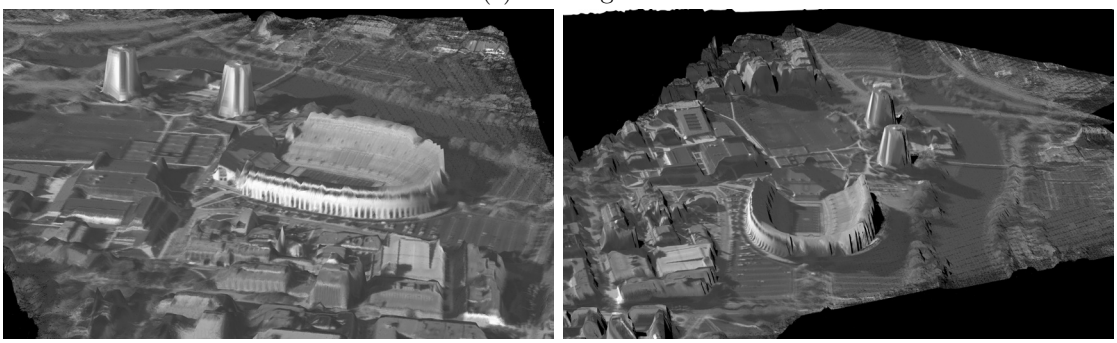
(a) Initial Configuration



(b) 49 images

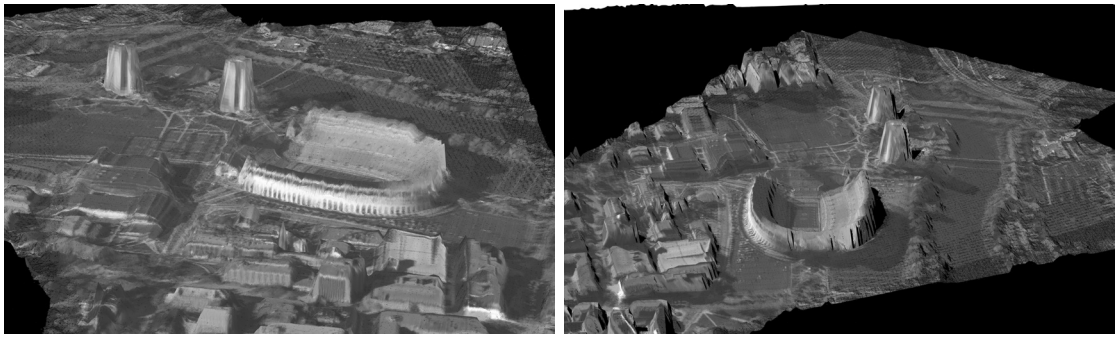


(c) 20 images

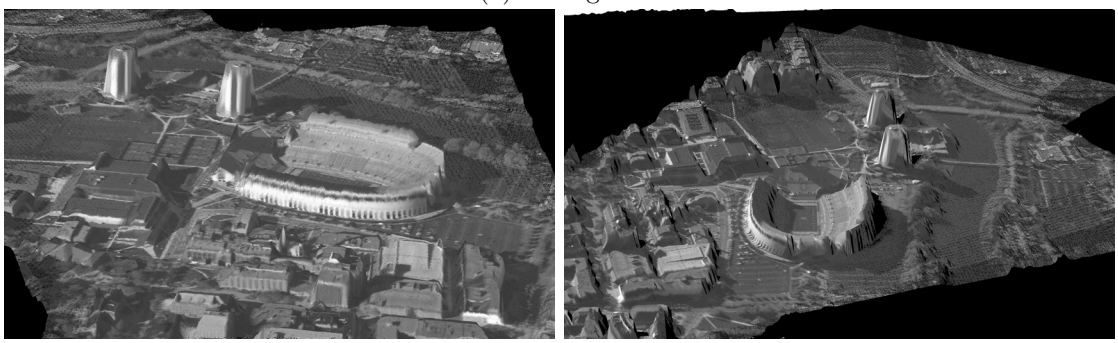


(d) 10 images

Figure 5.29: *Top row:* Initial Parameter configuration for appearance reconstructions using all 49 images. *Other rows:* Reconstruction as a function of number of input images (fixed world geometry and LiDAR, same number of optimizations run for all).



(a) 5 images



(b) 5 images (half noise)

Figure 5.30: Reconstruction as a function of number of input images, two views. (a) original noise level; (b), noise level reduced by a factor of two. (fixed world geometry and LiDAR, same number of optimizations run for all)

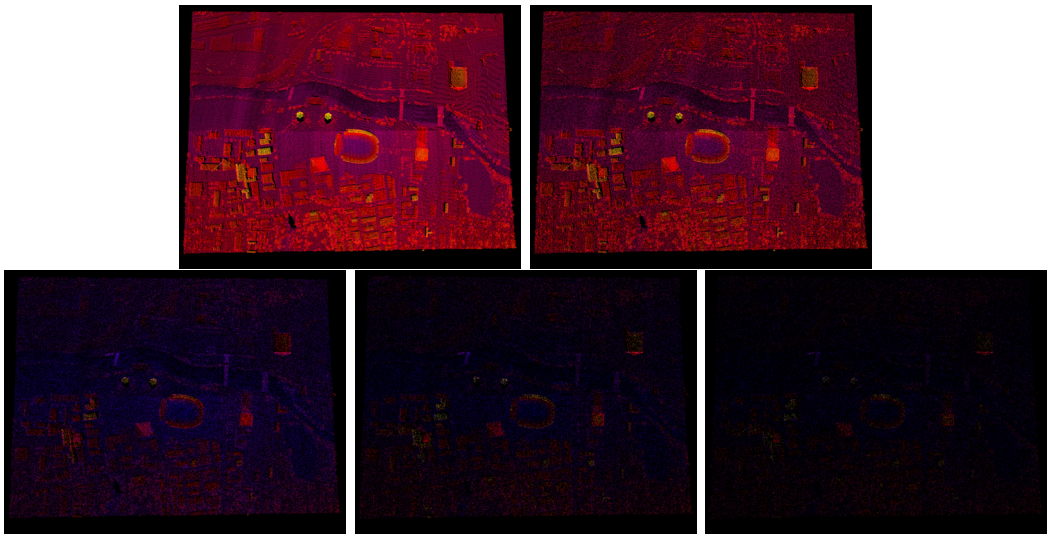


Figure 5.31: LiDAR density, 700k, 500k, 200k, 100k, 50k respectively.

to recover a good set of camera parameters, for better pose estimates we should further reduce the noise level.

Reconstruction as a function of LiDAR measurements

In order to further characterize the appearance geometry trade-off we analyzed the reconstruction obtained by the proposed model as we varied the number of LiDAR measurements (while maintaining a constant number of images). The number of LiDAR measurements in the sets varied from 700,000, 500,000, 200,000, 100,000 to 50,000 by randomly removing measurements from the previous set of measurements; the LiDAR measurements clouds can be seen in Figure 5.31. As before, we maintained all other parameters fixed, including the initialization and the number of images. The latent geometry for each of the sets was inferred by running a single iteration of the multi-plane geometry estimate.

The results of this experiment can be seen in Figure 5.32. From the figure we can see that quality of the results is fairly constant for the first three sets (700k-200k measurements), and differs for the remaining cases. We note that as the number of measurements decreases, the number of measurements associated with each primitive decreases, as a consequence, LiDAR has a smaller influence on the parameter estimate. In addition, some primitives will have no LiDAR measurements associated with them and their parameter estimates would be based on image evidence only.

Furthermore, as the number of observation decreases, the chance of incorrect data associations also decrease, for example the incorrect plane on the side of one of the towers (cf. Figure 5.24) is no longer present in the 300k and lower experiments since the lamp measurements are not in the LiDAR set. Typically these type of miss-association

Algorithm 5.1 Geometry Primitive Initialization

1: Divide LiDAR point cloud into a grid of $M \times M$ elements.
2: Allocate memory for new sub-sampled point cloud, consisting of M^2, 3D points.
3: **for** $m = 1 : M^2$ **do**
4: Obtain the mean height of each LiDAR measurement inside grid element m.
5: Save the 3D point in the sub-sampled point cloud using the grid element's center and mean height.
6: **end for**
7: Obtain initial primitive estimate by triangulating (e.g. Delaunay Triangulation) the grid element center and projecting them to their mean height.

causes the biggest change in parameter estimates due to the model's need to explain every measurement, including these outliers.

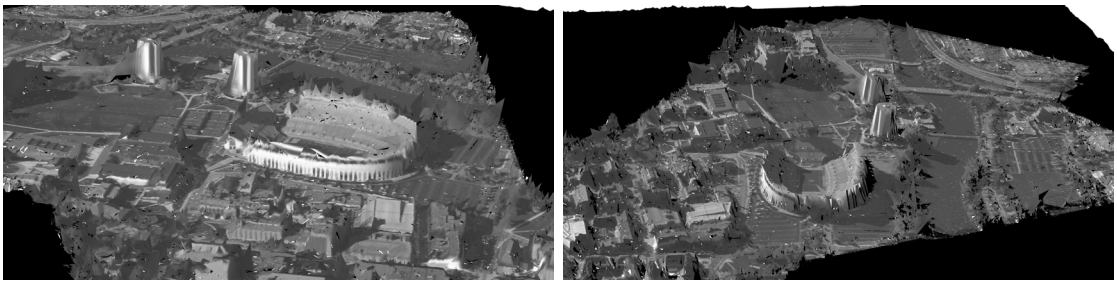
■ 5.2.5 Geometry Initialization

All results disused so far have had the same world geometry initialization. This initialization consists of quantizing the 2D projection of LiDAR measurement cloud into a grid of specified size, see algorithm 5.1. This initialization has several advantages, first it allows us to control the number of primitives used; secondly it creates mostly right triangles of the same size since the sub-sampled point cloud is in grid shape –only triangles on the boundary of significant height changes deviate from this.

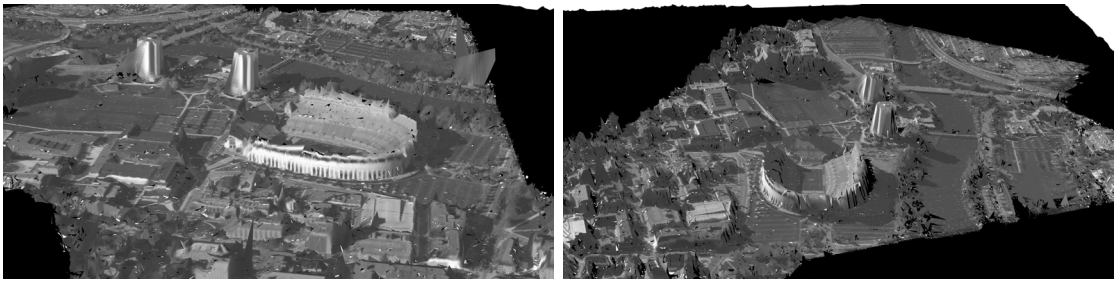
As we have seen so far, this initialization performs well but it has some disadvantages, namely there is no overlap between triangles, so a slight shift in their parameters produces background. Furthermore, because of the averaging occurring in grid elements it is unlikely that the plane produced for the grid fits exactly the LiDAR measurements (the exception to this is when there is a single measurement in the element), this further reinforces the earlier observation that most of the likelihood improvement in the geometry estimation comes from the LiDAR likelihood.

In order to avoid these disadvantages, and test the ability of the model to reconstruct a variety scenes we attempted a different initialization. This alternative initialization consisted of assigning a geometric primitive to each LiDAR observation as seen in Figure 5.33. As can be seen from the figure each plane is centered on a LiDAR observation, and the plane orientation was chosen by fitting a local plane to the set of neighboring measurements, 5 neighbors throughout this work. Furthermore the size of the primitive was chosen to produce some overlap between neighboring primitives. In contrast, to the Delaunay initialization, the resulting primitives do not share boundaries. While this property is not preserved during the inference procedure, it does reduce initial ambiguity as to which primitive explains which observation pixel, c.f. Figure 5.34.

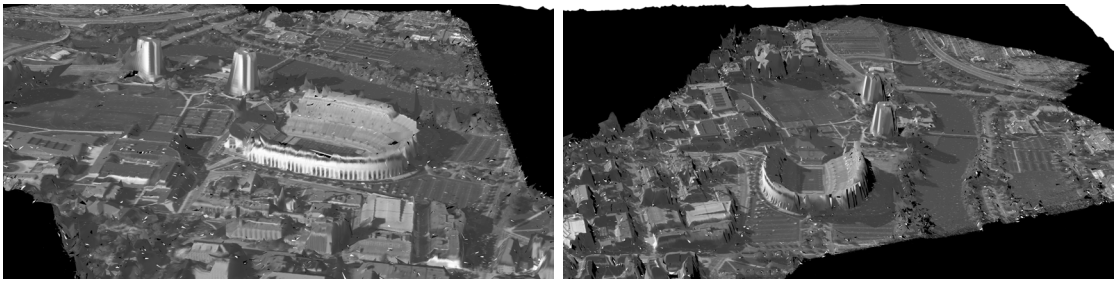
The latent primitives obtained through this initialization were updated as before. The results of the updates can be seen in Figure 5.35. At first glance it appears that the geometry update was not very successful, but looking closer, especially from image



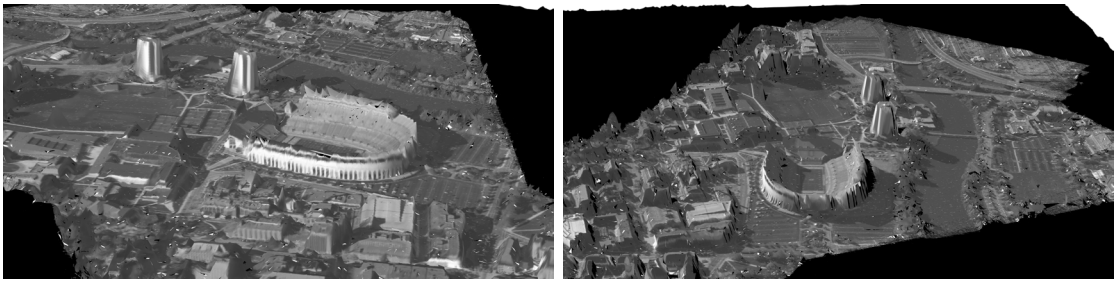
(a) 700k measurements



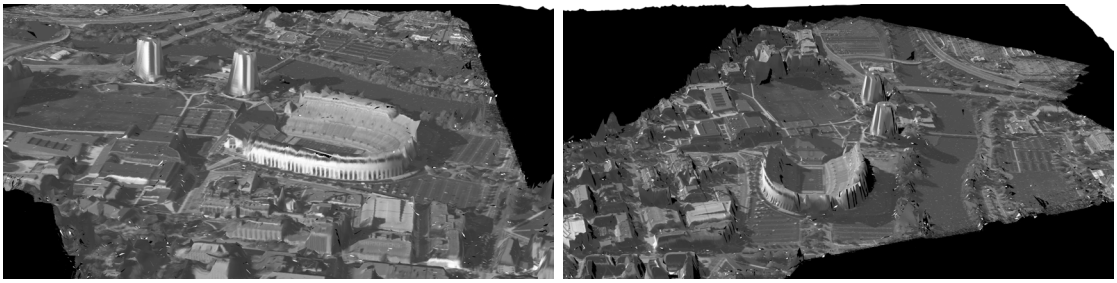
(b) 500k measurements



(c) 200k measurements



(d) 100k measurements



(e) 50k measurements

Figure 5.32: Reconstruction as a function of LiDAR measurements (fixed number of images)

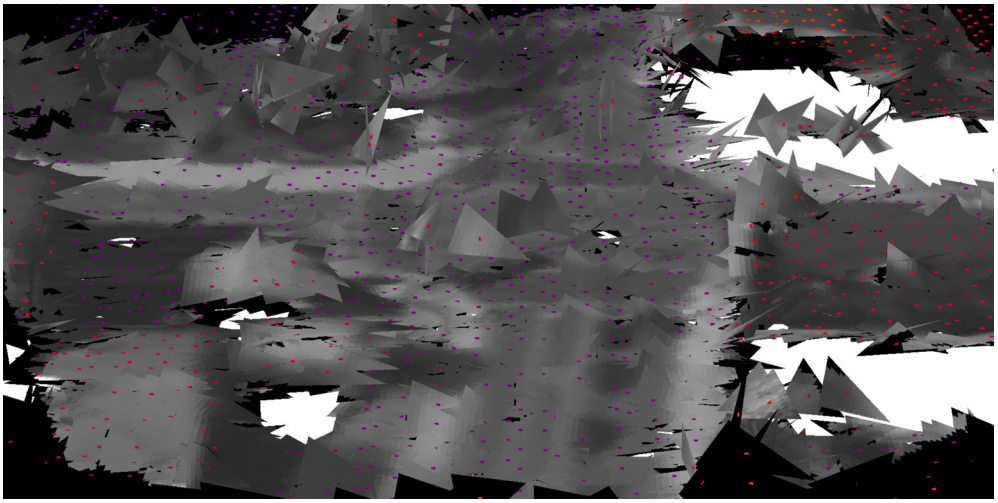


Figure 5.33: Unique planes per LiDAR measurement, each plane centered at its corresponding measurement.

perspective (as the top image in the figure), we can see that the previously empty space has now been filled, i.e. planes have expanded and/or moved to fill in the background regions. This is not surprising for a few reasons: first, by construction the initial LiDAR likelihood is as good as it can be, with the distance between each point and a plane being zero. This means that initially only the image likelihood can improve, and since the model slightly penalizes background the main gain stems from reducing the number of background pixels.

This reduction of background pixels can come at a cost as seen by the oversize primitive in the top left corner of the intersection in Figure 5.35 . In this case, the inference procedure covered the background in the region since the image likelihood dominated the size penalty and the LiDAR likelihood. This is the typical trade off behavior of the model.

To summarize, this section shows that the results obtained with the proposed model are initialization dependent.

■ 5.3 Structure from Motion Comparison

The goal of this section is to compare the performance of the proposed model to well established structure from motion techniques. Results of the proposed model will be compared to results obtained using the implementation of Snavely *et al.* [50] (i.e., Bundler) and the post-processing method of Furukawa *et al.* [23] (i.e., PMVS2). We will be comparing three scenes on reconstruction quality and computation time.

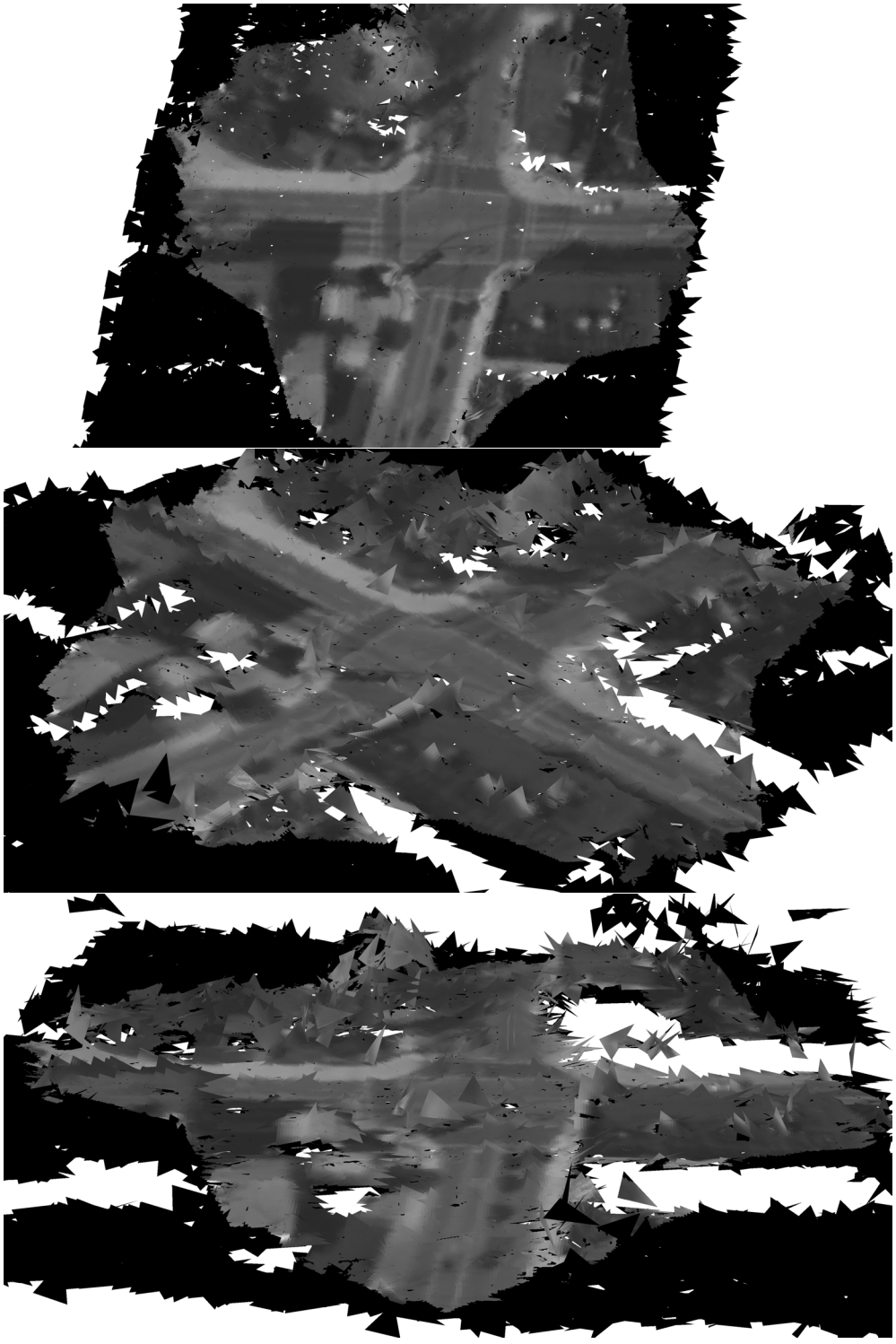


Figure 5.34: One plane per LiDAR measurement initialization (planes with no texture appear black, background is white).

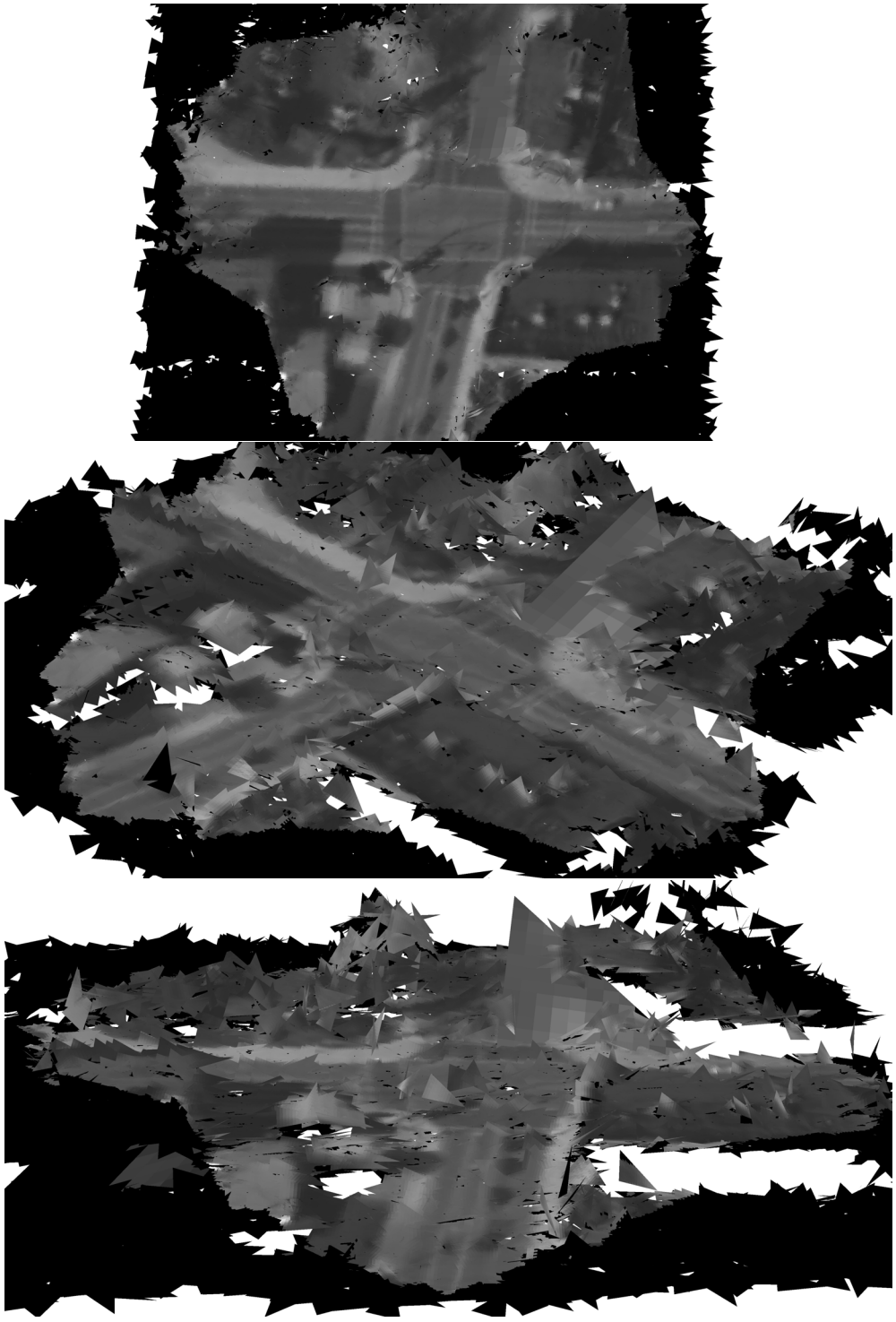


Figure 5.35: LiDAR initializing after updating all planes once (planes with no texture appear black, background is white).

Figure 5.36: Bundler reconstruction of CLIF Intersection - Ortho view (1,715 points)

5.3.1 Reconstruction Comparison

Reconstruction comparison will be based on the CLIF scenes, namely the Intersection, the Stadium Image Stack and the Stadium Only datasets. For each of these datasets we ran Bundler followed by PMVS with their default parameters. The features used for Bundler were SIFT features [37]. We note that these reconstructions are fundamentally different, both in intent and approach, for example we are interested in recovering higher order primitives, triangles, while Bundler reconstructs a sparse set of points. Due to such variations we are only considering a qualitative comparison.

CLIF Intersection

We begin the comparisons with the intersection scene. The results after running Bundler can be seen in Figure 5.36. We note that Bundler produces a sparse scene, with only over 1500 points. However, despite the high sparsity we are able to see the basic outline of the intersection in the ortho view, where three out of the four corners are recovered and the sidewalk markings can be clearly seen.

The results after running PMVS can be seen in Figure 5.37. From the three views we can see that PMVS expands the output of Bundler, increasing the reconstructed points to over 12,000, covering the entire visible area. Furthermore we can see from the views that the flat horizontal surfaces such as the street and rooftops are well reconstructed, and maintain their flat nature. The sides of buildings on the other hand do not always maintain their vertical orientation and are at times slanted.

In terms of appearance each point is colored with the corresponding image pixel values, which leads to a reasonable distinction between road and sidewalk for example. From the bottom image of Figure 5.37 we can see that there are a number of outliers

in the figure, easily seen for points with heights above building levels. Despite these outliers the reconstruction is fair.

Reconstructions for this scene using the proposed method can be seen in Figure 5.38. From the figure we can see that the reconstruction is dense, and that the appearance of the scene is easier to distinguished since appearances are based on image patches, rather than single pixel colors. In terms of geometry we can see that the reconstruction of horizontal surfaces are comparable to PMVS. Similarly to Bundler the reconstruction of vertical features can be slanted and not purely vertical, c.f. the top right intersection corner (same in all images). Furthermore the sharp planar discontinuities on trees and sides of buildings can be visually displeasing.

CLIF Stadium Image Stack

The Bundler reconstruction for the Stadium image stack was fairly sparse for the size of this scene, over 3,000 points, which makes it difficult to distinguish any features. As a result we will begin the discussions with the PMVS results, cf. Figure 5.39. The PMVS results are dense with over 77,000 points covering the entire scene. The collection of points let us clearly see the underlying scene structure, from the Stadium to the buildings surrounding the stadium. Again the horizontal surfaces are very well reconstructed, leading to excellent ground coverage.

Building sides are fairly dense and vary from being highly vertical to more slanted. As examples of these we can use the front wall of the stadium (as seen in the top image), and the towers. As before there are a few reconstruction errors, the most noticeable are the severe slant in the towers and the outliers in the back wall of the stadium.

Reconstruction of the scene using the proposed method can be seen in Figure 5.40. From the images we can see that as before the use of patches allows a clear scene interpretation, and allows rapid and easy identification of scene parts. Taking a closer look at the images we note that horizontal surfaces are well reconstructed and fine details such as parking lines and cars can be easily distinguished. The performance of our algorithm on side of buildings varies, with some sides being highly vertical, such as the buildings in front of the stadium), and some being vertical but oversize, such as the buildings behind the stadium in the middle image.

As pointed out earlier, the performance of the algorithm on trees is less than ideal, leading to the highly peaky areas around the river. There are other interesting artifacts visible such as the incorrect data association example of Figure 5.24 in the side of the tower.

CLIF Stadium Only

The Stadium Only dataset was run through Bundler and PMVS, however, the reconstructions obtained are mirror inverses of the underlying scene. The Bundler reconstruction can be seen in Figure 5.41, containing over 25,000 points. This reconstruction appears to be mirrored, so that the head of the stadium (where the scoreboard is) appears in the bottom of the figure, where it should be at the top.

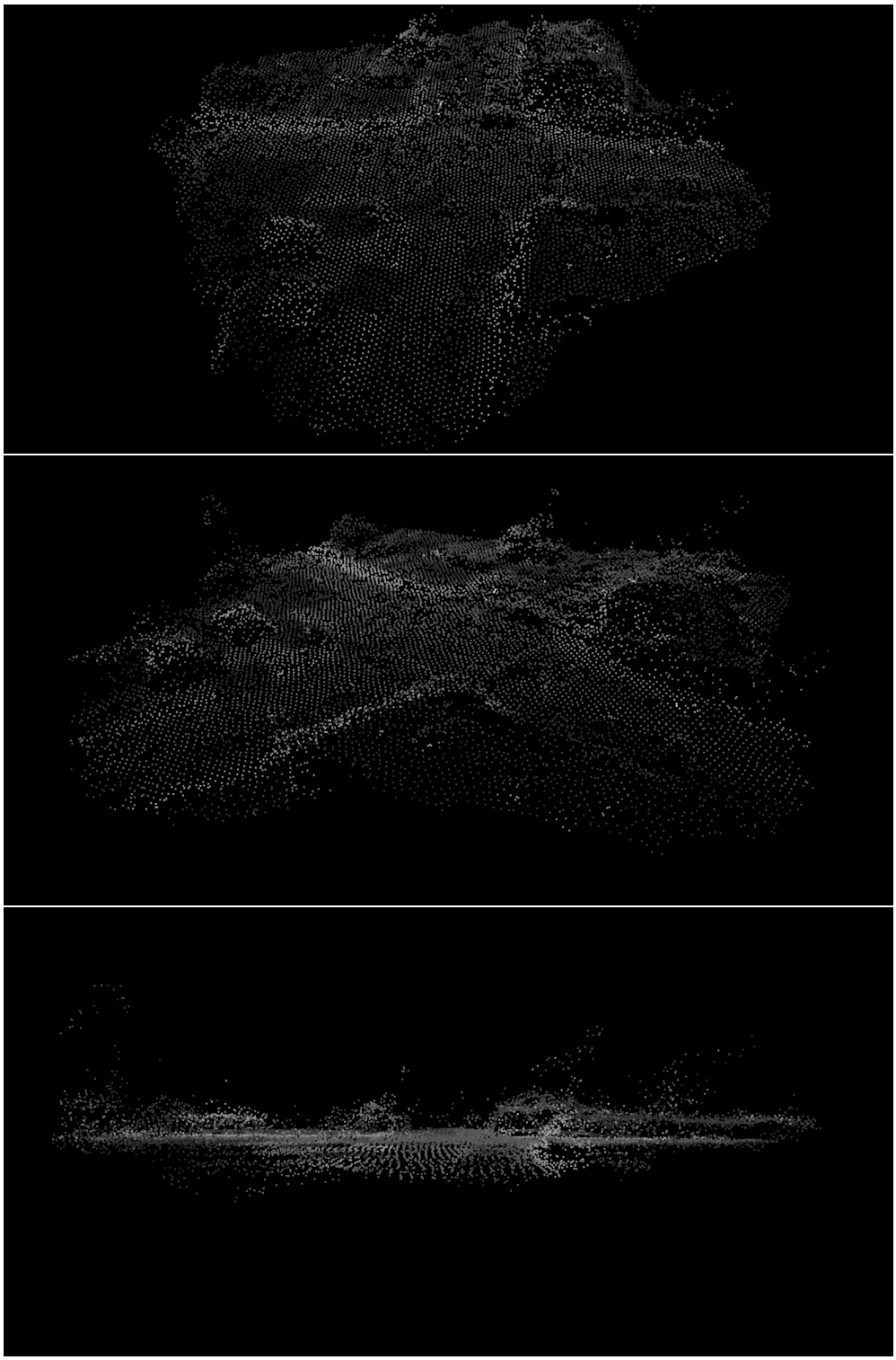


Figure 5.37: PMVS reconstruction of CLIF Intersection, 3 views. (12,208 points)

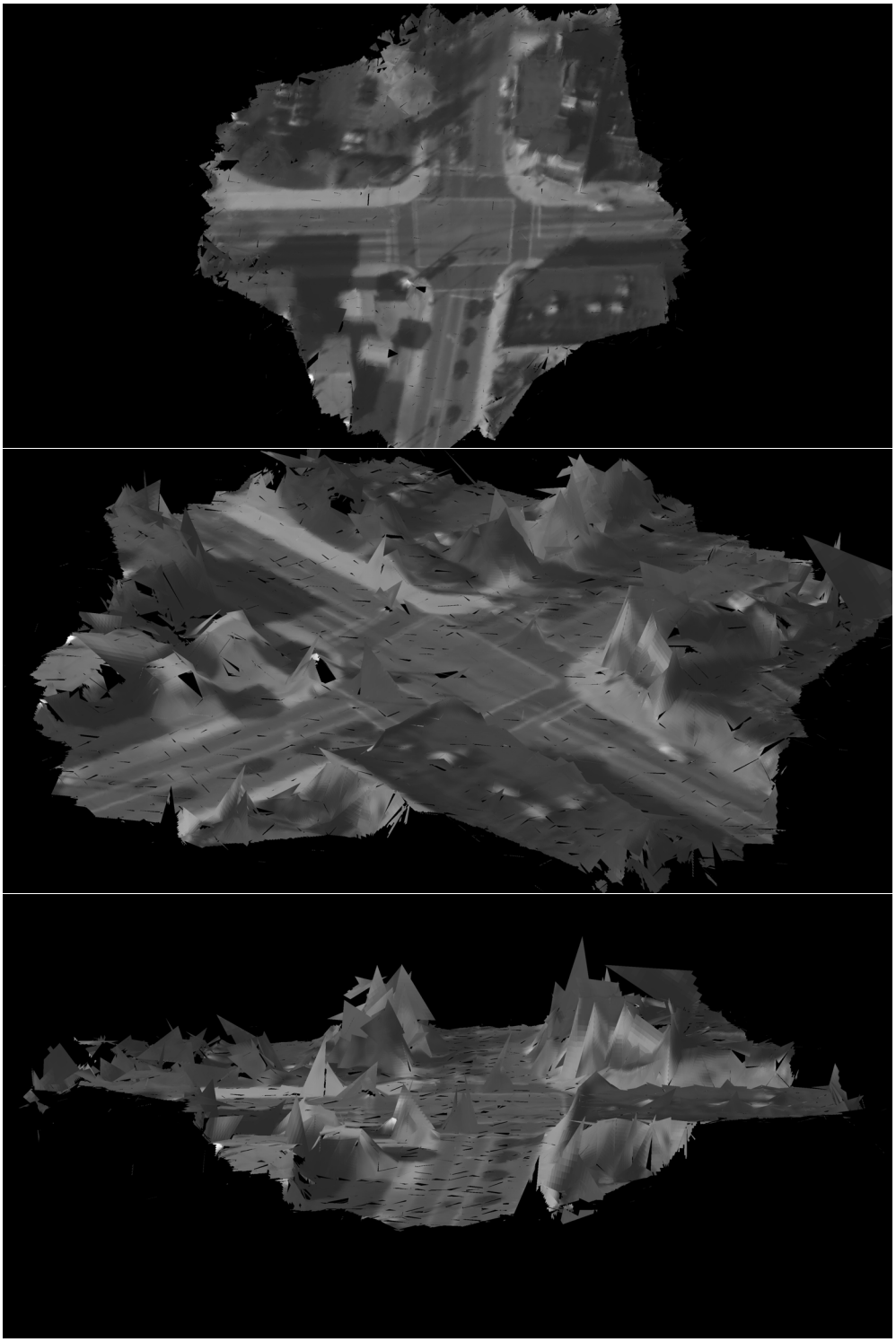


Figure 5.38: Reconstruction of CLIF Intersection using proposed algorithm, 3 views. (3,490 visible planes)

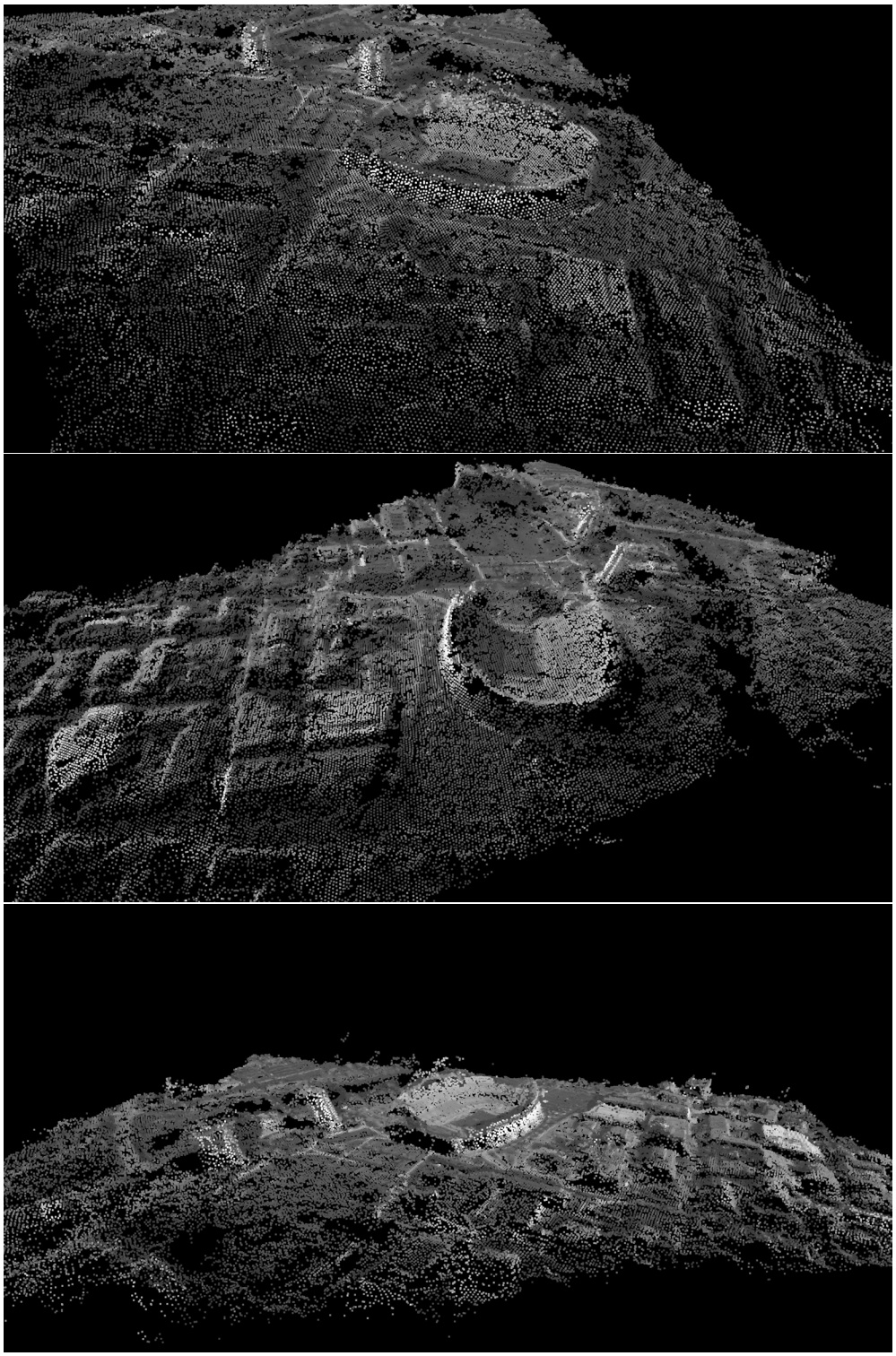


Figure 5.39: PMVS reconstruction of CLIF Image Stack, 3 views. (77,552 points)

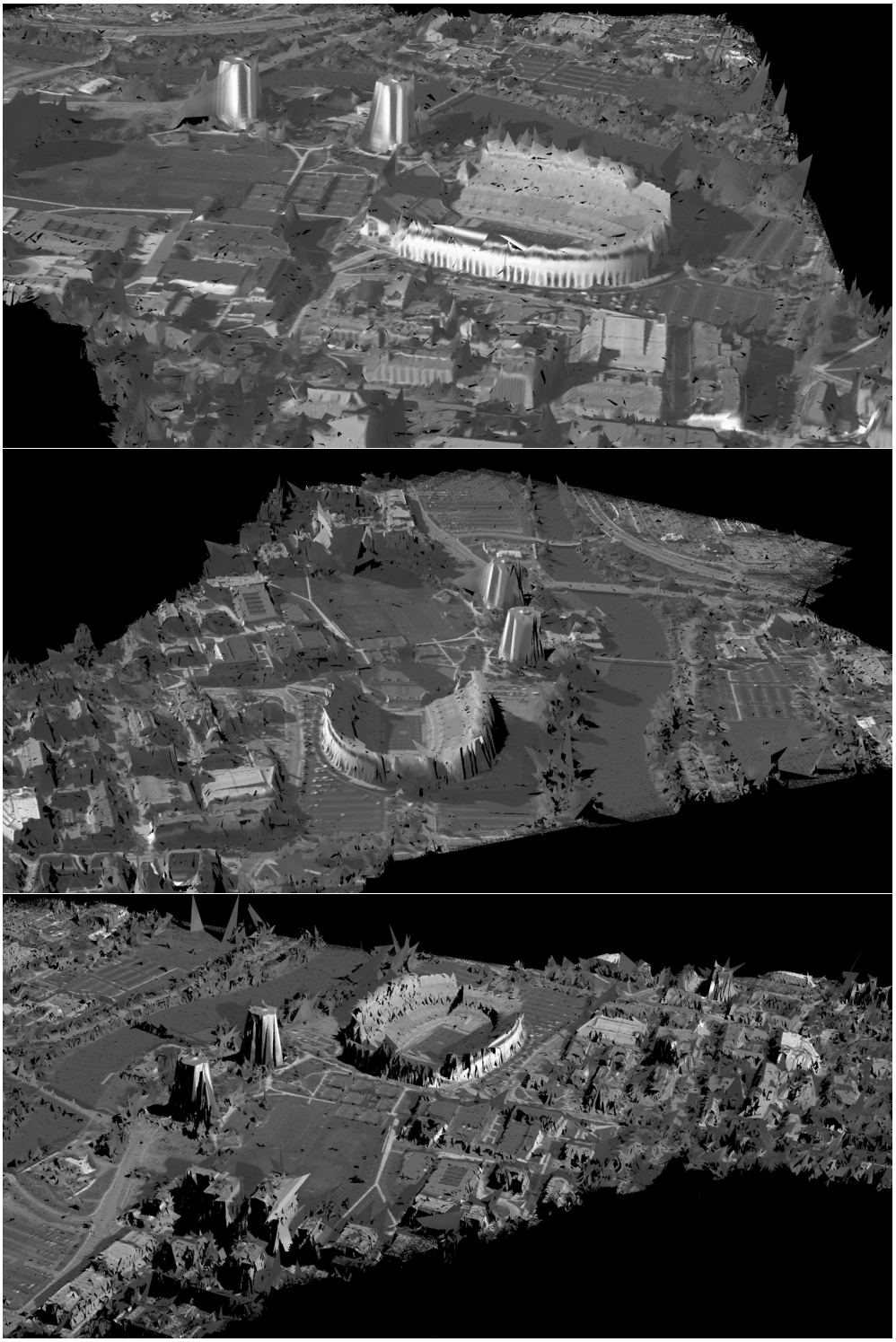


Figure 5.40: Reconstruction of CLIF Image Stack using proposed algorithm, 3 views. (81,573 visible planes)

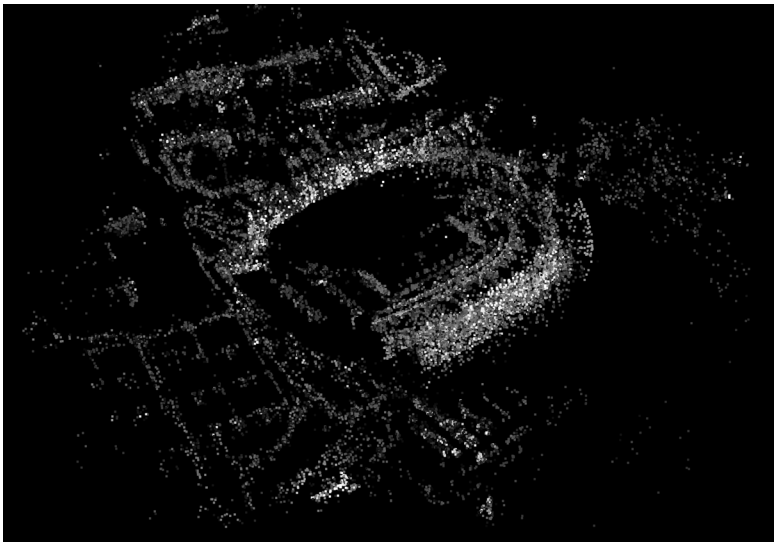


Figure 5.41: Bundler reconstruction of CLIF Stadium Only. Oblique view, note that the reconstruction is inverted (25,134 points)

Running PMVS, further confirmed that the scene is mirrored, cf. Figure 5.42. Closer look at the images reveals that the scene is inside out, meaning what we are perceiving to be the up direction of the reconstruction is actually the bottom. The PVMS results are very dense with over 156,000 points and allow very detailed account of ground plane, including roads and other interesting details.

Our reconstruction can be seen in Figure 5.43. As in the previous two cases, we can use the triangular reconstruction to model very accurately the scene details in planar regions. For example the street markings and parking lot lines can easily be identified from the reconstruction. Furthermore, all sides of the stadium are well reconstructed, including those in the shadow of the front wall. Other than the trees next to the river, this reconstruction is commendable.

■ 5.3.2 Computation Time

In this section we discuss the computational complexity in terms of running time for Bundler+PMVS; and the model presented in this thesis. The run time of Bundler+PMVS can be seen in Table 5.1. From the table we can see that anywhere between 26% to 72% of the time Bundler+PMVS is either finding or matching keypoints. The scene parameter optimization takes the remaining portion of the time. Overall, the computation time for each of the scenes is quite low.

The run time for the proposed model can be seen in Table 5.2. From the table we can see that the algorithm spends a significant amount of time computing scene geometry. These high run-times are mostly due to the large number of visible primitives in the scene, ranging from 3,000 to 80,000. We further note that on average the time per

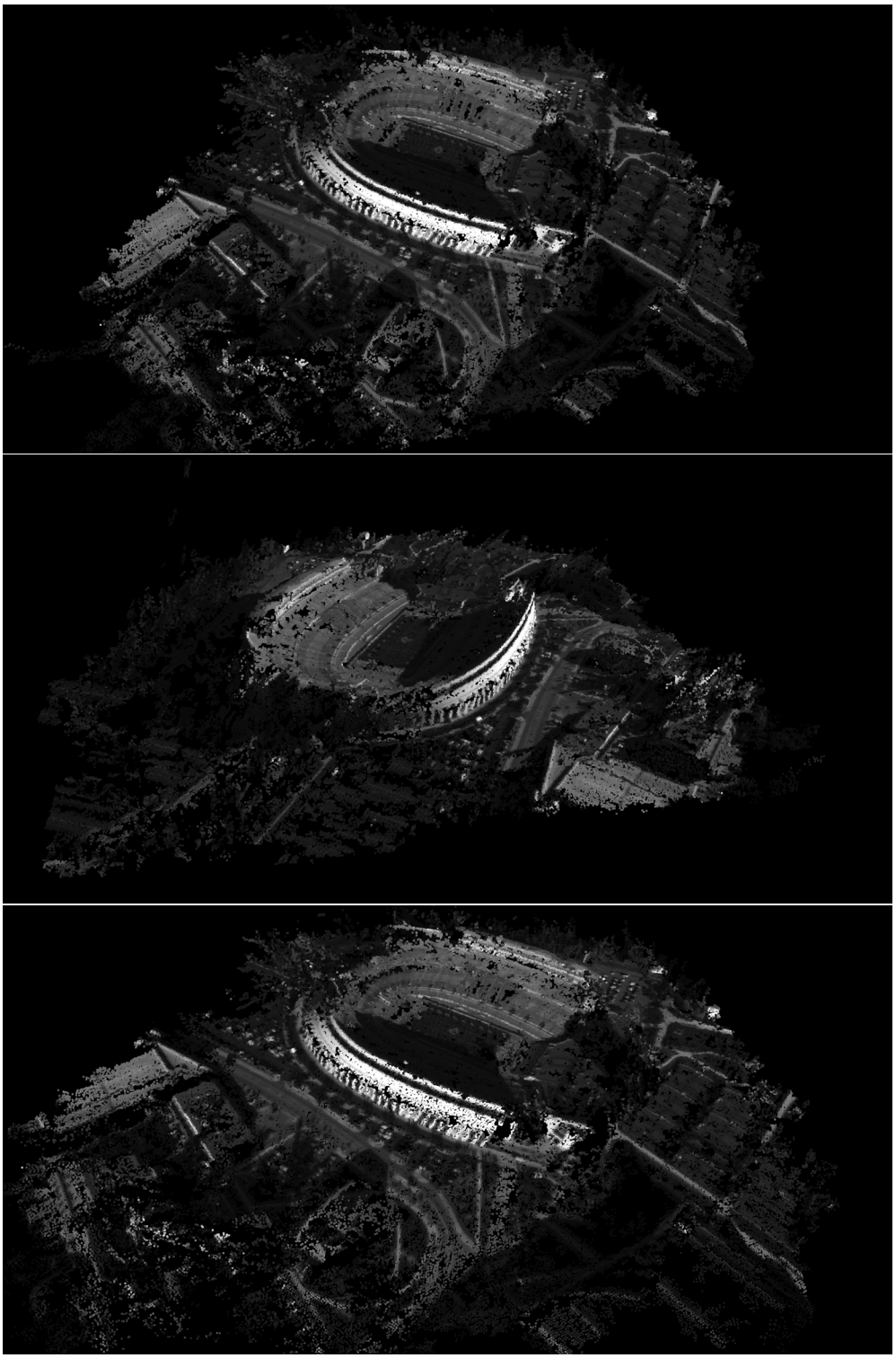


Figure 5.42: PMVS reconstruction of CLIF Stadium Only, 3 views. Note that the reconstruction is inverted (156,290 points)

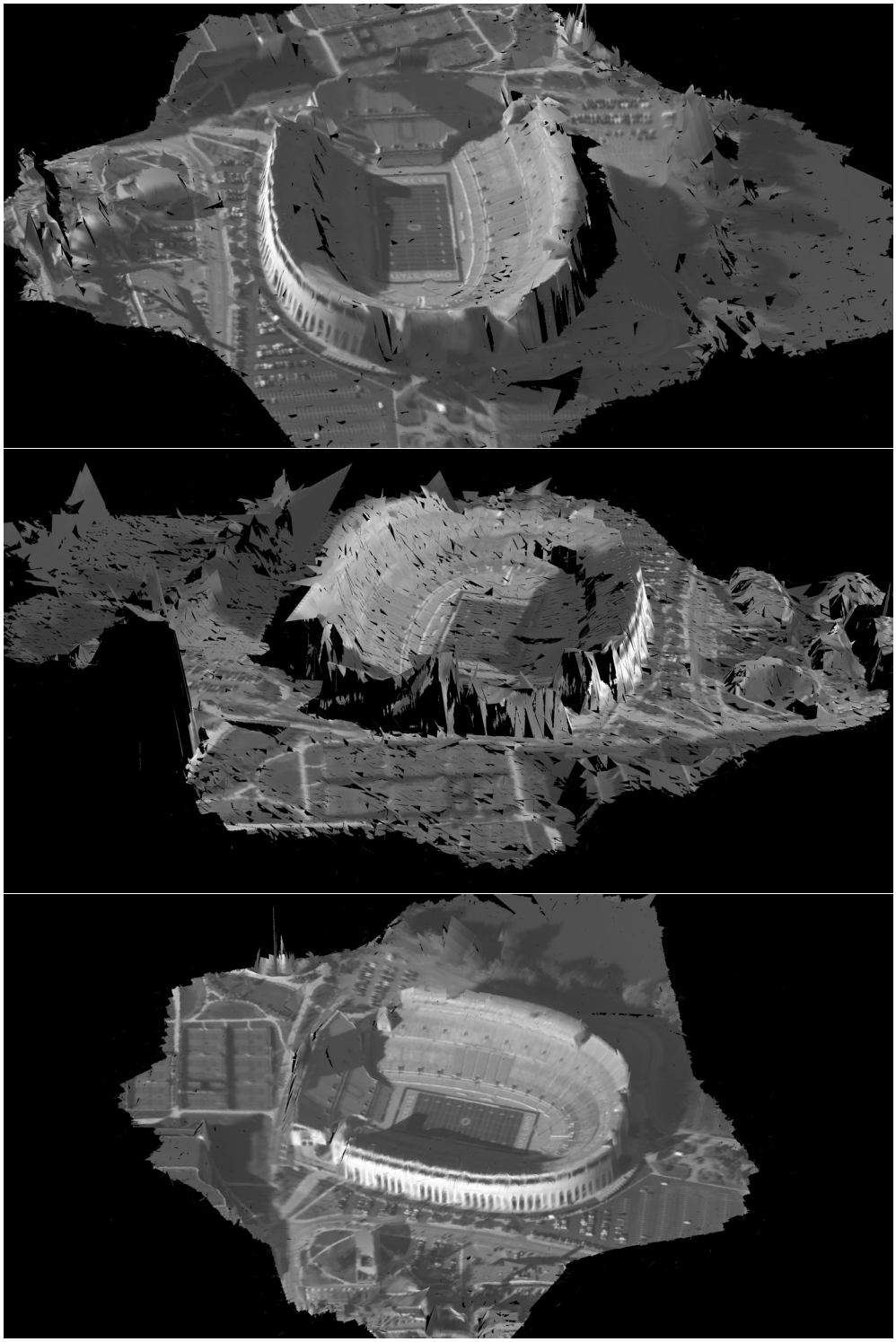


Figure 5.43: Reconstruction of CLIF Stadium using proposed algorithm, 3 views. (12,912 visible planes)

Scene	KeyPoint		Bundler	PMVS	Total
	Finding	Matching			
CLIF Intersection	47	80	294	68	489
CLIF Stadium Images	461	1,414	909	330	3,114
CLIF Stadium Only	293	725	286	90	1,401

Table 5.1: Bundler and PMVS time breakdown (all times in seconds)

iteration for a single plane is under a second with the multi-plane optimization method. We further note that computing appearances is extremely fast, mostly due to the use of CUDA.

Scene	Camera Pose			Geometry			Appearance
	Time/Iter	# Iter	Time	Time/Iter	# Iter	Time	Time
CLIF Intersection	7.26	20	6,534	170	2	5,100	1.52
CLIF Stadium Images	9.75	20	4,680	110	2	74,580	2.15
CLIF Stadium Only	6.86	20	7,260	197	2	20,882	2.01

Table 5.2: Our time breakdown. All times in seconds, iterations in camera pose refers to updating each camera once; iterations in geometry refers to updating 250 planes once (each update runs until specified tolerance is met or specified number of likelihood evaluations reached, whichever first). Total time for camera pose is obtained by $T = N_{images} * N_{iter} * T_{iter}$, time for geometry is obtained by $T = \lceil N_{planes}/250 \rceil * N_{iter} * T_{iter}$.

■ 5.4 Additional Reconstructions

This section provides additional reconstructions of the scenes analyzed earlier. The views presented here are obtained from observation viewpoints.

Figure 5.44 shows the reconstruction of the Stadium Only dataset. Each of the images in the figure corresponds to the reconstructed world as viewed from a particular observation –cameras: 0, 16, 32, and 48 . From the figure we can see that the reconstructions are sharp enough to mimic actual observations. It only by looking at certain features such as trees that we can ascertain that we are seeing a reconstruction.

Figure 5.45 shows a reconstruction of the Intersection dataset. As with the prior figure, each of these images corresponds to the reconstruction being seen from an observation perspective, cameras 0, 14, 29, and 44. Again from the figures we can see

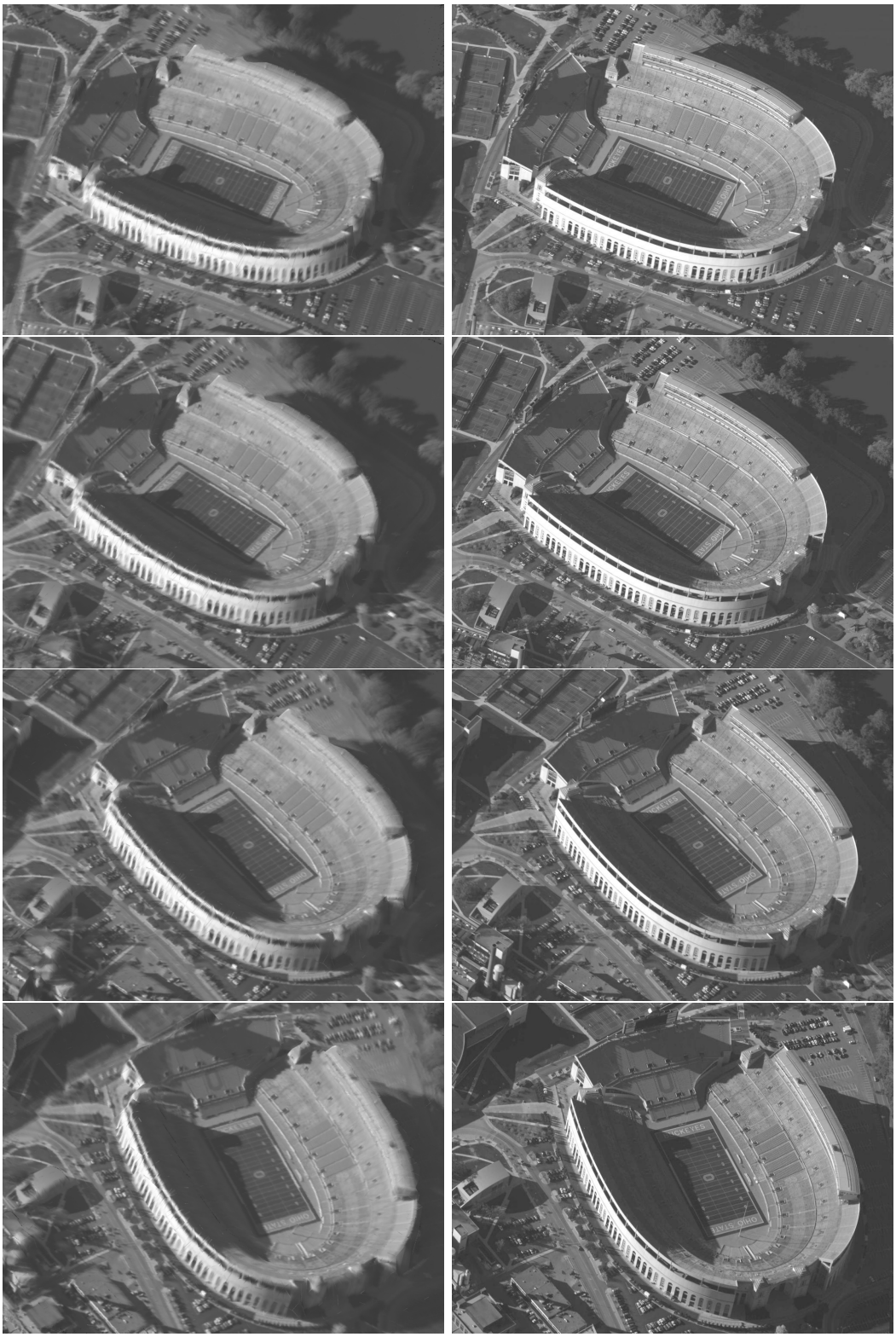


Figure 5.44: Stadium Only Reconstruction, *left:* reconstruction, *right:* original image, top to bottom cameras: 0,16,32,48.

the subtle details of the scene such has the double lines of the roads, or the left turn marking on the pavement. Furthermore, if one looks carefully one can see the shadow of what appears to be a pole with traffic lights on it in the center of the intersection. It is only by looking at building walls that we are able to deduce that this is a reconstruction and not an actual image.

■ 5.5 Beyond Reconstructions

In this section we discuss two aspects of the model that go beyond simply reconstructing the underlying scene. These added features are: having absolute scale and orientation; and being able to identify moving objects in the scenes. As we will see these features follow from our modeling choices and are key distinctions between the proposed model and traditional SfM.

Furthermore, these features are primary motivators for the proposed model. We note that while scene reconstruction is the principal focus of many algorithms (SfM included) it is generally an intermediate step to more complex reasoning about a scene. However, traditional reconstruction methods don't readily address or support this possibility. Our goal in this section is to highlight the potential of the proposed model in order to address those higher level queries.

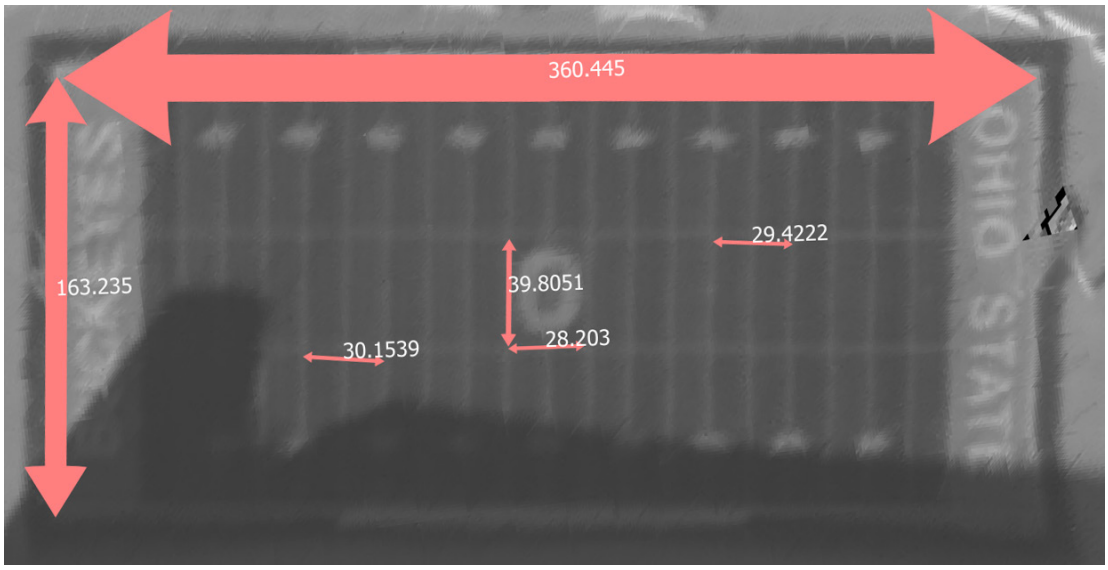
■ 5.5.1 Absolute Scale and Orientation

As pointed out earlier, we can use LiDAR measurements to identify the absolute scale and orientation for our reconstructions since LiDAR measurements are geocoded. This allows us to recover the scale factor that is typically unknown in traditional SfM. One benefit of incorporating scale and orientation into the model is that it allows for inference and analysis beyond simple reconstruction. One can reason about such quantities as the absolute distance between two points.

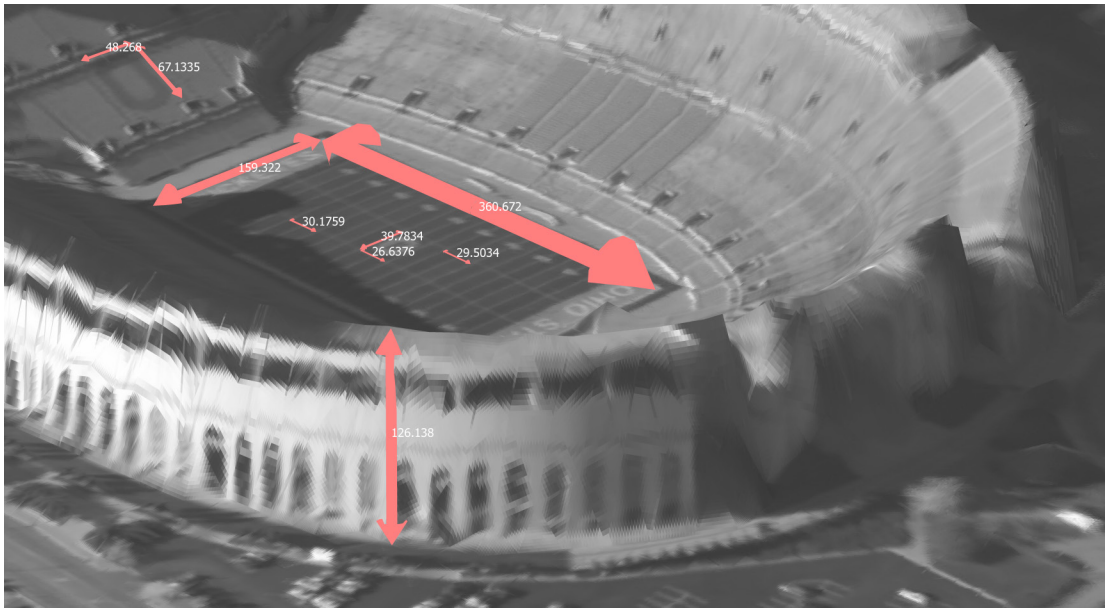
Knowing the scale and orientation is crucial for applications such as route planning, where being able to measure distances in world coordinates is fundamental. Furthermore, being able to reason about physical units via remote sensing as opposed to traditional surveying techniques can result in both financial and time savings.

As an example of how the scale and orientation knowledge can be used within the proposed mode we demonstrate the ability to measure distances in the football field, Figure 5.46a. From the figure we can see that we are able to measure the distance of the field to 360.445×163.235 feet, which corresponds to the actual distance of 360×160 feet for a football fields. This estimate is consistent with the uncertainty in both the measurements and user interaction with the marking tools. Additional distance measurements between the ten yard (30 feet) marks in the field can also be seen. We can use the same distance measuring tools to directly measure unknown quantities such as the size of the "O" in the field or stands, or the height of the stadium (cf. Figure 5.46b), which are approximately 40×28ft , 48×67ft and 126ft respectively.

Figure 5.45: Intersection Reconstruction, *left:* reconstruction, *right:* original image, top to bottom cameras: 0,14,29,44.



(a) Field View



(b) OSU Stadium

Figure 5.46: Two distance annotation examples of OSU Stadium.

5.5.2 Identifying Scene Movers

An implicit assumption of the proposed model is that the appearance and geometry of the primitives is static. Due to movement in the scene, this assumption is violated in some places. When this assumption is combined with two of our modeling choices we are able to detect moving objects in reconstructions. The modeling choices of interest are modeling scene geometry as higher order primitives and maintaining the temporal order of images. With these components in place detection can be achieved by examining image locations with low-likelihood under the inferred appearance model.

To see how these modeling choices provide a mechanism to detect movers in a scene, let us analyze the individual choices. The dense primitives allow us to have support over the entire image, this ensures that all image values get projected to a reconstructed primitive and thus allow for likelihood computations for all pixels in the image. These likelihood computation would only be possible in SfM if keypoint matches of the object were made across multiple frames, which is unlikely for moving objects. Even if the matches are established SfM triangulation methods assume that the matches correspond to the same 3D location, and thus reconstruct the moving object to a single location. On the other hand, the temporal order ensures that the object motion is consistent from frame to frame. This is not strictly necessary since likelihood computations do not depend on image order.

To demonstrate the model's ability to detect moving objects, we computed image likelihood of the CLIF Intersection dataset after estimating world geometry and appearance, and camera parameters. The results can be seen in Figure 5.47, where low likelihood regions have been color coded in red on the original images and the images have been ordered according to their temporal order. From the figure we can see that the model captures scene movers fairly well. In this example the movers consist of cars passing (or turning) across the intersection.

It is interesting to point out that there are some low likelihood regions that do not correspond to movers, such as building edges (see bottom right portion of the first image). These regions are highlighted since the model is not reconstructing them correctly. As pointed out earlier, the model's ability to reconstruct sharp edges depends on viewing directions, particularly wide baseline to determine correct geometry. Despite these spurious miss assignments it is clear that movers are reliably detected.

In summary, this chapter has shown the abilities of the proposed model. We have shown that the model is able to infer geometry, appearance and camera parameters reliably, obtaining reconstructions that are qualitatively similar to traditional structure from motion techniques. Furthermore, we have seen the trade off present between the multi-modal data sources used, and how this trade-off can lead to improvements in reconstructions. We have also seen how the proposed model can be used to extend traditional reconstruction results, and allow more complex reasoning about scenes.

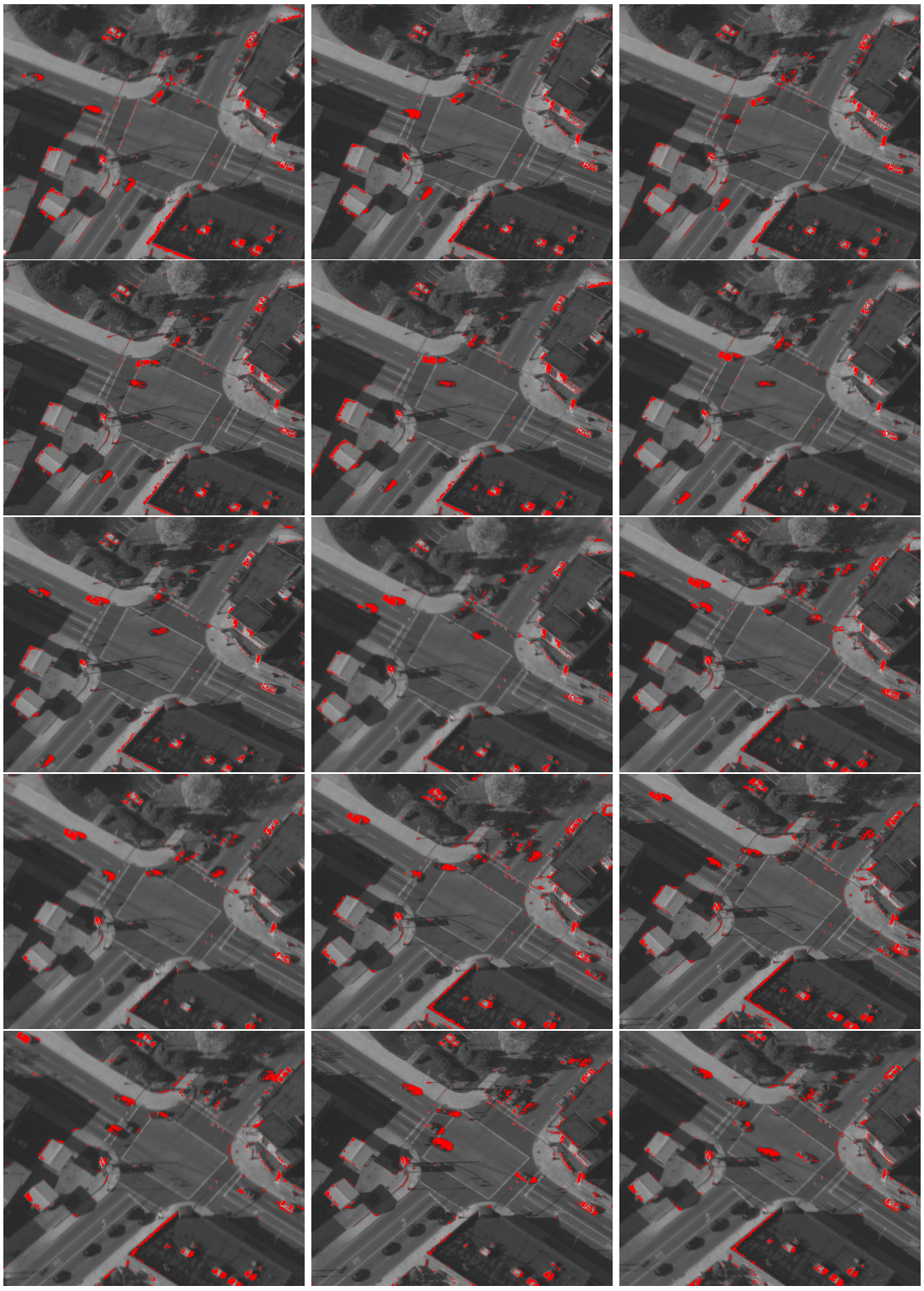


Figure 5.47: Low likelihood regions colored in red. Regions correspond to moving objects and sharp edges (sequence left to right CLIF Intersection images 14-28)

Chapter 6

Conclusion

In this thesis we considered the problem of scene reconstruction from multi-modal data, specifically LiDAR and aerial imagery. In so doing, we adopted a Bayesian view of data fusion and formulated an approximate generative model for combining the aforementioned data sources. The purpose of doing so is manifold. The content of the thesis focused primarily on inference of geometric and appearance parameters across a wide and diverse geographic region. We also demonstrated the capacity to utilize the model for higher level reasoning, specifically estimating physical dimensions of objects in the scene and detecting moving objects against a static background.

Our modeling approach has several advantages: first, the use of noisy observations and probabilistic model allows us to capture uncertainty in reconstructions, second, the model has the ability of incorporating additional scene measurements easily when available; third the model seamlessly handles missing data, and fourth the overall modeling approach simplifies the process of incorporating new measurement modalities provided a physical sensor model is available.

From an implementation stand point, this thesis has shown how to leverage the power of graphics hardware, e.g. GPUs, to allow fast renderings and inference. Using OpenGL we have been able to render scenes with hundreds of thousands of triangles in real time. This in turn has allowed us to infer the appearance for each triangle in the order of a few seconds. The combination of OpenGL and CUDA has allowed us to render a textured mapped scene and compute image likelihoods at the microsecond level. When combined the fast rendering and computation allow for camera pose inference in the order of ten or twenty seconds. Similarly the geometry estimate of a single triangle can be achieved on average in under a second.

From a reconstruction standpoint, we have shown that the model is able to recover the correct camera pose if initialized within a broad capture range. Our experiments demonstrate that the model is suitable for describing extended, relatively smooth surfaces (e.g. roads, buildings, and similar structures), but suffers when representing more complicated surfaces such as foliage or "thin" (relative to the measurement resolution) structures (e.g. goal posts on a football field, and street lamps). Moreover, the dense reconstructions and the use of images to represent scene appearances allow us to identify small but crucial scene details such as pavement markings.

Furthermore, we have shown that the appearance and geometry trade-off present

in the model between data sources can be used to obtain a comparable (and sometime superior) reconstruction of complex urban scenes with fewer image observations over traditional reconstruction method. When compared to classical structure from motion the model performs well. The resulting reconstructions are comparable. Horizontal surfaces appear to be qualitatively better than reconstructions of vertical surfaces. We hypothesize that this is a consequence of a greater number of LiDAR measurements being associated with the former as compared to the latter. Precisely quantifying this is the subject of future experiments.

Extending beyond simple reconstructions the model presented here has two desirable features: first we are able to determine orientation and absolute scale as demonstrated by making distance measurements. Secondly, the higher order primitives in this model and the temporal order of the image sequence allow us to detect moving objects in scene. These features come from the modeling choices made when building the model.

■ 6.1 Possible Changes

Some very explicit choices were made in this work. While these decisions typically simplified implementations or derivations they are not the only possibilities. In this section we speculate on aspects of our work that could be changed or improved

The first of such improvements is the latent geometry parameterization. As formulated, the geometry of primitives are independent in the generative model, the independent triangle parameterization works well within the proposed model but has the major disadvantage of requiring a very large number of triangles to represent the scene. This large number of triangles increases the total geometry update time and renders time. Moreover, since triangles are generally not overlapping changing triangle geometry leads to regions where background is visible. A possible improvement to alleviate these problems could be to incorporate statistical dependency between as a function of proximity.

Another geometry change could be to allow the model to change the number of triangles. Right now, the number of primitives must be chosen apriori and remains constant, only update the parameters of each triangle. By letting the model modify the number of primitives, we would have the ability to increase expressibility in areas where it is needed, and reduce the number of primitives in benign area to help mitigate the computation time problems.

To further ease the computation burden, we can change the multi-plane implementation to remove the constraint that only a max of 253 planes can be optimized in parallel. This constraints follows from the observation that we can save the association render between image pixels and world primitives if we encode the primitive number in the alpha channel of the appearance map. Since the appearance map is an 8-bit number, we only have 255 possible labels and two are occupied, which leaves only 253 possible labels. This constraint can be removed if we perform the association render.

In theory the proposed change would allow all primitives to be optimized in parallel;

however, care must be taken since the memory requirements far exceed the available memory in graphics hardware. This stems from the implementation that each optimized plane must have its own copy of background appearance. As an alternative, each primitive could maintain a dynamic list of background indices it influences.

■ 6.2 Future Work

There remain a great number of unanswered questions in the topics covered in this thesis. In this section we describe future research topics related to the work presented.

The data association problem between LiDAR measurement and latent primitives should be further studied. The main challenge in this problem stems from the combinatorics, that is, which LiDAR measurement corresponds to which latent primitive. We have simplified the association by having a fixed kd tree over the measurements, translating primitives into points and performing nearest neighbor queries on those. This simplification allows for fast reasoning over the large space of measurements and primitives. However, as we have seen this simplification can lead to data association errors, where large planes are associated with more measurements. A remaining challenge is that of formulating exact data association in a computationally feasible manner.

Throughout this work we have optimized to obtain point estimate of the probability distributions. The optimization techniques used do not require gradient computation. While this is ultimately simple, it is also inefficient and requires many more function evaluations. An interesting and potentially very useful future research topic would be to obtain expressions for the gradients for the parameters of interest.

An important extension to the work presented here is to perform quantitative model evaluation and comparisons. These evaluations were not carried out due the lack of ground truth data or general performance metrics for reconstruction algorithms. Moreover the difference in reconstruction types, points vs. triangles, between SfM and the model presented here adds another level of complexity since creating a triangulation from SfM can introduce errors, and generating points from our triangular model can be misleading and lead to sampling issues. Future work should either obtain ground truth data to validate the reconstructions or outline a comprehensive set of criteria in which to quantify reconstructions.

Last but certainly not least, future work should investigating other types of noisy observations to include in the model. The two observation we currently have complement each other, but more realistic reconstructions could be achieved if we model material properties, such as reflectivity and luminosity, or texture (as in smoothness or roughness). The introduction of lighting for the purpose of shadow modeling could also be interesting. Observations that attempt to capture these properties should be considered and further investigated.

Appendix A

Data Overview

In this appendix we briefly introduce and discuss the data sources used in this thesis: toy dataset, Lubbock dataset, and CLIF dataset.

A.1 Toy Dataset

The toy Dataset is a synthetic dataset consisting of 15 images, each image is 1200×900, cf. Figure A.1. This dataset is primarily used to demonstrate key concepts since all the parameters are known. It is helpful when computing appearances since the camera pose is simply a rotation around two cubes that have been texture mapped with book covers and other texture images. Four sample images of the sequence can be seen in Figure A.2.

A.2 Lubbock Dataset

The Lubbock Dataset, cf. Figure A.3, consist of three image of Campus Road, Lubbock Texas. The three images are approximately ninety degrees apart, and can be seen in Figure A.4, each image is 1336×891.

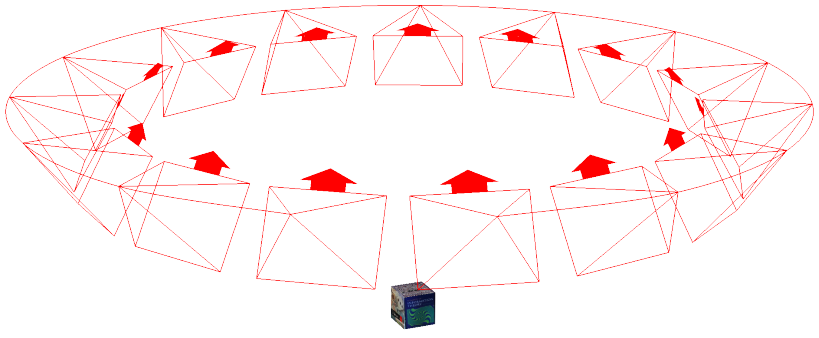


Figure A.1: Overview of Toy dataset, 15 images (outline location shown in red).

Figure A.2: Four Images of the toy dataset (left to right noiseless images 0,4,8,12)

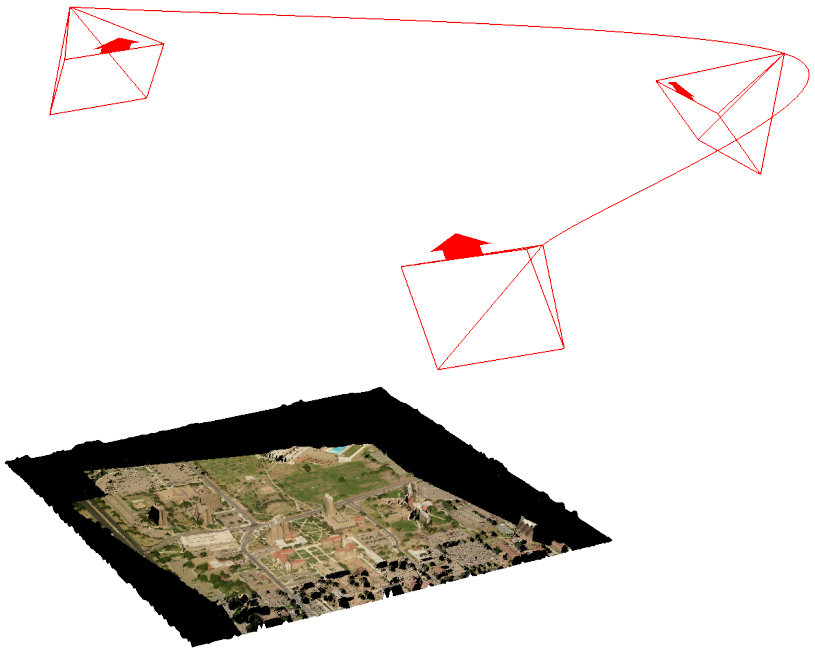


Figure A.3: Overview of Lubbock dataset, 3 images (outline location shown in red).

Figure A.4: The three images of the Lubbock dataset

The Lubbock dataset also contains over five million LiDAR returns in a single tile (c.f. Fig. A.5). This tile is 1000×1000 meters and encompasses all of the Campus Road Section that we are interested in. Density of LiDAR returns is about one per meter, with vertical resolution of 10 centimeters.

This dataset has been annotated, that is 2D-3D correspondences have been hand marked between LiDAR and images, and approximate ground truth camera pose has been computed via the gold standard method of [28]. As a result this dataset is particularly useful when comparing learned camera parameters. In addition, it can also be used to demonstrate the performance of algorithms when a small number of images with a wide baseline are considered.

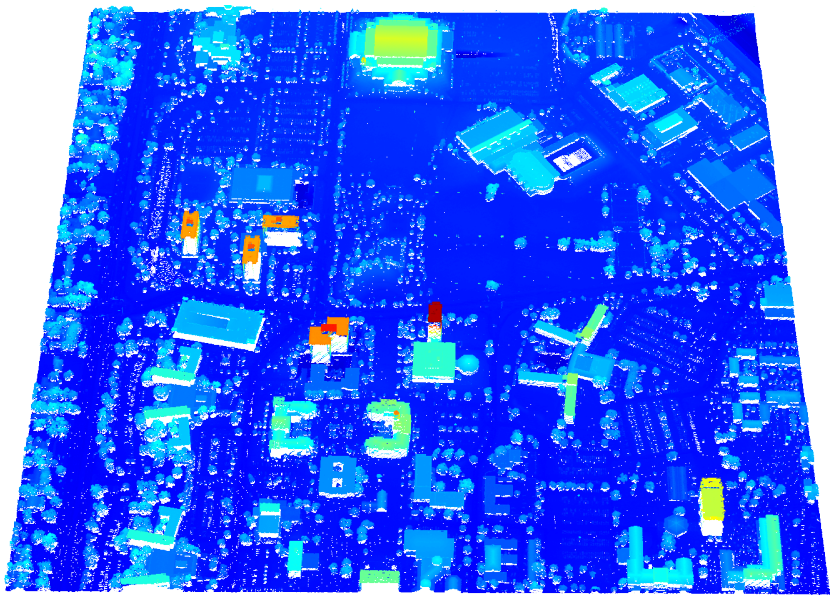


Figure A.5: Overview of Lubbock LiDAR, (color coded height above ground).

A.3 CLIF 2007 Dataset

The Columbus Large Image Format (CLIF) 2007 Sample dataset [59] is used throughout this work. It consist of 50 frames, each frame has 6 cameras, cf. Figure A.6. Each frame is originally 2672×4016 pixels, and this work has been downsampled to 822×1326. The area covered in one frame is approximately 1700×2200 feet, the approximate area of the sample set is 3500×3800 feet (i.e. visible across the entire sequence).

LiDAR for the Ohio State area was obtained from [41]. The tile containing the OSU stadium and surrounding area has over 727,000 returns. The density is approximately one return in 3×3 feet. An overview of the LiDAR tile can be seen in figure A.7.

In this work we focus on specific cites in the CLIF dataset, the stadium Image stack (top center in Figure A.6), a crop of the stadium and a crop of the intersection (in the bottom center in Figure A.6). We'll discuss these three subsets next.

A.3.1 CLIF Stadium Image Stack

The CLIF Stadium Image Stack consist of 49 images, one for each frame of camera one (top center in figure A.6) from the CLIF dataset. One image was not included since it was corrupted. An overview of the cameras can be seen in figure A.8, and four sample images can be seen in figure A.9.

This dataset uses the LiDAR tile described earlier.

Figure A.6: Overview of CLIF Cameras, frame 0, six cameras. Stadium Image stack, top center; intersection seen in the bottom center image.

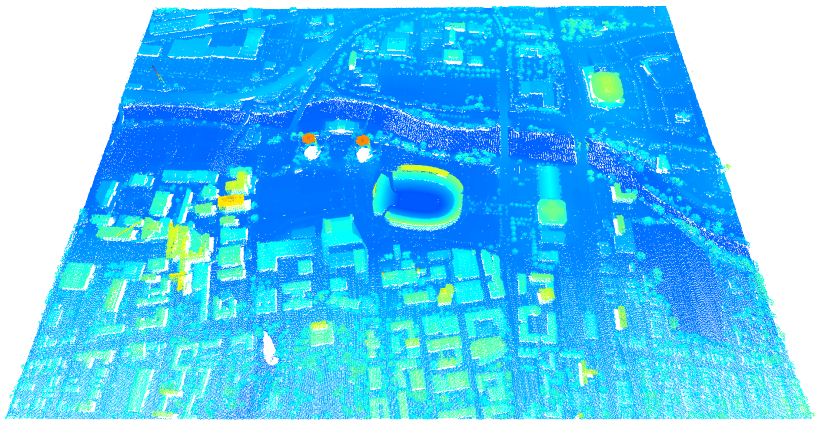


Figure A.7: Overview of CLIF LiDAR, (color coded height above ground).

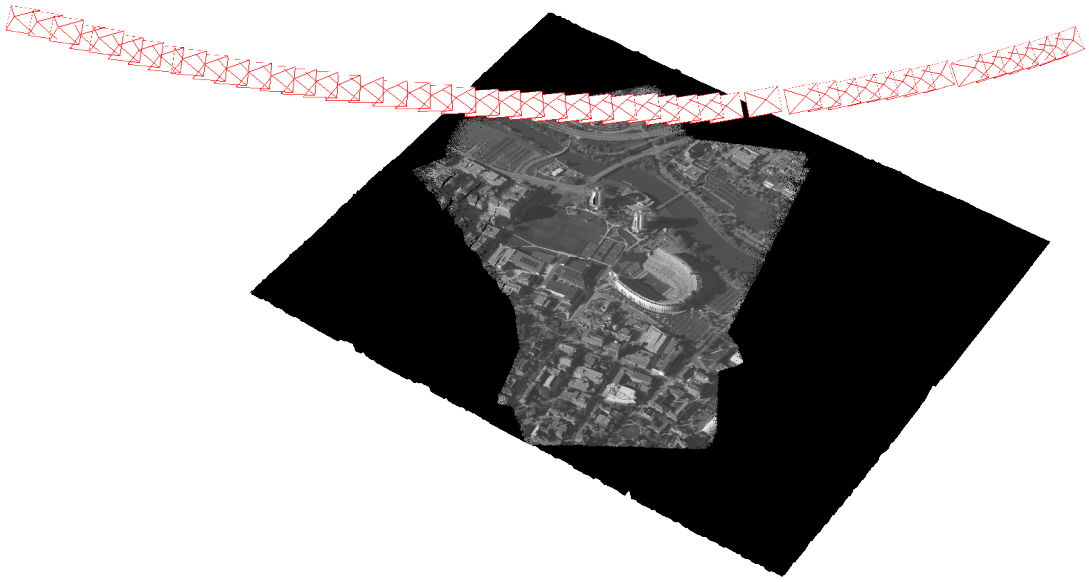


Figure A.8: Overview of CLIF Image Stack dataset, 50 images (outline location shown in red).

Figure A.9: Four Images of the CLIF Image Stack dataset (left to right images 0,16,32,48)

Figure A.10: Four Images of the CLIF Stadium Only dataset (left to right images 0,16,32,48)

■ A.3.2 CLIF Stadium Only

The CLIF Stadium Only Dataset consist of a cropped version of the original CLIF frames (not downsampled). Each image in this dataset is 1024×768, four sample images can be seen in Figure A.10.

This dataset uses the LiDAR tile described earlier.

■ A.3.3 CLIF Intersection

The CLIF Intersection dataset consist of 45 cropped portions of images from camera two and zero (bottom left and center in Figure A.6) of the CLIF dataset. Only one image for each frame was used, i.e. the sequence was maintained. Instances where the scene was split between two images where discarded, four sample images can be seen in Figure A.11.

The Intersection dataset falls outside the LiDAR tile described earlier. As a result, a small cropped version of another tile is used for this dataset, cf. Figure A.12. This mini-tile has the same properties as before, i.e. same density; it is simply smaller with just over 20,000 returns.

Figure A.11: Four Images of the CLIF Intersection dataset (left to right images 0,14,29,44)

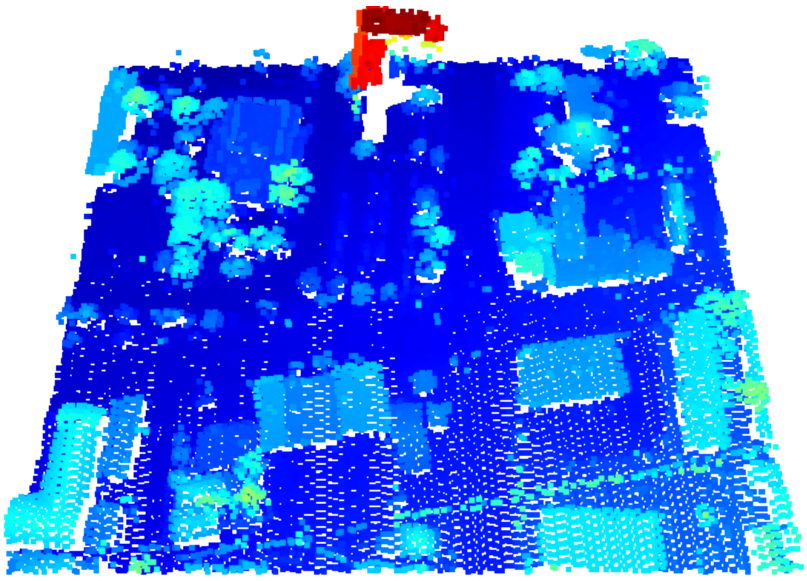


Figure A.12: Overview of CLIF Intersection LiDAR, (color coded height above ground).

Appendix B

Appendix B

Appearance Computation

In this chapter we derive the update equations for the appearance in our model conditioned on all other parameters. We begin by replicating our model in Figure B.1 and the joint probability distribution, equation (B.1).

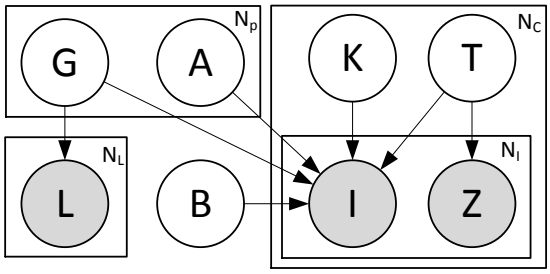


Figure B.1: Our Graphical Model Representation.

Joint Probability:

$$p(\mathbf{L},\mathbf{I},\mathbf{Z},\mathbf{G},\mathbf{A},\mathbf{T},\mathbf{K},\mathbf{B}) = \underbrace{\prod_{c=1}^{N_c}\prod_{i=1}^{N_I} p(I_i^c|\mathbf{G},\mathbf{A},\mathbf{B},K_c,T_c)p(Z_i|T_c)\prod_{l=1}^{N_l} p(L_l|\mathbf{G})}_{\text{Likelihoods}} \times \underbrace{\prod_{k=1}^{N_p} p(G_k)p(A_k)\prod_{c=1}^{N_c} p(T_c)p(K_c)p(\mathbf{B})}_{\text{priors}} \tag{B.1}$$

■ B.1 Observation Model

As discussed in chapter 3, the image observation model is:

$$I_n^c(u,v) = A_{m^*}(u',v') + Q_n \tag{B.2}$$

where
$Q_n \sim \mathcal{N}(q;0,R(m^*))$, I_n^c is the n^{th} image of camera c, A_{m^*} is the m^* triangle appear-

ance, and (u, v) is the projected image coordinates of the appearance at coordinate (u', v').

We note that (u, v) and (u', v'), are related implicitly via the 3D point (x, y, z), or explicitly via the homography between image A_k and I_n.

For any pixels (u, v), and (u', v') that are in correspondence, we can simply the notation by letting:

$$z = I_n(u, v)$$
$$a = A_{m^*}(u', v'),$$

where z denotes the observation and a denotes the underlying latent appearance. So then, equation (B.2) can be written as:

$$z = a + q$$

It follows that:

$$z|a \sim \mathcal{N}(z; a, R),$$

and

$$z \sim \mathcal{N}(z; \mu, \Sigma + R),$$

if we let A have a Normal prior, such that $A \sim \mathcal{N}(a; \mu, \Sigma)$.

So then our task is to estimate A. We are interested in several cases, first what would the estimate of A be from a single observation; for multiple observations with the same noise characteristics; and from multiple observations with different noise characteristics.

■ B.2 Summary

Before diving into the derivations we present the results. As we will demonstrated: $p(a|z) \sim \mathcal{N}(a; \hat{\mu}, \hat{\sigma})$, where $\hat{\mu}$ and $\hat{\sigma}$ vary for the different cases considered:

Single Observation, $z = a + q$:

$$\hat{\mu} = \frac{\sigma^2 z + r^2 \mu}{\sigma^2 + r^2} \tag{B.3}$$
$$\hat{\sigma} = \frac{\sigma r}{\sqrt{\sigma^2 + r^2}} \tag{B.4}$$

Multiple Observation (same noise), $z_n = a + q$:

$$\hat{\mu} = \frac{r^2 \mu + \sigma^2 \sum_{i=0}^{n-1} z_i}{r^2 + n\sigma^2} \tag{B.5}$$
$$\hat{\sigma} = \frac{\sigma r}{\sqrt{r^2 + n\sigma^2}} \tag{B.6}$$

Multiple Observations (different noise), $z_n = a + q_n$ (where $q_n \sim \mathcal{N}(q; 0, r_n)$):

$$\hat{\mu} = \frac{\mu \dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 z_i}{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2} = \frac{\mu \prod_{i=0}^{n-1} r_i^2 + \sigma^2 \sum_{i=0}^{n-1} \left(\prod_{j=0, j\neq i}^{n-1} r_j^2 \right) z_i}{\prod_{i=0}^{n-1} r_i^2 + \sigma^2 \sum_{i=0}^{n-1} \left(\prod_{j=0, j\neq i}^{n-1} r_j^2 \right)} \tag{B.7}$$

$$\hat{\sigma} = \frac{\dot{r}\sigma}{\sqrt{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2}} = \frac{\sigma \prod_{i=0}^{n-1} r_i}{\sqrt{\prod_{i=0}^{n-1} r_i^2 + \sigma^2 \sum_{i=0}^{n-1} \left(\prod_{j=0, j\neq i}^{n-1} r_j^2 \right)}} \tag{B.8}$$

■ B.3 Derivations

In this section we derive the forms of the posterior $p(a|z)$, for the multiple observations with different noise, next section shows how to obtain the special cases.

■ B.3.1 Multiple Observations - different noise

Assume that observation is the same as before $I_n(u, v) = A_k(u', v') + Q_n$, but with $Q_n \sim \mathcal{N}(q; 0, r_n)$. i.e. the noise variance depends on the image.

As before $z = a + q$, but let's denote the different variance explicitly as $z = a + q_n$. Then,

$$\begin{aligned}
p(a|\mathbf{z}) &= \frac{\prod_{i=0}^{n-1} p(z_i|a) p(a)}{\prod_{j=1}^{n} p(z_j)} \\
&\propto p(a) \prod_{i=0}^{n-1} p(z_i|a) \\
&\propto \mathcal{N}(a; \mu; \sigma^2) \prod_{i=0}^{n-1} \mathcal{N}(z_i; a, r_i^2) \\
&\propto \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{(a-\mu)^2}{2\sigma^2} \right\} \prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi r_i^2}} \exp\left\{ -\frac{(z_i - a)^2}{2 r_i^2} \right\}
\end{aligned}$$

Let us define

$$\dot{r} = \prod_{j=0}^{n-1} r_j$$

$$\dot{r}_{\setminus i} = \prod_{j=0, j\neq i}^{n-1} r_j$$

then,

$$\begin{aligned} p(a|z) &\propto \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(a-\mu)^2}{2\sigma^2}\right\} \prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi r_i^2}} \exp\left\{-\frac{(z_i-a)^2}{2r_i^2}\right\} \\ &\propto \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(a-\mu)^2}{2\sigma^2}\right\} \frac{1}{(2\pi)^{\frac{n}{2}}\,\dot{r}} \prod_{i=0}^{n-1} \exp\left\{-\frac{(z_i-a)^2}{2r_i^2}\right\} \\ &\propto \frac{1}{(2\pi)^{\frac{n+1}{2}}\,\dot{r}\sigma} \exp \underbrace{\left\{-\frac{(a-\mu)^2}{2\sigma^2} - \sum_{i=0}^{n-1} \frac{(z_i-a)^2}{2r_i^2}\right\}}_{*} \end{aligned}$$

Aside:

$$\begin{aligned} * &= -\frac{(a-\mu)^2}{2\sigma^2} - \sum_{i=0}^{n-1} \frac{(z_i-a)^2}{2r_i^2} \\ &= -\frac{1}{2}\left[\frac{(a-\mu)^2}{\sigma^2} + \sum_{i=0}^{n-1} \frac{\dot{r}_{\setminus i}^2 (z_i-a)^2}{\dot{r}^2}\right] \\ &= -\frac{1}{2}\left[\frac{(a-\mu)^2}{\sigma^2} + \frac{1}{\dot{r}^2}\sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 (z_i-a)^2\right] \\ &= -\frac{1}{2\dot{r}^2\sigma^2}\left[\dot{r}^2(a-\mu)^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 (z_i-a)^2\right] \\ &= -\frac{1}{2\dot{r}^2\sigma^2}\left[\dot{r}^2 a^2 - 2\mu\dot{r}^2 a + \mu^2\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 \left(z_i^2 - 2z_i a + a^2\right)\right] \\ &= -\frac{1}{2\dot{r}^2\sigma^2}\left[\dot{r}^2 a^2 - 2\mu\dot{r}^2 a + \mu^2\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 z_i^2 - 2\sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 z_i a + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 a^2\right] \\ &= -\frac{1}{2\dot{r}^2\sigma^2}\left[\left(\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2\right) a^2 - 2\left(\mu\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 z_i\right) a + \mu^2\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 z_i^2\right] \\ &= -\frac{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2}{2\dot{r}^2\sigma^2}\left[a^2 - 2\frac{\left(\mu\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 z_i\right)}{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2} a + \frac{\mu^2\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 z_i^2}{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2}\right] \\ &= -\frac{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2}{2\dot{r}^2\sigma^2}\left[(a-\hat{\mu})^2 + c'\right] \end{aligned}$$

where

$$\hat{\mu} = \frac{\mu\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 z_i}{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2}$$
$$c' = \frac{\mu^2\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 z_i^2}{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2} - \hat{\mu}^2$$

then

$$\begin{aligned}
p(a|\mathbf{z}) &\propto \frac{1}{(2\pi)^{\frac{n+1}{2}}\,\dot{r}\sigma} \exp\left\{-\frac{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2}{2\dot{r}^2\sigma^2}\left[(a-\hat{\mu})^2 + c'\right]\right\} \\
&\propto \underbrace{\frac{1}{(2\pi)^{\frac{n+1}{2}}\,\dot{r}\sigma\frac{\sqrt{\dot{r}^2+\sigma^2\sum_{i=0}^{n-1}\dot{r}_{\setminus i}^2}}{\sqrt{\dot{r}^2+\sigma^2\sum_{i=0}^{n-1}\dot{r}_{\setminus i}^2}}}}_{\text{roll part to proportionality}} \exp\left\{-\frac{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2}{2\dot{r}^2\sigma^2}(a-\hat{\mu})^2\right\} \underbrace{\exp\left\{-\frac{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2}{2\dot{r}^2\sigma^2}c'\right\}}_{\text{roll to proportionality}} \\
&\propto \frac{1}{(2\pi)^{\frac{1}{2}}\,\frac{\dot{r}\sigma}{\sqrt{\dot{r}^2+\sigma^2\sum_{i=0}^{n-1}\dot{r}_{\setminus i}^2}}} \exp\left\{-\frac{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2}{2\dot{r}^2\sigma^2}(a-\hat{\mu})^2\right\} \\
&\propto \frac{1}{(2\pi)^{\frac{1}{2}}\,\frac{\dot{r}\sigma}{\sqrt{\dot{r}^2+\sigma^2\sum_{i=0}^{n-1}\dot{r}_{\setminus i}^2}}} \exp\left\{-\frac{(a-\hat{\mu})^2}{2\frac{\dot{r}^2\sigma^2}{\dot{r}^2+\sigma^2\sum_{i=0}^{n-1}\dot{r}_{\setminus i}^2}}\right\} \\
&\propto \frac{1}{(2\pi)^{\frac{1}{2}}\,\frac{\dot{r}\sigma}{\sqrt{\dot{r}^2+\sigma^2\sum_{i=0}^{n-1}\dot{r}_{\setminus i}^2}}} \exp\left\{-\frac{(a-\hat{\mu})^2}{2\left(\frac{\dot{r}\sigma}{\sqrt{\dot{r}^2+\sigma^2\sum_{i=0}^{n-1}\dot{r}_{\setminus i}^2}}\right)^2}\right\} \\
&\propto \mathcal{N}(a;\hat{\mu},\hat{\sigma}^2)
\end{aligned}$$

where

$$\hat{\mu} = \frac{\mu\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2 z_i}{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2} = \frac{\mu \prod_{i=0}^{n-1} r_i^2 + \sigma^2 \sum_{i=0}^{n-1}\left(\prod_{j=0,j\neq i}^{n-1} r_j^2\right) z_i}{\prod_{i=0}^{n-1} r_i^2 + \sigma^2 \sum_{i=0}^{n-1}\left(\prod_{j=0,j\neq i}^{n-1} r_j^2\right)} \tag{B.9}$$

$$\hat{\sigma} = \frac{\dot{r}\sigma}{\sqrt{\dot{r}^2 + \sigma^2 \sum_{i=0}^{n-1} \dot{r}_{\setminus i}^2}} = \frac{\sigma \prod_i^{n-1} r_i}{\sqrt{\prod_{i=0}^{n-1} r_i^2 + \sigma^2 \sum_{i=0}^{n-1}\left(\prod_{j=0,j\neq i}^{n-1} r_j^2\right)}} \tag{B.10}$$

B.4 Special Cases

As a sanity check, we double check that the multiple observation with same noise and the single observation are special cases of the updates derived in §B.3.1.

To obtain the multiple observation case, we can let $r_j = r$, $\forall j \in [0, n-1]$, with this definition, $\dot{r} = r^n$, and $\dot{r}_{\setminus i} = r^{n-1}$.

Plugging the new values for $\dot{r}$ and $\dot{r}_{\setminus i}$ into the general updates of equations (B.9) and (B.10), we obtain

$$\hat{\mu} = \frac{\mu r^{2n} + \sigma^2 r^{2n-2} \sum_{i=0}^{n-1} z_i}{r^{2n} + n\sigma^2 r^{2n-2}}$$
$$\hat{\sigma} = \frac{r^n \sigma}{\sqrt{r^{2n} + n\sigma^2 r^{2n-2}}}$$

from which we can factor out r^{2n-2} from numerator and denominator to obtain:

$$\hat{\mu} = \frac{\mu r^2 + \sigma^2 \sum_{i=0}^{n-1} z_i}{r^2 + n\sigma^2} \tag{B.11}$$
$$\hat{\sigma} = \frac{r\sigma}{\sqrt{r^2 + n\sigma^2}} \tag{B.12}$$

Thus equations (B.11) and (B.12) refer to the multiple observation with same noise model. Furthermore, if we let $n = 1$ in the equations above we obtain the single observation case.

Appendix C

Gaussian Marginalization

In this appendix we derive the marginalization of a multiplication of Gaussian distributions.

■ C.1 General Derivation

$$\int \prod_{n=1}^{N} \mathcal{N}(x_n;\mu,\sigma^2)\mathcal{N}(\mu;\mu_0,\sigma_0^2)\, d\mu \tag{C.1}$$

$$= \int \prod_{n=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{(x_n-\mu)^2}{2\sigma^2}\right\} \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left\{-\frac{(\mu-\mu_0)^2}{2\sigma_0^2}\right\} d\mu \tag{C.2}$$

$$= \underbrace{\frac{1}{\left(\sqrt{2\pi\sigma^2}\right)^n} \frac{1}{\sqrt{2\pi\sigma_0^2}}}_{c} \int \prod_{n=1}^{N} \exp\left\{-\frac{(x_n-\mu)^2}{2\sigma^2} - \frac{(\mu-\mu_0)^2}{2\sigma_0^2}\right\} d\mu \tag{C.3}$$

$$= c \int \exp\left\{-\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n-\mu)^2 - \frac{(\mu-\mu_0)^2}{2\sigma_0^2}\right\} d\mu \tag{C.4}$$

$$= c \int \exp\left\{-\frac{1}{2\sigma^2}\sum_{n=1}^{N}(x_n^2 - 2\mu x_n + \mu^2) - \frac{1}{2\sigma_0^2}(\mu^2 - 2\mu_0\mu + \mu_0^2)\right\} d\mu \tag{C.5}$$

$$= c \int \exp\left\{-\frac{1}{2\sigma^2}\left[\sum_{n=1}^{N} x_n^2 - 2\mu\sum_{n=1}^{N} x_n + n\mu^2\right] - \frac{1}{2\sigma_0^2}(\mu^2 - 2\mu_0\mu + \mu_0^2)\right\} d\mu \tag{C.6}$$

$$= c \int \exp\left\{-\frac{1}{2}\left[\underbrace{\left(\frac{n}{\sigma^2}+\frac{1}{\sigma_0^2}\right)}_{a}\mu^2 - 2\underbrace{\left(\frac{1}{\sigma^2}\sum_{n=1}^{N} x_n + \frac{\mu_0}{\sigma_0^2}\right)}_{b}\mu + \underbrace{\left(\frac{1}{\sigma^2}\sum_{n=1}^{N} x_n^2 + \frac{1}{\sigma_0^2}\mu_0^2\right)}_{d}\right]\right\} d\mu \tag{C.7}$$

$$= c \exp\left\{-\frac{d}{2}\right\} \int \exp\left\{-\frac{1}{2}\left[a\mu^2 - 2b\mu\right]\right\} d\mu \tag{C.8}$$

$$= c \exp\left\{-\frac{d}{2}\right\} \int \exp\left\{-\frac{1}{2}\left[a\left(\mu - \frac{b}{a}\right)^2 - \frac{b^2}{a}\right]\right\} d\mu \tag{C.9}$$

$$= c \exp\left\{-\frac{d}{2}\right\} \int \exp\left\{-\frac{\left(\mu - \frac{b}{a}\right)^2}{2\left(\frac{1}{\sqrt{a}}\right)^2}\right\} \exp\left\{\frac{b^2}{2a}\right\} d\mu \tag{C.10}$$

$$= c \exp\left\{-\frac{d}{2}\right\} \exp\left\{\frac{b^2}{2a}\right\} \int \exp\left\{-\frac{\left(\mu - \frac{b}{a}\right)^2}{2\left(\frac{1}{\sqrt{a}}\right)^2}\right\} d\mu \tag{C.11}$$

$$= c \exp\left\{-\frac{d}{2}\right\} \exp\left\{\frac{b^2}{2a}\right\} \int \sqrt{\frac{a}{2\pi}}\sqrt{\frac{2\pi}{a}} \exp\left\{-\frac{\left(\mu - \frac{b}{a}\right)^2}{2\left(\frac{1}{\sqrt{a}}\right)^2}\right\} d\mu \tag{C.12}$$

$$= c \exp\left\{-\frac{d}{2}\right\} \exp\left\{\frac{b^2}{2a}\right\} \sqrt{\frac{2\pi}{a}} \int \sqrt{\frac{a}{2\pi}} \exp\left\{-\frac{\left(\mu - \frac{b}{a}\right)^2}{2\left(\frac{1}{\sqrt{a}}\right)^2}\right\} d\mu \tag{C.13}$$

$$= 1$$

$$= c \exp\left\{-\frac{d}{2}\right\} \exp\left\{\frac{b^2}{2a}\right\} \sqrt{\frac{2\pi}{a}} \tag{C.14}$$

$$\tag{C.15}$$

Back-substituting

$$\int \prod_{n=1}^{N} \mathcal{N}(x_n; \mu, \sigma^2)\mathcal{N}(\mu; \mu_0, \sigma_0^2)\, d\mu \tag{C.16}$$

$$= \frac{1}{\left(\sqrt{2\pi\sigma^2}\right)^n} \frac{1}{\sqrt{2\pi\sigma_0^2}} \exp\left\{-\frac{d}{2}\right\} \exp\left\{\frac{b^2}{2a}\right\} \sqrt{\frac{2\pi}{a}} \tag{C.17}$$

$$= \frac{\sigma}{(\sqrt{2\pi}\sigma)^n \sqrt{n\sigma_0^2 + \sigma^2}} \exp\left\{-\frac{1}{2}\left(\frac{1}{\sigma^2}\sum_{n=1}^{N} x_n^2 + \frac{\mu_0^2}{\sigma_0^2}\right)\right\} \exp\left\{\frac{\left(\frac{1}{\sigma^2}\sum_{n=1}^{N} x_n + \frac{\mu_0}{\sigma_0^2}\right)^2}{2\frac{n\sigma_0^2 + \sigma^2}{\sigma^2\sigma_0^2}}\right\} \tag{C.18}$$

$$= \frac{\sigma}{(\sqrt{2\pi}\sigma)^n \sqrt{n\sigma_0^2 + \sigma^2}} \exp\left\{-\frac{1}{2}\left(\frac{1}{\sigma^2}\sum_{n=1}^{N} x_n^2 + \frac{\mu_0^2}{\sigma_0^2}\right)\right\} \exp\left\{\frac{\left(\sigma_0^2\sum_{n=1}^{N} x_n + \sigma^2\mu_0\right)^2}{2\sigma^2\sigma_0^2\left(n\sigma_0^2 + \sigma^2\right)}\right\} \tag{C.19}$$

Appendix D

OpenGL Implementation Details

D.1 OpenGL projective texture transformation

In this section we discuss the implementation of projective texture transformations in OpenGL, and how to relate image pixels to texture texels.

D.1.1 Projective Transformation

We begin by noting that we can write the transformation from 3D world point to image space as:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \mathbf{P}_{cam}\mathbf{V}_{cam} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \tag{D.1}$$

where

$\mathbf{V}_{cam}$ is the projective camera view matrix,

$\mathbf{P}_{cam}$ is the projective camera projection matrix.

We complete the transformation from image projective space to pixels by

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ w' \end{bmatrix} = \begin{bmatrix} \frac{1}{2}w & 0 & \frac{1}{2}w \\ 0 & -\frac{1}{2}h & h - \frac{1}{2}h \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(x' + w')w \\ hw' - \frac{1}{2}(y' + w')h \\ w' \end{bmatrix} \tag{D.2}$$

Note that in equation (D.2) we have ignored the z' term since our images are 2 dimensional (we have also written the term $h - \frac{1}{2}h$ explicitly since it will be convenient later. Also note, that the matrix $diag(\frac{1}{2})$ used above is needed due to OpenGL internal storage of pixels.

We can obtain the final pixel location by:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{x'}{w'} \\ \frac{y'}{w'} \end{bmatrix} \tag{D.3}$$

■ D.1.2 Projective Texture Mapping

We begin by noting that we can write the texture transformations as:

$$\begin{bmatrix} s \\ t \\ r \\ q \end{bmatrix} = \mathbf{T}_{object} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \tag{D.4}$$

where

$$\mathbf{T}_{object} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{P}_p \mathbf{V}_p \mathbf{M} \tag{D.5}$$

where

$\mathbf{M}$ is an object to world transformation (identity in our cases, i.e. the object is expressed in world coordinates).

$\mathbf{V}_p$ is the projective camera view matrix,

$\mathbf{P}_p$ is the projective camera projection matrix.

This is obtained from [14]

Let

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \mathbf{P}_p \mathbf{V}_p \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \tag{D.6}$$

Note that equation (D.6) is simply a projective transformation of a 3D point.

Then, using, equation (D.6) in equations (D.4) and (D.5), we can write:

$$\begin{bmatrix} s \\ t \\ r \\ q \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} \frac{1}{2}x' + \frac{1}{2}w' \\ \frac{1}{2}y' + \frac{1}{2}w' \\ \frac{1}{2}z' + \frac{1}{2}w' \\ w' \end{bmatrix} \tag{D.7}$$

Equation (D.7) is the projective texture mapping for a point in 3D world to a 3D texture. In our case, the textures are 2D, so we can disregard the t component of the texture. These are un-normalized texture coordinates. The next step is to do a range mapping to $[0, 1]$ for each dimension.

The easiest way to do so, is to equate it to the orthographic projection we had working before: $1-(h-(1/2y'/w'+1/2)h+0)/hTex$. So we want to map this expression, to something like $\frac{a}{w'}$. Where a is the term we are looking for

In what follows, h is the height of the original image, h_{tex} is the height of the texture image (power of 2), by creation $h_{tex} > h$. Similarly for w and w_{tex}, the width of image and texture respectively.

We can achieve the range mapping as follows:

$$\frac{1-\left[h-(\frac{1}{2}\frac{y'}{w'}+\frac{1}{2})h+0\right]}{h_{tex}}=\frac{h_{tex}-\left[h-(\frac{1}{2}\frac{y'}{w'}+\frac{1}{2}\frac{w'}{w'})h\right]}{h_{tex}}=$$
$$\frac{h_{tex}-h+(\frac{1}{2}\frac{y'}{w'}+\frac{1}{2}\frac{w'}{w'})h}{h_{tex}}=\frac{h_{tex}-h+(\frac{1}{2}y'+\frac{1}{2}w')\frac{h}{w'}}{h_{tex}}=$$
$$\frac{(h_{tex}-h)w'+(\frac{1}{2}y'+\frac{1}{2}w')h}{h_{tex}w'}=\frac{\frac{(h_{tex}-h)w'+(\frac{1}{2}y'+\frac{1}{2}w')h}{h_{tex}}}{w'}$$

From above we can see that:

$$\frac{(h_{tex}-h)w'+\frac{1}{2}(y'+w')h}{h_{tex}} \tag{D.8}$$

is the term we are looking for. Note that the mirroring introduces an extra offset of $(1-\frac{h}{h_{tex}})w'$ (the first term in the equation above).

Similar procedure can be used to obtain s (without having to do the mirroring).

The final projective texture coordinates can be written as:

$$\begin{bmatrix} s \\ t \\ 0 \\ q \end{bmatrix} = \begin{bmatrix} \frac{1}{2}(x'+w')\frac{w}{w_{tex}} \\ \left[(h_{tex}-h)w'+\frac{1}{2}(y'+w')h\right]h_{tex}^{-1} \\ 0 \\ w' \end{bmatrix} \tag{D.9}$$

■ D.1.3 Pixels to Textures

Here we try to connect the pixel locations to texture locations. We can compare equations (D.2) and (D.9). We can see that the un-normalized pixel coordinates $[\tilde{x} \quad \tilde{y} \quad \tilde{w}]$ are very similar to the $[s \quad t \quad r \quad q]$ coordinates.

A substitution of common terms yields:

$$\begin{bmatrix} s \\ t \\ 0 \\ q \end{bmatrix} = \begin{bmatrix} \underbrace{\frac{1}{2}(x'+w')w}_{\tilde{x}} \; {w_{tex}}^{-1} \\ \left[(h_{tex}-h)w'+\underbrace{\frac{1}{2}(y'+w')h}_{hw'-\tilde{y}}\right]h_{tex}^{-1} \\ 0 \\ w' \end{bmatrix} = \begin{bmatrix} \tilde{x}\,{w_{tex}}^{-1} \\ (h_{tex}w'-\tilde{y})\,h_{tex}^{-1} \\ 0 \\ w' \end{bmatrix} \tag{D.10}$$

Bibliography

[1] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski. Building Rome in a day, 2009.

[2] J. Alon and S. Sclaroff. Recursive Estimation of Motion and Planar Structure. *IEEE Conference on Computer Vision and Pattern Recognition (2008)*, 2(Cvpr):2550, 2000.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[4] C. Baillard and A. Zisserman. Automatic reconstruction of piecewise planar models from multiple views. *Proceedings 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Cat No PR00149*, 2:559–565, 1999.

[5] A. Bartoli and P. Sturm. Constrained Structure and Motion From Multiple Uncalibrated Views of a Piecewise Planar Scene. *International Journal of Computer Vision*, 52(1):45–64, 2003.

[6] A. Bartoli, P. Sturm, and R. Haraud. Projective structure and motion from two views of a piecewise planar scene. *Proceedings Eighth IEEE International Conference on Computer Vision ICCV 2001*, 1(C):593–598, 2001.

[7] P. A. Beardsley, A. Zisserman, and D. W. Murray. Sequential updating of projective and affine structure from motion. *International Journal of Computer Vision*, 23(3):235–259, 1997.

[8] D. C. Brown. The bundle adjustment - progress and prospects. In *Int Archives Photogrammetry*, volume 21, pages 1–33, 1976.

[9] L.-c. Chen, T.-a. Teo, Y.-c. Shao, Y.-c. Lai, and J.-y. Rau. Fusion of LiDAR Data and Optical Imagery for Building Modeling. *Building*, 35(B4):2–7, 2003.

[10] L.-C. C. L.-C. Chen, T.-A. T. T.-A. Teo, J. Y. Rau, J.-K. L. J.-K. Liu, and W.-C. H. W.-C. Hsu. Building reconstruction from LIDAR data and aerial imagery, 2005.

[11] F. Dellaert, S. M. Seitz, C. E. Thorpe, and S. Thrun. Structure from motion without correspondence. *IEEE Transactions on Medical Imaging*, 26(2):557–564, 2000.

[12] A. R. Dick, P. H. S. Torr, S. J. Ruffle, and R. Cipolla. Combining Single View Recognition and Multiple View Stereo For Architectural Scenes. In *IEEE International Conference on Computer Vision*, volume 00, pages 268–280. IEEE Computer

Society, 2001.
[13] Digia. Qt Project, 2013.
[14] C. Everitt. Projective texture mapping.
[15] O. Faugeras, Q. Luong, S. Maybank, and G. Sandini. Camera self-calibration: Theory and experiments. In *European Conference on Computer Vision*, volume 588, pages 321–334. Springer Berlin / Heidelberg, 1992.
[16] O. Faugeras and F. Lustman. Motion and Structure from Motion in a Piecewise Planar Environment, 1988.
[17] A. Fischer. Extracting Buildings from Aerial Images Using Hierarchical Aggregation in 2D and 3D. *Computer Vision and Image Understanding*, 72(2):185–203, 1998.
[18] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.
[19] A. W. Fitzgibbon and A. Zisserman. Automatic camera recovery for closed or open image sequences. *Computer VisionECCV98*, 1:311–326, 1998.
[20] D. A. Forsyth, S. Ioffe, and J. Haddon. Bayesian structure from motion. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 00(648):660–665 vol.1, 1999.
[21] D. A. Forsyth and J. Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2002.
[22] K. Fujii and T. Arikawa. Reconstruction of 3D urban model using range image and aerial image, 2001.
[23] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1362–76, 2010.
[24] D. Gallup, J.-M. Frahm, and M. Pollefeys. Piecewise planar and non-planar stereo for urban scene reconstruction. In *Computer Vision and Pattern Recognition (CVPR)*, 2010.
[25] Gilles Debunne. libQGLViewer, 2013.
[26] P. Gurram, H. Rhody, J. Kerekes, S. Lach, and E. Saber. 3D Scene Reconstruction through a Fusion of Passive Video and Lidar Imagery, 2007.
[27] K. Hammoudi and F. Dornaika. A Featureless Approach to 3D Polyhedral Building Modeling from Aerial Images. *Sensors (Peterboroug*, 11(1):228–259, 2010.
[28] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*, volume 16. Cambridge University Press, 2004.
[29] R. I. Hartley and P. Sturm. Triangulation. *Computer Vision and Image Understanding*, 68(2):146–157, Nov. 1997.
[30] M. E. Hodgson and P. Bresnahan. Accuracy of Airborne Lidar-Derived Elevation : Empirical Assessment and Error Budget. *Photogrammetric Engineering Remote Sensing*, 70(3):331–339, 2004.
[31] J. H. J. Hu, S. Y. S. You, and U. Neumann. Integrating LiDAR, Aerial Image and Ground Images for Complete Urban Building Modeling, 2006.
[32] T. S. Huang and A. N. Netravali. Motion and structure from feature correspondences: a review. *Proceedings of the IEEE*, 82(2):252–268, 1994.

[33] M. Huber, W. Schickler, S. Hinz, and A. Baumgartner. Fusion of LIDAR data and aerial imagery for automatic reconstruction of building surfaces, 2003.
[34] Khronos Group. Open Graphics Library (OpenGL), 2013.
[35] H.-H. Liao, Y. Lin, and G. Medioni. Aerial 3D reconstruction with line-constrained dynamic programming, 2011.
[36] M. A. Lourakis and A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Trans. Math. Software*, 36(1):1–30, 2009.
[37] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.
[38] D. J. C. Mackay. *Information Theory , Inference , and Learning Algorithms*, volume 22. Cambridge University Press, 2003.
[39] NVIDIA. Compute Unified Device Architecture (CUDA), 2013.
[40] C. Nvidia. NVIDIA CUDA C Programming Guide. *Changes*, (350):173, 2011.
[41] Ohio Statewide Imagery Program Data and Ohio Geographically Referenced Information Program. Ohio Statewide Imagery Program Data.
[42] M. Pollefeys, D. Nistér, J. M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S. J. Kim, P. Merrell, C. Salmi, S. Sinha, B. Talton, L. Wang, Q. Yang, H. Stewénius, R. Yang, G. Welch, and H. Towles. Detailed Real-Time Urban 3D Reconstruction from Video. *International Journal of Computer Vision*, 78(2-3):143–167, 2007.
[43] G. Qian and R. Chellappa. Structure from Motion Using Sequential Monte Carlo Methods. *International Journal of Computer Vision*, 59(1):5–31, 2004.
[44] C. Rasmussen and C. Williams. *Gaussian processes for machine learning*. The MIT Press, 2006.
[45] H. Samet. *Foundations of multidimensional and metric data structures*. Morgan Kaufmann Publishers, San Francisco, 2006.
[46] J. Sanders and E. Kandrot. *CUDA by Example*, volume 54. Addison-Wesley, 2010.
[47] D. Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, versions 3.0 and 3.1*. Addison Wesley, 7 edition, 2010.
[48] S. N. Sinha, D. Steedly, and R. Szeliski. Piecewise planar stereo for image-based rendering, 2009.
[49] N. Snavely, S. M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. *Image Rochester NY*, 25(3):835–846, 2006.
[50] N. Snavely, S. M. Seitz, and R. Szeliski. Modeling the World from Internet Photo Collections. *International Journal of Computer Vision*, 80(2):189–210, 2007.
[51] H. Steuer. Height snakes: 3D building reconstruction from aerial image and laser scanner data, 2011.
[52] P. Sturm and B. Triggs. A factorization based algorithm for multi-image projective structure and motion. *Lect Notes Comput Sci*, 1065(April):709–720, 1996.
[53] I. Suveg and G. Vosselman. Reconstruction of 3D building models from aerial images and maps. *ISPRS Journal of Photogrammetry and Remote Sensing*, 58(3-4):202–224, Jan. 2004.
[54] R. Szeliski. Computer Vision : Algorithms and Applications. *Computer*, 5:832, 2010.

[55] R. Szeliski and P. H. S. Torr. Geometrically Constrained Structure from Motion : Points on Planes 1 Introduction. *Lecture Notes in Computer Science*, 1506:171–193, 1998.

[56] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9(2):137–154, 1992.

[57] B. Triggs, P. F. Mclauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle Adjustment A Modern Synthesis. *Synthesis*, 34099:298–372, 2000.

[58] F. T. F. Tsai, T.-A. T. T.-A. Teo, L.-C. C. L.-C. Chen, and S.-J. C. S.-J. Chen. Construction and visualization of photo-realistic three-dimensional digital city, 2009.

[59] US Air Force. Columbus Large Image Format (CLIF) 2007 Dataset .

[60] J. Vandekerckhove, D. Frere, T. Moons, and L. V. Gool. Semi-automatic modelling of urban buildings from high resolution aerial imagery, 1998.

[61] S. W. S. Weng, G. Z. G. Zhao, and B. H. B. He. Rapid reconstruction of 3D building models from aerial images and LiDAR data, 2010.

[62] M. X. M. Xie, K. F. K. Fu, and Y. W. Y. Wu. Building Recognition and Reconstruction from Aerial Imagery and LIDAR Data, 2006.

[63] G. X. G. Xu, J. Terai, and H.-Y. S. H.-Y. Shum. A linear algorithm for camera self-calibration, motion and structure recovery for multi-planar scenes from two perspective images, 2000.

[64] Y. Zhang, Z. Zhang, J. Zhang, and J. Wu. 3D Building Modelling with Digital Map, Lidar Data and Video Image Sequences. *The Photogrammetric Record*, 20(111):285–302, 2005.