Reconstruction and Visualization of Architectural Scenes

Avanish Kushal

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

Doctor of Philosophy

University of Washington

2014

Reading Committee:

Steven M. Seitz, Chair

Sameer Agarwal

Ali Farhadi

Program Authorized to Offer Degree:
UW Computer Science and Engineering

0

University of Washington

Abstract

Reconstruction and Visualization of Architectural Scenes

Avanish Kushal

Chair of the Supervisory Committee:
Professor Steven M. Seitz
Computer Science and Engineering

Can we experience a scene virtually, such as the Colosseum in Rome, without ever having to visit it? Such an experience should replicate the feeling of being physically present, in terms of being able to visualize the scene from different viewpoints as well as quickly assimilating the highlights of the scene. An Internet search can increasingly provide us with a complete photographic record for the scene, but the challenge is in displaying such imagery in a coherent and informative way. In this document, I propose an approach for this problem based on 3D reconstruction and path planning.

To make this approach feasible, we need to overcome three primary challenges. The *first* is in scaling the reconstruction algorithms to process millions of 3D points and thousands of images in an efficient manner. We design effective preconditioners to solve the non linear Bundle Adjustment problem efficiently, to obtain significant reductions in execution time. The *second* challenge involves improving the quality of the 3D reconstructions. Despite decades of research, state-of-the-art stereo algorithms cannot produce quality reconstructions everywhere, due to their dependence on the presence of texture. We complement stereo with monocular cues to overcome this challenge to compute more accurate and complete reconstructions. Finally having computed the reconstructions, a $third$ challenge is in creating compelling visual experiences to aid a user in effectively navigating through the scene. We automatically compute movies, *or photo tours*, as paths through the reconstruction that are coherent, informative and efficient, for famous sites all over the world.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I wish to thank my advisor Prof. Steven M. Seitz for nurturing me and helping me grow into an independent researcher, providing me a nice blend of direction when I needed it, and yet giving me freedom to trace my independent path. He taught me the importance of identifying the correct questions first before marching towards finding the answers. Striving for perfection while prioritizing things is an important lesson that I will carry with me throughout my life. Thank you Steve for being patient and supportive with my mistakes and guiding me through this tremendous journey, as well as for advice for life post my PhD.

I would also like to thank Prof. Ali Farhadi and Prof. Sameer Agarwal for serving on my supervisory committee and providing useful feedback at different stages of this dissertation. I was fortunate to work with a number of wonderful collaborators during the course of my PhD – Prof. Brian Curless, Prof. Yasutaka Furukawa, Dr. Carlos Hernandez, Prof. Sameer Agarwal, Dr. David Gallup, Dr. Sing Bing Kang, Mr. Ben Self. Every one of them brought their unique perspective and working style that I admire and respect. I would also like to thanks my undergraduate advisors Prof. Subhashis Banerjee and Prof. Prem Kalra at IIT Delhi, for introducing me to the wonderful field of computer vision.

I would like to thank the UW Department of Computer Science and Engineering for providing such a conducive environment for academic growth. The top class infrastructure combined with great professors, peers and staff indeed make it one of top computer science departments in the world. In particular I would like to thank Stephen Spencer and Lindsay Michimoto for always being there for administrative or infrastructure support during my stay at the Paul Allen Center.

Last but not the least, I thank my family (Mom, Dad, my brother Akash, and my sister-in-law Amrita) for their unconditional love. I am also indebted to my closest friend Jagriti, for her emotional support during these past five years.

DEDICATION

to my parents

Chapter 1

INTRODUCTION

Can we experience a scene virtually, such as the Colosseum in Rome, without ever having to visit it? Such an experience should replicate the feeling of being physically present, in terms of being able to visualize the scene from different viewpoints, and quickly assimilating the highlights of the scene. What makes this problem challenging? The challenge is not a lack of good imagery—there's plenty available via Internet search—but rather displaying such imagery in a coherent and informative way. In this document I propose solutions for creating compelling visualizations for scenes in a scalable and efficient manner. One immediate application is in creating virtual tours for famous tourist sites. Indeed, our approach has been commercialized as the Photo Tours feature in Google Maps to create tens of thousands of tours all over the world [3]. Figure 4.1 shows the *world scale* deployment of our approach — each photo tour (shown as a red dot on the map) is a movie which includes fluid 3D transitions between a sequence of photographs of the scene. To achieve this, we crawled more than a million geo-tagged user photos, clustered them into thousands of individual sites, reconstructed camera positions, scene geometry, and popular viewpoints, planned optimal tours, and rendered fly-through movies of each site.

To create such a visual experience, we require hundreds or thousands of photographs. The spread of the Internet and the growing amount of visual information online, provides us with a rich source of data for this problem. For example, a query for the "colosseum" on $Google$ or $Flickr$ yields millions of images taken from different viewpoints and at different times of the day. But displaying this collection of images to users does not provide them with a full understanding of the scene, primarily because it is difficult to tell how these individual images physically relate with each other. Further, the sheer size and redundancy in these collections makes it difficult for a user to make sense of them. Instead, what we seek is an experience that is coherent and informative. Coherent, so that as users transition from one image to the next, they can maintain context. To achieve this we need to know the position of where these images were taken in the scene, and as we transition, how the scene

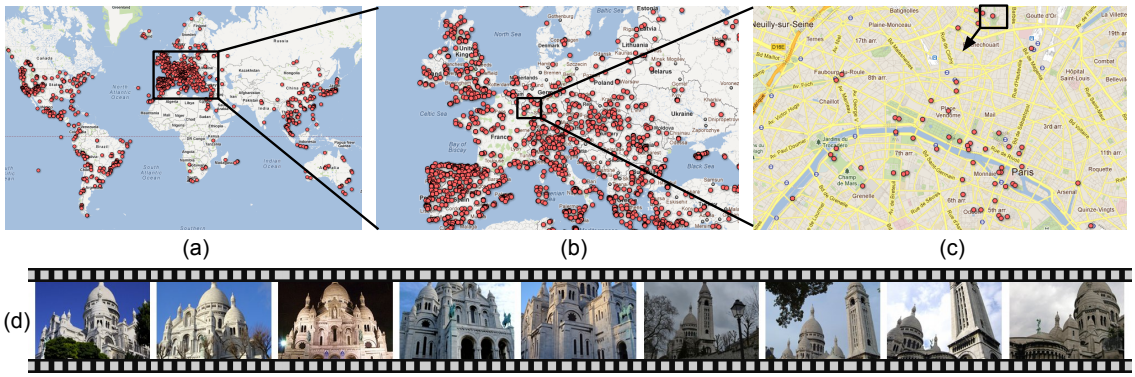


Figure 1.1: We have computed thousands of *photo tours* across the globe, shown as red dots (a). (b) shows a closeup of Europe, and (c) a zoom-in to one neighborhood in Paris. (d) shows the sequence of views in the photo tour for Sacre-Couer near Paris; the movie itself includes 3D transitions between consecutive views.

looks from a virtual viewpoint. Informative, so that the user can quickly and efficiently assimilate the highlights of a scene. To achieve this we need to identify paths through the scene that visit the highlights in an efficient and smooth manner. Thus one approach to creating such experiences is to compute the 3D geometry of the scene and the 3D location of the images to achieve coherence, and then plan efficient and informative paths through the reconstruction. I use this path planning approach of 3D reconstruction and visualization of scenes to create these experiences.

While this 3D reconstruction and path planning approach can in principal provide coherent and informative experiences, we need to overcome three primary challenges to make this approach feasible. The *first* challenge is in efficiently scaling the reconstructions. With the growing size of the reconstructions in terms of both coverage and quality, we need algorithms that can process millions of 3D points and thousands of images in an efficient manner. The *second* challenge involves improving the quality of the 3D reconstructions. While this problem has been researched for decades, state-of-the-art algorithms cannot produce quality reconstructions everywhere. In particular, these reconstructions rely on the presence of texture in the scenes for feature matching across images. Absence of texture leads to holes or incorrect geometry in the reconstruction. Finally having computed the reconstructions, a $third$ challenge is in creating compelling visual experiences for the users to aid them in effectively navigating through the scenes. Below, I discuss these three challenges and propose improvements over the state-of-the-art for each to create compelling experiences.

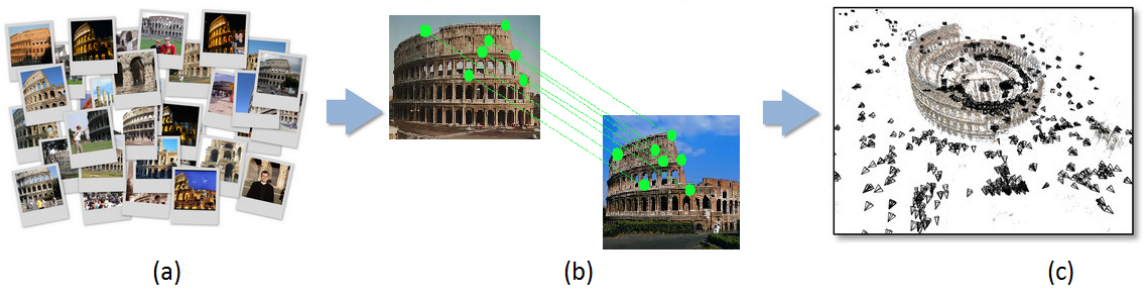


Figure 1.2: The figure (adapted from [92]) shows the reconstruction pipeline. Given a collection of photographs shown in (a), features are detected and matched across pairs of images as shown in (b). These features are then used to compute the 3D point cloud for the scene and the 3D location and parameters of the cameras using bundle adjustment, shown in (c).

1.1 Scalability of the Reconstructions

With the growing size of the reconstructions, we constantly require faster algorithms to deal with the large amounts of data. Given a collection of images, reconstruction approaches first compute discriminating $features$ in the images [69]. These features provide invariance to scale, rotation and lighting changes across different images. These features are then matched robustly across images [6, 90] to generate correspondences between images. Given the feature matches, the bottleneck in the reconstruction system is the problem of bundle adjustment – the joint non-linear refinement of camera parameters and the 3D scene to minimize the reprojection error [101]. This reconstruction pipeline is shown in Figure 1.2. What makes this problem challenging is the non linear objective function which is difficult to optimize, especially with the increasing dimensionality of the problem. We introduce a novel technique for solving this problem by constructing efficient, high quality preconditioners. These preconditioners when coupled with an inexact Levenberg-Marquardt algorithm [106], give a $3-5$ times reduction in execution time over the state-of-the-art methods.

With a few exceptions [22, 100], most of the successful bundle adjustment methods formulate the bundle adjustment problem as a non-linear least squares problem and use some variant of the Levenberg-Marquardt algorithm to solve it [77]. Levenberg-Marquardt operates by repeatedly linearizing the objective function into a linear least squares problem and solving its *normal equations*. Thus, reducing the cost of bundle adjustment comes down to reducing the number of times the

normal equations are solved, and reducing the cost of each individual solve. Traditionally, bundle adjustment algorithms have exploited the primary sparsity structure of the bundle adjustment problem (also known as the Schur complement trick) and a sparse or dense Cholesky factorization of the resulting Schur complement matrix [68, 101], but the huge memory requirements of factorization based methods have made them infeasible for large problems. Instead, interest has shifted to Conjugate Gradient (CG) based methods [5, 13, 49, 50, 107], and in particular in designing effective preconditioners for them [81]. Preconditioning is a technique for improving the condition number of a linear system, by considering easy to factorize approximations of the system. Agarwal et al. [5] look at constructing such preconditioners by considering the block diagonal structure of the matrix. However these preconditioners do not take into account the coupling between pairs of cameras, and thus are not a good approximations of the original matrix. Based on the idea that the number of 3D points visible to a pair of cameras is an indicator of the strength of their coupling, we compute better approximations for the system in our work titled *Visibility Based Preconditioning* [59]. In chapter 2, I describe the bundle adjustment problem in detail and our technique to solve this problem efficiently.

1.2 Quality of the Reconstructions

A *second* challenge in creating effective visualizations is the quality of reconstructions — poor 3D reconstructions lead to hallucination or ghosting artifacts when the scene is visualized from a novel view point. The problem of computing the 3D structure from 2D images (or $stereo$) is one of the oldest problems in computer vision, and has been studied across different domains such as robotics, medical imaging, artificial intelligence and machine learning. Indeed, state-of-the-art stereo algorithms can reconstruct complex objects up to sub-millimeter precision [85]. Yet, despite decades of research on stereo algorithms [11, 27, 84, 85, 97, 110], stereo methods fail on poorly textured, piece-wise planar scenes such as houses and room interiors [32]. Ironically, such scenes are composed of the simplest possible shapes: large planes, axis-aligned boxes, etc.

Instead, we seek to obtain reconstructions, that exhibit the same metric, highly accurate results we've come to expect from stereo algorithms, together with the ability to accurately represent large planar, textureless surfaces. Our approach is motivated in part by the human vision system which

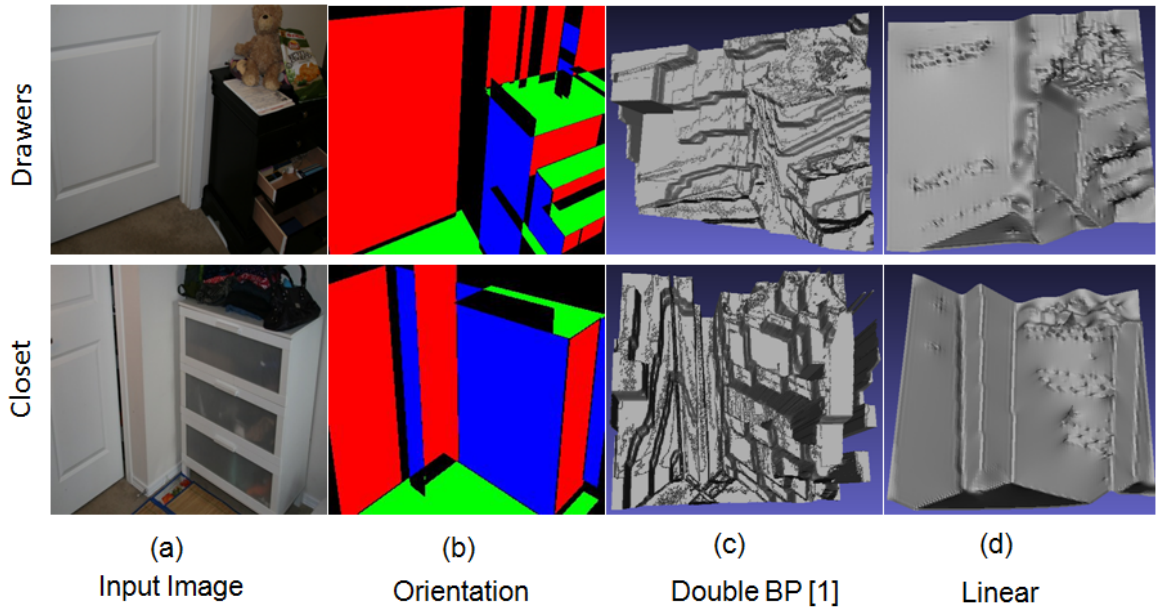


Figure 1.3: The figure shows for *two* scenes, (a) one input image part of a stereo pair, (b) the orientation map computed using the monocular cues (the colors R, G, B correspond to the *three* primary orthogonal directions), (c) the reconstructed range map using DoubleBP [111], a state-of-the-art stereo method on the Middelbury datasets [84] and (d) the reconstructed range map by integrating monocular cues with stereo using our linear relaxation.

interprets depths using both binocular cues as well as monocular cues such as linear perspective, texture gradient, shading etc. [55] and has no difficulty in interpreting the geometry for feature less environments. Our solution combines monocular cues (those readily obtained for architectural scenes, shown in Figure 3.1(b)) with binocular cues, to obtain reconstructions that outperform the state-of-the-art stereo algorithms, as shown in Figure 3.1(d). While stereo algorithms are typically non linear and operate in a discrete solution space, our approach computes a linear relaxation in a continuous solution space, thus reducing the time complexity as well as modelling the non fronto parallel surfaces in architectural scenes without aliasing and metric errors.

Significant progress has been made in recent years in modeling architectural structures using single-view techniques[23, 25, 46, 62, 83]. However, a limitation of these SVR methods is that only a coarse approximation of the scene is reconstructible, owing to the strong assumptions needed to make the problem tractable. Further, most of these techniques, both automatic [25, 62] as well as

interactive techniques [23, 96] are often restricted to a specific class of scenes called *Manhattan* scenes, which contain planes along 3 mutually orthogonal directions. By combining stereo with these cues, we can overcome the ill-posed nature of SVR methods. We also generalize these approaches to a new class of *Piecewise swept surfaces* [60], thereby capturing a broader range of architectural scenes.

1.3 Visualizing the Reconstructions

Having constructed the 3D models, a *third* challenge is creating useful scene visualizations, that achieve our objectives of being coherent, informative and efficient. We encode these objectives as constraints and objectives in our formulation and optimize them directly. Rather than relying on user interaction, we instead pose the problem of automatically generating the sequence of frames that best conveys the essence of that scene. We call such a sequence a *photo tour* — an automatically generated movie that serves as a informative guide for the scene.

We look upon creating such a *photo tour* as a path planning problem through an image graph consisting of a node for each image and an edge between a pair of nodes if they share common visible 3D points. A tour on this graph is a sequence of nodes that we would like to visit to convey the feel of the scene. We make the tour informative by computing a set of *canonical views [88]* capturing the most frequently photographed scene content, and constraining the tour to include these nodes, as shown in Figure 1.4(b). We address efficiency by posing this as a traveling salesman problem (TSP) [7] on this graph to compute the shortest tour (shown in Figure 1.4(b)), under an appropriate cost function, that enforces coherence by choosing edges in the graph that encourage high quality transitions — the transitions themselves are created by moving a virtual camera along the edges between the nodes, using techniques from image based rendering(IBR) [15, 24, 41] to generate 3D movies.

Our system represents the first attempt to deploy an image-based rendering (IBR) system *at world-scale* by harvesting the vast stores of community photo collections on the Internet. The photo tours feature in Google Maps implements our method to generate movies for thousands of sites. These sites are indicated as red dots on the maps in Figure 4.1, which shows the distribution of *photo tours* across the globe, in Europe and around Paris. Photo Tourism [92], behind the Microsoft

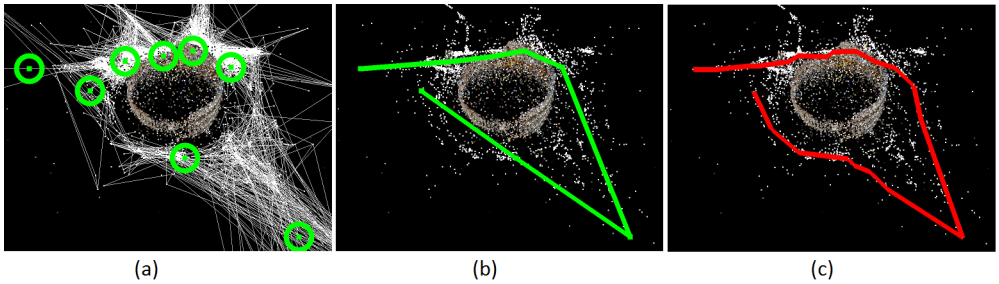


Figure 1.4: For the Colosseum (a) shows the point cloud overlayed with the graph and canonical views highlighted as green circles. (b) shows the ordering amongst the canonical views after solving the Traveling Salesman Problem. (c) shows the final computed tour through the camera centers in red that maximizes our objectives.

Research's popular product Photosynth [2], is another popular system for browsing large collections of photographs in 3D. Being interactive, it requires considerably mastery to use its different viewing modes and controls, and even for an expert it can be difficult to find his way around a new scene. Instead, watching a photo tour is like looking over the shoulder of an expert Photosynth user, who knows all the highlights and how to traverse them. Furthermore, the movie can communicate the most interesting aspects of the scene in a relatively short amount of time.

The primary contribution of this thesis is in computing reconstructions and creating compelling visual experiences for architectural scenes, both outdoor and indoor. We harvest millions of images to create *photo tours* at world scale. To achieve this, we make the following $three$ contributions to create such experiences.

- We design effective preconditioners to solve the non linear Bundle Adjustment problem efficiently, to obtain 3-5 times reduction in execution time for state-of-the-art BAL dataset [5]. I describe the bundle adjustment problem techniques and our proposed technique to solve it efficiently in Chapter 2.

- I show how combining stereo cues with monocular cues for architectural scenes can lead to reconstructions that outperform the state-of-the-art stereo algorithms on scenes containing planar textureless regions. In Chapter 3, I describe our approach and show results that outperform the state-of-the-art stereo systems.

- We automatically compute paths through the reconstruction that are coherent, informative and efficient, to create 3D movies for famous sites all over the world. I describe the tour generation algorithm in Chapter 4.

Chapter 2

SCALABILITY OF THE RECONSTRUCTIONS : VISIBILITY BASED PRECONDITIONING

As we have seen in Chapter 1, one way to create effective visualizations is to reconstruct the scene in 3D. Computing a 3D representation enables rendering the scene from new viewpoints and under different lighting conditions. This gives users the ability to explore the scene while maintaining context as they switch from one part of the scene to the other. 3D geometry can be represented in many different forms such as 3D point clouds or volumetric models in the scene domain, or simply as depth maps in the image domain.

With the creation and ubiquitous spread of the Internet, the amount of visual data available for human consumption is immense. Thus, a search for the colosseum in Rome returns millions of images on *Flickr* or *Google Images*. But processing this data also brings about its own challenges. Traditionally, multi-view stereo data sets used for reconstruction were captured in controlled settings and under known lighting conditions, often with the cameras parameters known, or easily recoverable. In contrast the Internet provides us with a collection of photographs from unknown view points, captured at different times of the day and night, using different camera parameters. Further, the growing size of the reconstructions, routinely involving *millions* of 3D points and *tens of thousands* of cameras, has led to the need for scalable algorithms for this problem. I now briefly describe the reconstruction pipeline that is used to compute 3D models from Internet photographs, and in particular focus on optimizing one of the time consuming steps in this pipeline.

2.1 Structure from Motion for Internet Photos

Recent work in Structure from Motion (SfM) has enabled 3D reconstructions from large unstructured community photo-collections [6, 29, 90] and reconstructions with thousands of images are now routinely computed, as shown in Figure 2.1(b). SfM algorithm relies on the ability to robustly find *features* and match them across images. Features refer to discriminative regions that can be

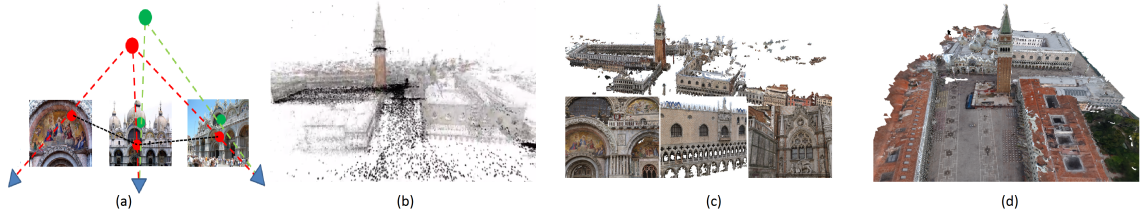


Figure 2.1: (a) shows the triangulation process of bundle adjustment, where detected and matched features across images, are used to recover the camera parameters shown in (b) for the San Macro data set, computed using [6]. Each black dot show a reconstructed cameras. (c) shows the result for [33] for the San Marco data set. The top image shows the entire reconstruction whereas, the bottom layer shows us zoomed in regions. (d) shows the result for the approach in [86], using aerial streetview photography as well.

matched across different images. Many features have been proposed for this [102] — a popular feature is called the Scale Invariant Feature Transform, or *SIFT* [69], which provides invariance to scale and $2D$ rotation. Having computed the features, the next step is to match the features (and tracks of features) robustly across images [6, 90].

Given the feature matches between images, the computational bottleneck in an SfM system is the bundle adjustment process – the joint non-linear refinement of camera parameters and the 3D scene to minimize the reprojection error [101]. The need to solve this problem with the growing size of reconstructions has sparked a renewed interest in scalable bundle adjustment algorithms [5, 49, 50, 64, 76, 93–95, 107].

However minimizing the energy function is very difficult. Firstly, it is non linear in the unknowns (camera parameters and 3d point cloud)Further the size of the problems can be very large with *thousands* of cameras and *millions* of points. With a few exceptions [22, 100], most of the successful bundle adjustment methods formulate the bundle adjustment problem as a non-linear least squares problem and use some variant of the Levenberg-Marquardt algorithm to solve it [77]. Levenberg-Marquardt operates by repeatedly linearizing the objective function into a linear least squares problem and solving its *normal equations*. Thus, reducing the cost of bundle adjustment comes down to reducing the number of times the *normal equations* are solved, and reducing the cost of each individual solve. Solving these equations efficiently is the focus of this chapter.

Bundle adjustment recovers a sparse representation for the scene. Typically, the next step in

the reconstruction is to use a multi-view stereo algorithm [85]. However, owing to the massive size of these collections (tens of thousands of photographs, and millions of 3D points), traditional multi-view stereo approaches are not directly usable. Recent work [33, 42] provide approaches enabling existing MVS methods to operate on these large unstructured collections to reconstruct dense point cloud representations, as shown in Figure 2.1(c). Further, with ariel photography now readily available to complement ground photographs of popular sites, the quality of reconstructions have further improved [86]. In particular, arial photography helps to geo-register individual models together for city-wide modeling. Further, they help in capturing ground plane details, often missed in personal photo collections, as shown in Figure 2.1(d).

The rest of the chapter is organized as follows. Section 2.2 presents a brief overview of the Bundle Adjustment problem and recent work on the use of preconditioned iterative methods for solving it. Section 2.3 describes the construction of two new preconditioners. Section 2.4 compares these new preconditioners to the state of the art using problems from the BAL dataset. We conclude with a discussion in section 2.5.

2.2 Bundle Adjustment

In this section, I present a brief overview of the bundle adjustment problem and methods to solve it. Please see Triggs et al. [101] for a comprehensive survey.

Given a set of measured image feature locations and correspondences, the goal of bundle adjustment is to find 3D point positions and camera parameters that minimize the reprojection error [101]. More formally, let x be the parameter vector (comprising all the 3D point positions and camera parameters) and $f(x) = [f_1(x), \ldots, f_k(x)]$ be the vector of reprojection errors (the error between the actual reprojected position on the images) for the 3D reconstruction. Then the bundle adjustment problem is formulated as the non-linear least squares problem:

$$x^* = \arg\min_x \sum_{i=1}^{k} \|f_i(x)\|^2. \tag{2.1}$$

Let $J(x)$ be the Jacobian of $f(x)$. Then in each iteration LM solves a linear least squares

problem of the form

$$\delta^* = \arg\min_{\delta} \left\| \begin{bmatrix} J \\ \sqrt{\lambda} D \end{bmatrix} \delta + \begin{bmatrix} f \\ 0 \end{bmatrix} \right\|^2 \tag{2.2}$$

and updates $x \leftarrow x + \delta^*$ if $\|f(x + \delta^*)\| < \|f(x)\|$. Here, $D(x)$ is a non-negative diagonal matrix, typically the square root of the diagonal of the matrix $J(x)^\top J(x)$ and λ is a non-negative parameter that controls the strength of regularization [77].

Before going further, lets make some notational simplifications. We will assume that the matrix $\sqrt{\lambda} D$ has been concatenated at the bottom of the matrix J (i.e., $\begin{bmatrix} J \\ \sqrt{\lambda} D \end{bmatrix}$) and similarly a vector of zeros has been added to the bottom of the vector f (i.e., $\begin{bmatrix} f \\ 0 \end{bmatrix}$) and the rest of our discussion will be in terms of J and f, i.e. the linear least squares problem.

$$\min_{\delta} \|J(x)\delta + f(x)\|^2. \tag{2.3}$$

Further, let $g(x) = -J(x)^\top f(x)$ and for notational convenience let us drop the dependence on x. Then by taking vector derivatives, we see that solving (2.3) is equivalent to solving the *normal equations*

$$J^\top J \delta = g \tag{2.4}$$

The solution of (2.4) in each iteration of the LM algorithm is the dominant computational cost in bundle adjustment.

Let the parameter vector be organized as $x = [x_c; x_p]$, where x_c is the camera parameter vector and x_p the point parameter vector. Similarly for δ, g, and J, we use subscripts c and p to denote the camera part and the point part respectively . Let $U = J_c^T J_c$, $V = J_p^T J_p$ and $W = J_c^T J_p$, then (2.4) can be re-written as the block structured linear system

$$\begin{bmatrix} U & W \\ W^T & V \end{bmatrix} \begin{bmatrix} \delta_c \\ \delta_p \end{bmatrix} = \begin{bmatrix} g_c \\ g_p \end{bmatrix}. \tag{2.5}$$

It is worth noting that for most bundle adjustment problems V is a block diagonal matrix, with blocks of size 3×3 and thus trivial to invert. This observation lies at the heart of the *Schur complement*

trick [71] used to solve this linear system efficiently, where by applying Gaussian elimination to the point parameters, we obtain a linear system consisting of just the camera parameters:

$$S\delta_c = r \tag{2.6}$$

where $r = g_c - WV^{-1}g_p$ and $S = U - WV^{-1}W^T$ is the Schur complement or the reduced camera matrix. Given the solution to (2.6), δ_p, the point parameters vector can be obtained by back-substitution:

$$\delta_p = V^{-1}(g_p - W^\top \delta_c) \tag{2.7}$$

Since S is symmetric positive-definite, Cholesky factorization can be used to exactly solve (2.6) [66]. But Cholesky factorization libraries, even ones like CHOLMOD [16] which exploit the sparsity structure of S, are space and time intensive making them prohibitively expensive for large problems. Thus, there has been a recent focus on Conjugate Gradients(CG) based methods for solving (2.4) and (2.6) [5, 13, 49, 50, 107]

CG based methods have a fraction of the memory usage of factorization based methods and can even be run matrix-free, where the Jacobian is never stored in memory [107]. However, the rate of convergence of CG depends on the condition number of the linear system being solved and bundle adjustment problems are notoriously ill-conditioned. This ill-conditioning occurs because of gauge ambiguity (the solution is invariant to global position and rotation if the scene) and wide variability in the sensitivity of the objective function to the different parameters, e.g., small changes in the translation of a camera affect the objective less than small changes in the radial distortion parameters.

One approach is to improve the condition number of the linear system [81] by using a *preconditioner*. A good preconditioner has the competing goals of reducing the condition number as much as possible while still being efficiently computable. Constructing such preconditioners is the subject of this chapter. I present *Visibility Based Preconditioning*, a new technique for constructing efficient preconditioners for bundle adjustment problems that improve the condition number of the system substantially. Based on the idea that the number of 3D points mutually visible to a pair of cameras is an indicator of the strength of their coupling, we present two preconditioners, cluster-jacobi and cluster-tridiagonal. The former is a block-diagonal preconditioner and the latter a block-tridiagonal preconditioner. When coupled with an inexact Levenberg-Marquardt algorithm [106],

these preconditioners provide significant improvements over the state-of-the-art performance on the BAL dataset [5].

In this chapter, I will look at solving (2.6) using Preconditioned Conjugate Gradients. Note that CG can be run on the Schur complement S without actually computing and storing it in memory [5, 107], by exploiting the relation

$$Sx = J_c^\top \left[J_c x - J_p \left[V^{-1} \left[J_p^\top \left[J_c x \right] \right] \right] \right]. \tag{2.8}$$

Thus, it is possible to perform one iteration of unpreconditioned CG on S, which is a smaller better conditioned linear system, at practically the same cost as one unpreconditioned iteration of CG on the Hessian $H = J^\top J$ [5, 107].

2.2.1 Related Work

Jeong et al. proposed using the band block diagonals of the Schur complement matrix S as preconditioners. They observed that amongst the various banded preconditioners, the block Jacobi preconditioner was a cheap and robust choice[49]. Byröd & Åström avoided the computation of H and S, and instead ran CG on J directly with an incomplete QR factorization based preconditioner [13]. Their construction is equivalent to running CG on H with a block Jacobi preconditioner. Agarwal et al. proposed the use of the block diagonal of $J_c^\top J_c$ and the block diagonal of S as preconditioners for S without storing S in memory explicitly [5]. Wu et al. [107] extended this work to a multicore Jacobian free bundle adjustment method. It ran CG on S by using (2.8) with the block diagonal of $J_c^\top J_c$ as the preconditioner.

More recently, there has been work towards designing preconditioners based on the combinatorial structure of the bundle adjustment problem [26, 50]. Inspired by the work in combinatorial preconditioning, the authors have proposed using low-stretch spanning tree approximations to H as preconditioners for (2.4).

2.3 Visibility Based Preconditioning

Recall that we are interested in the efficient iterative solution of the symmetric positive definite linear system $S\delta_c = r$. The convergence rate of CG on this linear system, depends on the condition number $\kappa(S)$ of S. If we use a preconditioner matrix M, the condition number changes to $\kappa(SM^{-1})$. The

computational cost of using M is the cost of computing M and evaluating the product $M^{-1}y$ for arbitrary vectors y. Thus, there are two competing factors to consider: 1) How much of S's structure is captured by M so that the condition number $\kappa(SM^{-1})$ is low, and 2) the computational cost of constructing and using M. It is usually the case that the more information M has about S, the more expensive it is to use. For example, Incomplete Cholesky factorization based preconditioners [63] have much better convergence behavior than the Jacobi preconditioner, but are also much more expensive to construct and factorize.

In designing new preconditoners for S, our point of departure is the block Jacobi preconditioner for S [5, 49] which is a simple approximation to S, but ignores all pairwise camera interactions. A better approximation would be one that includes off-diagonal block from S in the form of its band block diagonals [49]. Constructing these preconditioners however, has *t*wo challenges . First, the ordering of the cameras in S dictates which off diagonal blocks are included, and second, band diagonals of positive definite matrices are not guaranteed to be positive definite (unless the matrix being preconditioned is diagonally dominant). These challenges makes their use in CG problematic.

So the task ahead of us is to construct a symmetric positive definite matrix M, that accounts for the significant camera-camera interactions in S. This of course begs the question, how do we measure the interaction/coupling between a pair of cameras? A number of heuristic choices are possible — some that use the numerical values of the entries of S, while others only pay attention to its sparsity structure. We propose the use of scene visibility as a predictor of the coupling between cameras, i.e., In an SfM reconstruction, the strength of coupling depends positively on the number of points visible in both the cameras. Scene visibility has previously been used to speed up image matching and bundle adjustment [6, 30, 90]. We will now exploit this structure for preconditioning.

A significant advantage of using scene visibility over other measures is that it is independent of the actual numerical values of the camera and point parameters and in turn the numerical entries of S (which change in each iteration of the Levenberg-Marquardt algorithm [77]). Thus it can be computed once and used to determine the sparsity structure of the preconditioner before the start of the bundle adjustment algorithm. Yet, it contains more information than just the 0-1 structure of S, giving more weight to the interaction of two cameras that have 100 points in common, as compared to cameras that have 1 point in common. Our experimental results show that despite ignoring the magnitude of the entries in S, scene visibility leads to excellent preconditioning performance.

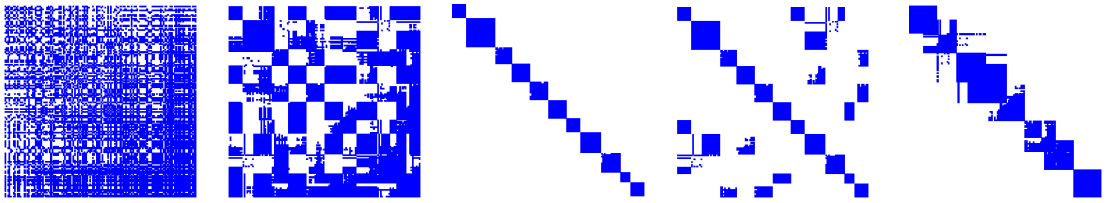


Figure 2.2: Visibility based preconditioning. S is the sparsity pattern of the Schur complement matrix for the ladybug-138 [5]. S_1 is the same matrix, with its rows and columns permuted using the permutation matrix P_1 induced by the clustering. cluster-jacobi preconditioner is the block diagonal of S_1. We show two views of the cluster-tridiagonal preconditioner, as a *degree 2* tree in S_1 and as a block tridiagonal matrix, after permuting it by P_2

2.3.1 Clustering

SfM problems, especially the ones involving community photo collections have highly non-uniform visibility — popular points of interest like entrances to landmarks and interesting locations have a very high concentration of images. On the other hand as we move from one popular viewpoint to another it is very common that there is little or no interaction between the images. Thus most of the interactions occur within dense clusters of cameras and show up as dense sub blocks in S. Identifying and accounting for these clusters in M should lead to a good approximation to S. Therefore the first step in our algorithm is to cluster the cameras.

If the scene contains n_p points, then each camera i can be described by a binary visibility vector $v_i \in \{0,1\}^{n_p}$, where the k^{th} entry is 1 if the point k is visible in camera i and 0 otherwise. Given these visibility vectors, we can now define a similarity measure between a pair of camera as the dot product of their corresponding normalized visibility vectors to cluster the cameras using a variety of clustering algorithms. We use the Canonical Views algorithm of Simon et al.[88]. The Canonical Views algorithm has been shown to be effective in identifying image clusters in 3D reconstructions. It greedily computes a set C of *canonical views* that maximizes the objective function

$$\sum_{i \in I} \max_{j \in C} \frac{v_i^\top v_j}{\|v_i\| \|v_j\|} - \alpha |C|. \tag{2.9}$$

Here, the first term maximizes coverage while the second term tries to minimize the number of canonical views selected. We use a fixed value of $\alpha = 2.2$ in all our experiments. Given the

canonical views, the clusters can be obtained by assigning each camera to the canonical view to which it is most similar. We observed that the average sizes of the clusters returned were independent of the problem.

2.3.2 Cluster-Jacobi

Given a clustering, the rows and columns of S can be permuted so that all the cameras from cluster 1 appear before all the cameras from cluster 2 and so on. Let's denote the matrix that performs this permutation by P_1 and let $S_1 = P_1 S P_1^\top$. Figure 2.3 shows the sparsity structure of a Schur complement matrix before and after the clustering induced permutation. Cameras within a cluster are expected to interact strongly with each other – this is reflected in the near dense super-blocks (one for each cluster) along the diagonal of S_1. Comparing S_1 to S we can see that we have moved a significant amount of the mass in S closer to the diagonal of S_1. We can now treat S_1 as a block matrix with blocks corresponding to the clusters. We will refer to the block diagonal of this matrix, as shown in Figure 2.3, as the cluster-jacobi preconditioner, in analogy with the block Jacobi preconditioner. The block Jacobi preconditioner is a strict specialization of cluster-jacobi with each cluster containing exactly one camera. Note that since S is positive definite, S_1, which is a symmetric permutation of S, is also positive definite. Thus cluster-jacobi, which is the block diagonal of a positive definite matrix, is also a positive definite matrix.

2.3.3 Cluster-Tridiagonal

The cluster-jacobi preconditioner only accounts for intra-cluster interactions. As can be seen in Figure 2.3, inter-cluster interaction can also be quite significant, showing up as dense off-diagonal blocks in S_1. Of course, accounting for all of them is not feasible, because then we would end up factorizing and inverting S. Instead, we want to add only those off-diagonal blocks which capture significant interactions between clusters and yet, still lead to an easily factorable matrix M. This also means that matrix M when factorized, should not introduce any fill-in, i.e. maintain the sparsity structure of M.

One class of matrices suitable for this are is the set of block band diagonal matrices whose Cholesky factorization does not introduce any fill-in . We consider the simplest of these - the block-

tridiagonal matrices. However, simply taking the block-tridiagonal of S_1 is not optimal, as the permutation P_1 will determine which off diagonal blocks will be included. Thus, our aim is to find a second permutation P_2 such that $S_2 = P_2 S_1 P_2^\top$ has as much mass as possible in its block-super and sub diagonals, i.e., as many of the inter cluster interactions as possible are accounted for. Finding such permutations is studied in the literature on *Bandwidth Minimization* and is known to be NP-Hard [40]. We propose a greedy heuristic suitable for bundle adjustment.

Define an undirected weighted graph $G(\mathcal{V}, \mathcal{E})$, whose vertices are the clusters, i.e, $\mathcal{V} = \{\mathcal{C}_1, \cdots, \mathcal{C}_k\}$. The weight on the edge connecting two vertices is the number of 3D points mutually visible in the two camera clusters it connects.

Now if S_2 were a block-tridiagonal matrix i.e. each cluster interacts with at most 2 clusters, the graph G would need to be a *degree* 2 tree – a tree where all its vertices have degree at most 2. A degree-2 tree is nothing more than a set of vertices on a line, and the corresponding permutation matrix P_2 can be obtained by traversing this line from one terminal vertex to another.

Thus to get a block-tridiagonal approximation of S_2, we first compute G_T, a *degree* 2 tree approximation of G. However, such an approximation, may not span the graph. For example a star graph, i.e. a graph with n-nodes where one node is connected to the other $n-1$ nodes does not have any spanning trees of degree less than $n-1$. To address this problem, we propose to approximate the graph G with a disjoint collection of degree-2 trees (i.e., a degree-2 forest) and P_2 is then computed by concatenating the orderings implied by the trees – the ordering amongst the trees themselves can be arbitrarily.

To compute the degree-2 forest we use a constrained variant of Kruskal's Minimum Spanning Tree (MST) algorithm. We start with a graph $G_T(\mathcal{V}, \{\})$, that has the same vertices as G and an empty edge set. We then iterate over the edges of G in decreasing order of weight, adding them to G_T if doing so does not create a cycle in G_T and the degree of all the vertices in G_T remains bounded by two. This $O(|E|)$ algorithm results in a disjoint collection of *degree* 2 trees that span the graph G.

Now S_2, like S_1, is a block matrix, where the blocks correspond to the clusters and the preconditioner cluster-tridiagonal contains the block diagonal, and the first super and sub-diagonal of S_2. Observe that an off-diagonal block corresponding to two clusters is included in the preconditioner if and only if the two clusters are connected by an edge in G_T. Figure 2.3 shows the *degree 2* tree

approximation, and the result of applying P_2 to it.

However, cluster-tridiagonal in its current form is not guaranteed to be positive definite. As we mentioned earlier, band diagonals of positive definite matrices can be indefinite. There are two ways to address this problem. The first is to find a *modified* Cholesky factorization $M = LL^\top - R$, where a correction R is applied to M to make it positive definite. Methods for determining a good R require complicated book keeping and are not challenging to implement [81]. For block tridiagonal matrices, there is, however, a simpler and more efficient static strategy – increase the diagonal dominance of the matrix M by scaling down the off-diagonal blocks of M. The following lemma (stated in scalar form, but easily generalizable to block matrices), whose proof can be found in the Appendix A, describes an *optimal* static scaling strategy.

Lemma 1. *Let A be a positive semidefinite symmetric matrix, then the tridiagonal matrix $M(\nu)$*

$$m_{ij}(\nu) = \begin{cases} a_{ij} & i = j \\ \nu a_{ij} & |i-j| = 1 \\ 0 & \textit{otherwise} \end{cases} \tag{2.10}$$

is positive semidefinite for $\nu = 0.5$ and for every $\epsilon > 0$, there exists a positive semidefinite matrix A, such that $M(0.5+\epsilon)$ is indefinite.

As if we have seen it is possible, but not necessary for the cluster-tridiagonal matrix, $M(1)$ to be indefinite. Thus we first construct the preconditioner for the unscaled version (i.e., without increasing the diagonal dominance). If its Cholesky factorization exists, we are done. If we encounter a non-positive pivot, we apply the scaling suggested by Lemma 2, i.e., use $M(0.5)$ and recompute the Cholesky factorization. This re-computation effort is a negligible fraction of the setup time, and helps us avoid unnecessary loss of mass in the preconditioner.

2.3.4 Implementation

Given the permutation matrices P_1 for cluster-jacobi and P_1P_2 for cluster-tridiagonal, we identify which of the blocks of the original matrix S contribute to each of the preconditioners. Only these entries of the Schur complement are computed. The resulting preconditioner matrix M is factorized

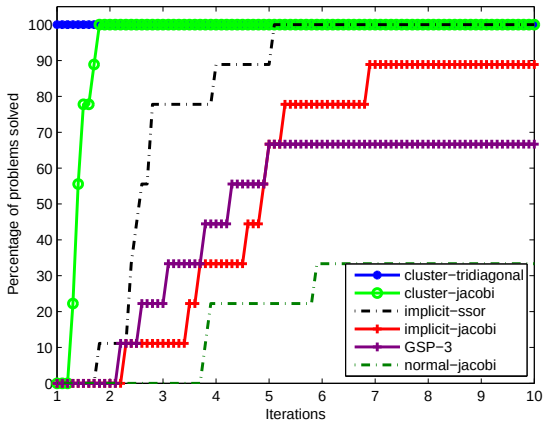


Figure 2.3: Performance profiles for the six preconditioners on small linear problems from the BAL dataset. Performance is based on the number of iterations to achieve the desired level of relative tolerance of $\tau = 10^{-6}$.

using CHOLMOD [16]. CHOLMOD's optimized triangular solve is also used to implement the matrix-vector product $M^{-1}y$ in each iteration of CG.

2.4 Experiments

We compare the performance of our preconditioners cluster-tridiagonal, cluster-jacobi with $five$ other linear solvers/preconditioners. implicit-jacobi and implicit-ssor perform CG on S using (2.8) with the block diagonal of $J_c^\top J_c$ and S as preconditioners respectively. normal-jacobi uses the block Jacobi preconditioner for H. gsp-3 is a combinatorial preconditioner for H that uses a low stretch maximum weight spanning tree [50]. Finally, explicit-sparse is a direct factorization based method that computes S and factorizes it using CHOLMOD [16]. All the preconditioners with the exception of gsp-3 were implemented as part of the same C++ library. Only a MATLAB implementation of gsp-3 is available from the authors.

2.4.1 Performance Profiles

We use *Performance Profiles* to report and compare the performance of the various solvers [28]. We briefly describe the method here.

Let $\mathcal{P}$ be a set of problems, $\mathcal{S}$ be a set of solvers and let $0 < \tau < 1$ be a user specified tolerance.

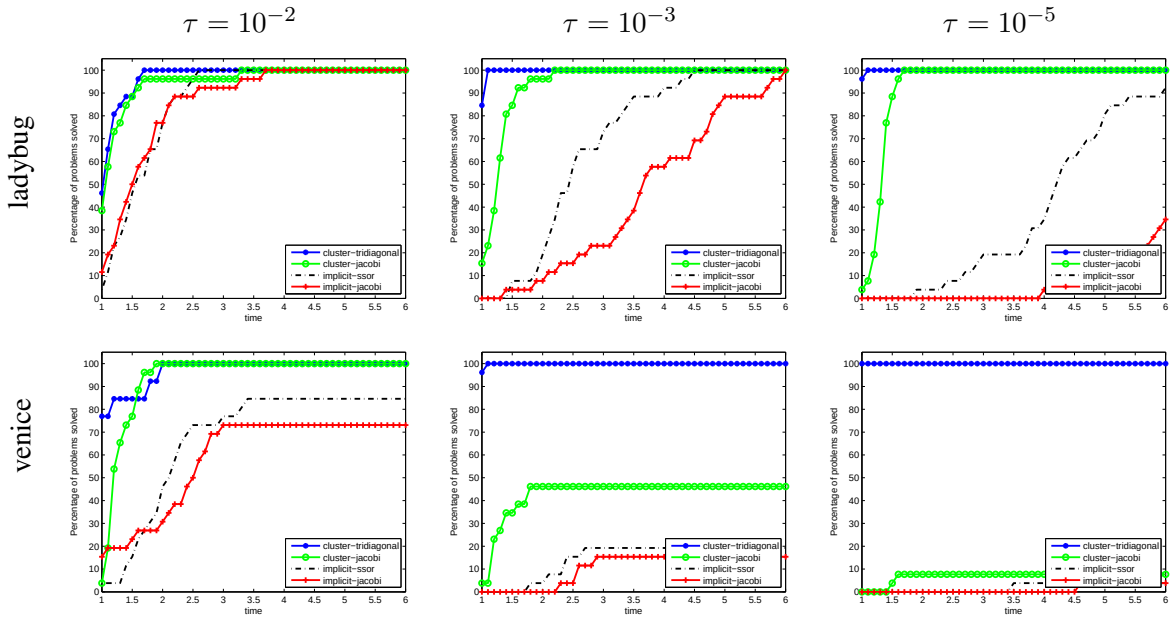


Figure 2.4: Performance profiles for the four Schur complement based preconditioners on linear problems from the BAL dataset. Performance measures the percentage of problems solved as a function of time.

For each problem $p \in \mathcal{P}$, run all the solvers $s \in \mathcal{S}$ until some convergence criterion (time, iteration or error tolerance) is satisfied.

Let $f(p, s)$ denote the minimum function value achieved on problem p using solver s. Let $f^*(p) = \min_s f(p, s)$, be the minimum value across all solvers for problem p. Define

$$f_\tau(p) = f^*(p) + \tau(f_0(p) - f^*(p)), \tag{2.11}$$

where $f_0(p)$ is the initial value of p.

Now to characterize the time performance of the solvers in $\mathcal{S}$, let $t(p, s)$ denote the time it takes solver s to reach $f_\tau(p)$, where $t(p, s) = \infty$ if the solver is unable to reach $f_\tau(p)$. Then the performance profile of a solver s over the problem set P is the curve

$$\rho(s, \alpha) = 100 \times \frac{|\{p : t(p, s) < \alpha \min_s t(p, s)\}|}{|\mathcal{P}|} \tag{2.12}$$

$\rho(s, \alpha)$ is a non-decreasing function of α. It measures the percentage of problems that are solved to a relative tolerance τ by solver s in time bounded by $\alpha \min_s t(p, s)$. A perfect solver (one that solves all the problems the fastest) will have a performance profile $\rho(s, \alpha) = 100, \forall \alpha \geq 1$.

2.4.2 Linear Problems

We begin by comparing the performance of the six iterative solvers (minus explicit-sparse which is not an approximate solver) on linear least squares problems, based on the number of iterations to converge. Since there is no high-performance implementation of gsp-3 available, comparisons based on execution time would not be fair. Further, the MATLAB implementation made available by the authors does not scale to large problems. Therefore, we selected five small bundle adjustment problems from the ladybug dataset and four small problems from the venice dataset[1]. For each problem we generated a linear least squares problem at the initial estimate of the solution. Each of the six solvers was run until achieving a relative residual tolerance of 10^{-6} or 1000 iterations, whichever came first. Figure 2.3 shows the performance profiles for $\tau = 10^{-5}$. Since time is not a factor in this experiment, the setup time for all the preconditioners is ignored and the iteration time (time taken per iteration) for each preconditioner is assumed to be the same. Notice that the cluster-jacobi and cluster-tridiagonal preconditioners perform extremely well, with cluster-tridiagonal the better of the two – in fact it is the perfect solver (it solved a 100% for $\alpha = 1$, i.e., it was the fastest solver for all the problems). The next best solver implicit-ssor requires up to five times as many iterations to solve the same problems. The other three solvers including gsp-3 were not even able to solve all of the problems. normal-jacobi, the simplest of the six preconditioners, is the worst performer. This is by no means a comprehensive experimental suite, but it does indicate that Schur complement based methods perform better than Hessian based methods. These performance profiles are also correlated with the condition number of the corresponding preconditioned matrices as shown in the Table 2.1.

Table 2.1 shows the condition numbers for the six preconditioners cluster-tridiagonal, cluster-jacobi, implicit-jacobi, implicit-ssor, normal-jacobi, gsp-3 we considered for the problems in Experiment 1. We also include the condition number of the Schur complement S, and the Hessian H, to show how the various preconditioners improve them. The smallest condition number for each problem is indicated in bold. Observe that in every instance cluster-tridiagonal has the best condition number. Observe also that gsp-3, has a worse condition number than the Schur complement S, thus at least in this small test it seems that eliminating the points from the linear problem provides

[1]Details of problem selection are provided in the Appendix B

problem	Hessian	normal jacobi	gsp-3	S	implicit jacobi	implicit ssor	cluster jacobi	cluster tridiagonal
p-162-22824	4.58e+08	3.43e+08	8.01e+07	1.04e+07	4.47e+06	1.14e+06	1.35e+06	**1.05e+06**
p-282-37322	1.15e+09	7.40e+08	2.75e+07	3.29e+07	1.34e+06	1.27e+06	1.24e+05	**8.63e+05**
p-339-44056	1.49e+09	1.87e+09	5.63e+07	6.00e+07	3.00e+07	2.18e+06	8.67e+05	**6.71e+05**
p-384-49181	1.15e+09	1.18e+09	3.31e+07	3.39e+07	1.98e+07	8.81e+05	3.82e+05	**3.31e+05**
p-412-52215	1.20e+09	1.00e+09	8.12e+07	3.36e+07	1.53e+07	2.04e+06	1.24e+06	**5.07e+05**
p-52-64053	5.73e+09	8.37e+09	2.69e+09	1.99e+07	7.44e+06	8.08e+06	3.54e+06	**1.00e+00**
p-89-110973	7.40e+09	1.08e+10	2.69e+09	2.55e+07	9.89e+06	9.08e+06	4.66e+06	**1.67e+06**
p-245-198739	8.90e+09	1.23e+10	2.72e+09	4.32e+07	1.46e+07	1.15e+07	5.94e+06	**2.55e+06**
p-427-310384	1.09e+10	1.91e+10	2.37e+09	5.29e+07	2.10e+07	1.68e+07	7.00e+06	**2.62e+06**

Table 2.1: Condition Numbers for Preconditioners.

as much benefit or more than constructing a low stretch spanning tree and using it to precondition H. Like Agarwal et al. [5], the Hessian matrix ($J^\top J$) has been scaled by the inverse of its diagonal (the scalar Jacobi preconditioner).

A more strenuous and fair test of linear solver performance is one that accounts for the total time (rather than iterations) taken to solve a problem. Thus, in our second experiment we took 26 problems each from the ladybug and venice datasets with more than 400 cameras each and ran the Schur-based preconditioners (as the Hessian based preconditioners are clearly worse), cluster-tridiagonal, cluster-jacobi, implicit-jacobi and implicit-ssor on them. Each solver was allowed a time budget of 300 seconds with a convergence tolerance of 10^{-6}. We account for the time needed to compute the entries of the Schur complement for the cluster-tridiagonal, cluster-jacobi and implicit-ssor as well as the time needed to factor the resulting matrix. The time for computing the clustering and the tridiagonal permutation is only spent once per bundle adjustment problem and amortized across all linear solves and therefore not accounted for. Figure 2.4 shows the profile curves for each of these two datasets for $\tau = 10^{-2}$, $\tau = 10^{-3}$ and $\tau = 10^{-5}$. Note that the cluster-jacobi and cluster-tridiagonal solvers dominate all the other solvers, especially for $\tau = 10^{-3}$, $\tau = 10^{-5}$. Only for the extremely loose tolerance of $\tau = 10^{-2}$ is the performance of the other

solvers even comparable, where the setup time of the preconditioner is a factor. As we decrease τ, cluster-tridiagonal beats all other solvers, especially for the venice problems, where we expect the clustering structure and some inter-clustering interaction to be present. The extra mass in cluster-tridiagonal means the time spent computing it is well worth it. It is also worth noting, that the ladybug is not a community photo collection dataset, rather it was collected by mounting cameras on a moving vehicle, and thus isn't likely to have significant cluster structure. This would explain the similar performance of cluster-jacobi and cluster-tridiagonal on this dataset.

In Figures 2.5 and 2.6 we show the detailed convergence behavior of the four preconditioners on the first $nine$ problems in the ladybug and venice data sets (the remaining problems are shown in Appendix B). For each preconditioner, we plot the log relative residual $log(\frac{||Ax_k - b||}{||Ax_0 - b||})$ as a function of time. The time needed to compute the preconditioner is also accounted for. Note that CG does not reduce the residual monotonically, thus the relative residual plots are oscillatory. As can be seen from the plots for most of the problems, cluster-tridiagonal performs the best (seen as the relative residual error falling the fastest), followed by cluster-jacobi, implicit-ssor and implicit-jacobi in that order.

2.4.3 Bundle Adjustment

We now look at the performance for these iterative solvers and explicit-sparse on the bundle adjustment problems. All of the iterative solver based bundle adjustment algorithms were run inside an inexact Levenberg-Marquardt (LM) loop [106], with the forcing sequence set to a constant $\eta_k = 0.1$ and the termination rule suggested by Nash & Sofer [73]. We use the same set of bundle adjustment problems as the previous experiment. As the ladybug problems are smaller they were run with up till a maximum time of 600 seconds; while the venice problems are denser and were run up to a maximum time of 1200 seconds. No other convergence tolerances were used. The solver time now accounts for time spent in computing the visibility, clustering and *degree-2* tree structures. Figure 2.7 show the performance profiles for $\tau = 10^{-1}, 10^{-2}$ and 10^{-3}.

As observed in [5], explicit-sparse is dominated by the iterative solvers. Only for $\tau = 10^{-3}$ in the venice dataset is explicit-sparse doing better than implicit-jacobi, the simplest of the four preconditioners. Also worth noting is that the visibility based preconditioners find better solutions

faster for the vast majority of the problems. For $\tau = 10^{-1}$, the high setup cost of the more sophisticated preconditioner dominates and the simpler preconditioners are able to get to the solution faster. However once the tolerance is tightened, both implicit-ssor and implicit-jacobi are unable to compete with the visibility based preconditioners either in solution quality or in time. For the same percentage of problems solved for $\tau = 10^{-2}$, the visibility based preconditioners are up to two times faster than the next best solver, and for $\tau = 10^{-3}$ this gap widens to up to 5 times.

Perhaps the most surprising fact is the similar performance of cluster-jacobi and cluster-tridiagonal and that the former at times, performs a bit better. We believe the reason for this is the rather large constant value of the forcing sequence $\eta_k = 0.1$, which only rarely requires the full power of the more sophisticated preconditioner to achieve that convergence threshold. Thus the extra time needed to construct the cluster-tridiagonal preconditioner is not always worth it. Having said that, even with their higher setup times and complexity, both visibility based preconditioners are a clear win over the existing state of the art and we recommend their use.

Figures 2.8 and 2.9 show the detailed convergence behavior of Levenberg-Marquardt as a function of the linear solvers used for the first $nine$ problems of the ladybug and venice data sets (the remaining can be found in the Appendix C). Again, we plot the log relative error for each problem,

$$e_{k,s} = \log \frac{f_{k,s} - f^*}{f_0 - f^*}, \tag{2.13}$$

where, f_0 is the initial error, f^* is the lowest error across all solvers, and $f_{k,s}$ is the error for solver s at iteration k. The log relative error $e_{k,s}$ is plotted against time. As can be seen, for most of the plots, cluster-tridiagonal and cluster-jacobi compete for the best preconditioner (seen as the relative error falling the fastest).

2.5 Conclusion

In this chapter, I presented bundle adjustment as one of the key steps in the reconstruction pipeline for Internet photo collections. I presented *Visibility Based Preconditioning*, a new technique for preconditioning the linear least squares problems arising in large scale bundle adjustment problems. Using the visibility information in the scene, we cluster the cameras into tightly interacting clusters. These clusters form the basis of our block diagonal and block tridiagonal preconditioners. When combined with an inexact step LM algorithm, these preconditioners offer equal or better solution

quality compared to the best available methods at 3-5x less execution time on problems from the BAL dataset.

The theory of preconditioning is still in its infancy. For certain classes of matrices, e.g., symmetric diagonally dominant(SDD) matrices and matrices arising from finite element methods, Support Graph Theory [10, 10] provides bounds on the condition number, but for general positive semi definite (PSD) matrices very few techniques exist. Even for well known preconditioners like incomplete cholesky the theory only deals with existence and breakdown of the preconditioner and not its performance. Thus, a theoretical analysis of visibility based preconditioning remains an interesting open problem.

Our experiments are limited to the BAL dataset. The sparsity patterns present in the BAL dataset are only a subset of the sparsity patterns encountered in real world SfM problems. A notable exception for example is the presence of camera blocks with long range interactions, e.g., aerial views of a scene that correspond to near dense rows in the Schur Complement S. Visibility based clustering is not the optimal approach here, and better approximations can be obtained by isolating such views and dealing with them separately.

And finally, while preconditioners excel at solving linear least squares problems to high precision, the extra work of setting up the block tridiagonal preconditioner is at times not worth the gain, when used with an inexact step LM algorithm for loose tolerances. In future work, better forcing sequences (η_k) can be explored along with ways of reducing the setup time of the block tridiagonal preconditioner.

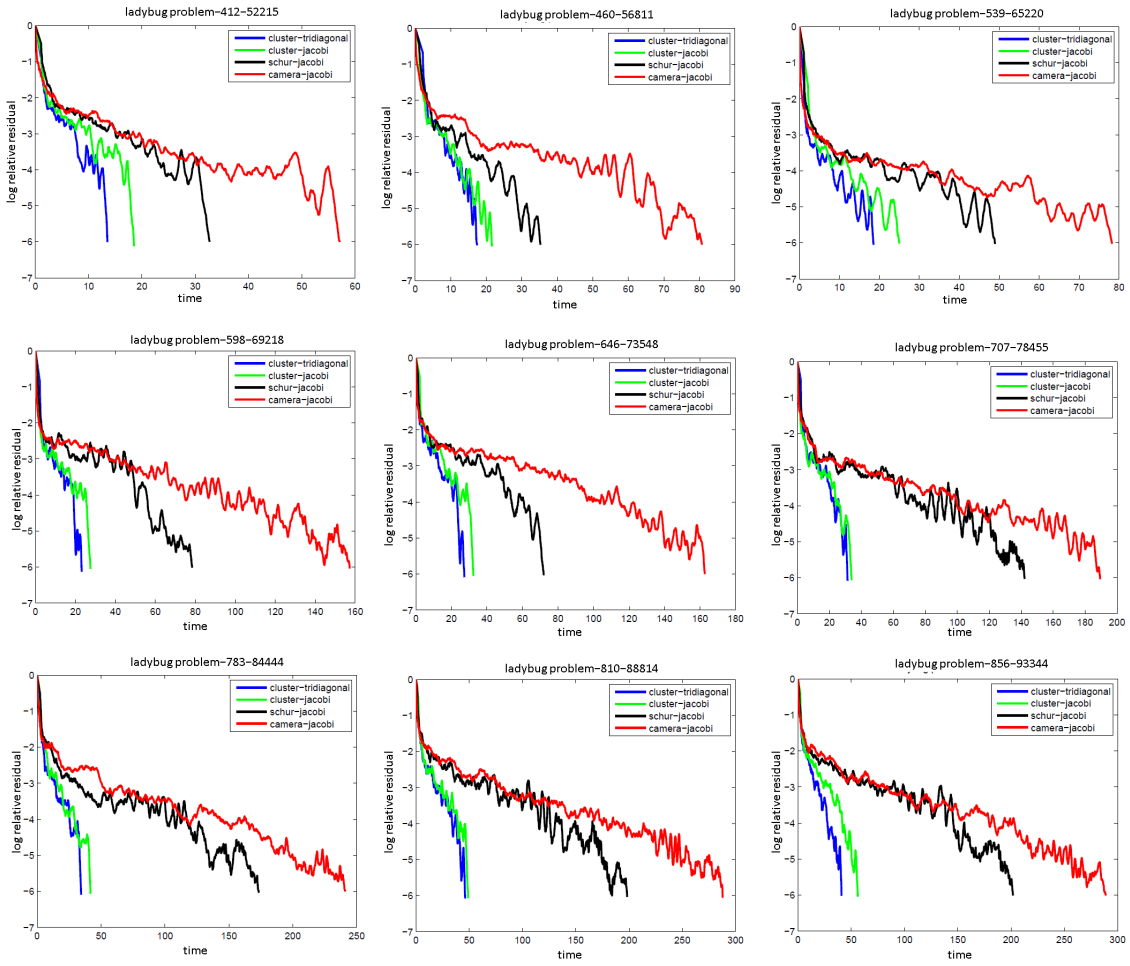


Figure 2.5: Convergence Plots for Linear Problems 1-9 of the ladybug data set, plotting the relative residual error as a function of time.

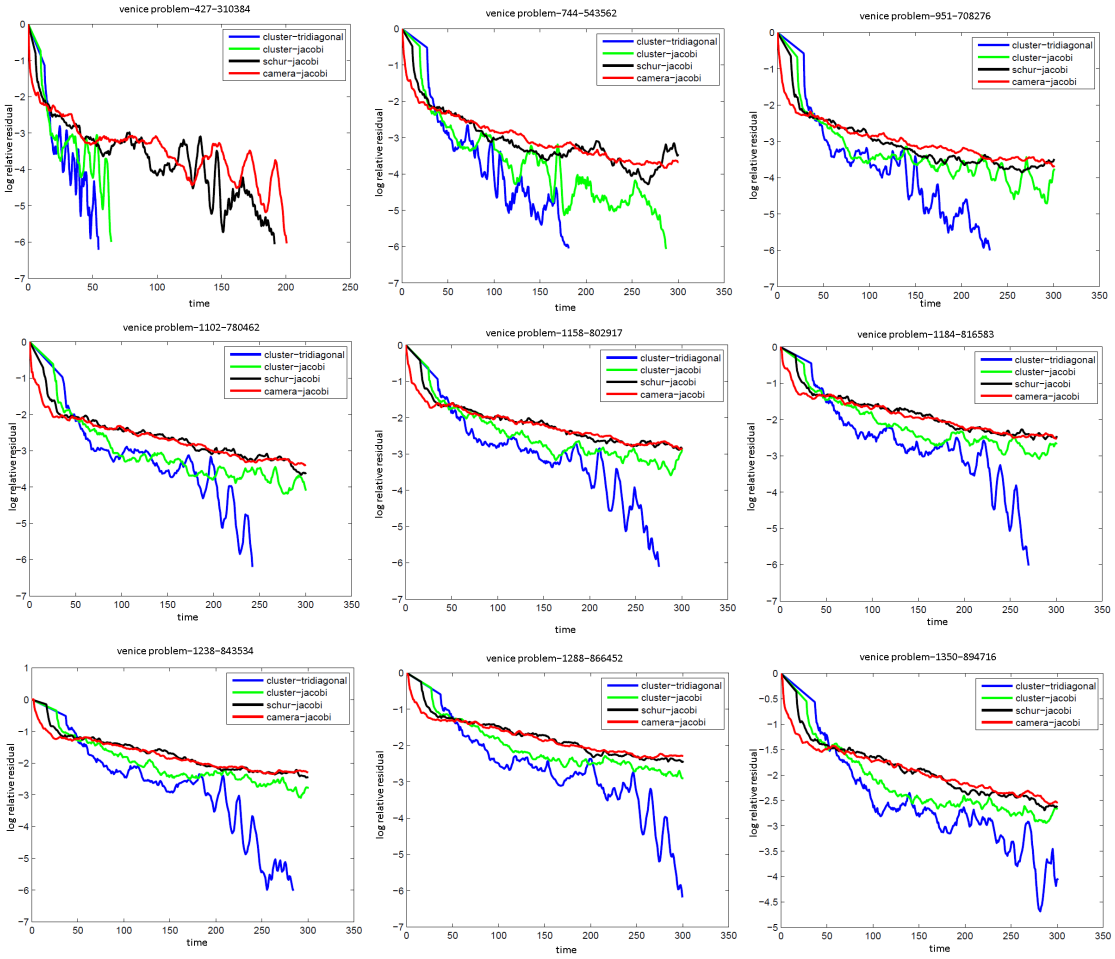


Figure 2.6: Convergence Plots for Linear Problems 1-9 of the venice data set, plotting the relative residual error as a function of time.

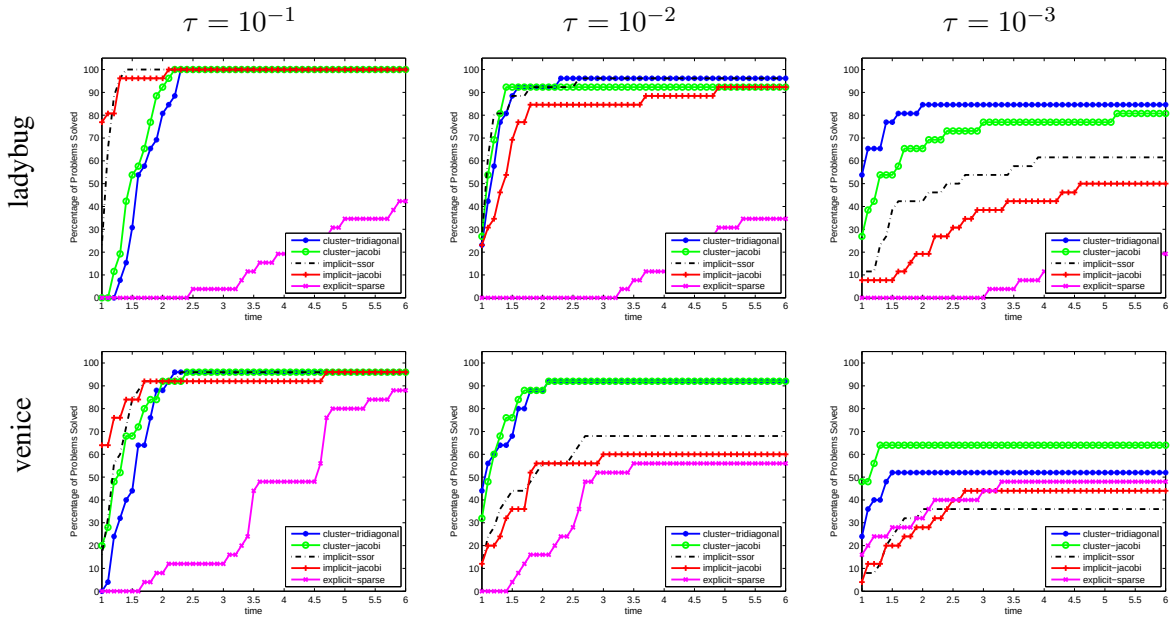


Figure 2.7: The Performance Profiles for the full bundle adjustment problems. Performance measures the percentage of problems solved as a function of time.

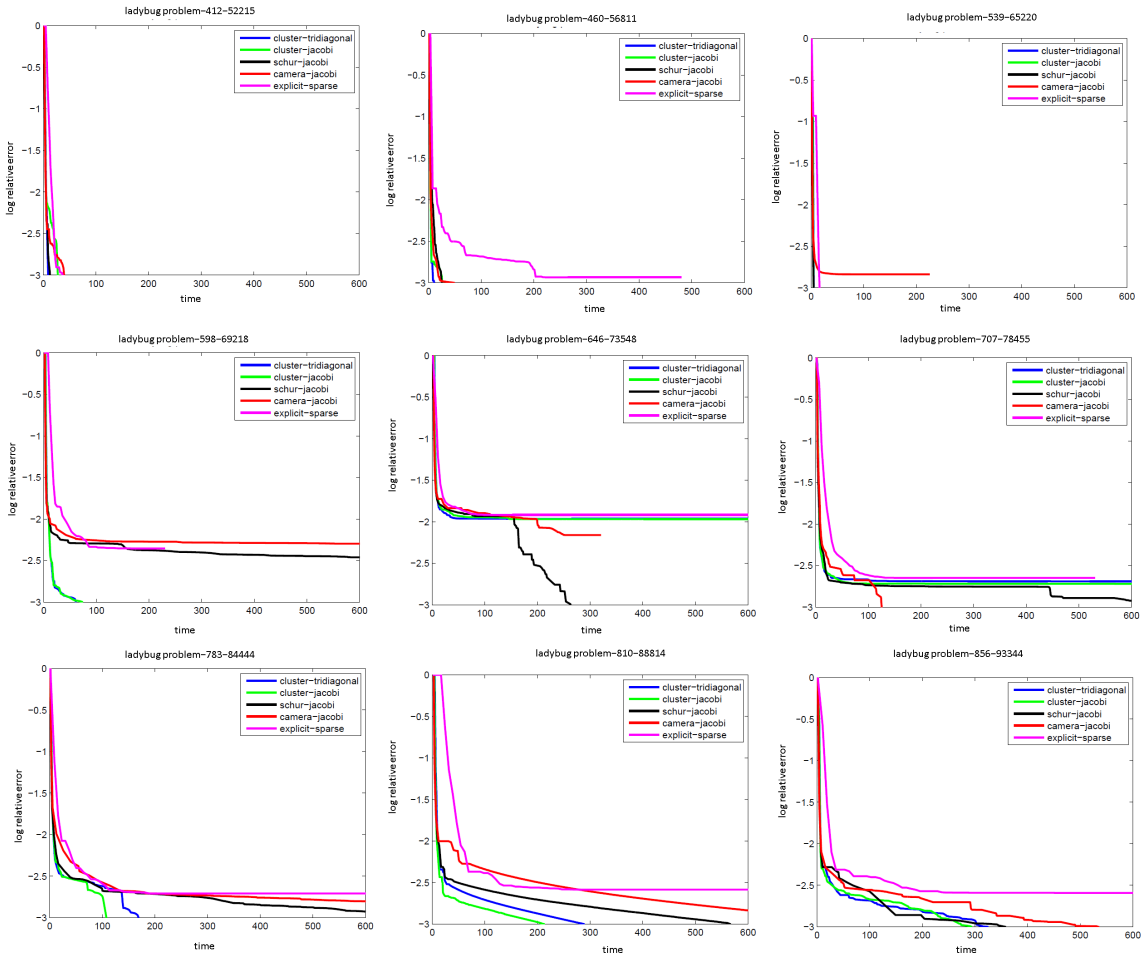


Figure 2.8: Convergence Plots for Bundle Adjustment Problems 1-9 of the ladybug data set, plotting the relative residual error as a function of time.

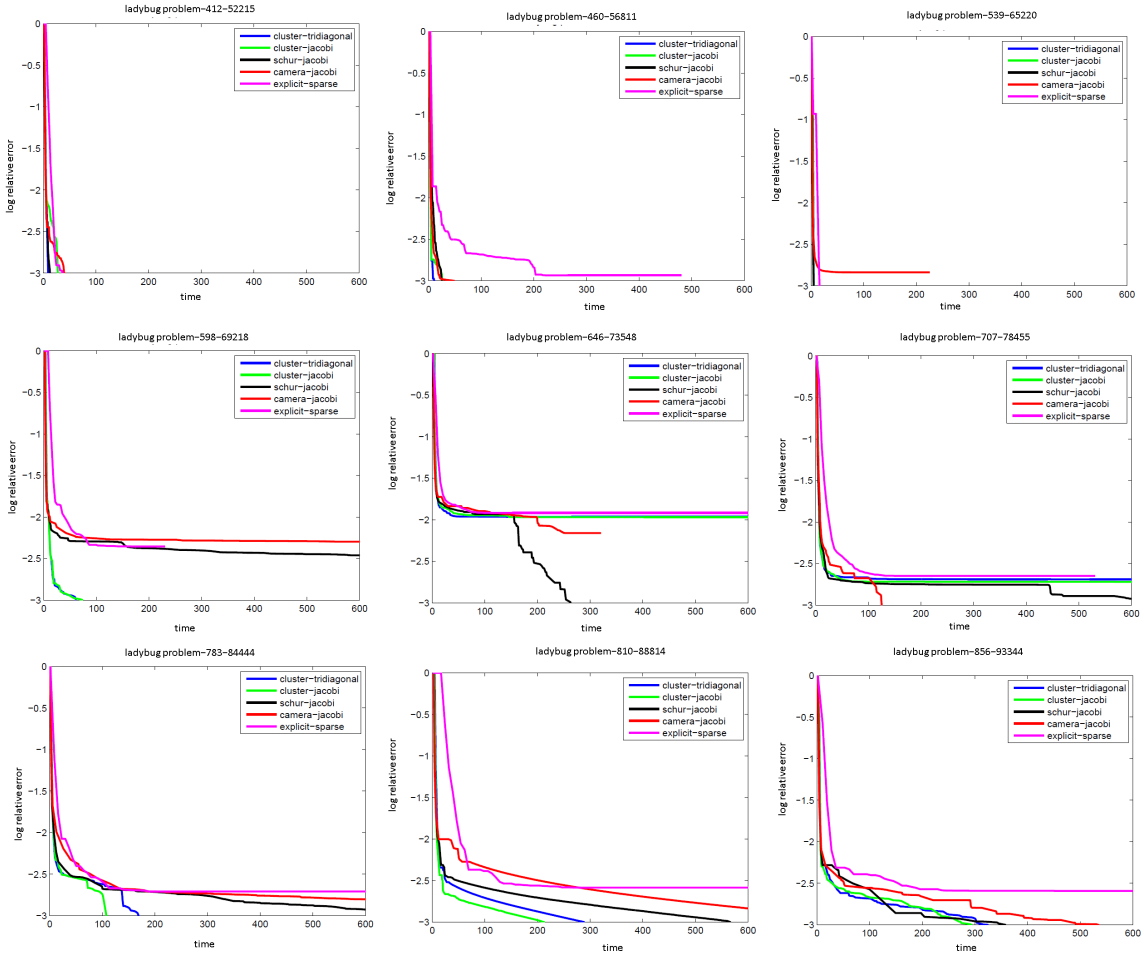


Figure 2.9: Convergence Plots for Bundle Adjustment Problems 1-9 of the venice data set, plotting the relative residual error as a function of time.

Chapter 3

QUALITY OF THE RECONSTRUCTIONS : MONOCULAR CUES FOR STEREO

Another challenge in creating effective visualizations is in computing high quality reconstructions — poor 3D reconstructions lead to hallucination or ghosting artifacts when the scene is visualized from a novel view point. A variety of cues have been investigated to determine the problem of computing the 3D structure from 2D image(s) under the title of $Shape - from - X$. The most widely applicable amongst these is $Stereo$, and has been studied across different domains such as robotics, medical imaging, artificial intelligence and machine learning.

Yet, despite decades of research on stereo algorithms [11, 27, 84, 85, 97, 110], state-of-the art methods fail on poorly textured, piece-wise planar scenes such as houses and room interiors [32]. Ironically, such scenes are composed of the simplest possible shapes: large planes, axis-aligned boxes, etc. Why should our stereo algorithms, which can reconstruct complex objects up to sub-millimeter precision [85], fail on the simplest possible scenes? These planar scenes, shown in Figure 3.1, occur in abundance in architectural scenes, both indoor and outdoor. Figure 3.1(a) shows one of the input images part of a stereo pair, and Figure 3.1(c) shows the result of using the DoubleBP algorithm [111], which is one of the top stereo algorithms on the Middlebury dataset [1], the most popular stereo benchmark. Note how the layout of the scene, e.g., the door and floor in the drawers data set, or the closet and walls in the the closet data set, is not well reconstructed using the stereo algorithm.

Arguably, large planar regions are better captured via monocular rather than binocular cues. Indeed, significant progress has been made in recent years in modeling architectural structures using single-view techniques [23, 25, 46, 62, 83]. A limitation of these methods, however, is that only a coarse approximation of the scene is reconstructible; owing to the strong assumptions needed to make the problem tractable. Figure 3.1(b) shows the orientation map computed using [62]. Thus while the overall layout of the scene is captured well in the orientation maps, the details of the teddy bear are completely missing.

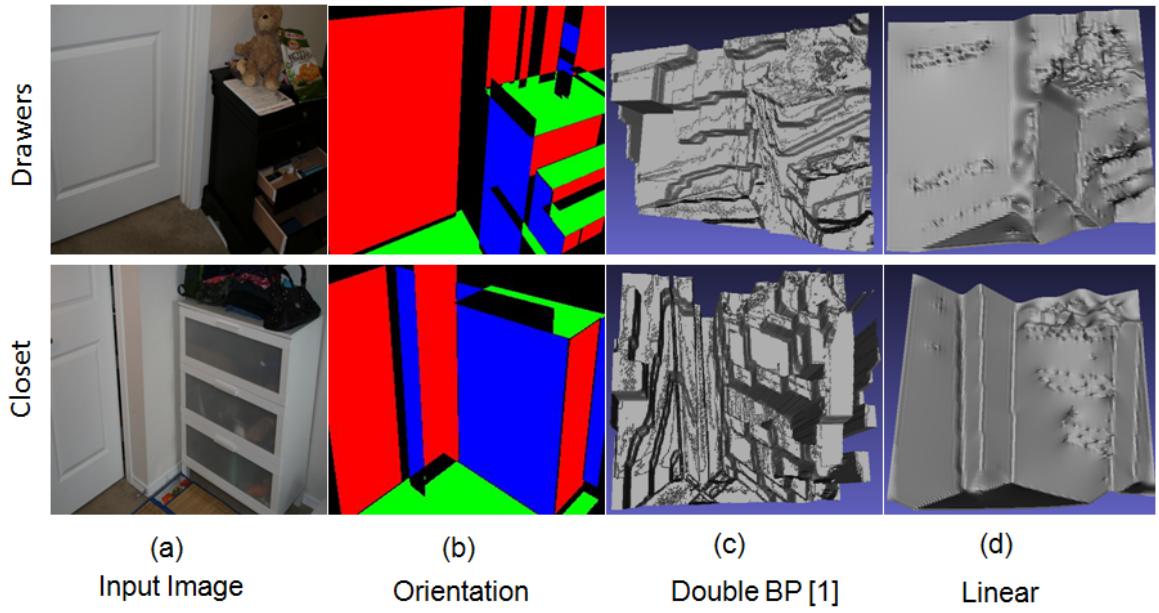


Figure 3.1: The figure shows for two indoor scenes (a) one input image part of a stereo pair, (b) the orientation map computed using the monocular cues (the colors R, G, B correspond to the $three$ primary orthogonal directions), (c) the reconstructed range map using DoubleBP [111] and (d) the reconstructed range map by integrating monocular cues with stereo using our linear relaxation.

We seek a best of both worlds solution: we want the metric, highly accurate results we've come to expect from stereo algorithms, together with the ability to accurately represent large planar, textureless surfaces. Humans have no trouble interpreting these scenes, as we combine both monocular and binocular cues to estimate depth. In this chapter, I present a solution combining monocular and binocular cues, to produce reconstructions shown in Figure 3.1(d), where both the layout and the details are reconstructed accurately.

Our approach is the first algorithm that addresses at the problem binocular stereo of man-made environments which is a very important application domain, e.g., for robots in urban spaces. While there has been prior work in modeling Manhattan Scenes [32, 34] and piecewise planar scenes [89], these approaches require several images as input, from a range of different viewpoints. Binocular stereo is more challenging due to the much more limited input (weaker data term), yet is an important special case due to the prevalence of binocular rigs. We show that our approach of combining

orientation information can improve the quality of multi-view stereo results too in Section 3.4. Further our approach has the following additional advantages:

- Linearity: we introduce a simple relaxation that leads to a closed form, linear stereo solution with results that rival or outperform nonlinear formulations.

- Continuity: while the vast majority of stereo algorithms require discretizing the solution space, our formulation is continuous; especially important for modeling non-frontal-parallel surfaces in architectural scenes without severe aliasing and metrication errors.

Our approach computes a range map from an image pair that best fits both a photo consistency term from stereo, as well as an orientation term derived from the monocular line sweeping approach of Lee et al. [62]. We show that parallax and orientation cues are best combined using a *range-based* rather than disparity-based formulation. Our linear, continuous formulation enables fast and globally optimal solutions on challenging data sets. Our results provide the ability to accurately reconstruct low-textured elements such as walls, floors, and beds, while also providing fine scale geometry for highly textured objects such as pillows, curtains, baskets, etc.

3.1 Related Work

Stereo Matching has been one of the most active research areas in computer vision. Scharstein et al. [84] provide a taxonomy of dense stereo correspondence algorithms. The algorithms are classified along various attributes such as the matching cost (e.g., SSD, SAD or NCC), cost aggregation (e.g., 2D square or gaussian windows of fixed or adaptive sizes [52]), optimization methods (e.g., local "Winner takes all" [21] or global optimizations [11, 97]), and disparity refinement. They show that while global methods outperform the local approaches, optimizing them using Graph Cuts [11], or Belief Propagation [97] can be slow for large images or a large label space. Recently, edge aware filtering [27] and Patch Match stereo [9] have been used for speeding up these formulations. Second-order smoothness priors have also been incorporated in these global formulations, by showing that inference with triple cliques can also be effectively optimized [105]. While cost aggregation is typically performed locally, in recent work Yang et al. [110] compute the matching costs adaptively on

a suitably defined minimum spanning tree to generate a more natural image pixel similarity measure. Typically stereo approaches work in a discrete setting, however, recent work [79] has looked at continuous formulations using variational models, to come up with convex relaxations for the problem.

Despite this progress, all stereo algorithms rely on the presence of texture in the scenes to identify corresponding regions in the images. While texture is often easy to find in photographs of monuments leading to successful reconstructions [33, 85, 92], indoor scenes have walls, ceilings and other surfaces that are often painted for a uniform appearance. In addition, these scenes often contain transparent, translucent or specular surfaces (e.g., polished table tops, washing machines, windows, lifts etc.), which although they have received attention in literature [65], still pose challenges for stereo. Thus these techniques do not generate reliable stereo cues on such surfaces which leads to poor reconstructions, as can be seen in the reconstruction in Figure 3.1(c), where incorrect geometry is computed for the wall, the floor and the polished drawers.

Monocular cues have the potential to overcome these challenges and have been studied in detail [8, 25, 43, 46, 47, 62, 70, 83, 103, 114]. One of the earliest attempts at the single view problem was undertaken by studying the variation of brightness, or *shading* in an image [12, 47], which relates the orientation of a patch with its intensity in the image. While these methods can reconstruct very fine details on the surface, they work well only under strong assumptions on the albedo of the surface and on the lighting distribution. Similarly other physics based cues such as the contours [103] or repeated textural patterns [70] in the scenes have been shown to work for a restricted class of scenes. In the last decade attempts have been made to overcome these restrictions and reconstruct photographs downloaded from the Internet, by using the vast amount of suitably labeled image data and machine learning approaches for reconstruction [45, 46, 82, 83]. While these approaches allow us to automatically reconstruct arbitrary images downloaded from the Internet, they have their limitations. Automatic Photo Pop-Up [45, 46] computes a very coarse representation of the scene as a collections of texture mapped planar bill boards (such as ground, vertical, sky). The only horizontal surfaces that can be modelled using their method are the ground and sky, not suitable for many architectural scenes, and clearly not applicable to indoor scenes. While Make3D [82, 83] looks at modeling more general scenes, their method also reports results only on outdoor scenes with partial success.

In contrast, the reconstruction of architectural scenes [23] has provided impressive results because of the natural constraints imposed on these scenes which help to overcome the ill-posed nature of the single view problem. These scenes have clusters of parallel lines which under perspective projection, intersect at vanishing points. Detecting lines and clustering them using vanishing points can help to determine their 3D direction. Further, many of these scenes exhibit a *Manhattan* structure, i.e., the lines directions correspond to *three* mutually orthogonal directions (the X, Y and Z axis). Computing the *three* dominant clusters helps to specify the layout of the scene. While some prior work [96] has focused on interactively reconstructing scenes using piecewise planar approximations, other approaches look at automatically computing the geometry from a single image [25, 62]. Our work takes inspiration from the line sweeping approach of Lee et al. [62] which showed that automatically detected line segments combined with prior knowledge about how building interiors are structured can be used to reconstruct 3D models. In this approach, detected line segments are clustered and swept towards each other to generate partial orientation maps for the image. While later extensions [60] have generalized it to include piecewise planar and curved surfaces, these methods work best on (uncluttered) manhattan world scenes, where all the planes are oriented along the $three$ mutually orthogonal directions and most of the lines (range and orientation discontinuities) are clearly discernible. Indoor scenes often have such a manhattan structure where the walls, ceiling, floor, tables and beds are placed along the manhattan axes. However there is often other significant geometry such as cushions, pillows, curtains, baskets, utensils etc. that do not exhibit this manhattan geometry and can be misclassified. This can be seen in Figure 3.1(b) where the overall structure of the scene is well captured by the orientation map, but the teddy and other objects on the table are missing or incorrectly assigned a manhattan orientation. Further, these methods rely on the detection of all discontinuity lines, which may not be visible because of occlusions or clutter present in scenes.

In our approach we use these two sources of complementary information (parallax and orientation) and solve for the range map for an image pair by casting it as an range integration problem. The orientation map is integrated as a pairwise interaction between neighboring pixels, which helps to complement stereo in textureless and structured regions and compute reconstructions that outperform those constructed using only photo consistency based methods. This can be seen in Figure 3.1(d), where both the overall layout as well as the details of the scene are well reconstructed.

This approach of combining positional and orientation information has been used in [74] where the information from a triangulation scanner and photometric stereo was combined. However, their approach is suitable for generating bump mapping on surfaces captured in a laboratory setting, and not for architectural scenes. Similarly while Make3D [83] has also investigated combining such cues, their approach is aimed at natural outdoor scenes and not suitable for texture less scenes. Plane sweeping techniques [37, 112] have used detected manhattan directions from the point cloud, to sweep planes along multiple directions to generate better matching scores, particularly for scenes with planes along grazing angles. While they show impressive results by using slanted windows to reconstruct the ground plane, they too focus on outdoor scenes which typically are more textured. We instead use the orientation information in a global optimization to complement stereo in regions where the stereo cues are truly inadequate. Finally, while the two view problem continues to be the focus of much research, multiple views (both from images [32, 35, 85] or video [53, 75]) have been shown to be extremely effective in disambiguating disparities, and getting sub-millimeter results.

3.2 Formulation

Given a pair of images I_1, I_2, along with their projection matrices P_1, P_2, our objective is to compute a range map (as a grid overlayed over the image). Thus, we compute the range z_p (the distance from the camera center to the 3D point along the ray) for each cell p in the grid.

It is well known that traditional stereo algorithms work well in regions with texture, while they perform poorly in planar textureless regions e.g., floors, ceilings, walls. On the other hand monocular cues such as those introduced in Lee et al. [62] work well in determining the overall structure of the room via an orientation map, but often miss or provide wrong orientations for (non-manhattan) objects and clutter present in these scenes. We formulate our approach to use both these complementary sources — parallax between the images as well as orientation information for each image to compute the range maps. We thus assume that we have a parallax cost function $c_{parallax}(p, z)$, which refers to the cost of assigning a range z to a cell p in the grid based on the photo consistency of the pair of images (Section 3.4.1). We also have a partial orientation map [62], i.e. we have a normal n_p for a subset of cells in the grid (Section 3.2.3).

While the vast majority of stereo algorithms require discretizing the disparity space, we seek

a formulation that is continuous. This is especially important for modeling non-frontal-parallel surfaces in architectural scenes without suffering from aliasing artifacts. Further, while most global stereo formulations (and our photo consistency cost function $c_{parallax}(p, z)$) are non linear, we seek a linear formulation that provides a closed form solution. We thus introduce a simple relaxation that apart from significantly minimizing computational time generates results that rival or outperform the nonlinear formulations.

This is achieved by minimizing a novel objective function that requires that the range map minimize the consistency between the images, and that the inferred normals in the range map be consistent with the orientation map generated by using [62], and is given by

$$\min_{z} \lambda_s c_s(z) + \lambda_m c_m(z) + c_r(z) \tag{3.1}$$

Here c_s denotes the stereo matching cost between the images, c_m denotes the monocular cost that ensures consistency between the inferred normals in the depth map and those generated using [62]. c_r denotes the regularizing cost across neighboring cells. All the costs c_s, c_m, c_z are constructed as continuous and linear functions in z, the *range* for every cell. While stereo methods usually operate in the disparity space [84], we operate in the *range* space as this is critical to the integration of the orientation information. We now describe these individual terms.

3.2.1 Linear Relaxation

The matching cost between two images (and so also our cost $c_{parallax}$) is typically a non linear function of range and is computed by measuring the photo consistency between the images at discretely sampled values $z \in [z_{\min}, z_{\max}]$. We interpolate this function between these samples to compute a continuous cost function $c_{parallax}$ (we defer the construction of the photo consistency measure used to Section 3.4.1).

Figure 3.2 shows this function $c_{parallax}$ for $three$ separate cells. For most of the cells, there is a clearly identifiable minima in the function (first two cells in Figure 3.2) which is close to the correct range at the cells, while for some of the cells there are multiple candidates for range (third cell in Figure 3.2). In computing our linear relaxation c_s for the parallax cost $c_{parallax}$, we restrict our attention to a subset of cells N_s which satisfy two conditions. Firstly, we reject cells for

which $\min_z c_{parallax}(p, z) >= T$, i.e, cells that do not have a good match in the other image for any range value. Secondly, we reject cells that have multiple minima. While it might be possible to disambiguate between these multiple minima, our experiments show that ignoring these cells does not hurt the quality of the reconstructions because of two reasons. First we found that these cells were rare in our scenes (typically less than 2% of cells). Second as opposed to pure stereo algorithms, we also use orientation as a cue, so dropping the stereo cue for these cells is often mitigated by the presence of orientation cues which can serve to compute range here.

The shape of the matching cost function $c_{parallax}$ also determines the confidence in the minima. The first cell in Figure 3.2(a) has a sharp minima whereas the second one in Figure 3.2(b) comes from a textureless region and thus $c_{parallax}$ varies smoothly across the minima. We are looking for a linear approximation of this function, that both captures the minima, as well as the confidence in the estimate. We thus construct c_s as linear approximation for $c_{parallax}$, where

$$c_s(z) = \sum_{p \in N_s} \alpha_p \cdot |z - z_p^0| \tag{3.2}$$

Here for a cell p, z_p^0 denotes the range where the matching cost function achieves the minimum, and α_p indicates the slope of the function in the neighborhood of z_p^0, as shown in Figure 3.2(a),(b) which show the matching cost functions for *two* different cells. The first cell has a sharp minima and α_p is large, while for the second cell α_p is small indicating the weaker confidence in the range estimate. The red dotted curves show the linear approximation $c_s(z)$ for the *two* cells. Thus we are able to capture not only the minima but also the shape of the function $c_{parallax}$, to a first order approximation. We do not include the third cell in N_s for which we have two peaks in the $c_{parallax}$ curve as show in Figure 3.2(c). The slope α_p is computed as the slope of the best fitting tent function that approximates $c_{parallax}$ in a local neighborhood (within $+/-10z$ values of z_p^0).

To complement the parallax cues, particularly in low-texture regions we also use the orientation information that these scenes readily provide. Thus we first construct a partial orientation map, i.e. the normal n_q for certain cells q in the image following the approach of [62] (the details are deferred to Section 3.2.3).

Given the normal at a cell q in the image grid, we use an approach similar to that used in photometric stereo [58, 78] to impose constraints on the range around the cell q. Given the projection

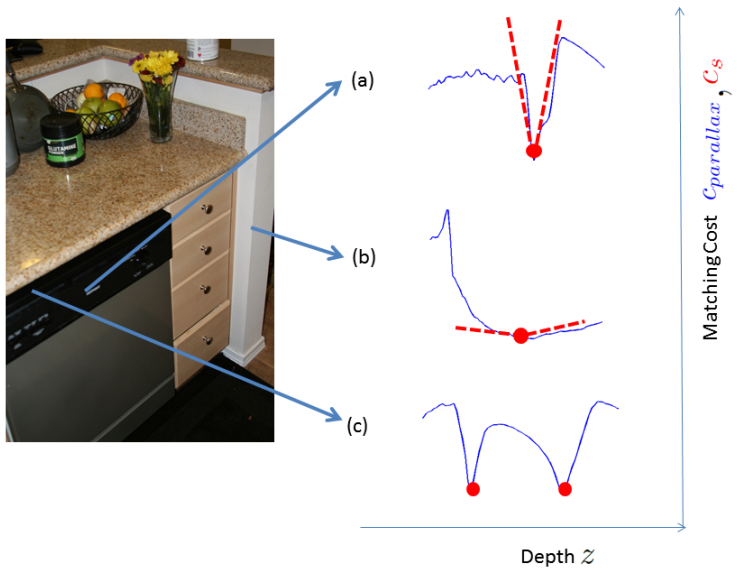


Figure 3.2: The figure shows the matching cost $c_{parallax}$ in blue and the approximation c_s in dotted red as a function of z for *three* different cells in the image. The first *two* cases have one minima in $c_{parallax}$ and are included in N_s, while the third pixel has two minima and is excluded from N_s

matrix, we know $\hat{t_q}$, the unit vector along the ray in 3D that projects onto the cell q. Then $3d_q$, the 3D point corresponding to q, has one degree of freedom given by the range of the cell z_q, and is computed as $3d_q = \hat{t_q} \cdot z_q$ (shown in Figure 3.3).

We can thus get constraints on the range z_q by requiring that the surface be locally perpendicular to the computed normal at the cell q. Let N_m be the set of pairs of pixels $\{q, r\}$ that are connected by a 4-neighborhood and for which we the normal n_q is known. Then, for each pair of pixels $q, r \in N_m$, we require the local surface direction $(3d_q - 3d_r)$, be perpendicular to the normal n_q (shown in Figure 3.3). We thus setup the monocular cost as,

$$c_m(z) = \sum_{q,r \in N_m} |(n_q) \cdot (z_q \hat{t_q} - z_r \hat{t_r})|, \tag{3.3}$$

which enforces this orthogonality constraint, wherever such orientation information is present. This integration approach dictates why it is critical to work in the *range* space as oppose to the disparity space.

While we observe in our experiments that the regions in the scenes are typically either highly textured and thereby have a reliable photo consistency score, or are planar and manhattan and thus have

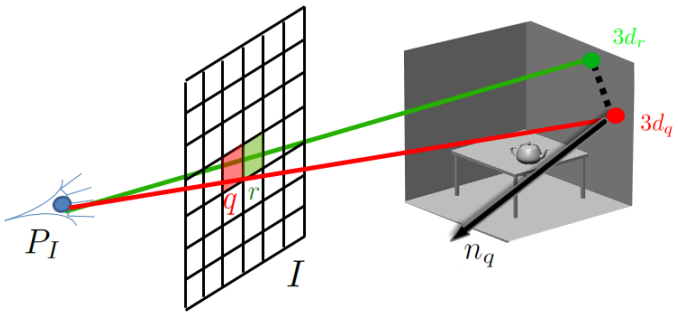


Figure 3.3: The figure shows construction of the monocular cost c_m from the orientation cues. Given two neighboring cells q and r, the range of the cells is related by Equation 3.3

reliable orientation information, there are regions where neither source of information is present. To propagate to these regions we require that the range vary in a smooth manner in these regions. We use a 4-neighborhood N_r in our grid, i.e. two cells $q, r \in N_r$, if q is the left, right, top or bottom neighbor of r. The regularization cost is set as

$$c_r(z) = \sum_{q,r \in N_r} |z_q - z_r|, \tag{3.4}$$

which penalizes the difference in the range of neighboring cells.

As all the terms c_s, c_m, c_r are linear in z, any linear solver can be used to solve Equation 3.1 to solve these constraints efficiently.

Other regularizers, e.g., second-order smoothness, total generalized variation can be used, while still maintaining this linear formulation. Dependence on the regularizer, however is lower in our approach as compared to stereo, as stereo and orientations cues, being complementary, together cover most of the image. In our experiments, we also compared $L1$ and $L2$ norms to minimize this system. As expected, while $L1$ norm gives more piecewise planar results, $L2$ norm yields more smoother results. The artifacts are very similar in both cases.

3.2.2 Photo consistency

We use a Normalized Cross Correlation (NCC) matching cost and a square window as the cost aggregation function to compute the matching between the two pixels in the two images. We use a plane sweeping approach to discretely sample ranges $z \in [z_{\min}, z_{\max}]$. Let $NCC(p, z)$ denote the

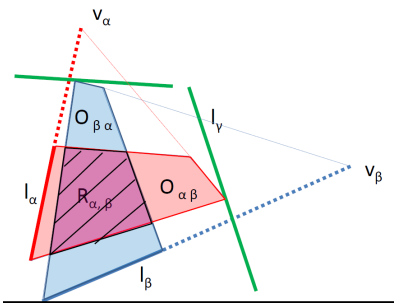


Figure 3.4: The figure shows how lines are swept towards each other to sweep out regions with known normals, and generate orientation maps

normalized cross correlation score for a cell p at a range of z. We then set the matching cost as

$$c_{parallax}(p, z) = e^{-NCC(p,z)}, \tag{3.5}$$

and interpolate the cost between these samples to get a continuous function.

3.2.3 Orientations

In this section we describe the algorithm used to compute the orientation map for an image by clustering the detected lines into the manhattan directions. While we use the approach of Lee et al [62] to generate this orientation map, other approaches to generate such manhattan orientations [25] can also be used interchangeably.

We first apply the Canny edge detector [14] to an image and then extract and cluster lines using the method described in [57]. The lines are clustered based on the vanishing points (parallel lines meet at a vanishing point in the image) — in our example we restrict ourselves to the top $three$ clusters only. Having computed the vanishing point v and all the cluster of lines that pass through it, we can compute the 3D direction d_l of these parallel lines by using

$$d_l = P^{+}v. \tag{3.6}$$

Here P^{+} refers to the Moore-Penrose pseudo inverse of the known projection matrix. To compute the orientation map associated with the image, we then use the approach of Lee et al. [62]. They introduced a line sweeping approach where a detected line l having one of the Manhattan directions

d_l is swept towards the vanishing point of another line $l^{'}$ and (Manhattan) direction $d_{l'}$ to sweep out an area which is likely to have the normal $d_l \times d_{l'}$.

We briefly describe the approach here. Consider two clusters α and β, and let $l_\alpha \in \alpha$ and $l_\beta \in \beta$ be any two lines in these clusters as shown in Figure 3.4. We sweep l_α towards v_β, the vanishing point for β cluster, to sweep out a red region $O_{\alpha\beta}$ until we intersect a line from another cluster, l_γ. Similarly $O_{\beta\alpha}$ is computed (blue trapezoid). Then

$$R_{\alpha,\beta} = O_{\alpha\beta} \cap O_{\beta\alpha} \tag{3.7}$$

is the intersection of these regions (shaded in the figure), and the orientation for $R_{\alpha,\beta}$ is set to $d_{l_\alpha} \times d_{l_\beta}$. If any two such regions with different normals ever intersect at any pixel (i.e., there is a conflict), then the orientation information is removed. The result of the sweeps is shown in Figure 3.5(b), where the $three$ colors (R,G,B) correspond to the $three$ mutually orthogonal normal directions. The regions in black are regions which are either not included within any sweep, or where there is a conflict.

3.3 Results

In this section we show the reconstructions obtained by minimizing Equation 3.1. Figure 3.5(a) shows one of the two input images part of a stereo pair, and Figure 3.5(b) shows the orientation maps generated using [62] for the input image. Figure 3.5(f) shows the range maps reconstructed using the linear relaxation c_s and the orientation cost c_m. We compare our results with those obtained using only the non linear matching cost $c_{parallax}$ in Figure 3.5(c). This energy function is solved using graph cuts [11], which provides guarantees on its convergence.

Comparing the stereo results with the linear result using both the stereo and orientation costs, we can see how the textureless parts of the image which determine the layout of the scene are poorly reconstructed in the stereo result, but are much better modeled once we include the orientation information. This can be seen clearly in the drawers data set, in the shape of the door and walls in closet data set, in the correction of the bulge at the bottom left in the kitchen data set. Similarly the walls and ceilings are recovered for the livingroom, corridor and lab data sets which were otherwise incorrectly reconstructed or missing in Figure 3.5(c). Figure 3.6 shows more results. These results

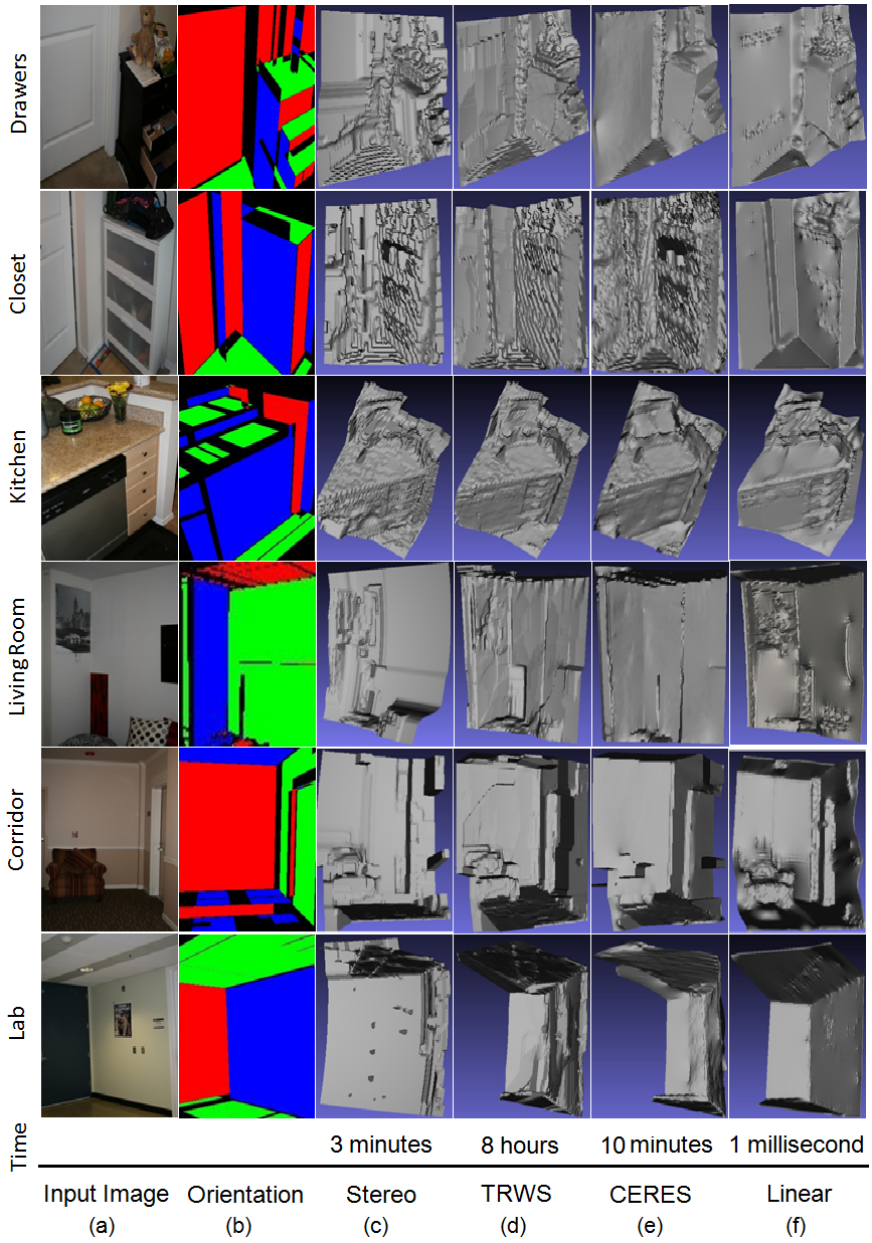


Figure 3.5: The figure (a) shows one of the two input images (b) the orientation map created by sweeping the lines towards each other, (c) the range map constructed with just the matching cost $c_{parallax}$. The range map constructed by including the orientation cost c_m are shown in (d) optimized using TRWS [56], (e) optimized using CERES [4] and (f) optimized using the linear relaxation in Equation 3.1.

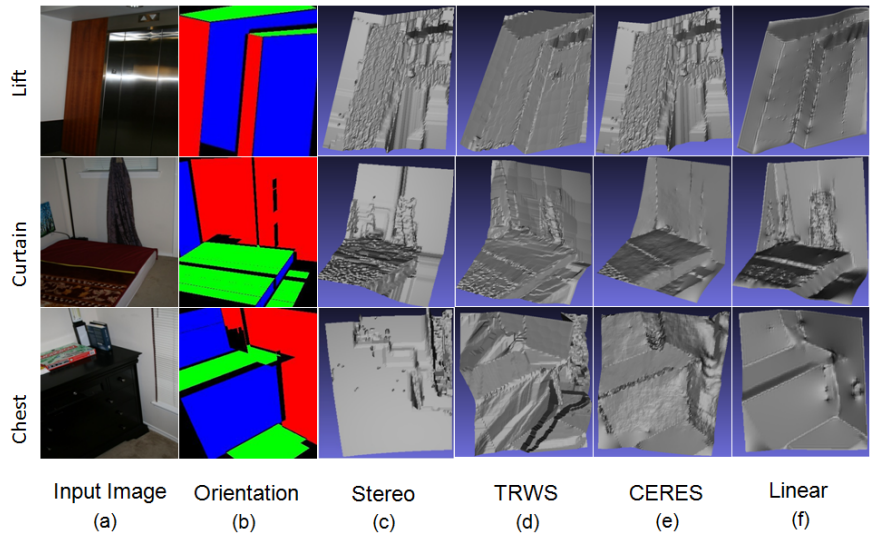


Figure 3.6: The figure (a) shows one of the two input images (b) the orientation map created by sweeping the lines towards each other, (c) the range map constructed with just the matching cost $c_{parallax}$. The range map constructed by including the orientation cost c_m are shown in (d) optimized using TRWS [56], (e) optimized using CERES [4] and (f) optimized using the linear relaxation in Equation 3.1.

show that including the orientation information helps stereo in the these traditionally challenging regions of the scene, while still preserving the details in well-textured regions.

3.3.1 Linear vs Non Linear

As discussed in Section, the linear relaxation is an approximation, as certain multi model cells (roughly 2%) are dropped and the cost $c_{parallax}$ is approximated as a linear function elsewhere. As an alternative, we can instead optimize the raw matching cost $c_{parralax}$ instead of the relaxed cost c_s in Equation 3.1, using a more sophisticated nonlinear optimizer.

One popular approach is to consider this as a discrete optimization problem, and use approximate algorithms like Graph Cuts [11], or Belief Propagation [97] to compute suitable local maxima for the energy function, with some guarantees on the convergence. This energy function can also be put in the requisite form with the unary term set to $c_{parallax}$, and the pairwise term set to $c_m + c_r$. However, a closer examination of the pairwise term shows that it is not sub modular owing to the form of c_m, and thereby the convergence guarantees of graph cuts are lost. Empirically too, we see

that the depth maps generated by graph cuts are of poor quality. We instead use a variant of loopy belief propagation, called Sequential Tree Reweighted Belief Propagation TRWS [56, 99] as the discrete optimizer, which works the best amongst the possible choices in [98]. We show the results of minimizing the non linear formulation in Figure 3.5(d) using TRWS.

A second approach to minimize the non linear function is to treat it as a continuous optimization problem and use a generic non linear solver to minimize this. We use Ceres [4] to optimize the energy function. This is highly sensitive to the initialization, and we use the stereo solution (in Figure 3.5(c)) to initialize this. These results obtained are shown in Figure 3.5(e).

Comparing our linear approach to the nonlinear TRWS and CERES alternatives, the results are of comparable quality, with the linear solution producing more consistently correct results and at a fraction of the runtime (described later). This can be seen in the case of the teddy bear in the drawer data set, the cushions in the living room data set and the couch in the corridor data set. Further, in certain cases, CERES gets stuck close to the stereo solution to which it is initialized as can be seen in the closet or corridor data set.

3.3.2 Computational Time

In terms of computation speed the linear relaxation provides significant gain. While the linear relaxation takes less than a millisecond to compute these reconstructions, the stereo result takes 3 minutes using Graph cuts. The non linear formulation (including the monocular cost c_m) on average takes 10 minutes using CERES, and about 8 hours to converge using TRWS (as shown in Figure 3.5. As mentioned before c_m makes the binary term non sub-modular, and thus we cannot use graph cuts to optimize this non linear function. For all the approaches we compute a 100 X 100 depth map and use 800 range samples uniformly sampled between $z_{\min}$ and $z_{\max}$ to sample the parallax cost. These timings include only the time taken within the optimizations. In addition all the algorithms require computing the photo consistency cost (on average 8 minutes) and detecting lines and line sweeping to compute orientations (on average 3 minutes, though this depends on the number of lines detected).

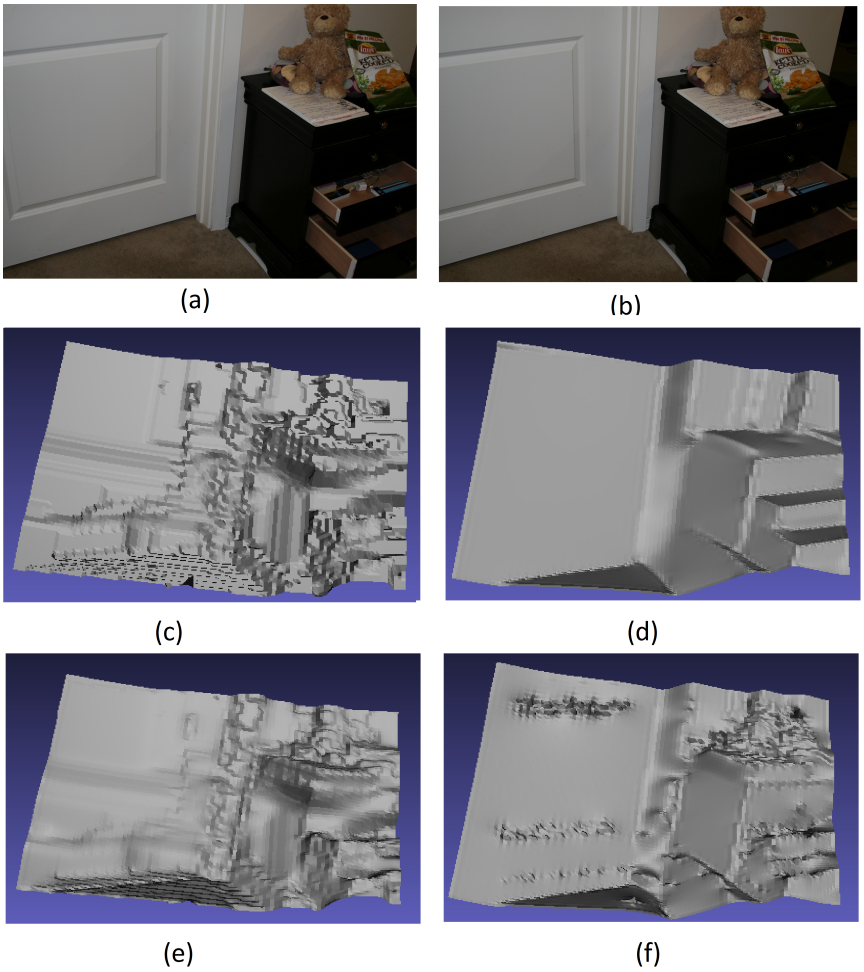


Figure 3.7: The figure (a),(b) shows the *two* input images of the stereo pair (c) the depth map using just the stereo information, (d) the depth map using just the orientation information, (e) the depth computed using the approach of Nehab et al. [74] to combine normals and depth maps, (f) the result using our linear approach to combine the normals and parallax cues jointly.

3.3.3 Comparisons

Nehab et al. [74] propose an approach of generating depth maps from laser scans and from normals obtained using photometric stereo separately, and then linearly combining them. However, the problems domains, are quite different. While they work on controlled scenes in the lab, our settings are more general. They rely on photometric stereo to provide the high frequency details that stereo lacks, whereas our orientation information fills in the low frequency details missing in stereo.

The reconstructions obtained using [74] on our indoor data sets had a lot of artifacts. In contrast to the laser scans used in [74], state-of-the-art binocular stereo depthmaps produce major artifacts for low-textured architectural scenes that a linear merge with normal maps cant fix. Our approach optimizes the range map to jointly conform to the NCC curves and orientation cues, thereby avoids poor depth estimates. Furthermore, their approach doesnt work under perspective projection (necessary for binocular stereo), whereas ours does—the key is our innovative formulation in range space instead of depth.

We show a comparison between the approaches in Figure 3.7. Figure 3.7(c), (d) show the reconstructions using the cues — binocular and monocular separately. Note that the monocular cues cannot reconstruct the finer details, whereas the stereo cues produce major artifacts in textureless regions. Figure 3.7(e) shows the result of combining them using the approach of [74] (this is the best result obtained using their method for different weights to both binocular and monocular cues), which continues to produce similar artifacts, as the normal maps cannot fix these depth estimates. In contrast, our linear approach shown in Figure 3.7(f) achieves high quality reconstructions, by jointly optimizing for parallax and orientation.

3.4 Extension to Multi-View Stereo

While our approach is the first algorithm to address the problem binocular stereo of man-made environments, the multi-view stereo problem for urban scenes has received some attention [32, 34, 89]. These methods rely on many views taken from different viewpoints, along with strong priors on the geometry to compensate for the lack of texture in these scenes. In this section, I demonstrate how our approach of combining orientation information with stereo can be used extended to the multi-view case. While prior works [32, 34, 89] require a manhattan or piecewise planar scenes, our approach can handle a mix of free form and piecewise planar geometry.

Consider a collection of images $\Gamma = \{I_1, I_2 ... I_m\}$. The objective as before is to compute a depth map over an (NXN) grid for each image $I \in \Gamma$. Thus, for each cell p in the grid, we wish to compute a depth z_p. While a natural extension of our previous approach is to compute a cost function $c^{mvs}_{parralax}$, set to the sum of the parallax functions for each pair of images in Γ, state-of-the-art multi view approaches use many sophisticated heuristics to build a visibility model (to handle occlusions),

priors [85]. Our experiments showed that the 3D points generated by state-of-the-art approaches like PMVS [35] serve as better depth candidates than those obtained by summing the parallax costs, for combining with orientation costs. This might seem counter-intuitive, especially in textureless regions, where PMVS [35] fails to recover any 3D points. In contrast, the sum of parallax functions, while cannot isolate the correct depth, can indeed provide a range of valid depths (matching white wall patch with other white patches), which includes the correct depth. However, this is a very weak cue, and as these regions are often well recovered using the orientation information, this cue is often not important in these regions. On the other hand, for very detailed textured regions, PMVS and other MVS algorithms have been finely tuned to give accurate depth estimates, which outperform the sum of parallax functions. Of course, this comes at the expense of the linear formulation.

Thus, we decide to use an off-the-shelf, state-of-the-art algorithm [35] to compute the point cloud. However as we have discuussed, for challenging indoor scenes these approaches are unable to capture the structure of the room, because of the low textured nature of walls, floors, ceilings. As before, line sweeping cues such as those introduced in Lee et al. [62] work well in determining the overall structure of the room but provide incorrect or missing orientations for (non-manhattan) objects as well as the clutter present in these scenes. Thus we combine these two cues to obtain high quality reconstructions by minimizing the objective function

$$\min_{z} \lambda_{mvs} c_{mvs}(z) + \lambda_m c_m(z) + c_r(z) \tag{3.8}$$

for the multi view stereo case which outperform state-of-the-art for urban scenes [32]. Here, c_{MVS} is the cost function obtained using the 3D point cloud generated using [35], and c_m and c_r are the orientation cost and regularizer as before. We now describe the construction of the cost function c_{mvs}

3.4.1 MVS Cost

I use the SfM + MVS pipeline in [108] to generate the 3D point cloud X, associated with the image set Γ and the projection matrices $P_I, \forall I \in \Gamma$. Further, the MVS algorithm used in the case (PMVS [36]) also returns a list of points X_I visible in each image I.

Given the set of 3D points visible in the image X_I and its projection matrix P_I, we can project each point $x \in X_I$ to some cell p on the depth grid. Let the collection of non empty cells (cells onto

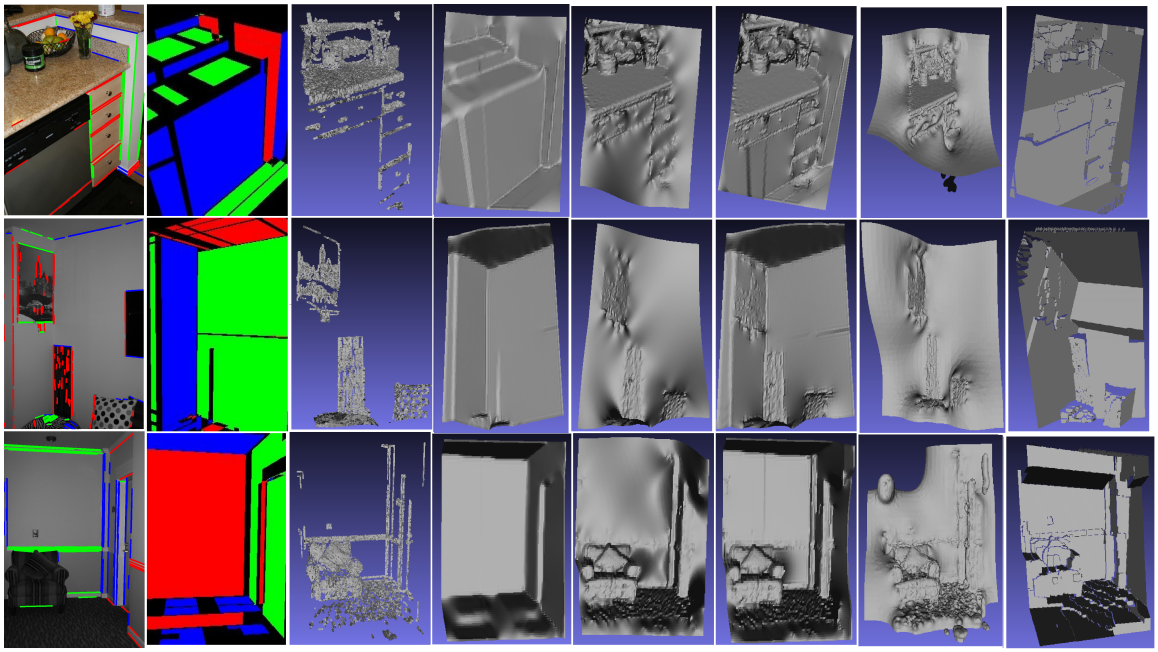


Figure 3.8: The figure (a) shows the detected and clustered lines for the 3 manhattan clusters, (b) the orientation map created by sweeping the lines towards each other, (c) the reconstructed point cloud, (d) the depth map constructed using just the orientation map, (e) the depth map constructed by interpolating just the MVS points, (f) the depth map reconstructed by combining the MVS points with the orientation information, (g) shows the result obtained by using Poisson Surface Reconstruction [54] and (h) shows the result using Manhattan World Stereo [32]

which at least *one* point projects) be given by N_{mvs}. Further, let $\overline{z_p}, p \in N_{mvs}$ be the average depth of all the 3D points projecting onto p.

Then I set

$$c_{mvs}(z) = \sum_{p \in N_{mvs}} |z_p - \overline{z_p}|_2, \tag{3.9}$$

which measures the disparity of the depth map from the point cloud for all the cell in N_{mvs} (non empty MVS points), in Equation 3.8

3.4.2 Results

In this section I show the reconstruction in Figure 3.8. Figure 3.4a shows one of the images in the image sequence for three different sequences. Figure 3.4b shows the result of using the line sweeping approach of [62] on the image, and Figure 3.4c shows the reconstructed point cloud. As

can be seen, many of the details are captured by the point cloud, but the overall structure is well represented in the orientation maps.

The reconstruction results by combining these cues is shown in Figure 3.4f. For comparison, the depth maps using just the monocular cues, i.e. setting $\lambda_s = 0$ are shown in Figure 3.8d. Similarly, the depth maps using just the MVS points, i.e. setting $\lambda_m = 0$, are shown in Figure 3.8e. The results in column Figure 3.8d brings out the overall layout of the scene. However it misses all the objects in the room (the objects on the cabinet in sequence 1, the cushion and the bean bag in sequence 2, and the cushion in sequence 3). Further, it introduces various other artifacts in the depth maps. In contrast, the results in Figure 3.8e show the finer details such as the cushion, bean bag, couch, as well as objects kept on the cabinet, but fails to reconstruct the planar facades. The results in Figure 3.8f bring out both the finer details as well as the layout of the scene.

3.4.3 Comparison

I compare the result of our algorithm with Poisson Surface Reconstruction [54] which is a popular choice of meshing and fusing point clouds and works in the 3D space. The results of their method is shown in Figure 3.8g. The results show that structure of the room is not reconstructed well, and the finer details too have a number of artifacts. I also show the results using [32] in Figure 3.8h which enforces the manhattan prior on the structure on the point cloud. In their approach, first (manhattan) planes are detected in the reconstructed 3D point cloud (in regions where texture was present) and then a labeling problem is solved to assign a plane label to each pixel in the image. The depth maps have disconcerting occlusions and the a manhattan structure is wrongly imposed on non manhattan objects such as the cushion, couch or objects on the cabinet. For the second result, the manhattan structure itself is incorrect due to the sparsity of points in the $three$ orthogonal directions.

3.5 Conclusions

In this chapter we introduced an approach for the reconstruction of architectural scenes, in particular indoor scenes, which have traditionally been challenging for stereo by complementing stereo cues with monocular cues, readily present in such structured scenes. Further, while traditional global formulations for stereo are non-linear, we propose a linear relaxation for our problem that produces

uniformly better results, and significantly reduces computational time. We also extend our approach to the multi view stereo case, to obtain reconstruction that outperform state-of-art-methods [32, 54].

Currently, we use the orientation information in only one of the two input images. In future it might be useful to incorporating the orientation map of the second image and use it to resolve any conflicts. However, projecting this orientation information into the first image, is possible only if we know the depth estimates first. While an E-M style approach could yield improvements, in our limited experiments we found that this showed very minor improvements, at the expense of more algorithmic complexity.

Future improvements would include extending the algorithm to handle non-Manhattan orientations. Our prior work [60] has been used to model a broader class of architectural scenes called *Piecewise Swept Surfaces*. It would be interesting to combine the results in [61] using SVR techniques, with stereo.

As we have seen, the broad literature on SVR has studied many cues such as shading, contours which can also be readily combined with stereo. Other cues such as brightness variation in rooms with height of the wall (as most light sources are on the ceiling) can also be used within this approach. Similarly other learning approaches [25, 46, 83] can also be used within this framework.

Chapter 4

VISUALIZING THE RECONSTRUCTIONS : A PATH PLANNING APPROACH

In the previous chapters, I focussed on techniques to create high quality 3D models, automatically and in a scalable fashion. Having constructed the 3D models, the next challenge is to create useful scene visualizations that can provide a rich and immersive experience for the user. Such an experience should replicate the feeling of being physically present, in terms of being able to visualize the scene from different viewpoints, and quickly assimilating the highlights of the scene. As we have seen in the previous chapters, Internet photographs provide a rich and diverse source of visual data that can be used to efficiently create 3D reconstructions for many popular tourist sites. One immediate application is in creating virtual tours for these famous tourist sites. Indeed, our approach [61] has been commercialized as the Photo Tours feature in Google Maps to create tens of thousands of tours all over the world [3]. Figure 4.1 shows the *world scale* deployment of our approach — each photo tour (shown as a red dot on the map) is a movie which includes fluid 3D transitions between a sequence of photographs of the scene.

As we want our approach to scale, rather than relying on user interaction, we instead propose the problem of automatically generating the sequence of frames that best conveys the essence of the scene. We call such a sequence a *photo tour* — an automatically generated movie that serves as a informative guide for the scene. We look upon creating such a *photo tour* as a path planning problem through an image graph consisting of a node for each image and an edge between a pair of nodes if they share common visible 3D points. A tour on this graph is a sequence of nodes that we would like to visit to convey the feel of the scene. We make the tour *informative* by computing a set of *canonical views [88]* capturing the most frequently photographed scene content, and constraining the tour to include these nodes. We address *efficiency* by posing this as a traveling salesman problem (TSP) [7] on this graph to compute the shortest tour, under an appropriate cost function, that enforces *coherence* by choosing edges in the graph that encourage high quality transitions — the transitions themselves are created by moving a virtual camera along the edges between the nodes,

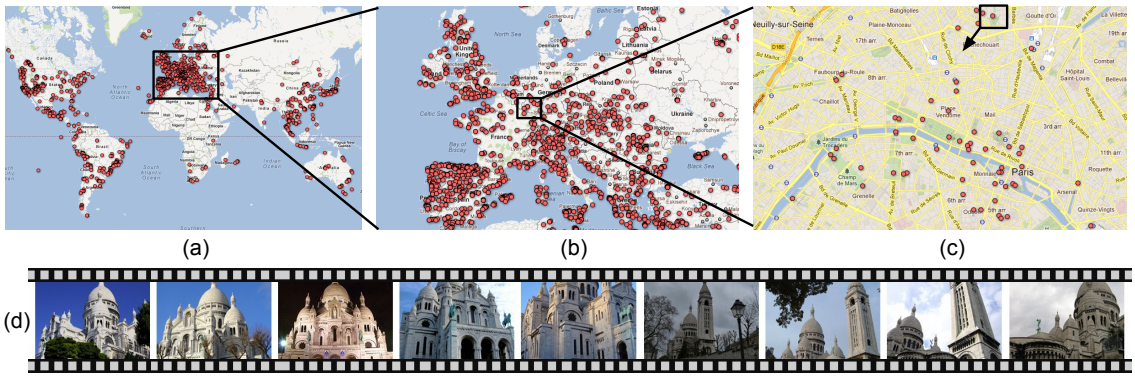


Figure 4.1: We have computed thousands of *photo tours* across the globe, shown as red dots (a). (b) shows a closeup of Europe, and (c) a zoom-in to one neighborhood in Par is. (d) shows the sequence of views in the photo tour for Sacre-Couer near Paris; the movie itself includes 3D transitions between consecutive views.

using techniques from image based rendering(IBR) [15, 24, 41] to generate 3D movies. To achieve this, we crawled more than a million geo-tagged user photos, clustered them into thousands of individual sites, reconstructed camera positions, scene geometry, and popular viewpoints, planned optimal tours, and rendered fly-through movies of each site.

4.1 Related Work

Work on image-based rendering [15, 24, 41] has yielded significant progress towards addressing coherence through more realistic transitions between photos which allow the viewer to maintain context and physical relationships. However, for a visualization to be informative, it must communicate what that scene is *about*, i.e., the most interesting and relevant content. Finding good content and navigating efficiently remain significant challenges for current state-of-the-art IBR systems like Photosynth [2] and Streetview. Photosynth, for example, has four different modes of viewing the scene (3D view, overhead, 2D view, point cloud), with a dozen different controls (buttons or other click behaviors) in the main 3D mode - which take time to master. Even for an expert, it can be difficult to find his way around a new scene. To overcome these problems, we propose to take away user interaction, and instead pose the problem of automatically generating a short video for each site of interest that conveys the essence of the scene. Watching a photo tour is like looking over the shoulder of an expert Photosynth user, who knows all the highlights and how to traverse them.

Furthermore, the movie can communicate the most interesting aspects of the scene in a relatively short amount of time. And while our focus is on automation, we note that photo tours can be used to compliment an interactive system, e.g., as a quick scene overview or auto-play mode.

Secondly the warps produced by the PhotoSynth based on planar proxies, work well for roughly planar facades (e.g., Trevi Fountain, Notre Dame), but are not as effective for non-planar scenes. While depth data, e.g., from multi-view stereo [33, 35, 42], can be used to produce more realistic warps [91], such data is typically incomplete - and thus the renderings produced often suffer from artifacts such as ghosting depth hallucination. Goesele et al. [41] introduced an approach to address this completeness issue which uses 3D data where it is available, and a blurred color wash effect (called the *ambient point cloud*) elsewhere. This combination results in fewer holes in the rendering, at the expense of blur. Our approach looks instead at carefully choosing only those transitions that can be rendered with high quality—thus bringing the domain of Internet Photo Collections within the domain of IBR. The massive size and redundancy in these photo collections allows us to chose only the informative images and high quality transitions. To create these tours, we draw upon prior work in path planning, image based rendering and depth generation, which I discuss now.

Path Planning

Most related to our work is the work by Snavely et al. [91] which proposed a way of computing good paths through photo collections, and guiding the user to these paths. They introduced the idea of optimizing for and rendering sequences of photos in internet collections. In that work, the authors computed circular paths (orbits and panoramas), but did not address the general planning problem (which is the focus of this work). They also addressed the problem of two-point planning, i.e., recovering the best path from one image to another—an important subcase which we generalize here. An important distinction is that [91], like prior work in image-based rendering, sought to produce an interactive system, whereas our objective is a completely automatically generated *movie*.

We produce photo tours of popular tourist sites by processing imagery from photo sharing sites like *Flickr.com*. The goal is to automatically plan a tour through the most interesting (i.e., frequently photographed) objects and viewpoints, with compelling 3D transitions that convey the spatial relations between views. In this chapter, we show how to convert this problem into a graph with images

as nodes, and edge weights encoding the quality of rendered transitions between each pair of images. Computing an efficient tour reduces to solving a form of the Traveling Salesman Problem (TSP) [7], which is NP-Complete [19]. However, it differs from the traditional TSP, in that we are required to visit only a subset of nodes in the graph. The most related formulation is the Traveling Tourist Problem [44], which solves for the shortest tour that would allow a tourist to *see* all the nodes in a graph, by visiting each node or one of its neighbors (it is assumed that every node is seen by its neighbors—the edges represent lines of sight). The formulation, however, is scene-based rather than photo based, doesn't account for partial visibility or rendering quality, and requires coverage of all the nodes (overkill in our case). Also related is the Art Gallery Problem [18], which looks at the problem of placing guards in a museum, modeled as a polygon, such that every point in the polygon is visible to at least one guard. This formulation, however, doesn't support the notion of a tour, let alone an optimal tour.

Image Based Rendering

We draw heavily on prior work in IBR for producing effective image transitions, but by optimizing for fixed *tours* rather than interactive experiences, we seek to side-step many of the challenges and pitfalls that have made prior systems hard to learn and navigate. Our technique uses depth maps to give a compelling sense of parallax, which is further enhanced by using a technique called the Ken-Burns effect from 2-D photography.

Our system represents the first attempt to deploy an image-based rendering (IBR) system *at world-scale* by harvesting the vast stores of community photo collections on the Internet. The photo tours feature in Google Maps implements our method to generate movies for thousands of sites. These sites are indicated as red dots on the maps in Figure 4.1, which shows the distribution of *photo tours* across the globe, in Europe and around Paris. Our approach resembles Photosynth in terms of scale and computer vision techniques, but differs in our use of community photo collections(CPC), rather than photos contributed by a single user. CPC's are useful in that they also capture the distribution of photos that people take of a scene, which can be mined to guide the user to the highlights.

Depth from Point Clouds

Our approach also draws on techniques for creating geometry from 3D point clouds. A number of approaches [24, 32, 34, 38, 72, 113, 116] have been proposed that attempt to construct a mesh model using the reconstructed point clouds under various assumptions. One approach is to generate a height field (often called a representation in 2.5 dimensions) where the height corresponding to every point on the ground is solved for. Gallup et al. [38] recover the height map of houses along a street from street level video. Furukawa et al. [32] using similar manhattan world assumptions, introduce stereo algorithms for such applications and use them in the reconstruction of building interiors [34], where stereo algorithms generally fail, due to the presence of low textured walls. However, in locations with concave structures this assumption often leads to fake geometry which can produce displeasing hallucinations.

Another approach to reconstructing mesh models from point clouds focuses on detecting symmetry and repetition in structures in order to fill in holes in the reconstruction. Mitra et al.[72] focus on discovering symmetry in geometric models by matching local shape signature pairs and using them as evidence for symmetries in an appropriate transformation space. However, Zheng et al. [116] note that the amount of noise typically present in LIDAR scans makes it very hard to detect symmetry for filling holes. Their approach instead involves having a user select a repeating element and using this selection to search for repetitions of this element using descriptors. Once such repetitions have been detected, all instances are brought into the same coordinate frame and merged after some de-noising.

Another approach attempts to fit primitives like planes and cylinders to the point cloud to fill in regions where these structures extend. Zebedin et al. [113] use primitives such as planes and surfaces of revolution for modeling urban scenes using aerial photography. Debevic et al. [24] exploit the characteristics of architectural scenes and represent the scene as a set of blocks constructed from polyhedral primitives linked together by spatial relationships in a hierarchical structure. However, this assumption too, is likely to break in places with complicated geometry, which is usually the case with internet photographs because structures with complicated and unique geometry are often photographed the most.

Our approach differs from these approaches in that we do not solve for a global model. Instead,

we focus on computing a depth map for each image. It avoids the inconveniences in previous approaches in that they do not have any recovered geometry for textureless regions like the sky or grassy lawns. As these textureless regions are part of the images but have no corresponding geometry, the textures are not mapped onto any surface and thus these regions do not appear in the transitions. We solve for the geometry of the entire image and can therefore display these regions in the transitions. The fact that they often have smooth texture implies that even with hallucinated geometry, the transitions produced are pleasing. It also has the benefit of being inherently adaptive in resolution depending on the detail of the image.

4.2 Problem Definition

Consider a collection of geo-registered images $I = \{I_1, I_2...I_m\}$ with known camera pose and sparse 3D points, recovered through a Structure from Motion (SfM) procedure, which also specifies which 3D points are visible in which images (details deferred to Section 4.6). Suppose we also have a depth map for each photo, e.g., recovered through stereo, represented as a set of depth values over a regular pixel grid.

Construct an image graph G_I, consisting of a node for each image and an edge between a pair of nodes if they share common visible 3D points. A tour $P = \{P_1, P_2...P_{|P|}\}$ on this graph is a sequence of nodes in G_I such that each node P_j is connected to the next node P_{j+1} through an edge, call it e_j. Our goal is to compute a tour that is informative, coherent, and efficient. We address the first objective by computing a set of *canonical views* C [88] capturing the most frequently photographed scene content, and constraining the tour to include these nodes. Simon et al. [88] select these images using a greedy approach so that together cover the 3D point cloud well.

We will achieve *efficiency*, by computing shortest paths through the graph. *Coherence* is achieved by ensuring the viewer maintains context as the tour transitions between photos, and encoded via two objectives defining *rendering* and *smoothness* costs, respectively, as follows.

- *Rendering*: We seek a tour P that leads to high quality visualizations. Thus, each edge on the tour should produce high quality renderings between the images.

- *Smoothness*: Geometrically, P should result in a smooth (not jerky) camera motion when

traversed. Thus, for each pair of consecutive edges on P, the corresponding transitions should vary smoothly.

The overall objective is then to find the tour $P : C \subset P$ that minimizes

$$\sum_j RenderingCost(e_j) + \alpha \sum_j SmoothnessCost(e_j, e_{j+1}) \tag{4.1}$$

where α trades off transition quality for the smoothness of the tour, and is set to 1.0 in all our experiments.

In the remainder of this section, we elaborate on the cost functions used to encode the objectives. Later (in Section 4.3), we describe our method for generating tours that approximately minimize Eq 4.1.

4.2.1 Cost Functions over the Image Graph

We now define the $RenderingCost$ for each edge in G_I and the $SmoothnessCost$ for each pair of edges that share a node in G_I. We can think of these two costs as, respectively, a first order cost that encourages high quality transitions between images and a second order cost that regularizes the tour to avoid jerky camera motion.

Once we have the key sites detected and the graph G_I setup, we set the weights over the edges (and pairs of edges) in G_I, in order to choose paths through the canonical images that are likely to have high quality renderings and geometrically smooth motion. We look at a variety of desirable properties that the transitions should have. For each of these we look at the both the first and second order terms which are then incorporated as edge costs and edge-pair costs respectively.

The first order terms look at each transition (between two images) individually — thus the smoothness of the path as a whole is not catered for. In particular, without the second order terms, the paths may not proceed in a consistent direction and may have a lot of wobble in the optical axis, which causes an unpleasant viewing experience. The second order terms, by considering a pair of edges, are able to enforce regularization over the path, and make it less jerky.

We construct these costs from a set of terms which we now describe. Let e_1 denote the transition from I_A to I_B and e_2 denote the transition from I_B to I_C; these edges correspond to a sequence of two consecutive transitions. We define the following desirable properties for a *good* tour:

1. **Camera Motion.** Larger transitions – in terms of motion of the camera center, as well as changes in the rotation matrix – can be expected to render poorly and thus should be penalized with higher costs. Let O_A be the optical center, and R_A the rotation matrix for image I_A (similarly for B and C). Then

$$t(e_1) = ||O_A - O_B||_2, \tag{4.2}$$

$$r(e_1) = ||R_A^{-1} R_B||_\theta, \tag{4.3}$$

$$t(e_1, e_2) = ||O_A - 2O_B + O_C||_2, \tag{4.4}$$

$$r(e_1, e_2) = ||(R_A^{-1} R_B)^{-1} (R_B^{-1} R_C)||_\theta \tag{4.5}$$

where $||R||_\theta$ is the angle of rotation. Equations 4.2 and 4.4 penalize the first and second derivatives of the camera motion, while Equations 4.3 and 4.5 penalize the first and second derivatives of the camera orientation.

2. **Optical motion.** Transitions which have large apparent scene motion – optical flow – in the renderings can be expected to have more artifacts. We approximate the overall flow as the average projected motion of the SfM points that are common to pairs of images. Denote this flow as M_{e_1}. Then,

$$m(e_1) = ||M_{e_1}||_2 \tag{4.6}$$

$$m(e_1, e_2) = ||M_{e_1} - M_{e_2}||_2 \tag{4.7}$$

penalize the first and second order optical motion.

We use SfM points here (rather than the dpeth-map samples, which we compute later) because the latter are often over-smoothed versions particulary in regions where no 3D data is recovered, and hence less reliable.

3. **Scene Area Coverage.** We would like to avoid transitions that zoom-in/ zoom-out of the scene a lot. We encode this by aiming to keep the surface area covered by an image roughly the same as we transition between images. We measure the surface area covered by each image's depth map as follows. For each pixel with depth z in its corresponding depth map, its surface area is roughly proportional to $\frac{z^2}{f^2}$, where f is the focal length of the image. Then the

surface area of the depth map is just the sum over all pixels. Letting S_A denote the surface area for image I_A (similarly for B and C), then

$$s(e_1) = |S_A - S_B| \tag{4.8}$$

$$s(e_1, e_2) = |S_A - 2S_B + S_C| \tag{4.9}$$

penalize the first and second order change in surface area.

4. **Stretching Artifacts.** We would like to exclude transitions where the depth map corresponding to one image has regions that are severely stretched when rendered from the viewpoint of the next camera on the tour. Consider a depth map pixel in I_A (but not on its boundary), corresponding to a point in 3D. The projection of this point into I_B gives a 2D location p; similarly, the original 4-neighborhood projects to locations N_p. We define the stretching associated with p as

$$L_p = ||p - \frac{1}{4} \sum_{q \in N_p} q||_2, \tag{4.10}$$

which measures how much the average of the position of the 4 points around p differs from p. [1] Then the stretching penalty $l(e_1)$ associated with the edge e_1, is set to the 99 percentile value of L_p over the two depth maps involved in the transition, thus ensuring that we penalize transitions where the (close to) maximum stretching is large, as that is a strong indicator of the quality of the transition.

These costs are now incorporated into the (first order) $RenderingCost$ and (second order) $SmoothnessCost$:

$$RenderingCost(e) \tag{4.11}$$
$$= t(e) + \alpha_r r(e) + \alpha_m m(e) + \alpha_s s(e) + \alpha_l l(e)$$

$$SmoothnessCost(e_1, e_2) \tag{4.12}$$
$$= t(e_1, e_2) + \alpha_r r(e_1, e_2) + \alpha_m m(e_1, e_2) + \alpha_s s(e_1, e_2)$$

We use $\alpha_r = 1, \alpha_m = 30, \alpha_s = .002$ and $\alpha_l = 100$ for all our experiments. Note we use the same constants to weight the first and second order weights. For the units, we scale translation so

[1] This is equivalent to applying the discrete Laplacian and computing the magnitude.

that the variance of the centers of canonical images is 1. We apply this same scale factor to the scene area measure. Rotation is measured in degrees, and optical motion is measured in terms of screen space so that the height of the screen is 1.

4.3 Generating Tours

We now describe our algorithm for generating tours through photo collections. In section 4.2, we defined the problem as finding the tour P that passes through canonical views C while minimizing Eq. 4.1. Computing the optimal tour on a large graph G_I, however, would be prohibitively expensive, under the second order smoothness costs.

To simplify the problem, we initially ignore the $SmoothnessCost$ when passing through the nodes corresponding to canonical views. A preliminary tour then can be seen as the concatenation of the shortest paths between pairs of canonical views in G_I as shown in Section 4.3.1, as each of the concatenated segments can be solved for independently. As a further simplification, we prune the graph before computing each of the shortest paths. Having computed the shortest paths between each pair of canonical nodes, we compute the optimal ordering of the canonical views in Section 4.3.2, by posing the problem as a TSP. We then retain this ordering of canonical views in the preliminary tour and compute the final tour after re-introducing the $SmoothnessCost$ across canonical views in Section 4.3.3, to ensure a tour that is smooth everywhere. Algorithm 1 summarizes our approach.

4.3.1 Approximate Shortest Paths

We can solve for the shortest path between pairs of canonical views in G_I under the first and second order cost functions described above, using Dijkstra's algorithm [20] on a modified, higher order graph G_H, called the line graph. In particular, each edge in G_I becomes a node with weight $RenderingCost$ in G_H and each pair of edges with a node in common in G_I becomes an edge with weight $SmoothnessCost$ in G_H. This graph construction to create the Line Graph G_H is shown in Figure 4.3.

Given a canonical view C_j in G_I, we can identify an edge emanating from C_j and thus its corresponding node in G_H. We then treat the node in G_H as a source node and run Dijkstra's

Algorithm 1 $P = Tour(I)$

(SfM, depth maps, C) $\leftarrow$ Pre-process(I)

Construct G_I and compute $RenderingCost$ per edge

for $C_j, C_k \in C$ **do**

Compute $\tilde{G}_I(j,k) \leftarrow Prune(G_I, C_j, C_k)$, using $RenderingCost$.

Compute $SmoothnessCost$ on $\tilde{G}_I(j,k)$

Compute shortest paths in $\tilde{G}_I(j,k) : C_j \rightarrow C_k$

end for

Construct G_C. Store cost of $C_j \rightarrow C_k$ as edge weight in G_C

$\Pi \leftarrow TSP(G_C)$

Construct $G_\Pi \leftarrow Splice(\tilde{G}(\Pi_1, \Pi_2), \cdots, \tilde{G}(\Pi_{|C|-1}, \Pi_{|C|}))$

Compute $SmoothnessCost$ per edge-pair across each C_i

$P \leftarrow$ shortest path in G_Π from C_{Π_1} to $C_{\Pi_{|C|}}$

RETURN P

algorithm. We repeat this for all edges emanating from C_j and find the shortest edge paths that reach each of the other C_k.

Pruning the Graph

While our shortest path algorithm is exact, it is again quite expensive. In particular, the constructed graph $G_H(V_H, E_H)$ has $|V_H| = |E_I|$ and $|E_H| = O(|V_I|^3)$, where V_I, E_I denote the vertices and edges of G_I. Thus, the complexity of computing shortest paths in G_H is much higher than in G_I [104].

To make the problem more tractable, we solve for approximate shortest paths on a pruned graph. For each pair of canonical views, we start from the original graph and remove vertices that are *unlikely* to be part of the shortest path. Consider a pair of canonical views C_j and C_k. We assign edge weights to G_I according to $RenderingCost$ and then run Dijkstra's shortest path algorithm on G_I from C_j(and C_k) to get the shortest distance $d(C_j, V)$ (and $d(C_k, V)$), to each other node V. Then we sort the vertices by $d(C_j, V) + d(C_k, V)$, the length of the shortest path under the

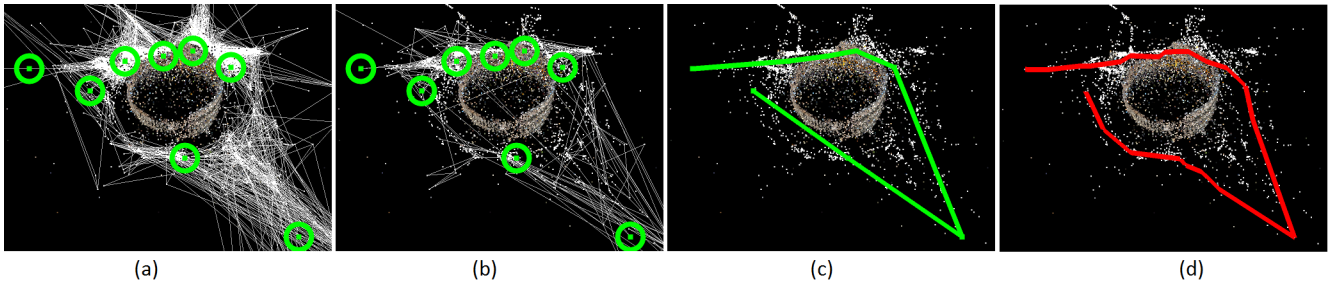


Figure 4.2: For the Colosseum (a) shows the point cloud overlayed with the unpruned G_I and canonical views highlighted as green circles. (b) shows the (overlaid) set of pruned graphs $\tilde{G}_I(j,k)$. (c) shows the ordered canonical views. (d) shows the final computed tour through the camera centers in red.

$RenderingCost$ from C_j to C_k, constrained to pass through V, and keep the ones with the K (set to 50) smallest values. These vertices and the edges between them form a pruned graph $\tilde{G}_I(j,k)$. This heuristic helps us to reduce the computational time (in subsequent steps) while keeping the most promising vertices. Figures 4.2a and 4.2b illustrate the reduction in graph size due to pruning for the Colosseum data set, while keeping the most promising edges. As another example, for one of our data sets – The St. Vitus Cathedral in Prague – the number of edges was 303,307 in G_H which was reduced to 1225 edges on average for each pair of canonical views.

After pruning, we include the $SmoothnessCost$ for the pairs of edges in $\tilde{G}_I(j,k)$ and compute shortest paths from C_j to C_k. We perform the pruning and shortest path computations in parallel for each pair of canonical views. At this we have computed the optimal paths between each pair of Canonical nodes, under both the $RenderingCost$ and the $SmoothnessCost$. If we ignore the $SmoothnessCost$ across the Canonical nodes, the tour can be constructed as a concatenation of these shortest paths. We solve for the order in which to visit the nodes Section 4.3.2. The $SmoothnessCost$ is then added across the canonical nodes too, in Section 4.3.3, to compute a path that is smooth everywhere.

4.3.2 Canonical Graph

After computing the shortest paths between pairs of canonical views, we solve for an ordering of the canonical views as though the tour were to take these shortest paths. To do so, we construct

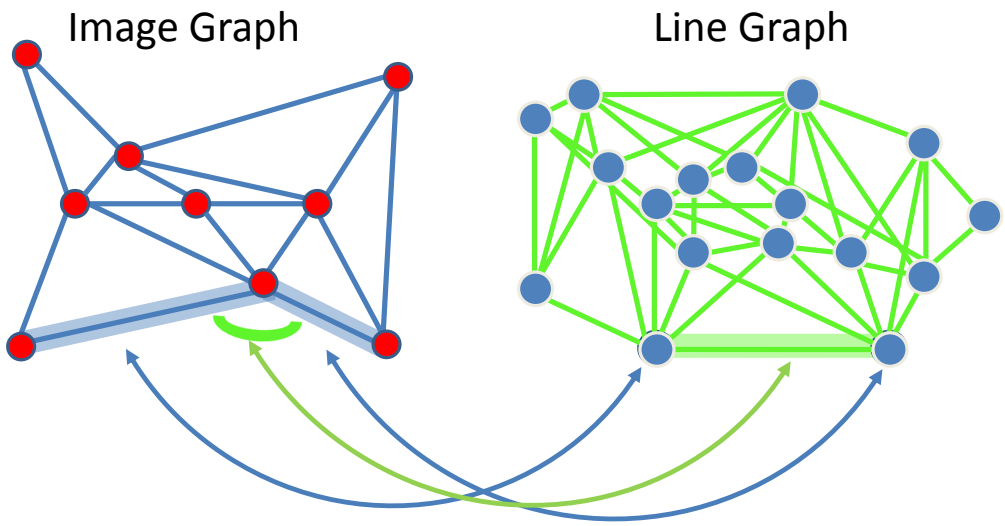


Figure 4.3: The figure shows the construction of the Line Graph G_H, from G_I, where each edge in G_I becomes an node in G_H, and each edge pair incident at a node in G_I, becomes an edge in G_H.

the canonical graph G_C, the complete graph over the canonical views, with the cost of the edge (C_j, C_k) between each pair of images in G_C set to the cost of the shortest path $(C_j \rightarrow C_k)$ between these images in $\tilde{G}_I(j,k)$, as shown in Figure 4.4. The problem now reduces to solving the traveling salesman problem(TSP) in G_C, to get an ordering Π. We now use an approximation algorithm Christofides [17] to solve our (metric)TSP.

4.3.3 Computing the Final Tour

At this stage, we could concatenate the shortest paths between canonical views according to the ordering Π, but the lack of smoothness across the canonical views leads to jerky camera motions when transitioning through them. To avoid this, we first splice together the pruned sub-graphs

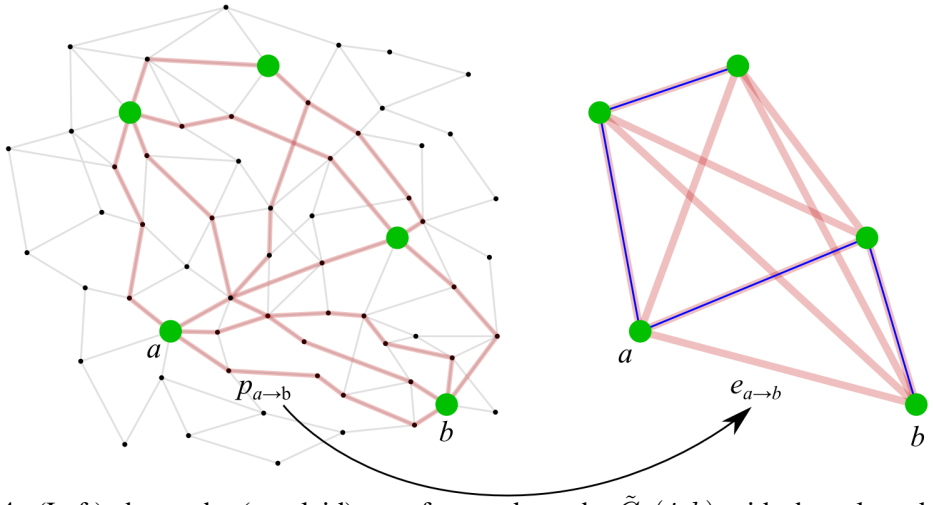


Figure 4.4: (Left) shows the (overlaid) set of pruned graphs $\tilde{G}_I(j,k)$ with the selected canonical view nodes in green and the shortest paths between each pair displayed in red. (Right) shows the construction of G_C with the blue line showing the optimal ordering of the nodes.

$\tilde{G}(\Pi_1,\Pi_2), \tilde{G}(\Pi_2,\Pi_3), \cdots, \tilde{G}(\Pi_{|C|-1},\Pi_{|C|})$ to form the graph G_Π. These graphs are spliced so that only the canonical views are in common; e.g., graphs $\tilde{G}(\Pi_1,\Pi_2)$ and $\tilde{G}(\Pi_2,\Pi_3)$ are spliced to meet exactly at C_{Π_2}. Any other nodes or edges that the sub-graphs have in common are duplicated rather than identified with each other across the sub-graphs when constructing G_Π. We can then construct the line graph for G_Π and use Dijkstra's algorithm to solve for the the shortest path from C_{Π_1} to $C_{\Pi_{|C|}}$, following the approach described at the beginning of Section 4.3.1. Note that we have to compute a shortest path starting from a single source (C_{Π_1}). By construction, this path – the final tour – will pass through all of the canonical views, and maintain the ordering previously computed. The construction of G_Π, and the final path is shown in Figure 4.5. Figures 4.2c and 4.2d illustrate the path through the canonical graph and the tour for one data set (Colosseum).

4.4 Depth-map Reconstruction

Having selected the transitions that comprise our, we need to make these transitions looks as realistic and compelling as possible. We use the 3D point cloud (generated using SfM and MVS techniques) to create suitable geometric proxies to generate image based renderings. There has been much prior work in this domain [24, 32, 34, 38, 72, 109, 113, 116] that attempt to construct a mesh model using the reconstructed point clouds under various assumptions, such as representing the

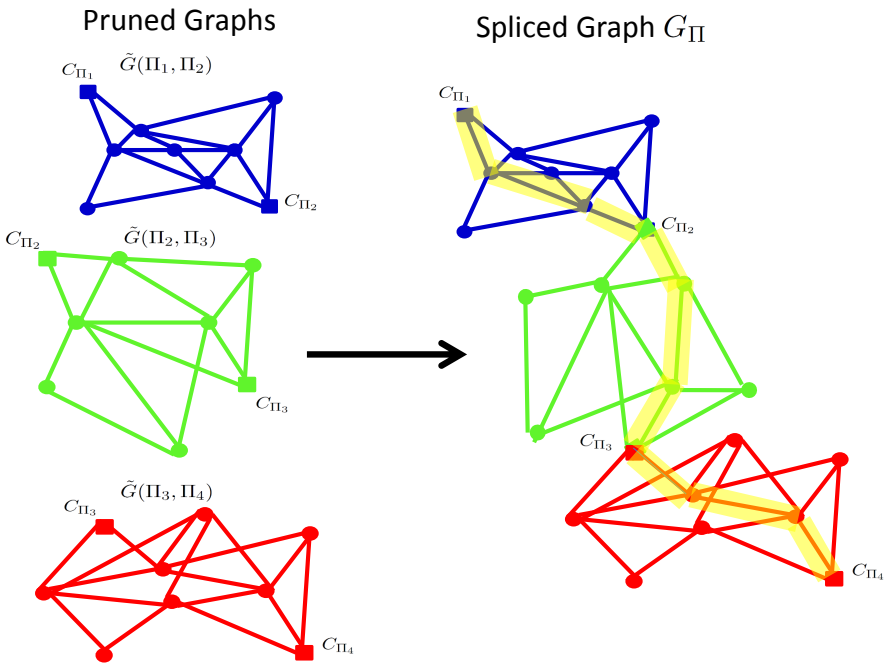


Figure 4.5: The figure shows the construction of the spliced graph G_Π from the pruned graphs $\tilde{G}(\Pi_1, \Pi_2), \tilde{G}(\Pi_2, \Pi_3), \cdots, \tilde{G}(\Pi_{|C|-1}, \Pi_{|C|})$. The final path P (Shown in yellow) passes through the all the Canonical nodes C under the optimal ordering Π, while still being smooth everywhere.

geometry as a height-field, using symmetry to fill in holes, or decomposing the point cloud into simple primitives. While these assumptions work for certain cases, modeling free form geometry present in architectural scenes is not possible using these approaches.

Instead we employ per-photo depth maps to serve as geometric proxies for image-based rendering. At one extreme, we could just use planar proxies, but this can lead to significant ghosting artifacts [91, 92]. At the other extreme, detailed depth maps reduce ghosting, but often significant outliers lead to other artifacts (e.g., stretching). Instead, we want to capture details where they are reliable, and fall back on (smooth) planar-proxy-like models elsewhere. We accomplish this by estimating partial depth maps which are then completed using a Markov Random Field (MRF) that downweights low-confidence data. As these depth maps will be relatively smooth, we compute them at a fairly low resolution, $20,000$ pixels per depth map, while preserving the aspect ratio.

Our depth-map reconstruction algorithm, building on existing techniques, consists of three steps. First, given a reference image I, we follow Goesele et al. [42] and automatically select k neighboring

images (k = 16) well-suited to recovering a depth map for I. Second, we run Furukawa's patch-based multi-view stereo software (PMVS) [36] on the k images to generate a semi-dense point cloud. We retain only the points X that were computed using I, as well as the confidence $\beta(x) \in [-1, 1]$ assigned to each point x. Third, we formulate an MRF objective and solve for the depth d_p at each pixel p that minimizes

$$\sum_p w(p)\rho_d\left(\frac{|d_p - \overline{d_p}|}{\sigma_I}\right) + \lambda \sum_{p,q\in\mathbf{N}} \rho_s\left(\frac{|d_p - d_q|}{\sigma_I}\right). \tag{4.13}$$

The data cost (first term) penalizes the discrepancy between d_p and the expected depth $\overline{d_p}$, set to the average depth of the points in X that project within the extent of pixel p (call this subset $X(p)$). We use a robust norm – the Cauchy function $\rho_d(a) = \ln(1 + a^2)$ – to handle outliers. The penalty for each pixel is also scaled by a confidence $w(p)$, where

$$w(p) = \frac{1}{W} \sum_{x\in X(p)} \max(\beta(x) - \beta_{\min}, 0). \tag{4.14}$$

The numerator is the sum of $\beta(x)$, where $\beta_{\min}$ $(= 0.7)$ is subtracted from each score, so that only 3D points with very high confidence values can contribute. If no point projects to the pixel p, then its confidence $w(p) = 0$. We normalize $w(p)$ to lie in the range $[0, 1]$ by dividing by $W = (1 - \beta_{\min})|X(p)|$, which is the upper bound on the value of the numerator, assuming a fully dense PMVS reconstruction with maximum per-point confidence.

The smoothness cost is chosen to encourage neighboring pixels (defined by a four-connected neighborhood $\mathbf{N}$) to have similar depths. In this case, where outliers are not a concern (as they were in the data term), we apply the less robust Huber loss function ρ_s [48] to the absolute depth difference between neighboring pixels. λ is a relative scaling factor for the smoothness term, set to 2.0 in all of our the experiments.

The Cauchy and Huber norms, which we apply to depth differences, are designed to operate on normalized, unitless quantities. The effective unit of depth differences can vary widely from image to images and across data sets. We compute a per-image normalization constant σ_I which depends on the observed scene depths and is proportional to the the average depth of the MVS points visible in I.

Finally, we minimize Eq. 4.13 and recover a depth map using Levenberg-Marquardt optimization [106].

4.5 Image-based Rendering

Given a sequence of photos along with the underlying 3D geometry, we use a standard view-dependent texture mapping technique to project the images onto their respective geometric proxies, render them to a virtual viewpoint, and blend the images to form a composite frame. As demonstrated by prior image-based rendering systems, this approach works fairly well and yields compelling viewing experiences [24, 34]. However, it is not free of rendering artifacts, particularly ghosting artifacts where the geometry is imprecise. While a more sophisticated IBR method (e.g., [41]) could be used to achieve further improvements; in this work, we focus on optimizing the camera path, rather than the IBR technique

A key observation is that rendering artifacts are minimal when the camera is close to an input photo location. Hence, the visualization should focus most time near input camera positions, speeding up during transition between input camera viewpoints. To generate appealing sequences near an input viewpoint, we take inspiration from the work of Zheng et al. [115], which adds parallax effects to create smooth cinematic camera motion across images. This 3D version of the Ken-Burns effect, a popular technique in 2D film production in which the viewer gently pans and/or zooms across a single image, provides a visually appealing experience. However, While they operate on small sets of images, each set carefully taken with one camera near a single viewpoint, we operate on large, unorganized photo collections, which makes the problem more challenging. Further, while the authors in [115] treat each set in isolation to create a parallax effect for a single view, our goal is to create a coherent path through several views; i.e., the Ken-Burns effect must fit within the context of flying through a photo collection.

In the remainder of this section, we describe how the camera motion is interpolated and parameterized.

4.5.1 Camera Path Interpolation

We move a virtual camera that starts from one input viewpoint and interpolates smoothly towards the next viewpoint on our computed tour. At any point, the virtual viewpoint can be defined by interpolating each camera parameter independently with Monotonic Piecewise Cubic Interpolation [31], which guarantees monotonicity between adjacent sample points and prevents overshooting or os-

cillations. However, simply interpolating the camera center guarantees monotonicity only in the camera parameter space, and not in the screen space which is a good measure for the speed perceived by the viewer. To remedy this, we first compute for each image P_j an *interest distance* d_j set to the average over the all depth map values, and an *interest point* $x_j = c_j + d_j O_j$, where c_j is the camera center and O_j is the optical axis (view direction), i.e., the point along the optical axis at the average depth d_j for the image. We then independently interpolate these interest points and interest distances, along with camera yaw, pitch, and field of view. Camera roll is set to zero throughout. Then, at any point along the curve, we use yaw and pitch to recover the camera rotation and thus the optical axis O. From the interpolated interest point x and the interpolated interest distance d, we compute the camera center for the virtual camera as $c = x - dO$. This determines the complete camera trajectory for our tour.

One approach to traversing this trajectory is to move to virtual camera at a uniform speed from one camera to the next along our tour, as shown in Figure 4.6(top). Here the contribution of the two neighboring cameras changes linearly (i.e., as one falls from 1 to 0, the others rises from 0 to 1 linearly) However as noted above, most of the artifacts occur near the middle of the transitions and the uniform parametrization results in a large portion of the tour focussing on the rendering artifacts. Further, it results in abrupt changes in all the camera parameters (such as camera center, optical axis, FOV) at the input viewpoints, which can be displeasing to the viewer. Instead we choose a different parametrization to overcome these shortfalls. Given a camera trajectory passing through a sequence of photos $\{P_1, P_2, \cdots P_{|P|}\}$, let S denote the parametrization of the trajectory curve, where $S = i$ when the camera is at P_i. The curve is divided into two types of segments: 1) Ken-Burns segments $\{[S_i^s, S_i^e]\}$ around each photo P_i, where rendering artifacts are minimal and the camera moves slowly and spends time D_i^{KB} in traversing it; and 2) transition segments $\{[S_i^e, S_{i+1}^s]\}$, where the camera moves from one photo to the next quickly in time D^{trans}. D^{trans} is set to a constant, while D_i^{KB} is different for each image, to allow the optical flow speed inside the rendering window to be roughly constant. This allows us to spend more time near the input images, thus minimizing artifacts. This parametrization can be seen in Figure 4.6(bottom). Further we adaptively slow down the virtual camera near the input cameras, so that there are no sudden changes in the speed of the camera (note the regions in the black circles where the curve shows no abrupt changes).

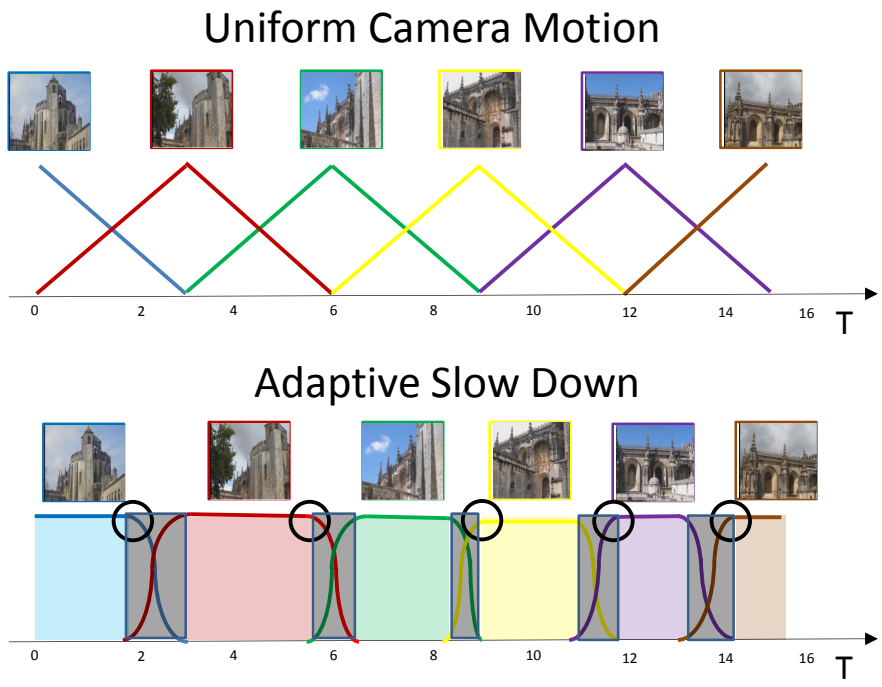


Figure 4.6: This figure shows our approach of adaptively changing speed and transitioning between segments in as opposed to using a uniform parametrization. At any time, the contribution in the blended warp is shown in a color coded fashion. For the uniform camera motion (top) the contributions rise from 0 to 1, as we approach a camera, and then fall back to 0 as we move away from it to the next camera, in a linear fashion. In our parametrization (bottom), the curve is divided into *two* types of segments Ken Burns (shown as color coded boxes) where only one image is used to render the scene, and transitional segments, as we transition from one segment to the other.

Below I describe how we estimate the Ken-Burns parameters $\{S_i^s, S_i^e, D_i^{KB}, D^{trans}\}$ for each segment and the final timing of the tour.

Minimizing Ken Burns Artifacts

The parameters S_i^s, S_i^e, and hence the Ken-Burns segments are computed to highlight the details of the photos. During these segments we want to avoid showing the boundary of the image and to minimize artifacts due to stretching.

As a pre-process, we narrow the field of view of every virtual camera by a factor of 0.8, to avoid seeing image boundaries during Ken-Burns panning. For each image P_i, we then set the Ken-Burns

segment parameters $S_i^s \in [i-0.3, i]$ and $S_i^e \in [i, i+0.3]$, in discretizations of 0.01, to be the minimum and maximum parameter values, such that each parameter sample in $[S_i^s, S_i^e]$ passes a screen coverage and stretchiness tests. The screen coverage test simply requires that the resulting rendered image fills the screen. The stretchiness test requires that the majority of the depthmap edges (at least 99.7 percents) do not stretch more than a constant threshold (1.8) at the virtual viewpoint.

The Ken-Burns segment is defined to be the longest segment $[S_i^s, S_i^e]$ where both the coverage and the stretchiness tests pass within.

Estimating Durations

We want the optical flow in each Ken-Burns segment to be roughly the same and compute an initial estimate of the duration $\overline{D_i^{KB}} = D \cdot m_i$. Here, m_i is the average optical motion between the virtual camera at S_i^s and S_i^e based on the motion of the depthmap grid points associated with P_i, where D is chosen so that the average value of $\overline{D_i^{KB}}$ becomes 2.0 seconds. To avoid abrupt changes in the durations, we penalize the first and the second order derivatives and compute the smoothed durations $\{D^{KB_i}\}$ by solving the following linear least squares problem:

$$\underset{\{D_i^{KB}\}}{\operatorname{argmin}} \sum_{i=1}^{|P|} (D_i^{KB} - \overline{D_i^{KB}})^2 + \sum_{i=1}^{|P|-1} (D_i^{KB} - D_{i+1}^{KB})^2 + \sum_{i=1}^{|P|-2} (D_i^{KB} - 2D_{i+1}^{KB} + D_{i+2}^{KB})^2.$$

Lastly, we clamp each D_i^{KB} to be the range [0.7, 2.7] (in seconds) to ensure no segment is too short or too long. Note D^{trans} is set to a constant (0.7 seconds).

Timing

Given the duration of every segment, i.e., $\{D_i^{KB}\}$ and D^{trans}, it is straightforward to compute times T_i^s and T_i^e (the start and end times for each Ken Burns segment). We have also computed the S parametrization for each segment boundary S_i^s and S_i^e, respectively. We can then interpolate these times to give a complete mapping between S and time, again using Monotonic Piecewise Cubic Interpolation to avoid abrupt changes in speed as illustrated in Figure 4.7.

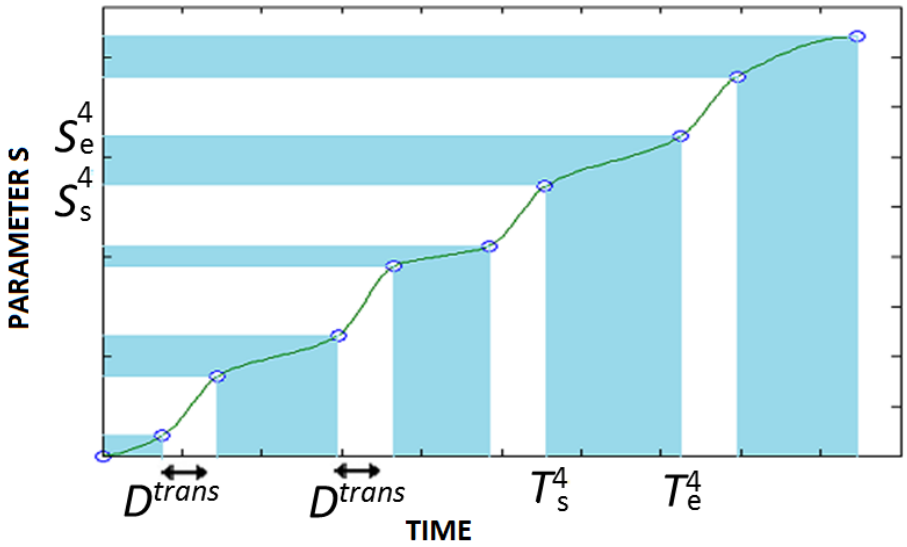


Figure 4.7: A tour is partitioned into a series of Ken-Burns segments (blue intervals) and transition segments (white intervals). The curve shows the mapping between the timing (x-axis) and the parameterization (y-axis) of a tour. Transition segments are fixed-length D^{trans} in time, while the lengths of Ken-Burns segments vary both in time and parameterization.

4.5.2 Rendering

For the rendering in a transition segment $[S_i^e, S_{i+1}^s]$, we simply render P_i and P_{i+1} into the virtual view and blend with a uniform weight per pixel. The blend weights are set to the fractional distances between images, where distance is measured in parameter space S.

During the Ken-Burns segment $[S_i^s, S_i^e]$ we warp a single image P_i, to render the photo tour. As mentioned before, the field of view is reduced so that as we pan over the images, we don't see any background pixels.

4.6 Deployment within Google Maps

We obtained 3.2 million (at the time of publication in July 2012) geo-tagged photos from Panaromio/Flickr. Based on the geo-tags, the photos were clustered into individual tourist sites. Each site was reconstructed using a structure-from-motion technique similar to [67], and placed on the map using a RANSAC method similar to [51]. These reconstructions were performed in parallel, over a cluster with 1000 CPUs, and took about 24 hours, consistent with the Rome-in-a-day

Landmark	# images	# canonical	#tour length	movie duration
Hagia Sophia	874	4	10	30.6 s
Marianplatz	1442	5	12	29.0 s
Arch of Titus	220	3	10	25.5 s
St. Vitus Cathedral	1093	4	8	20.3 s
Abu Simbel	984	3	10	26.3 s
Colosseum	3230	8	22	58.7 s
Hongcun	49	3	6	15.5 s
Loro Parque	76	3	6	15.5 s
Gundam Statue	281	3	5	12.8 s

Table 4.1: Number of images, number of canonical views, number of images in the tour, and movie duration for several landmarks from our results.

projects [6, 30], which processed roughly similar numbers of photos using massive parallelism in a 24 hour period. Our depth-map pipeline required 6 hours and 10 minutes on the same cluster.

About 2.5 million images were successfully reconstructed by our structure-from-motion and stereo algorithms. These images were then processed by our tour generation algorithm to produce 20,857(**new**) photo tours of popular landmarks all over the world. (See Figure 4.1.) Generating these photo tours took 3.5 hours on the same cluster. Figure 4.8 shows the geographical distribution and some of the popular photo tours playing simultaneously. Figure 4.10 shows a sampling of frames from three of our photo tours (More tours can be seen in the Appendix D). We encourage the reader to also look up the videos online [3], to see the fluid 3D transitions. Table 4.1 shows additional information about some of the popular landmarks(the number of images in the collection, the number of canonical nodes, and the tour length in terms of the number of images and time).

Figure 4.9 shows the size of the reconstructions and the lengths of the tours, measured in photos. Note the long-tail distribution: the most popular 2% of landmarks contain between 1000 and 25000 images. The remaining average less than 100 images. The landmarks on average have 3.2 canonical views (and 6 images), while the largest landmark has 16 views (and over 40 images).

(a) (b) (c)

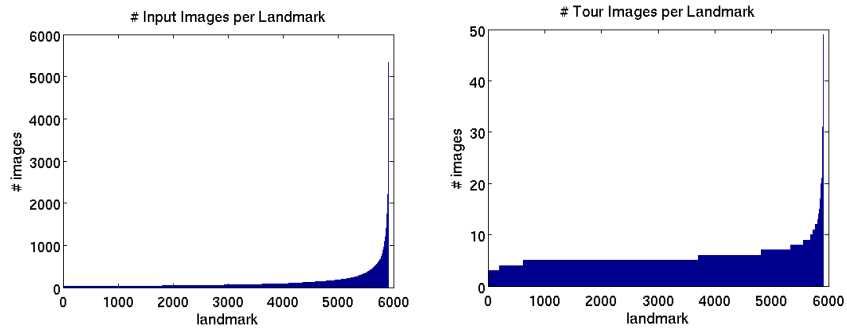
Figure 4.8: The figure shows (a) the geographical distribution of the computed photo tours (each red dot on the map), and some of the tours in different parts of the world playing together. (b), (c) show the most popular 25 and 100 tours respectively, playing simultaneously

4.6.1 Web-based Viewing

Our goal was to stream and render photo tours to any viewer over the Internet via their web browser. The viewer was implemented as a javascript web application running on any browser that supports the HTML5 WebGL spec (Chrome, Firefox, Safari, etc.). The images, depth maps, and sequencing information is stored on a server and streamed real-time to the client. We prefer this to pre-recording movies, primarily to keep the option of interactivity, if needed later. The bandwidth requirements are dominated by the images, the low-res depth maps and meta data being tiny in comparison. Secondly, the typical bandwidth required to play a tour of 10 views for a 1024x768 viewport is less than 3MB, which works out to 115kbits/sec, while for comparison, Youtube's standard video setting is 250kb/s and Netflix starts at 625kb/s, so again photo tours are significantly more efficient than streaming video. Most modern graphics cards have no problem keeping up; the application is able to render a tour at 30 fps on a GeForce 9400M and 60 fps on a Quadro FX 580.

4.6.2 Discussion

This approach yields high quality results for a broad range of scenes. Overall since launch, we have found that of all the tours rated by users, 89.7%(at the time of publication in July 2012) of our photo tours got a thumbs up rating. As with any "at-scale" system, however, there are failure cases. We now enumerate the types of failure cases we've observed.



(a) (b)

Figure 4.9: Size of each landmark SfM reconstruction, sorted by number of photos. Note the long-tail distribution: while most landmarks have less than a hundred photos, the top 2% have over a thousand. (b) Number of photos in each tour; most tours have half a dozen photos, whereas the largest tours have over 40.

The first category of failure cases is due to *data problems*, i.e., problematic photos, pose, or geometry. One effect of a bad photo (and corresponding bad depth map) is seen in our Trevi Fountain photo tour, where a toy monkey is prominently featured in the foreground of one of the photos (as shown in Figure 4.11). It may be possible to reduce or eliminate problematic images through a combination of moderation, user ranking, image quality measures, and detection of faces and other foreground elements.

Errors in depth maps can occur for thin objects (e.g., foreground statues, spans on bridges) as shown in Figure 4.12, reflective surfaces (water, windows), and textureless or partially occluded regions; such errors can produce warping artifacts. While future research on stereo techniques (or use of range sensing techniques like Kinect), may produce better quality geometry, another possibility is to fall back on planar proxies in such cases. Overall, however, the images with depth maps have turned out well, and we see major problems in a minority of cases.

A second category of failure cases is due to *tour problems*, i.e., shortcomings in our graph traversal approach. Some tours are too short, often due to a scarcity of photos, or cases where the reconstruction breaks up into multiple pieces. Other tours are too long; a good example is the Colosseum in Rome tour which does a 360 rotation around the outside of the Colosseum, which can get tedious. The rendering quality is quite good, but the pace is a bit tedious. Similarly, the Notre Dame tour spends too much time in the back of the Cathedral, then rotating to the side, before

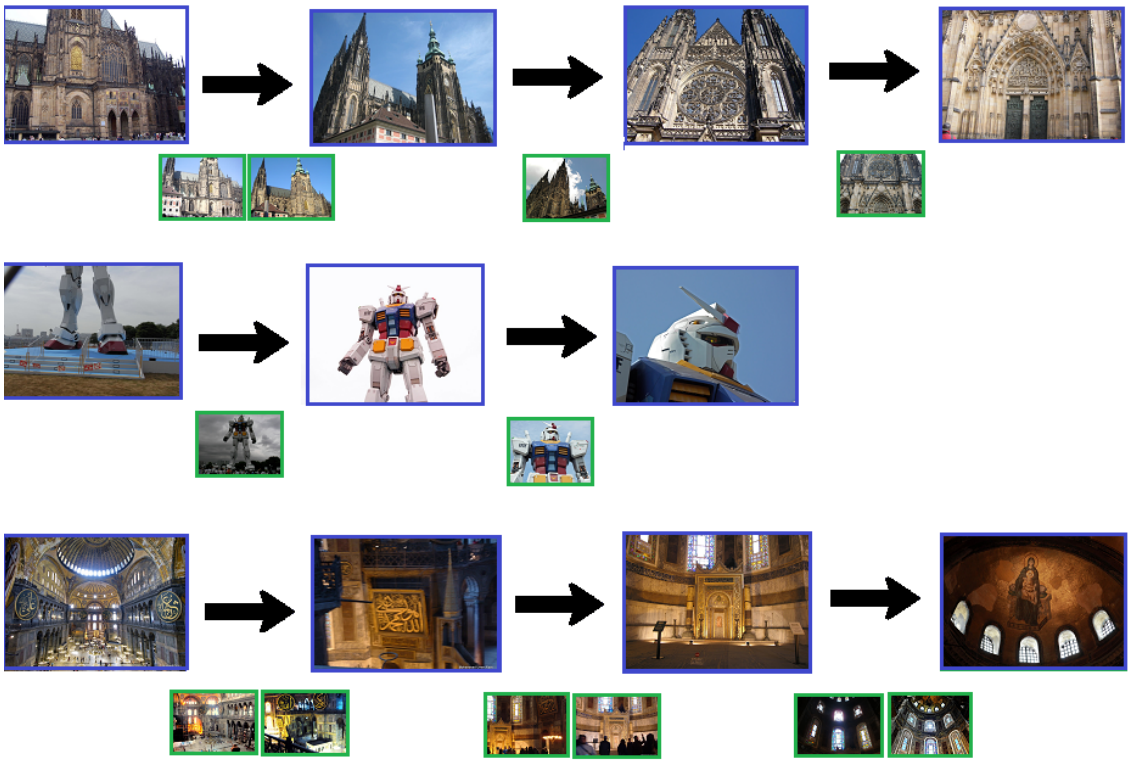
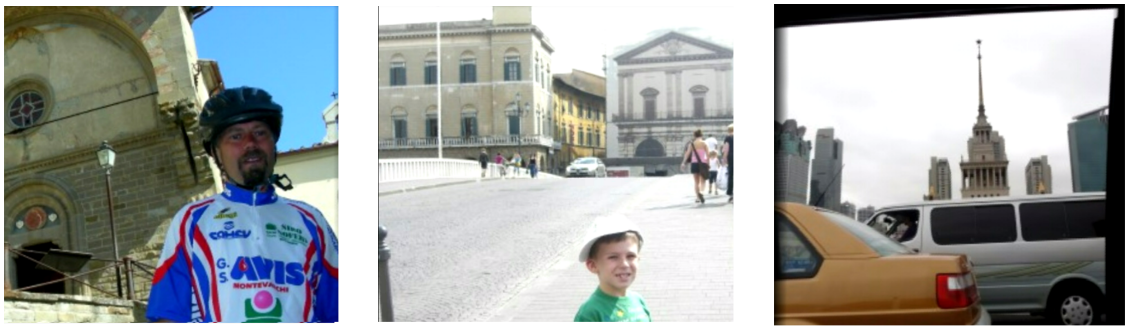


Figure 4.10: Sample photo tours, displayed as a sequence of images. Top: St. Vitus Cathedral (Partial) in Prague, Middle: Gundam (Full), Tokyo, Bottom: Hagia Sophia, Istanbul. The sequence of canonical images are outlined in blue, whereas the transitional views are outlined in green



(a) (b) (c)

Figure 4.11: The figure shows in (a) a person occluding the scene in the photo tour for Propositura di San Niccolo, in (b) a child occluding the scene in Ponte Mezzo, and in (c) traffic occluding the main attraction in the Shanghai Exhibition Center Photo Tour.

Figure 4.12: The figure shows on the left and right, *two* consecutive frames. The middle image shows the rendering from a view point in the middle of the transition. The foreground object, because of noisy 3D data or because of the limitation of representing geometry as a depth map, can result in stretching or ghosting artifacts.

Figure 4.13: The figure shows a sequence of consecutive images for the Palazzo Balbi photo tour. The tour jumps between images of a foreground element (purple dog statue) and images of the landmark, which leads to loss of context.

showing the more interesting details on the front. While repetitions are penalized, they do occur sometimes, as in the case of the Abu Simbel photo tour.

In most cases the scene traversal feels natural, and moves through the distribution of viewpoints in a coherent fashion owing to the smoothness term (e.g., an orbit for Trevi Fountain, a pan for Chicago, an upward pan for the statue of Gundam). The 2nd order smoothness term helps significantly in this respect. However, some tours are less coherent. A good example is the Palazzo Balbi tour which jumps randomly between images of a foreground element (purple dog statue) and images of the landmark, as shown in the Figure 4.13. Such cases could be identified and improved with user interaction, or perhaps with machine learning algorithms trained on good and bad tours.

While most of the photo tours we've generated have few artifacts, there are those that have one or more of the aforementioned failures. E.g., the single bad photo in the Trevi Fountain results in the entire tour to be labeled a failure. By inspection we have found that 50% of the photo tours we viewed are artifact-free and constitute high-quality results. Breaking down these results by landmark size, we have found that popular landmarks (top 2%) are more difficult with 42% being artifact-free. One reason is that larger landmarks result in longer photo tours and are therefore more likely to have one ore more artifacts. For a completely automatic system deployed at scale on unstructured photo collections, we consider it quite successful to achieve high-quality results 50% of the time, and even despite the artifacts, most of low-quality results show promise and are still interesting and useful. We obtained this estimate by having several people spot-check 5% of the tours and average their ratings. If this system were to be deployed on the web, it would be straightforward to check *all* of the tours and launch only the good ones; a hundred Amazon Mechanical Turk operators could

provide multiple ratings for each tour in a few hours. While we expect future research to further improve the success rate, we believe that the current system (with a manual quality check) is already suitable for large scale deployment.

In addition to addressing these problems, an interesting topic of future work is to add textual annotations and image captions, providing context on what's being depicted. For example, the visualization could identify "Raphael's Tomb" inside the Pantheon, and other significant scene elements. Recent work [80] attempts this and uses Wikipedia to automatically search and label reconstructed 3D point clouds.

Chapter 5

CONCLUSION

In my thesis, I have looked at the problem of reconstructing and visualizing 3D models, particularly for architectural scenes. The Internet does provide us with a rich and diverse source of visual imagery for this purpose, but the challenge is in using this imagery to create compelling visualizations for the users. One approach to creating such experiences discussed here is to compute the 3D geometry of the scene (to achieve *coherence*), along with the 3D location of the images and then plan *efficient* and *informative paths*, as a sequence of images, through the reconstruction. I use this path planning approach for scene visualization to create these experiences.

To achieve this goal, I propose solutions to challenges that need to be overcome to make this approach feasible. In Chapter 2, I look at improving the scalability of SfM algorithms used to reconstruct 3D models. I introduced *Visibility Based Preconditioning*, a new technique for preconditioning the linear least squares problems arising in large scale Bundle adjustment problems. Using the visibility information in the scene, we cluster the cameras into tightly interacting clusters. These clusters form the basis of our block diagonal and block tridiagonal preconditioners. When combined with an inexact step LM algorithm, these preconditioners offer equal or better solution quality compared to the best available methods at 3-5x less execution time on problems from the BAL [5] dataset.

A second challenge is in improving the quality of the reconstructions. The presence of texture in the scenes is a necessary requirement for stereo algorithms that are used for reconstruction. But many scenes and in particular indoor scenes, still pose challenges for stereo, due to the lack of texture. Monocular cues have the potential to overcome this limitation. Stereo cues and the line sweeping cues (one possible Monocular cue) are shown to be complementary in nature, and thereby perform well in different regions of the scene. Hence the reconstructions using both these cues are more accurate and complete than those that would be constructed using either of these cues independently. In Chapter 3, I setup the problem as a depth integration problem requiring that the

depth map be consistent with both photo-consistency term (for stereo), as well as the orientation information generated using line sweeping [62]. I show results that outperform state-of-the-art stereo algorithms on challenging indoor data sets. These techniques are also extended to the multiple view case [85] to create reconstructions that are more accurate, compared to other state-of-the-art methods [32, 54].

Having constructed the 3D models, the next challenge is to create useful scene visualizations that can provide a rich and immersive experience for the user. Such an experience should replicate the feeling of being physically present, in terms of being able to visualize the scene from different viewpoints, and quickly assimilating the highlights of the scene. We thus propose the problem of automatically generating the sequence of frames that best conveys the essence of the scene in Chapter 4. Our approach of creating such *photo tours* has been commercialized as the Photo Tours feature in Google Maps to create tens of thousands of tours all over the world [3].

5.1 Future Scope

While this is an encouraging first step in automatically and scalably creating compelling visualizations, there are many directions for future research, both at the *big picture* level as well at the level of the individual challenges. Here I discuss future research directions at both these levels.

The problem of replicating the feeling of actually being present at the scene, in fact has many different aspects to it. For e.g., an actual experience at the Trevi fountain in Rome would possibly involve hearing Italian in the background, which makes audio an integral part of the experience. Video alone cannot capture this. Similarly to obtain a truer experience, other factors such as the ambient atmosphere, weather, clouds etc. need to be replicated as well. Another thing that is often missed in these experiences is the sense of the scale, e.g. the massive size of the Trevi fountain or the Dome in the Pantheon. By creating a stereo display, metric depth could be conveyed to the viewer, to better provide them with a sense of scale.

Another limitation of the photo tours, in their ability to capture the experience, is that most input views and hence the virtual camera in our photo tours have a very restricted field of view. In contrast humans have peripheral visions, thus allowing us to experience almost a 180° view of the surrounding area. Recent work [87] address the problem of extending the field of view of a photo —

an operation they call uncrop. Given a reference photograph to be uncropped, their approach selects, reprojects, and composites a subset of Internet imagery into a larger image around the reference using the underlying scene geometry. This approach, modified suitably can help to increase the field of view for the virtual camera as well.

Further, while we generate one photo tour per site, experiences can vary depending on different times of day/ seasons etc. It would be interesting to categorize the dimensionality of this variation, similar to [39], and customize photo tours by varying these parameters. These experiences could then be part of virtual reality games, i.e, a user could load in a model of the Hagia Sophia for example, and fight aliens there. Projects like Google's Tango or Matteport, aim to create 3D models for indoor scenes that can be used for these purposes. The same could be done for these Internet photo collections as well.

Another direction for future research is to customize photo tours with personal photo collections of the scene, i.e., allowing users to put their personal photo collections, say for their trip to the Colosseum, and create a photo tour that traverses through their photo collections. One simple way to extend our framework would be to first recover the 3D position of these photographs, and to mark them as canonical nodes. This would then force the path to visit these images as part of the *photo tour*.

While the previous directions are ways to improve the experience at the *big picture* level by adding in other perceptory senses, such as audio, feel etc., there is scope for improvement within the individual challenges themselves to improve the quality of the visual tours. There are two clear directions for future work in terms of improving the scalability of our approach. Firstly, our experiments are limited to the BAL dataset. The sparsity patterns present in the BAL dataset are only a subset of the sparsity patterns encountered in real world SfM problems. Thus one future direction would be to increase the scope to other classes of SfM problems. A notable exception to the BAL dataset, for example, is the presence of camera blocks with long range interactions, e.g., aerial views of a scene that would correspond to near dense rows in the Schur Complement S. Visibility based clustering is not the right approach here, and better approximations can be obtained by treating such views separately. Similarly satellite imagery or photographs taken from a plane or a street-view car, would often have a very dense diagonal as it is, and thus a diagonal approximation with a fixed width (capturing the overlap between neighboring photographs) may be

more appropriate. Secondly, while the preconditioners excel at solving linear least squares problems to high precision, when used with an inexact step LM algorithm, which uses fairly high tolerance convergence criterion, the extra work of setting up the block tridiagonal preconditioner is at times not worth the gain. In future work, better forcing sequences (η_k) can be explored along with ways of reducing the setup time of the block tridiagonal preconditioner.

Further, while our approach of combining monocular cues with stereo outperforms the state-of-the-art stereo methods, the approach is restricted to corners and regions where the manhattan structure (all *three* orthogonal directions) are visible. It would be interesting to extend this to the case of panoramas. This provides the most promise in overcoming this challenge of obtaining orientation information even for images that don't capture the manhattan structure themselves, by transferring this information (orientation and 3D line direction) from neighboring images in the panoramas. Further, by working with panoramas, we can obtain full 360° models for indoor spaces rather than just depth maps.

On the visualization side, an interesting topic of future work is to improve the visualizations by adding textual annotations and image captions, and providing context on what's being depicted. For example, the visualization could identify "Raphael's Tomb" inside the Pantheon, and other significant scene elements. Recent work [80] attempts this and uses Wikipedia to automatically search and label reconstructed 3D point clouds. Merging this work with photo tours could be a direct extension.

BIBLIOGRAPHY

[1] Multi-view stereo evaluation. `http://vision.middlebury.edu/mview/eval/`, 2006.

[2] Photo synth. `http://photosynth.net/`, 2007.

[3] Photo tours. `http://maps.google.com/phototours`, 2012.

[4] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `https://code.google.com/p/ceres-solver/`.

[5] Sameer Agarwal, Noah Snavely, Steven M. Seitz, and Richard Szeliski. Bundle adjustment in the large. In *ECCV*, pages 29–42, 2010.

[6] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M. Seitz, and Richard Szeliski. Building Rome in a day. In *ICCV*, 2009.

[7] D.L. Applegate, R.E. Bixby, V. Chvtal, and W.J.Cook. The traveling salesman problem: A computational study. In *Princeton University Press*, 2006.

[8] Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 187–194, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[9] Michael Bleyer, Christoph Rhemann, and Carsten Rother. Patchmatch stereo - stereo matching with slanted support windows. In *BMVC*, pages 1–11, 2011.

[10] Erik Boman and Bruce Hendrickson. Support theory for preconditioning. In *SIAM J. Matrix Anal. Appl. 25(3)*, pages 694–717, 2003.

[11] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 23(11):1222–1239, 2001.

[12] Michael J. Brooks and Berthold K. P. Horn. Shape and source from shading. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'85, pages 932–936, San Francisco, CA, USA, 1985. Morgan Kaufmann Publishers Inc.

[13] M. Byröd, K. Åström, and S. Lund. Bundle adjustment using conjugate gradients with multiscale preconditioning. In *BMVC*, 2009.

[14] J. Canny. A computational approach to edge detection. In *IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679698*, 1986.

[15] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *SIGGRAPH '93*, pages 279–288, 1993.

[16] Y. Chen, T.A. Davis, W.W. Hager, and S. Rajamanickam. Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate. *TOMS*, 35(3), 2008.

[17] N. Christofides. Worst-case analysis of a new heurestic for the travelling salesman problem. In *Techincal Report 338, Grad School of Industrial Administartion, CMU*, 1976.

[18] V. Chvtal. A combinatorial theorem in plane geometry. In *Journal of Combinatorial Theory, Series B 18: 3941*, 1975.

[19] Stephen A. Cook. The complexity of theorem proving proceedures. In *Symposium on Theory of Computing*, 1971.

[20] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to algorithms. In *MIT Press and McGraw-Hill. pp. 595601. ISBN 0-262-03293-7*, 2001.

[21] Ingemar J. Cox, Sunita L. Hingorani, Satish B. Rao, and Bruce M. Maggs. A maximum likelihood stereo algorithm. *Computer Vision and Image Understanding*, 63:542–567, 1996.

[22] David Crandall, Andrew Owens, Noah Snavely, and Daniel P. Huttenlocher. Discrete-continuous optimization for large-scale structure from motion. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*, 2011.

[23] A Criminisi. *Accurate Visual Metrology from Single and Multiple Uncalibrated Images*. Springer, 2001.

[24] Paul Debevec, Camillo Taylor, and Jitendra Malik. Modelling and rendering architecture from photographs : A hybrid geometry and image based approach. In *SIGGRAPH '96*.

[25] Erick Delage, Honglak Lee, and Andrew Y. Ng. Automatic single-image 3d reconstructions of indoor manhattan world scenes. In *In ISRR*, 2005.

[26] Frank Dellaert, Justin Carlson, Viorela Ila, Kai Ni, and Charles E. Thorpe. Subgraph-preconditioned conjugate gradients for large scale slam. In *IROS*, pages 2566–2571, 2010.

[27] Minh N. Do. Cross-based local multipoint filtering. In *Proceedings of the 2012 IEEE Con-*

ference on Computer Vision and Pattern Recognition (CVPR), CVPR '12, pages 430–437, Washington, DC, USA, 2012. IEEE Computer Society.

[28] E.D. Dolan and J.J. Moré. Benchmarking optimization software with performance profiles. *Math. Prog.*, 91(2):201–213, 2002.

[29] Jan-Michael Frahm, Pierre Fite-Georgel, David Gallup, Tim Johnson, Rahul Raguram, Changchang Wu, Yi-Hung Jen, Enrique Dunn, Brian Clipp, Svetlana Lazebnik, and Marc Pollefeys. Building rome on a cloudless day. In *ECCV*, 2010.

[30] J.M. Frahm et al. Building rome on a cloudless day. *ECCV*, pages 368–381, 2010.

[31] Carlson R. E. Fritsch F. N. Montone piecewise cubic interpolation. In *SIAM J, Vol. 17*, pages 238–246, 1980.

[32] Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. Manhattan-world stereo. In *CVPR 09*, pages 1422–1429.

[33] Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. Towards internet-scale multi-view stereo. In *CVPR 10*, pages 1434–1441.

[34] Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. Reconstructing building interiors from images. In *ICCV 2009*, pages 80–87, 2009.

[35] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multi-view stereopsis. In *CVPR 07*.

[36] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multi-view stereopsis. *PAMI 2010*, 32(8):1362–1376.

[37] David Gallup, Jan-Michael Frahm, Philippos Mordohai, Qingxiong Yang, and Marc Pollefeys. Real-time plane-sweeping stereo with multiple sweeping directions. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 0:1–8, 2007.

[38] David Gallup, Jan-Michael Frahm, and Mark Polleffeys. A heightmap model for efficient 3d reconstruction from street-level video. In *3d pvt Conference Proceedings*, 2010.

[39] Rahul Garg, Hao Du, Steven M. Seitz, and Noah Snavely. The dimensionality of scene appearance. In *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009*. IEEE, 2009.

[40] Norman E. Gibbs, William G. Poole, Jr., and Paul K. Stockmeyer. A comparison of several bandwidth and profile reduction algorithms. *ACM Trans. Math. Softw.*, 2:322–330, December

1976.

[41] Michael Goesele, Jens Ackermann, Simon Fuhrmann, Carsten Haubold, Ronny Klowsky, Drew Steedly, and Richard Szeliski. Ambient point clouds for view interpolation. In *SIGGRAPH 2010*.

[42] Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, and Steven M. Seitz. Multi-view stereo for community photo collections. In *Proceedings of the 11th International Conference on Computer Vision (ICCV 2007)*, pages 265–270, Rio de Janeiro, Brazil, 2007. IEEE.

[43] Peng Guan, Alexander Weiss, Alexandru O. Balan, and Michael J. Black. Estimating human shape and pose from a single image. In *ICCV*, pages 1381–1388, 2009.

[44] Sudipto Guha and Samir Khullar. Connected facility location problems. In *DIMACS Workshop on Network Design*, 97.

[45] Derek Hoiem, Alexei A. Efros, and Martial Hebert. Automatic photo pop-up. SIGGRAPH '05, pages 577–584, New York, NY, USA, 2005. ACM.

[46] Derek Hoiem, Alexei A. Efros, and Martial Hebert. Geometric context from a single image. In *ICCV*, pages 654–661, 2005.

[47] B. K.P. Horn. Shape from shading: A method for obtaining the shape of a smooth opaque object from one view. Technical report, Cambridge, MA, USA, 1970.

[48] Peter J. Huber. Robust estimation of a location parameter. In *Annals of Statistics 53*, pages 73–101, 1964.

[49] Yekeun Jeong, David Nistér, Drew Steedly, Richard Szeliski, and In-So Kweon. Pushing the envelope of modern methods for bundle adjustment. In *CVPR*, pages 1474–1481, 2010.

[50] Yong-Dian Jian, Doru C. Balcan, and Frank Dellaert. Generalized subgraph preconditioners for large-scale bundle adjustment. In *ICCV*, 2011.

[51] Ryan Kaminsky, Noah Snavely, Steven M. Seitz, and Richard Szeliski. Alignment of 3d point clouds to overhead images. In *Workshop on Internet Vision, CVPR'09*.

[52] Takeo Kanade and M. Okutomi. A stereo matching algorithm with an adaptive window: theory and experiment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):920 – 932, September 1994.

[53] Kevin Karsch, Ce Liu, and Sing Bing Kang. Depth Extraction from Video Using Non-

parametric Sampling. In *IEEE ECCV*, 2012.

[54] M. Kazhdan, M. Bolitho, and H. Hoppe. Poisson surface reconstruction. In *Symposium on Computer Graphics Conference Proceedings*, 2006.

[55] J. J. Koenderink, A.J.V. Doorn, and A.M.L. Kappers. On so-called paradoxical monocular stereoscopy. In *Pereption Vol 23*, pages 583–594, 1994.

[56] Vladimir Kolmogorov. Convergent tree-reweighted message passing for energy minimization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(10):1568–1583, October 2006.

[57] J. Kosecká and W. Zhang. Video compass. ECCV, pages 476–490, 2002.

[58] P. Kovesi. Shapelets correlated with surface normals produce surfaces. ICCV, pages 994–1001.

[59] Avanish Kushal and Sameer Agarwal. Visibility based preconditioning for bundle adjustment. In *CVPR*, pages 1442–1449. IEEE.

[60] Avanish Kushal and Steven M. Seitz. Single view reconstruction of piecewise swept surfaces. In *3DV*, 2013.

[61] Avanish Kushal, Ben Self, Yasutaka Furukawa, David Gallup, Carlos Hernandez, Brian Curless, and Steven M. Seitz. Photo tours. *2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*, 0:57–64, 2012.

[62] D.C. Lee, M. Herbert, and T. Kanade. Geometric reasoning for single image structure recovery. CVPR, pages 2136–2143, 2009.

[63] Na Li and Yousef Saad. MIQR: A multilevel incomplete qr preconditioner for large sparse least-squares problems. *SIMAX*, 28:524–550, 2006.

[64] X. Li, C. Wu, C. Zach, S. Lazebnik, and J.M. Frahm. Modeling and recognition of landmark image collections using iconic scene graphs. *ECCV (1)*, pages 427–440, 2008.

[65] Stephen Lin, Yuanzhen Li, Sing Bing Kang, and Xin Tong. Diffuse-specular separation and depth recovery from image sequences. In *In Proceedings of European Conference on Computer Vision (ECCV*, pages 210–224, 2002.

[66] M.I. A. Lourakis and A.A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *TOMS*, 36(1):1–30, 2009.

[67] M.I. A. Lourakis and A.A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Trans. Math. Software*, 36(1):1–30, 2009.

[68] M.I.A. Lourakis and A.A. Argyros. Is Levenberg-Marquardt the most efficient optimization algorithm for implementing bundle adjustment. In *ICCV*, pages 1526–1531, 2005.

[69] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

[70] J. Malik and R. Rosenholtz. Computing local surface orienientation and shape from texture from curved surfaces. In *Internation Journal of Computer Vision 23(2)*, pages 149–168, 1997.

[71] J. Mandel. On block diagonal and Schur complement preconditioning. *Numer. Math.*, 58(1):79–93, 1990.

[72] N. J. Mitra and M. Pauly. Symmetry for architectural design. In *Advances in Architectural Geometry*, pages 13–16, 2008.

[73] S.G. Nash and A. Sofer. Assessing a search direction within a truncated-newton method. *Operations Research Letters*, 9(4):219–221, 1990.

[74] Diego Nehab, Szymon Rusinkiewicz, James Davis, and Ravi Ramamoorthi. Efficiently combining positions and normals for precise 3d geometry. *ACM Transactions on Graphics (Proc. SIGGRAPH*, 24:536–543, 2005.

[75] Richard A. Newcombe, Steven Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. In *ICCV*, pages 2320–2327, 2011.

[76] K. Ni, D. Steedly, and F. Dellaert. Out-of-core bundle adjustment for large-scale 3d reconstruction. In *ICCV*, 2007.

[77] J. Nocedal and S.J. Wright. *Numerical optimization*. Springer, 2000.

[78] Emmanuel Prados and Olivier D. Faugeras. Perspective shape from shading and viscosity solutions. ICCV, pages 826–831, 2003.

[79] Rene Ranftl, Stefan Gehrig, Thomas Pock, and Horst Bischof. Pushing the limits of stereo using variational stereo estimation. In *Intelligent Vehicles Symposium*, pages 401–407, 2012.

[80] B. C. Russell, R. Martin-Brualla, D. J. Butler, S. M. Seitz, and L. Zettlemoyer. 3D Wikipedia: Using online text to automatically label and navigate reconstructed geometry. *ACM Transactions on Graphics (SIGGRAPH Asia 2013)*, 32(6), November 2013.

[81] Y. Saad. *Iterative methods for sparse linear systems*. SIAM, 2003.

[82] Ashutosh Saxena, Sung H. Chung, and Andrew Y. Ng. Learning depth from single monocular images. In *In NIPS 18*. MIT Press, 2005.

[83] Ashutosh Saxena, Min Sun, and Andrew Y. Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE Trans. PAMI 31*, pages 824–840, 2009.

[84] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47:7–42, 2001.

[85] Steven M. Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, CVPR '06, pages 519–528, Washington, DC, USA, 2006. IEEE Computer Society.

[86] Qi Shan, Riley Adams, Brian Curless, Yasutaka Furukawa, and Steven M. Seitz‘. The visual turing test for scene reconstruction. In *3DV*, 2013.

[87] Qi Shan, Brian Curless, Yasutaka Furukawa, YCarloz Hernandez, and Steven M. Seitz. Photo uncrop. In *Proceedings of the 13th European Conference on Computer Vision*, ECCV, 2014.

[88] Ian Simon, Noah Snavely, and Steven M. Seitz. Scene summarization for online image collections. In *ICCV*, pages 1–8, 2007.

[89] Sudipta N. Sinha, Drew Steedly, and Richard Szeliski. Piecewise planar stereo for image-based rendering. ICCV, pages 1881–1888, 2009.

[90] N. Snavely, S. M. Seitz, and R. Szeliski. Skeletal graphs for efficient structure from motion. In *CVPR*, pages 1–8, 2008.

[91] Noah Snavely, Rahul Garg, Steven M. Seitz, and Richard Szeliski. Finding paths through the world's photos. In *SIGGRAPH 2008*, pages 15:1–15:11, New York, NY, USA, 2008.

[92] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Photo tourism: Exploring photo collections in 3d. In *SIGGRAPH 2006*.

[93] Noah Snavely, Steven M. Seitz, and Richard Szeliski. Skeletal graphs for efficient structure from motion. In *CVPR*, 2008.

[94] D. Steedly and I. Essa. Propagation of innovative information in non-linear least-squares structure from motion. In *ICCV*, pages 223–229, 2001.

[95] D. Steedly, I. Essa, and F. Dellaert. Spectral partitioning for structure from motion. In *ICCV*, pages 996–103, 2003.

[96] P. Sturm and S. J. Maybank. A method for interactive 3d reconstruction of piecewise planar objects from single images. BMVC, pages 119–126, 1999.

[97] Jian Sun, Nan-Ning Zheng, and Heung-Yeung Shum. Stereo matching using belief propagation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(7):787–800, July 2003.

[98] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Aseem Agarwala, and Carsten Rother. A comparative study of energy minimization methods for markov random fields. In *In ECCV*, pages 16–29, 2006.

[99] Marshall F. Tappen and William T. Freeman. Comparison of graph cuts with belief propagation for stereo, using identical mrf parameters. ICCV, Washington, DC, USA, 2003. IEEE Computer Society.

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

[101] B. Triggs, P.F. McLauchlan, Hartley R.I, and A.W. Fitzgibbon. Bundle Adjustment - A modern synthesis. In *Vision Algorithms'99*, pages 298–372, 1999.

[102] Tinne Tuytelaars and Krystian Mikolajczyk. *Local Invariant Feature Detectors: A Survey*. Now Publishers Inc., Hanover, MA, USA, 2008.

[103] F. Ulupinar and R. Nevetia. Perception of 3d surfaces from 2d contours. In *PAMI vol. 15*, pages 3–18, 1993.

[104] Stephan Winter. Modeling costs of turns in route planning. In *GeoInformatica*, volume 6, pages 363–380, 2002.

[105] Oliver J. Woodford, Philip H. S. Torr, Ian D. Reid, and Andrew W. Fitzgibbon. Global stereo reconstruction under second-order smoothness priors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 31(12):2115–2128, 2009.

[106] S. J. Wright and J. N. Holt. An inexact Levenberg-Marquardt method for large sparse nonlinear least squares. *J. Austral. Math. Soc. Ser. B*, 26(4):387–403, 1985.

[107] C. Wu, S. Agarwal, B. Curless, and S.M. Seitz. Multicore bundle adjustment. In *CVPR*, pages 3057–3064, 2011.

[108] Changchang Wu. Visualsfm: A visual structure from motion system,`http://ccwu.me/vsfm/`. 2011.

[109] Changchang Wu, Sameer Agarwal, Brian Curless, and Steven M. Seitz. Schematic surface reconstruction. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2012.

[110] Q. Yang. A non-local cost aggregation method for stereo matching. In *CVPR*, pages 1402–

1409, 2012.

[111] Q. Yang, L. Wang, R. Yang, H. Stewenius, and D. Nister. Stereo matching with color-weighted correlation, hierachical belief propagation and occlusion handling. *PAMI*, 31(3):492–504, 2009.

[112] Christopher Zach, David Gallup, Jan-Michael Frahm, and Marc Niethammer. Fast global labeling for real-time stereo using multiple plane sweeps. In Oliver Deussen, Daniel A. Keim, and Dietmar Saupe, editors, *VMV*, pages 243–252. Aka GmbH, 2008.

[113] Lukas Zebedin, Joachim Bauer, Konrad Karner, and Horst Bischof. Fusion of feature- and area based information fro urban buildings modelling from areil imagery. In *ECCV 2008*, 2008.

[114] Ruo Zhang, Ping-Sing Tsai, James Edwin Cryer, and Mubarak Shah. Shape from shading: A survey. *PAMI*, 21(8):690–706, 1999.

[115] Ke Colin Zheng, Alex Colburn, Aseem Agarwala, Maneesh Agrawala, David Salesin, Brian Curless, and Michael F. Cohen. Parallax photography: creating 3d cinematic effects from stills. In *Graphics Interface 09*.

[116] Qian Zheng, Andrei Sharf, Guowei Wan, Yangyan Li, Niloy J. Mitra, Baoquan Chen, and Daniel Cohen-Or. Non-local scan consolidation for 3d urban scene. *SIGGRAPH 2010)*, 29(3):Article 94, 2010.

Appendix A

PROOF OF LEMMA 1

Lemma 2. *Let A be a symmetric positive semidefinite(PSD) matrix, then the tridiagonal matrix $M(\nu)$*

$$m_{ij}(\nu) = \begin{cases} a_{ij} & i = j \\ \nu a_{ij} & |i-j| = 1 \\ 0 & \textit{otherwise} \end{cases} \tag{A.1}$$

is positive semidefinite for $\nu = 0.5$ and for every $\epsilon > 0$, there exists a positive semidefinite matrix A, such that $M(0.5+\epsilon)$ is indefinite.

Proof. Since A is symmetric PSD, there exists a matrix U such that $A = U^\top U$. Let the column vectors of $U = [u_1, u_2 ... \cdots u_n]$. Consider $M(0.5)$ .Then for any x,

$$\begin{aligned} x^\top M x =& \sum_{i=1}^{n} x_i M_{i,i} x_i + 2\sum_{i=1}^{n-1} x_i M_{i,i+1} x_{i+1} \\ =& \sum_{i=1}^{n} x_i A_{i,i} x_i + \sum_{i=1}^{n-1} x_i A_{i,i+1} x_{i+1} \\ =& \sum_{i=1}^{n} x_i^\top u_i^\top u_i x_i + \sum_{i=1}^{n-1} x_i^\top u_i^\top u_{i+1} x_{i+1} \\ =& \frac{1}{2}\sum_{i=1}^{n-1} \left(x_i u_i + x_{i+1} u_{i+1}\right)^\top \left(x_i u_i + x_{i+1} u_{i+1}\right) + \frac{1}{2} x_1^\top u_1^\top u_1 x_1 + \frac{1}{2} x_n^\top u_n^\top u_n x_n \\ \geq& 0 \end{aligned}$$

Next we show that this is the best static scaling strategy, i.e. for $\nu > 0.5$, we can find a PSD matrix A, s.t. $M(\nu)$ is indefinite. Choose an integer $n > \frac{1}{2\epsilon} + 1$, and set A to be the square matrix of size n with all entries set to 1. Then clearly $A = \frac{1}{n}(11^\top)$ is PSD. Now consider the matrix $M(0.5+\epsilon)$ that has 1 on the diagonals and $0.5+\epsilon$ on the super and sub diagonal and the vector

$v = [1, -1, 1, -1....]^\top$, then observe that

$$\begin{aligned} v^\top M v &= \sum_{i=1}^{n} v_i^2 + 2\sum_{i=1}^{n-1}(0.5+\epsilon)v_i v_{i+1} \\ &= \sum_{i=1}^{n} 1 - 2\sum_{i=1}^{n-1}(0.5+\epsilon) \\ &= 1 - 2(n-1)\epsilon \\ &< 0 \end{aligned}$$

Thus $\nu = 0.5$ is the largest number that can be used in any static scaling strategy for scaling the sub and super diagonals, and ensuring that the matrix M(ν) still remains PSD. □

Appendix B

COMPARISON FOR LINEAR PROBLEMS

The venice problems are community photo collections, are expected to have a clustered structure and some inter-clustering interaction present. In contrast the ladybug is not a community photo collection dataset, rather it was collected by mounting cameras on a moving vehicle, and thus isn't likely to have significant cluster structure.

Table B.1 lists the problems used for comparing the performance based on iterations, which compared the six preconditioners (including gsp-3) by the number of iterations it took for them to converge.

Ladybug	Venice
problem-162-22824	problem-52-64053
problem-282-37322	problem-89-110973
problem-339-44056	problem-245-198739
problem-384-49181	problem-427-310384
problem-412-52215	

Table B.1: List of Problems for iteration based convergence.

Table B.2 lists the problems used for the experiment to compares the four Schur-based preconditioners on the time it takes for them to solve large linear least squares problems to convergence. It consists of 26 problems each from the Venice and Ladybug datasets.

Figure B.1 reproduces at full scale the performance profiles of the four Schur based preconditioners for varying values of $\tau = 10^{-2}, 10^{-3}$ and 10^{-5}. The first row shows the performance profiles for ladybug, the next for venice, and the third for the combined dataset(all 52 problems together). Note that cluster-tridiagonal outperforms the others in all the charts.

In Figures 2.5–B.5 we show the detailed convergence behavior of the four preconditioners. For each preconditioner, we plot the log relative residual $log(\frac{||Ax_k - b||}{||Ax_0 - b||})$ as a function of time. To be fair to each preconditioner, the time needed to compute the preconditioner is accounted for. Note that Conjugate Gradients algorithm does not reduce the residual monotonically, thus the relative residual plots are oscillatory. As can be seen from the plots for most of the problems, the cluster-tridiagonal performs the best, followed by cluster-jacobi, implicit-ssor and implicit-jacobi in that order.

LADYBUG	VENICE
problem-412-52215	problem-427-310384
problem-460-56811	problem-744-543562
problem-539-65220	problem-951 -708276
problem-598-69218	problem-1102-780462
problem-646-73548	problem-1158-802917
problem-707-78455	problem-1184-816583
problem-783-84444	problem-1238-843534
problem-810-88814	problem-1288-866452
problem-856-93344	problem-1350-894716
problem-885-97473	problem-1408-912229
problem-931-102699	problem-1425-916895
problem-969-105826	problem-1473-930345
problem-1031-110968	problem-1490-935273
problem-1064-113655	problem-1521-939551
problem-1118-118384	problem-1544-942409
problem-1152-122269	problem-1638-976803
problem-1197-126327	problem-1666-983911
problem-1235-129634	problem-1672-986962
problem-1266-132593	problem-1681-983415
problem-1340-137079	problem-1682-983268
problem-1469-145199	problem-1684-983269
problem-1514-147317	problem-1695-984689
problem-1587-150845	problem-1696-984816
problem-1642-153820	problem-1706-985529
problem-1695-155710	problem-1776-993909
problem-1723-156502	problem-1778-993923

Table B.2: List of Problems for time based convergence.

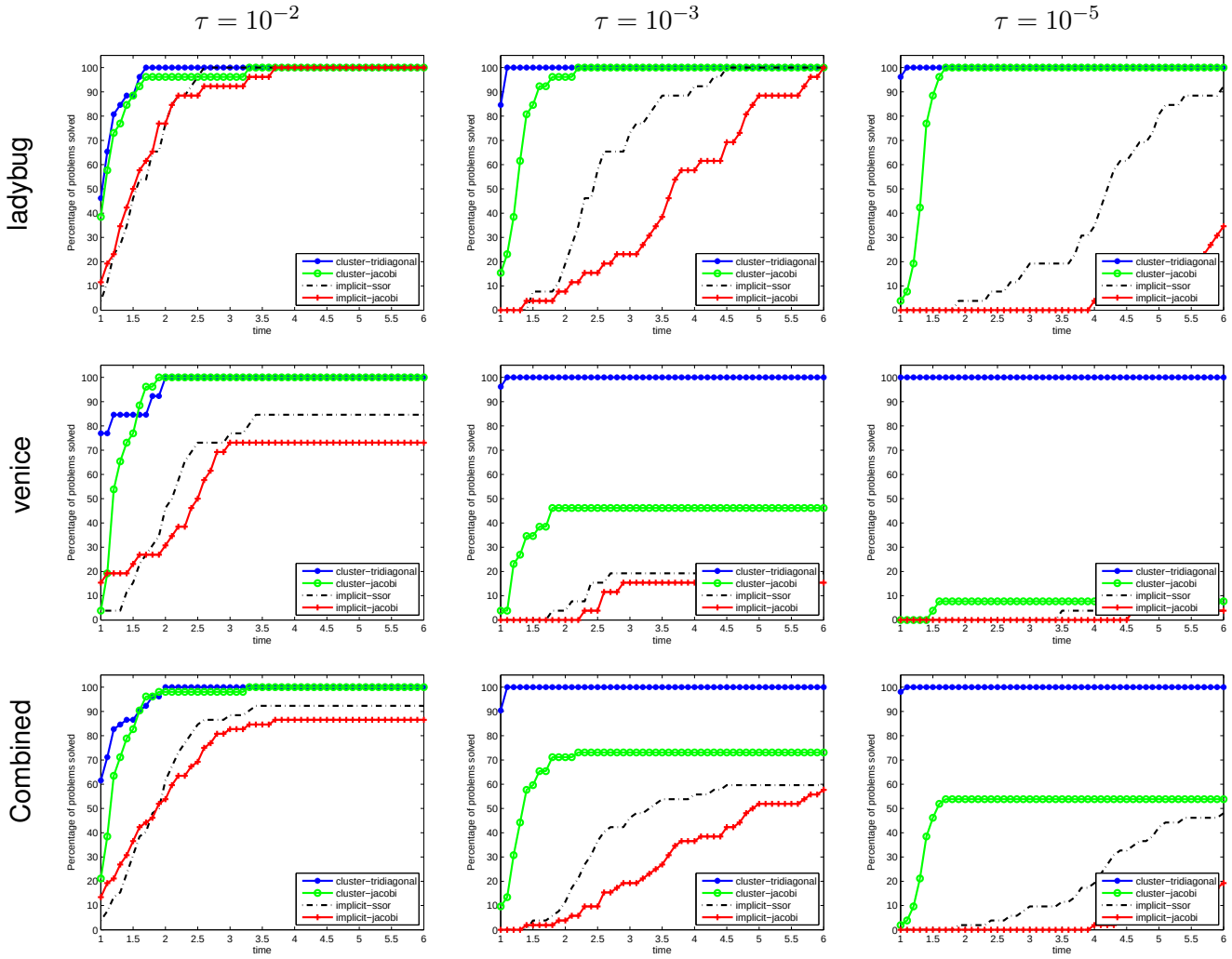


Figure B.1: Time based performance profiles.

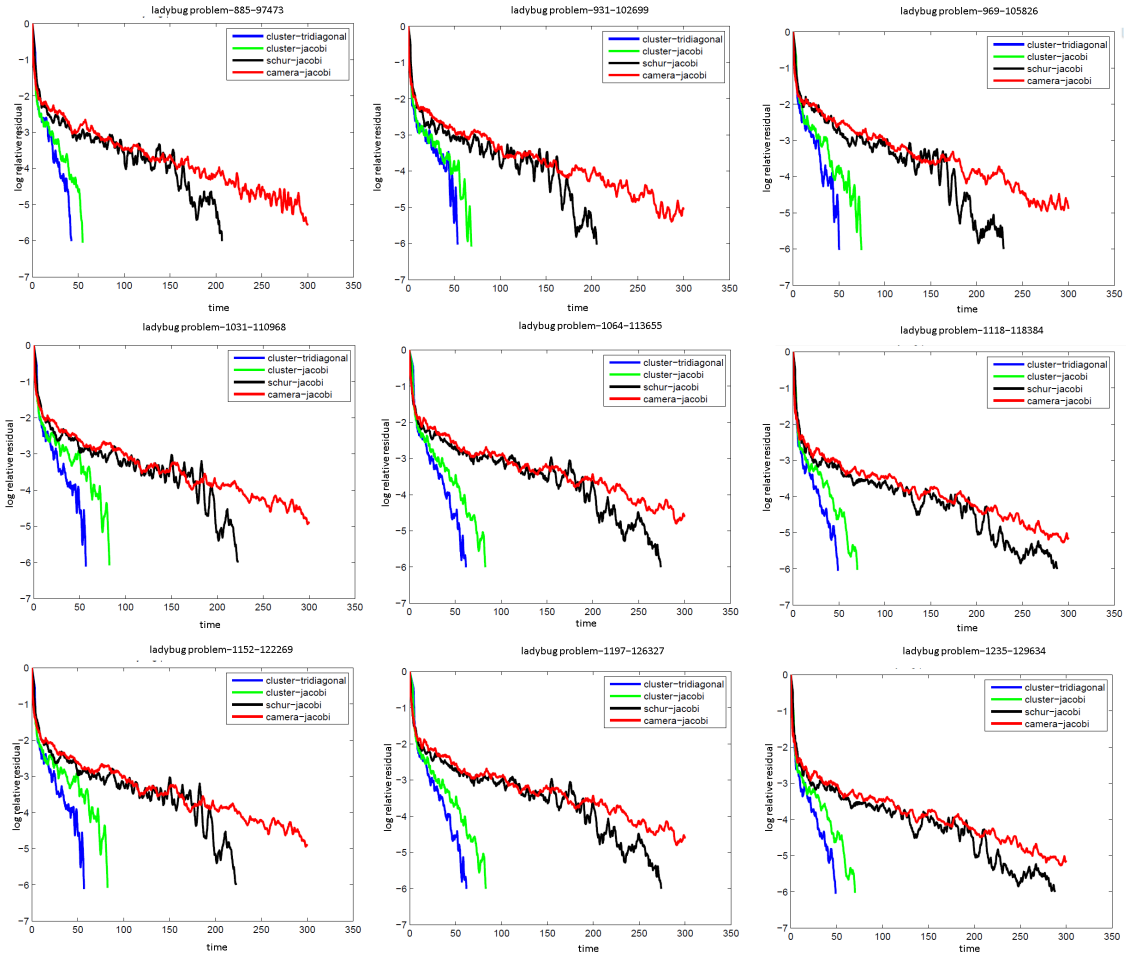


Figure B.2: Convergence Plots for linear problems 10-18 of the ladybug data set, plotting the relative residual error as a function of time.

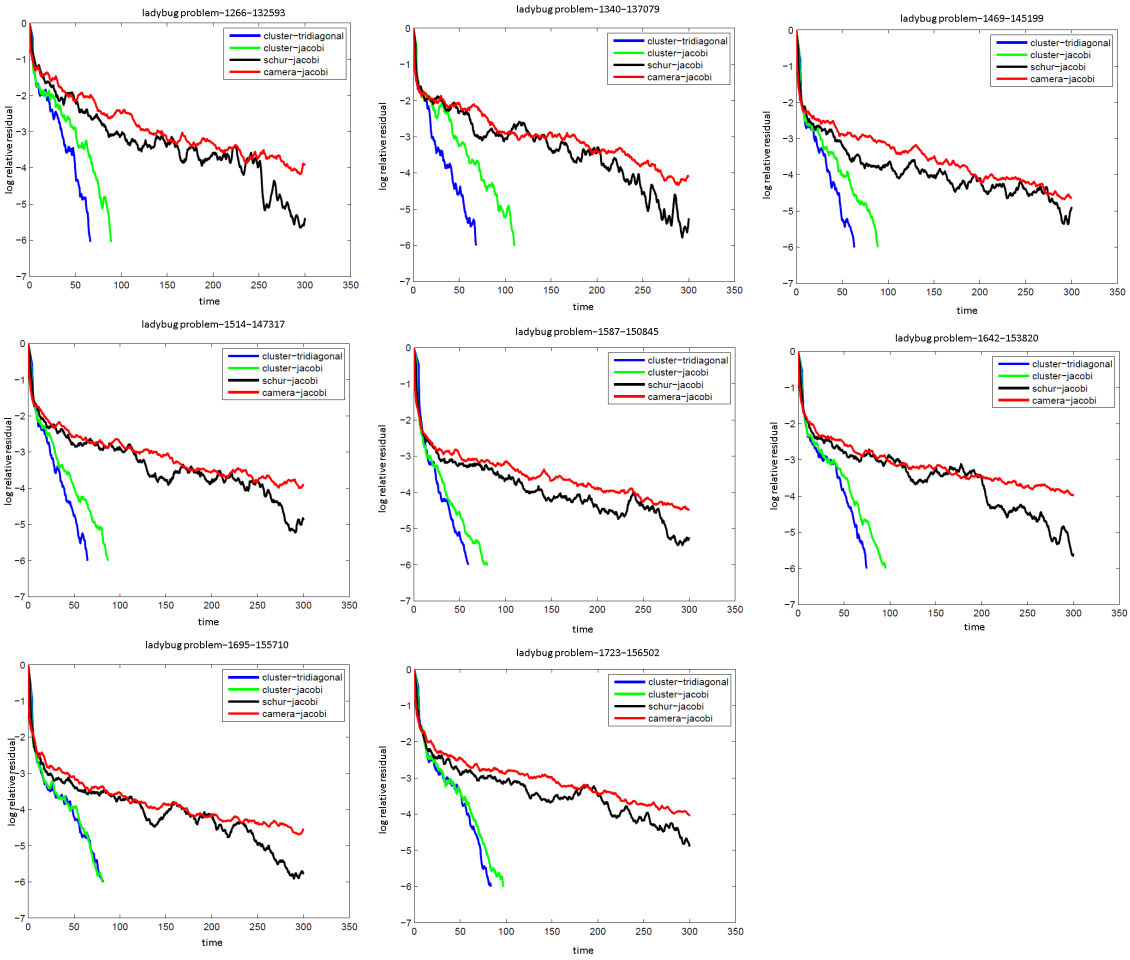


Figure B.3: Convergence Plots for linear problems 19-26 of the ladybug data set, plotting the relative residual error as a function of time.

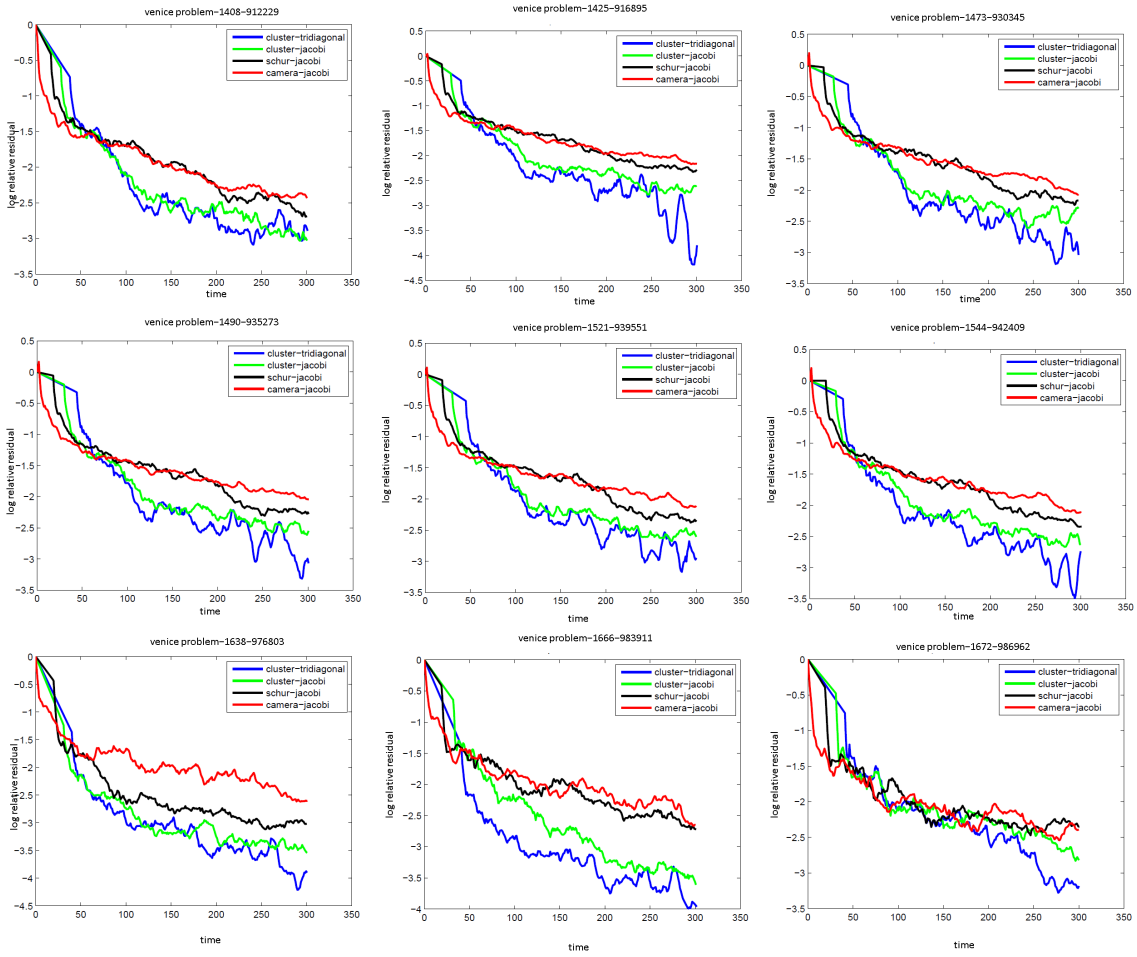


Figure B.4: Convergence Plots for linear problems 10-18 of the venice data set, plotting the relative residual error as a function of time.

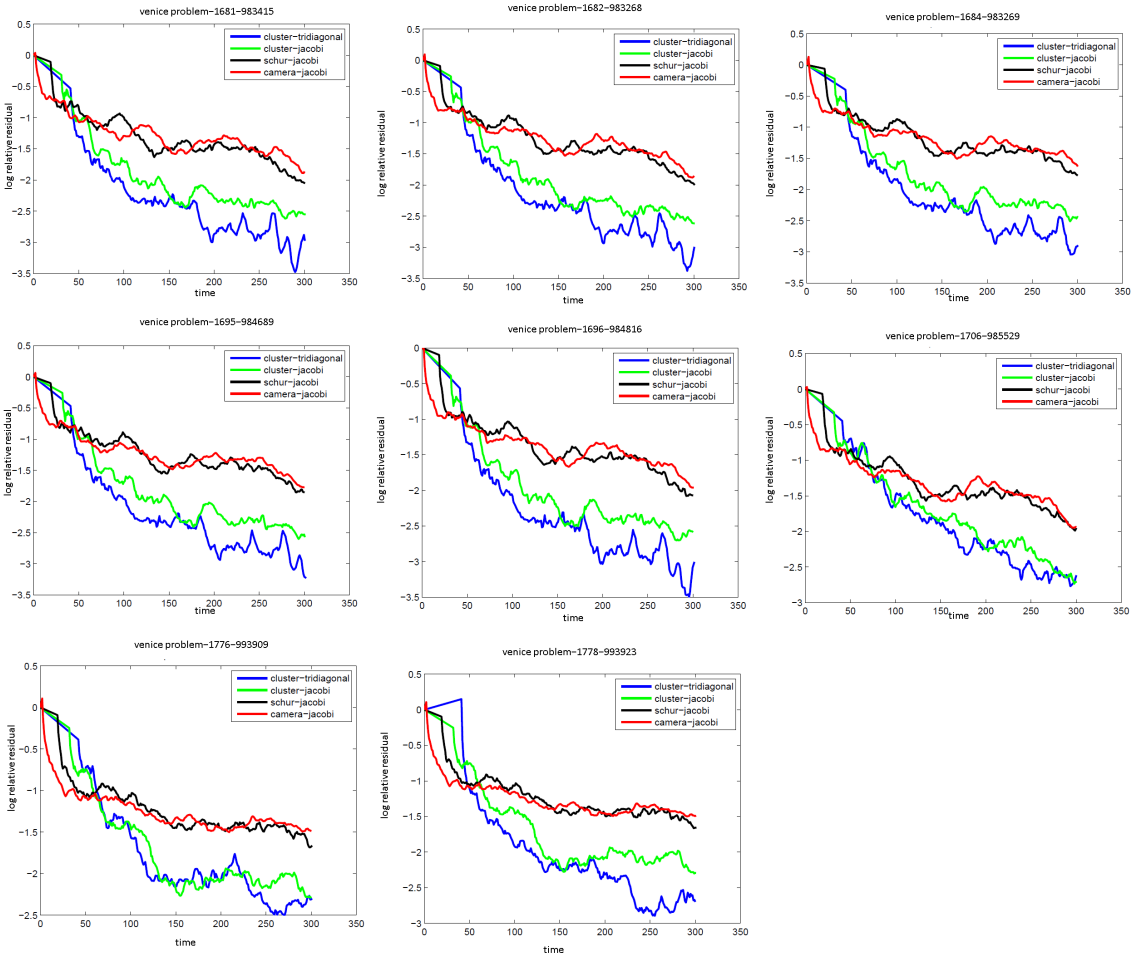


Figure B.5: Convergence Plots for linear Problems 19-26 of the venice data set, plotting the relative residual error as a function of time.

Appendix C

COMPARISON FOR BUNDLE ADJUSTMENT PROBLEMS

We use the problems listed in Table B.2(same as that used for the experiment on the linear problems based on time) to report the performance of the non-linear solvers.

Figure C.1 shows the performance profiles for the four preconditioners and the explicit-sparse solver for $\tau = 10^{-1}$, $\tau = 10^{-2}$, $\tau = 10^{-3}$. Here, the initial setup time to compute the clustering as well as the *degree 2* forest is also taken into account. As can be seen from the plots, for tighter values of τ, cluster-tridiagonal outperforms the rest. Even for $\tau = 10^{-1}$, where the initial setup cost becomes a factor, the preconditioners cluster-tridiagonal, cluster-jacobi catch up quickly with the others as the value of α is increased. The first row shows the performance profiles for ladybug, the next for venice, and the third for the combined dataset(all 52 problems together).

Figures2.8–C.5 show the detailed convergence behavior of Levenberg-Marquardt as a function of the linear solvers used. Again, we plot the log relative error for each problem.

$$e_{k,s} = \log \frac{f_{k,s} - f^*}{f_0 - f^*}, \tag{C.1}$$

Where, f_0 is the initial error, f^* is the lowest error across all solvers and $f_{k,s}$ is the error for solver s at iteration k. The log relative error $e_{k,s}$ is plotted against time. As can be seen, for most of the plots, cluster-tridiagonal and cluster-jacobi compete for the best preconditioner.

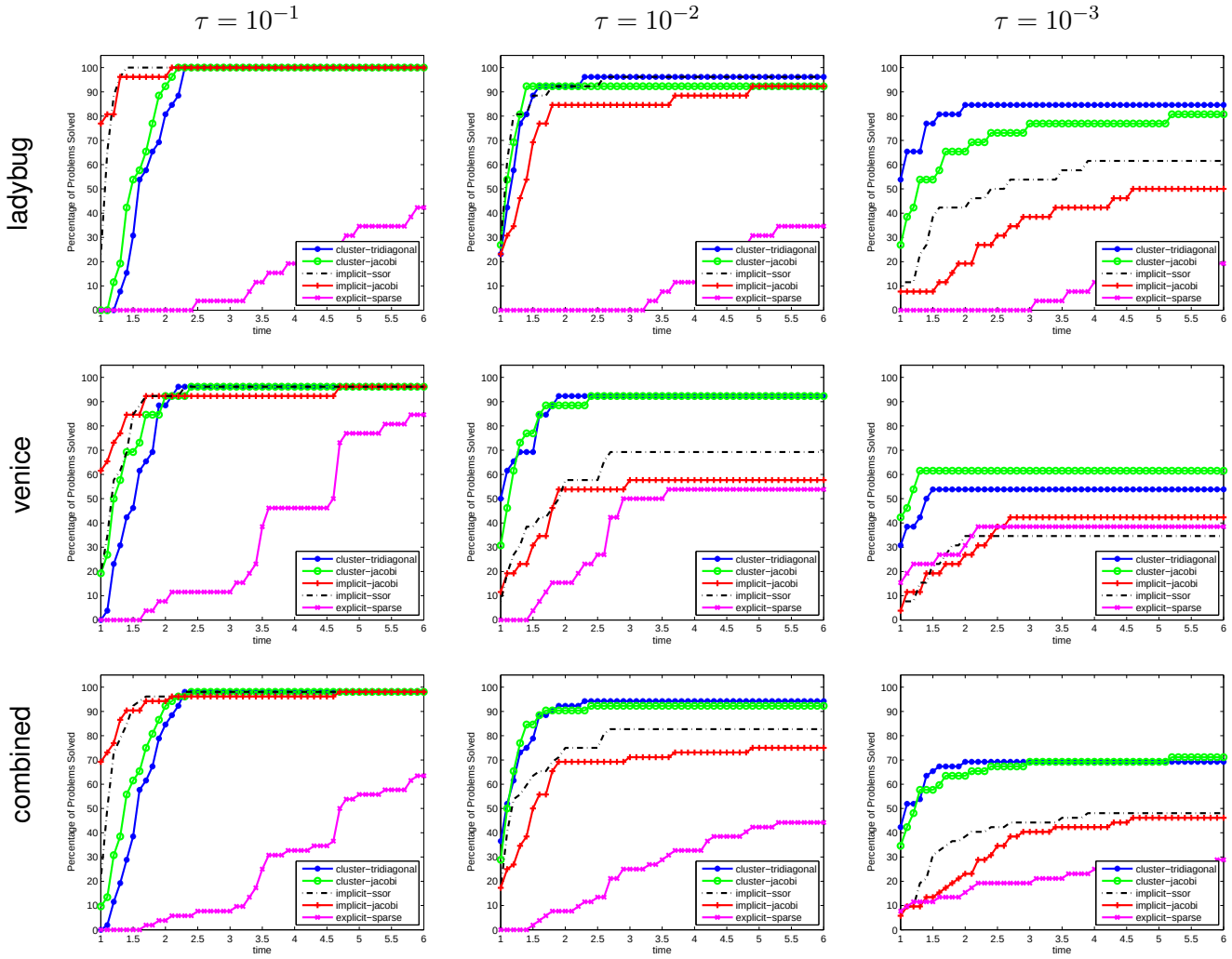


Figure C.1: Performance profiles for the bundle adjustment problems for different data sets.

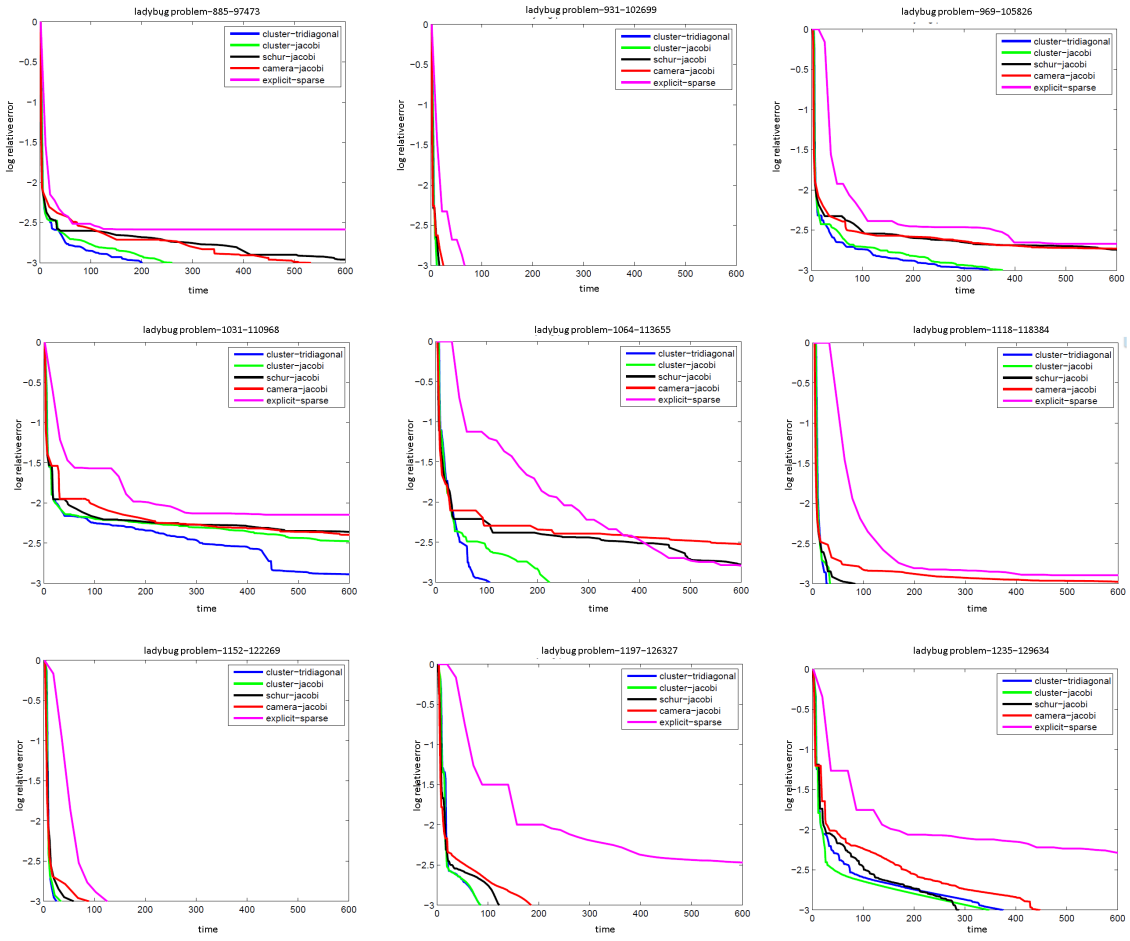


Figure C.2: Convergence Plots for bundle adjustment problems 10-18 of the ladybug data set, plotting the relative residual error as a function of time.

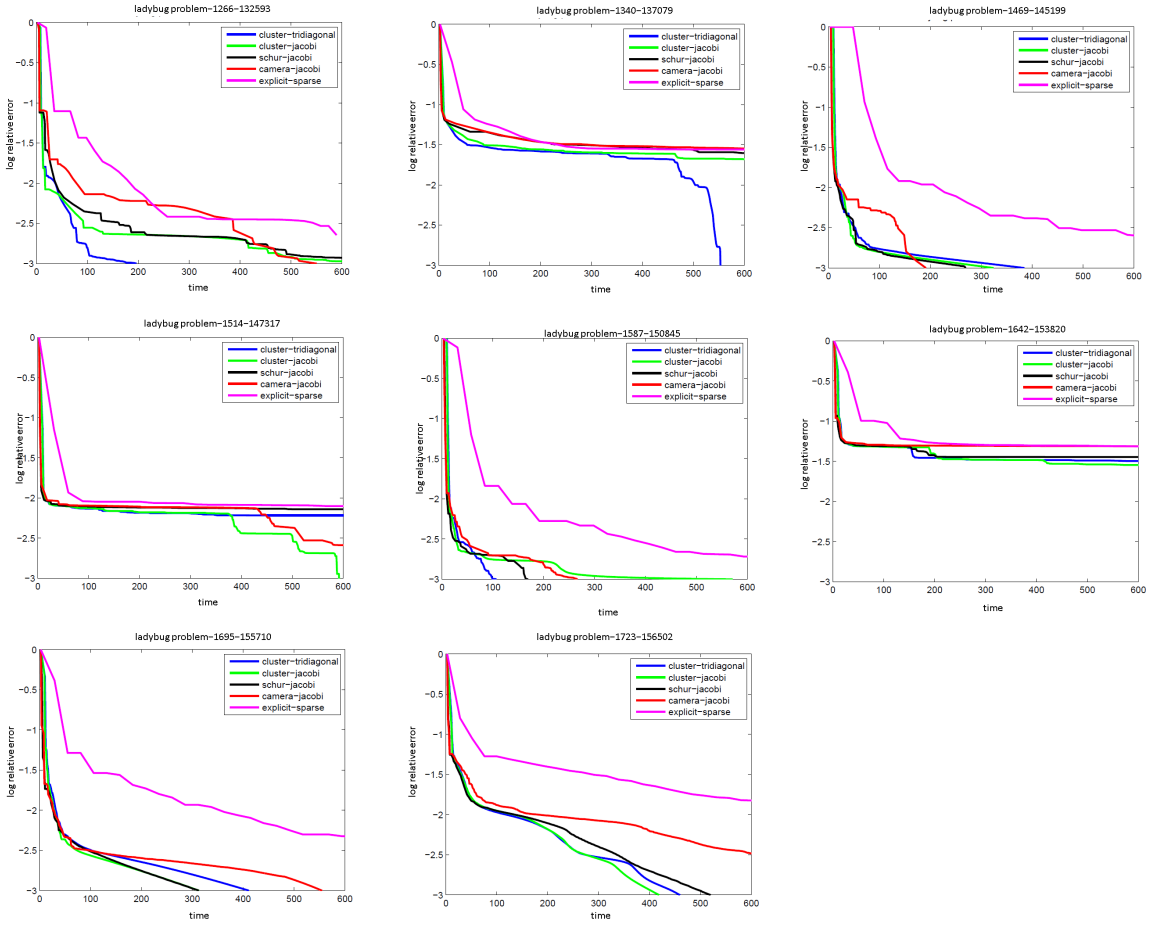


Figure C.3: Convergence Plots for bundle adjustment problems 19-26 of the ladybug data set, plotting the relative residual error as a function of time.

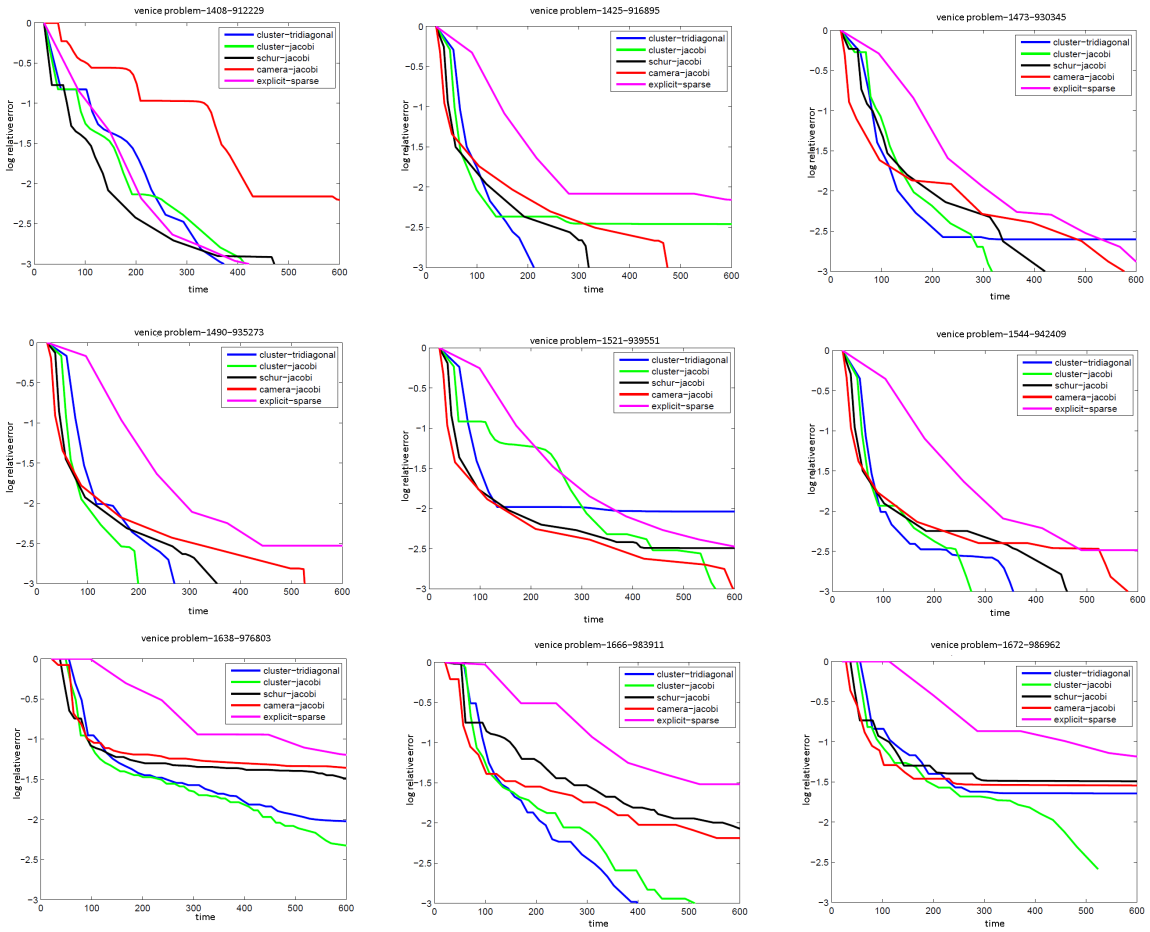


Figure C.4: Convergence Plots for bundle adjustment problems 10-18 of the venice data set, plotting the relative residual error as a function of time.

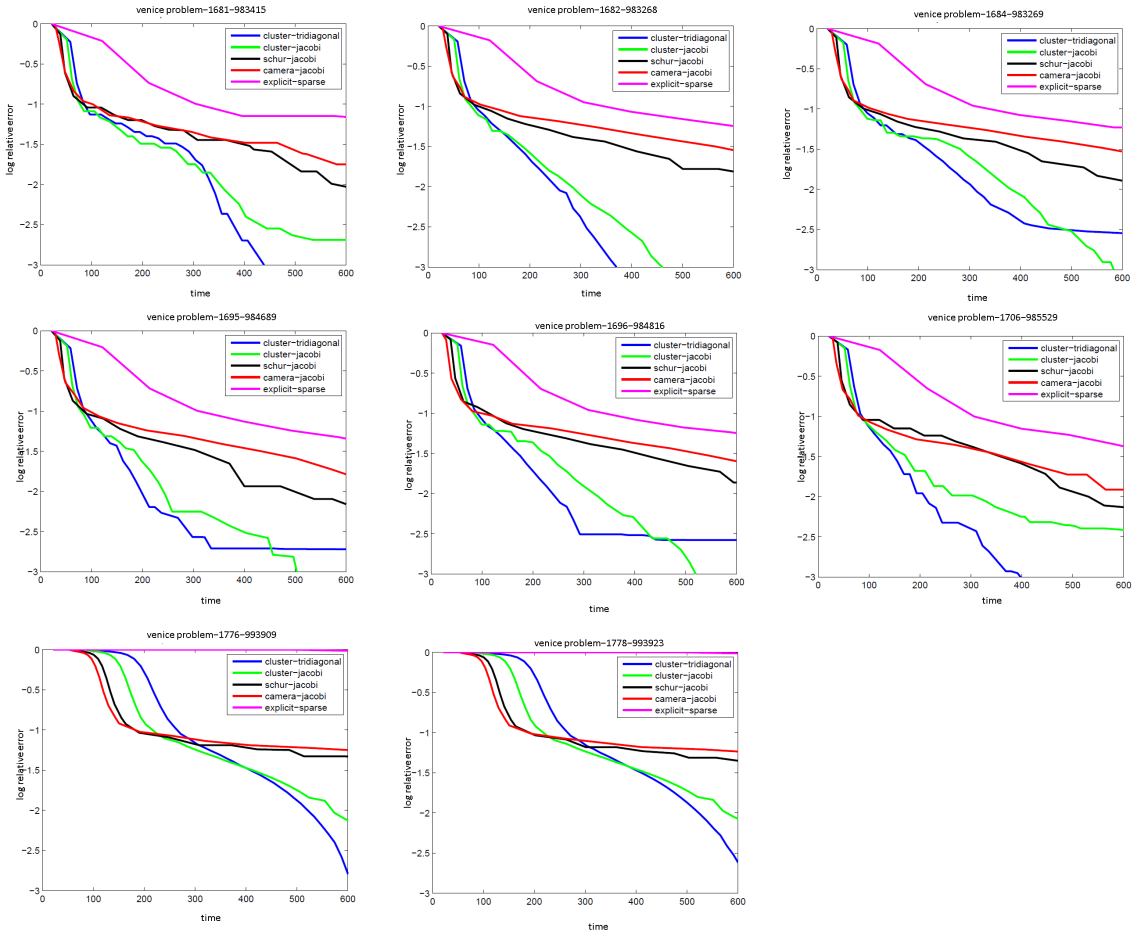


Figure C.5: Convergence Plots for bundle adjustment problems 19-26 of the venice data set, plotting the relative residual error as a function of time.

Appendix D

EXAMPLE PHOTO TOURS

Here, we show the sequence of images in some of the popular photo tours. The actual photo tours are movies which includes fluid 3D transitions between a sequence of photographs of the scene. The canonical images are shown as larger images, and the images that are used to transition between in the path are shown as smaller image. The arrows indicate the order of the images in the tour.

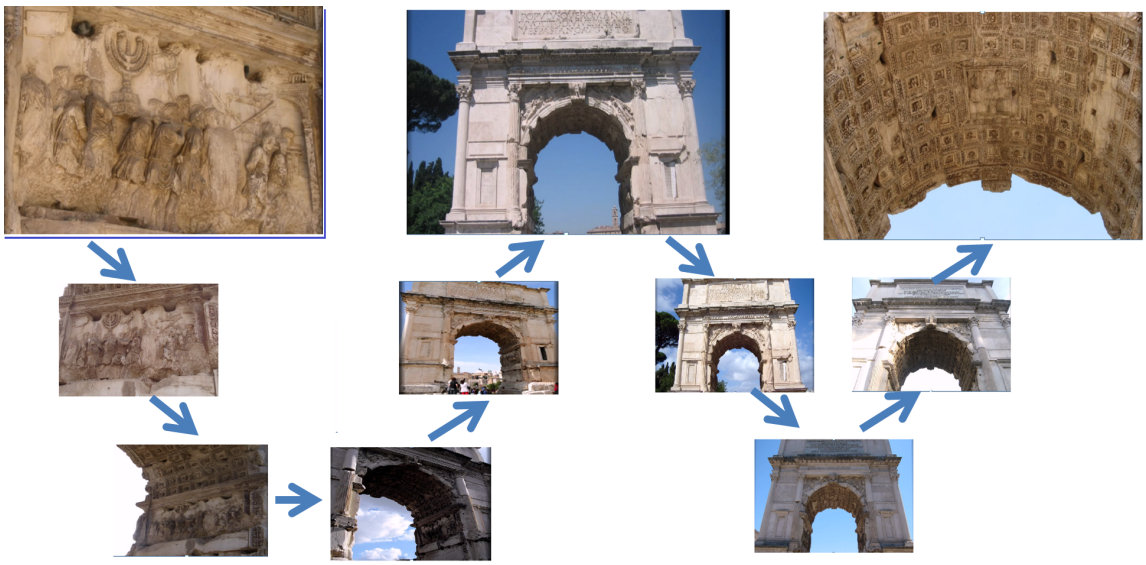


Figure D.1: The figure shows the photo tour for the Arch of Titus in Rome, Italy. The tour starts from the main inscriptions, zooms out to show the full arch with the inscriptions on the top, before zooming into the details of the central coffers.

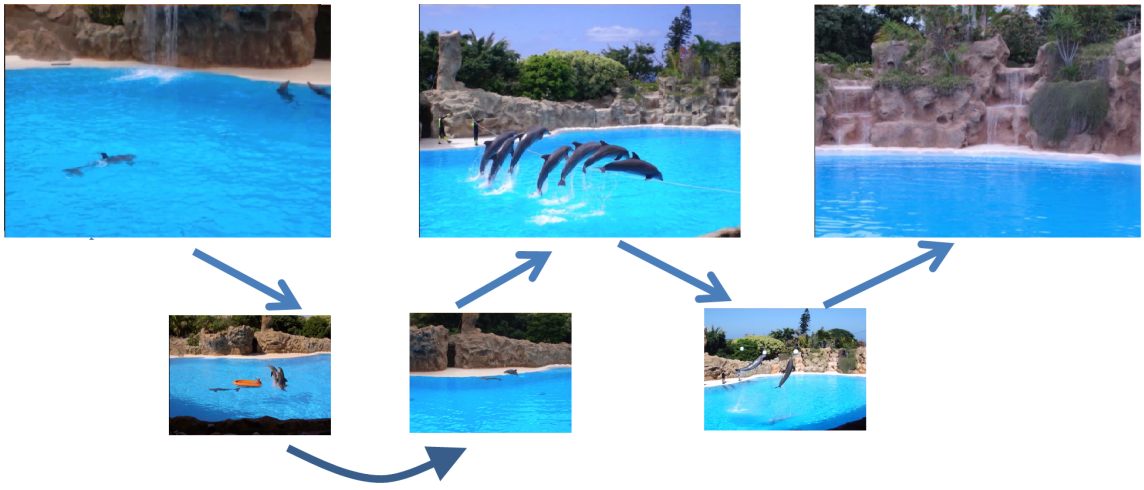


Figure D.2: The figure shows the photo tour for Loro Parque, in Tenerife, Spain. The tour pans across the arena of the dolphin show. The sequence of frames almost gives a temporal feel of the dolphin show.

Figure D.3: The figure shows the photo tour for the central clearing in Hongcun, in China.

Figure D.4: The figure shows the photo tour for Svarifos, in Iceland. The tour starts out from a zoomed out image and moves in a coherent fashion to show the details of the waterfall.

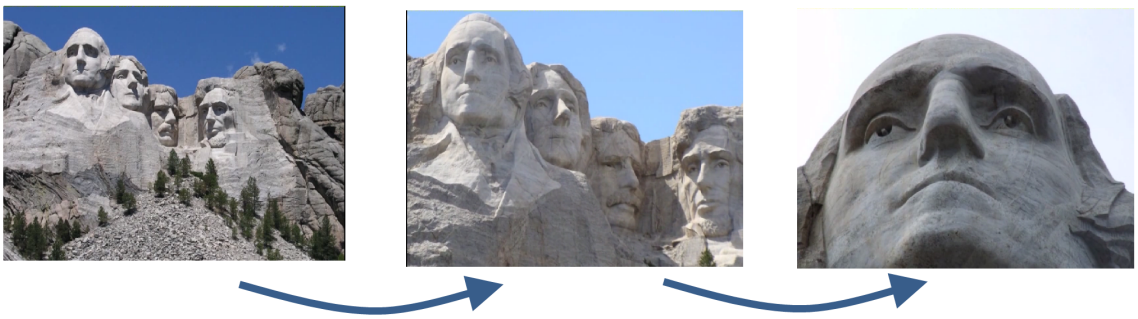


Figure D.5: The figure shows the photo tour for Mt. Rushmore. The tour starts out from a zoomed out image and moves in a coherent fashion to show the details of the carvings.

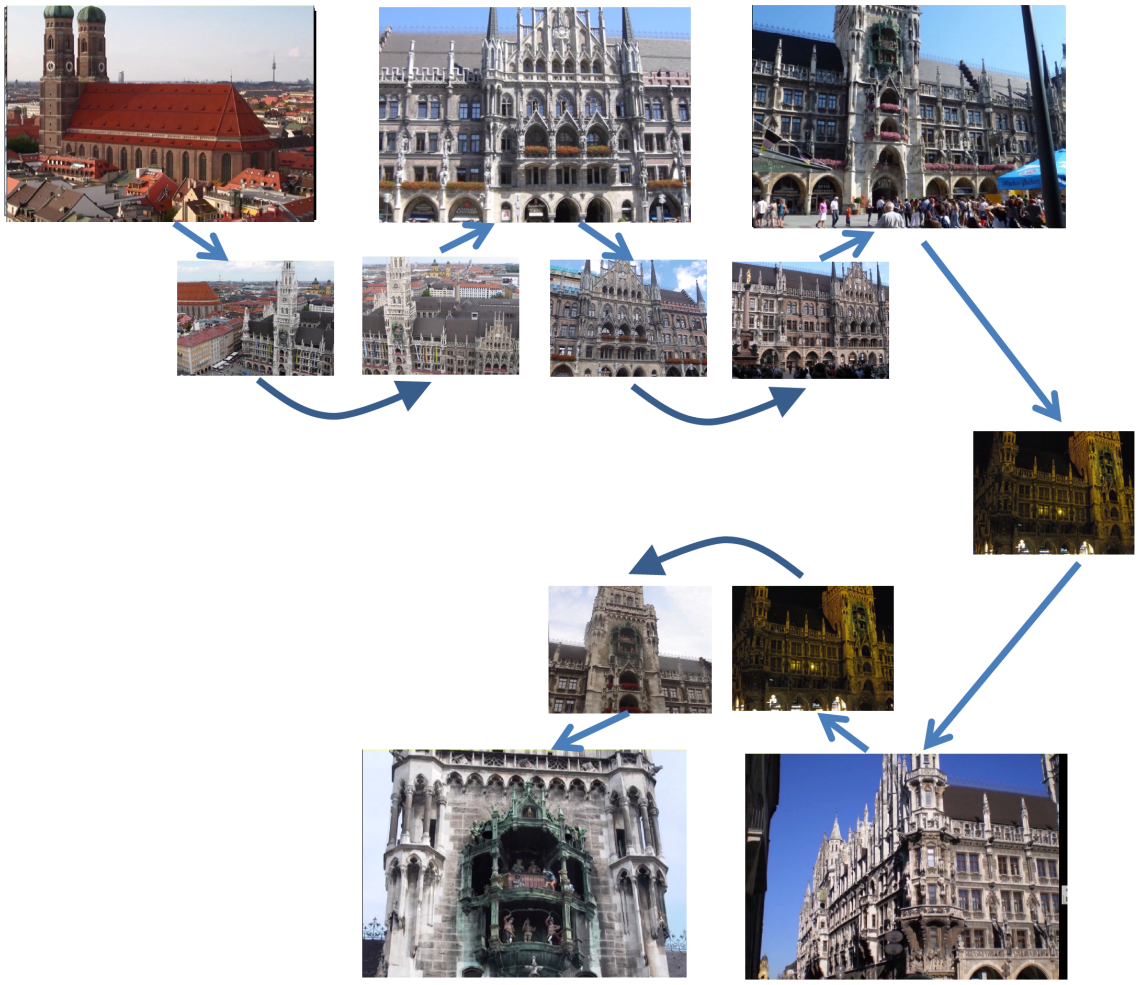


Figure D.6: The figure shows the photo tour for Marian Platz, in Munuch, Germany. The tour starts from a city scale view of the square, before zooming in on the individual attractions.

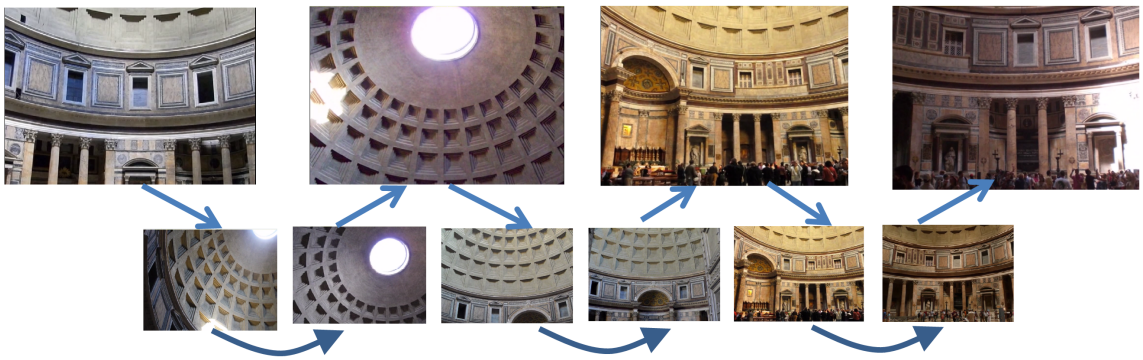


Figure D.7: The figure shows the photo tour for Pantheon, in Rome, Italy. The tour starts at the central dome, before zooming in on the details at the ground level.

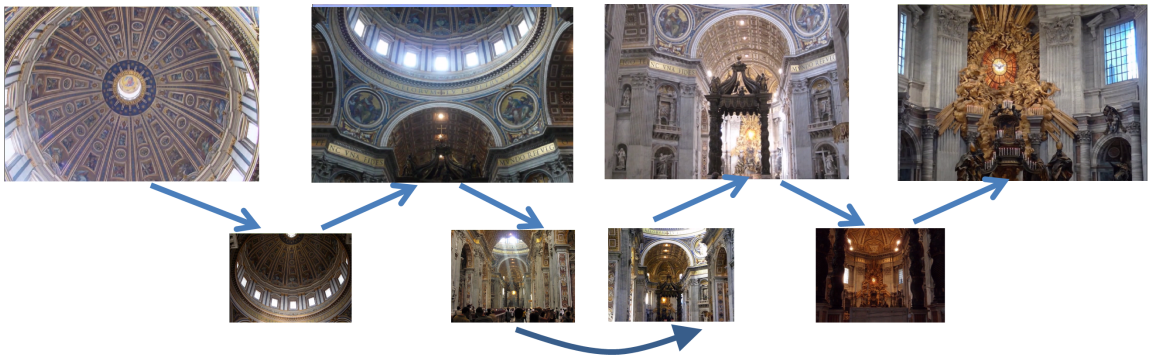


Figure D.8: The figure shows the photo tour for St. Peters, in Rome, Italy. The tour starts at the central dome, before zooming in on the details at the ground level, the altar with Bernini's Baldacchino, and Bernini's Cathedra Petri and Gloria

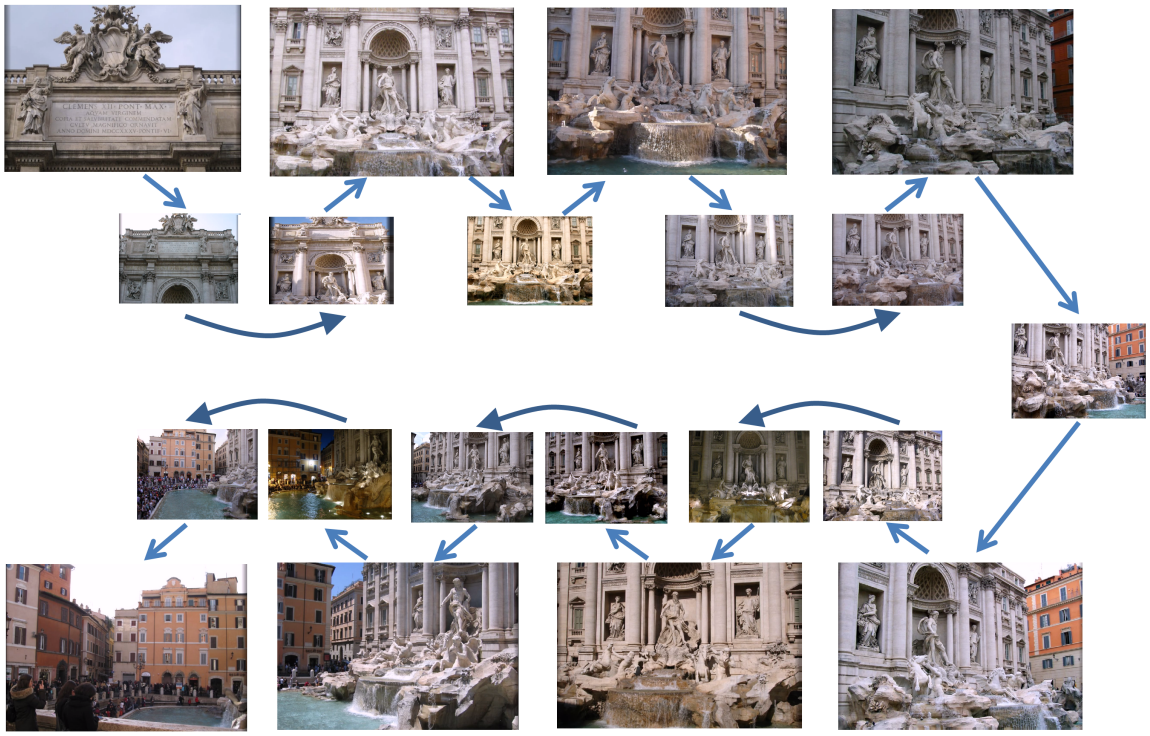


Figure D.9: The tour starts by showing us the inscriptions on top of the Trevi fountain. Then the tour focusses on the central statue, and as it revolves about it capturing it from every angle, it takes us through different appearances — both day and night shots. Finally the tour shows us the surroundings, the people and crowds giving a feel of how it is to be present there.