Pose Imagery and Automated Three-Dimensional Modeling of Urban Environments

by

Satyan R. Coorg

B.Tech., Indian Institute of Technology, Madras (1992)
S.M., Massachusetts Institute of Technology (1994)

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 1998

Author . . . .
Department of Electrical Engineering and Computer Science
August 20, 1998

Certified by . . . .
Seth Teller
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Pose Imagery and Automated Three-Dimensional Modeling of Urban Environments

by

Satyan R. Coorg

Submitted to the Department of Electrical Engineering and Computer Science
on August 20, 1998, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Three-dimensional (3-D) modeling of urban environments has numerous applications, including virtual environments, urban planning, and physical simulation. Constructing 3-D models from photographs (images) is thus an important area of research in computer vision, and increasingly, computer graphics. However, despite many years of research, a system that automatically recovers realistic 3-D models remains elusive; most practical systems require significant human input. Unlike automatic algorithms, human-assisted systems are not scalable, both in terms of the number of images processed and the complexity of the generated 3-D model.

This thesis describes novel techniques to automatically extract textured 3-D models of urban environments from *pose imagery*, i.e., images annotated with camera position and orientation in a single global coordinate system. Physical instruments (e.g., surveying, Global Positioning System (GPS), inertial sensors, etc.) are used to provide accurate initial pose estimates to the proposed algorithms. As these estimates are not perfect, I first describe two optimization techniques that *refine* pose estimates using information present in the images: *spherical mosaicing* recovers relative rotations between images taken from a single position, and *mosaic registration* accurately locates mosaics in a global coordinate system. Next, I describe an algorithm that extracts vertical facades from mosaics annotated with accurate pose. The algorithm employs horizontal line segments to detect likely facade orientations and locates these facades using a space-sweep technique. Textures are robustly computed for the facades by combining information from several mosaics using median statistics.

I present results for a large pose image dataset (consisting of about four thousand images taken from eighty-one positions) of an urban office complex. These techniques were successful in recovering all significant vertical facades in the complex, as well as several neighboring facades.

Thesis Supervisor: Seth Teller
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgements

I first would like to thank Prof. Seth Teller for being an absolutely wonderful thesis supervisor through the entire process. He started me out with great ideas and problems, provided good feedback and suggestions during the research, and made sure that the final output is of high quality. As well as teaching me graphics, he showed me that good research can be fun. I thank him for making my time at MIT a very rewarding experience.

Thanks to Prof. Tomás Lozano-Pérez and to Prof. Leonard McMillan for serving as readers on my committee and providing very useful comments on the thesis draft. Thanks also to Prof. Julie Dorsey for her interest in my research.

I am grateful to Prof. Alan Edelman for clarifying some of the issues in numerical optimization. Also, thanks to Dr. Rick Szeliski (Microsoft) and Dr. Paul Beardsley (MERL) for pointing me to several related pieces of research in computer vision.

I received help from several other members of the City Project during various stages of this work. In particular, thanks to Neel for being the UI and dataset "master" and to Adam Holt for assistance in acquiring the pose image dataset.

The graphics graduate students – especially Becca, Eric A., Eric B., Hans, Justin, Kavita, Mike B., Mike C., – were great companions during the many long hours I spent in the lab.

A special note of thanks to Prof. Arvind, my masters thesis supervisor, for being a constant source of encouragement during my PhD.

I deeply appreciate my parents for patiently supporting me through this long academic journey. Thanks also to the rest of my family who are all excited about "one of them" getting a doctorate. I finally thank Sabitha for all the support she has given me during the final stages of my thesis.

To my parents,

C. S. Ramadas and Malathi Ramadas

Contents

List of Figures

Chapter 1

Introduction

Three-dimensional (3-D) models of urban environments contain descriptions of scene geometry (e.g., locations of buildings) as well as visual information associated with the geometry (e.g., texture). Such 3-D models are used as virtual environments [DTM96, JLF95] in several applications (movies, advertisements, games, ...). They are also useful in urban planning and physical simulation (e.g., fire simulation [BS97]).

The design and display of such 3-D models has thus been an important area of research during the past several decades. While advances in both graphics hardware [AJ88, Ake93, MBDM97, TK96] and software [CT97b, SLS$^+$96, TS91] permit efficient navigation through large models, model construction remains a tedious, manual process that involves significant input from a human designer.

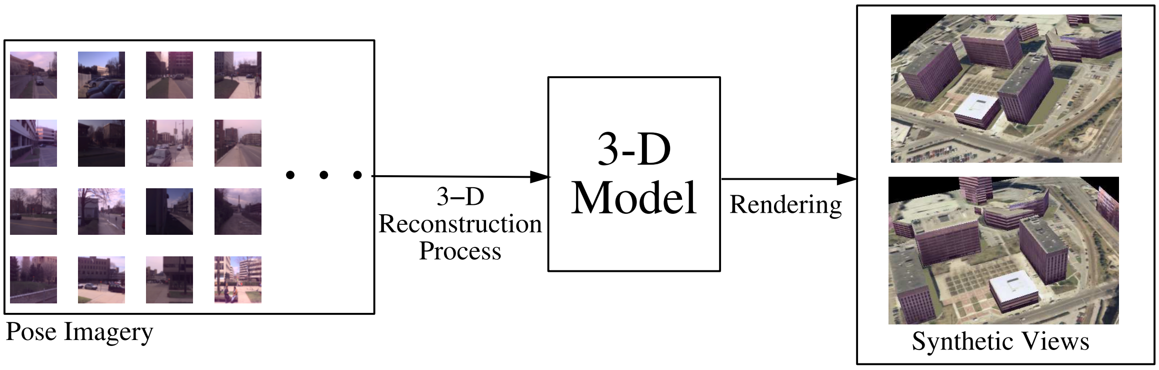


Figure 1-1: From images to 3-D models (and applications).

One method to simplify the modeling of urban environments is to use photographs (images) as input. Figure 1-1 illustrates the desired pipeline, where the process of *3-D reconstruction* is used to extract a 3-D model of an environment from images, and standard computer graphics *rendering* techniques [FvDFH90] are used to produce *synthetic views* from novel viewpoints. This approach offers several advantages. Images are easily acquired with digital cameras and stored inexpensively on computer disk drives; they serve as a reference to the accuracy of the extracted geometry; and they enhance visual realism of the extracted environments. Current approaches to perform 3-D modeling from images include:

- *Interactive modeling systems* [CJS+95, Vex] that allow a human designer to incorporate images in the modeling process. While such systems can generate high quality results [DTM96], the user must process each image and each structure, making them difficult to use for large sets of images or complex scenes.

- *Computer vision techniques* [Fau93, Hor86] that automatically analyze one or more images to generate 3-D information, thus eliminating the "human-in-the-loop". However, they possess several drawbacks for modeling extended outdoor environments. First, they are fragile with respect to common effects such as occlusion (e.g., a building partially obscuring another) and illumination variation (e.g., sunny vs. overcast days). Second, most vision techniques do not scale to very large sets of images, thereby limiting their applicability in complex 3-D modeling situations. Finally, their output (in the form of depth maps or floating edges) do not correspond to 3-D CAD models, which are required for realistic rendering on computer graphics workstations.

The goal of thesis is to *use a large set of images of an urban environment to generate a 3-D model consisting of texture-mapped polygons, involving minimal human-intervention in the reconstruction process.* The techniques proposed in this thesis toward achieving this overall goal are based on the following ideas:

- Instead of analyzing raw images, the proposed algorithms analyze *pose imagery* – images annotated with camera pose (position and orientation in a single global coordinate system). Estimates of pose can be derived from a variety of physical instrumentation such as surveying instruments, Global Positioning System (GPS), Inertial Sensors, etc. [Tel97, DeC98]. The pose estimates supplied are used in a variety of ways in this thesis. For example, they are used to initialize numerical optimizations (spherical mosaicing and global registration). They also permit an efficient organization of the images in a spatial data structure, yielding efficient reconstruction algorithms.

- The algorithms are designed to exploit *geometric constraints* in the extraction process. Such constraints arise both from knowledge of camera poses as well as geometric structure inherent in urban environments. For example, the spherical mosaicing algorithm exploits geometric properties of images taken from a fixed position. Also, the model extraction algorithm extracts *vertical facades* by using constraints imposed by horizontal edges on the facades.

- The algorithms naturally extend to *large image sets*, making them robust with respect to effects of occlusion and illumination variation. Two examples are: a vertical facade orientation detector based on histograms, and a texture generator based on median statistics.

While some of these ideas are partially explored in existing research work (described later), I demonstrate that their combination leads to novel algorithms that robustly extract 3-D models from images.

1.1 Thesis Overview

The rest of this chapter describes the acquisition of a pose image dataset used as input to the algorithms proposed in this thesis. The next chapter discusses previous work in the area of image-based 3-D reconstruction.

Figure 1-2 shows an overview of the novel 3-D modeling pipeline described in the thesis. The first phase addresses the problem of *pose refinement*, i.e., recovering accurate camera pose from estimates supplied by physical instrumentation. Chapter 3

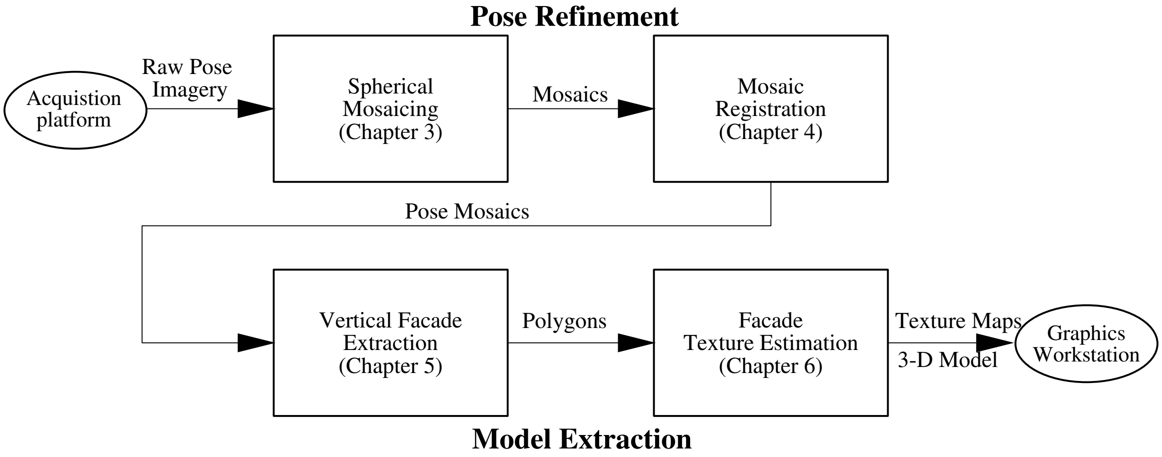


Figure 1-2: Overview of the 3-D modeling pipeline.

describes a *mosaicing* algorithm that computes accurate relative rotations between images taken from a camera with fixed optical center, generating *spherical mosaics*. Chapter 4 describes techniques using point correspondences to accurately locate these mosaics in a global coordinate system, generating *pose mosaics*.

The second phase addresses the problem of *model extraction* from pose mosaics. Chapter 5 describes a vertical facade extraction algorithm based on identifying 3-D horizontal line segments on such facades. Chapter 6 describes a technique that estimates textures for these vertical facades based on median statistics. All the algorithms presented in this thesis have been implemented and tested on a large pose image dataset (consisting of about four thousand images) of a real urban environment.

1.1.1 Definitions of Terms

The following table lists definitions of terms frequently used in this thesis. They are presented roughly in order of appearance.

- **Node**: A set of images taken from the same position, but with different camera orientations. It is also used to refer to a *spherical mosaic* derived from these set of images.

- **Node Pose**: The position and orientation of a reference image (e.g., the first image) of the node.

- **Horizontal Edge**: A building feature (e.g., window boundary) parallel to the ground plane.

- **Vertical Facade**: A vertical plane bounded at the top by a horizontal edge, and at the bottom by the ground plane.

- **Grid**: A two-dimensional subdivision of the 3-D environment being modeled. A **cell** is one unit of this subdivision.

- **Tile**: A piece of a vertical facade contained entirely within a single grid cell.

1.2 Pose Image Dataset

Acquisition of pose imagery used in this thesis was performed in several stages:

- The camera was calibrated so that the mapping from 3-D world space to 2-D image space is completely determined (Section 1.2.1).

- Images were acquired in a special configuration to facilitate image storage and 3-D model extraction (Section 1.2.2).

- Pose estimation was performed using surveying instruments (Section 1.2.3).

- Raw images were processed to detect important features such as edges and corners (Section 1.2.4).

1.2.1 Camera Calibration

Camera calibration involves computing camera parameters such as the focal length, principal point, etc., that determine the mapping from image space to 3-D. For this work, the camera was calibrated with the help of a measurement grid (Figure 1-3) and Tsai's calibration algorithm [LT87, Tsa87]. The input to this algorithm is a list of correspondences that determine the camera mapping. Each correspondence specifies the 3-D location of a point feature (e.g., a corner of one of the black squares in Figure 1-3) and its 2-D location in the image (obtained manually or by using feature

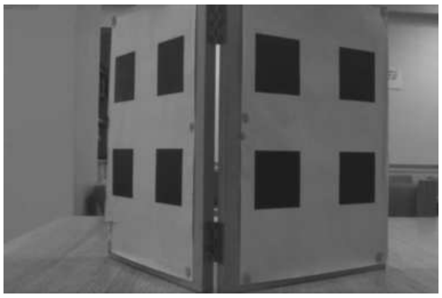


Figure 1-3: The camera calibration grid, courtesy Dr. Paul Beardsley at MERL.

detection). Based on this information, Tsai's algorithm solves for the unknown camera parameters using nonlinear optimization.

While Tsai's calibration algorithm provides useful estimates for the camera parameters, it is possible for mechanical variations (e.g., changes in camera focus) to alter the values slightly. Thus, these parameters are further modified based on image information by the mosaicing optimization algorithm described in Chapter 3.

1.2.2 Raw Image Acquisition

Images were acquired by a Kodak DCS 420 digital camera mounted with fixed optical center on an indexed pan-tilt head, itself attached to a tripod base. The tripod was positioned at eighty-one locations (*nodes*) among the buildings of an office complex. Typical inter-node distance is large (about ten meters) resulting in fairly dissimilar images across nodes due to differences in perspective and occlusion.

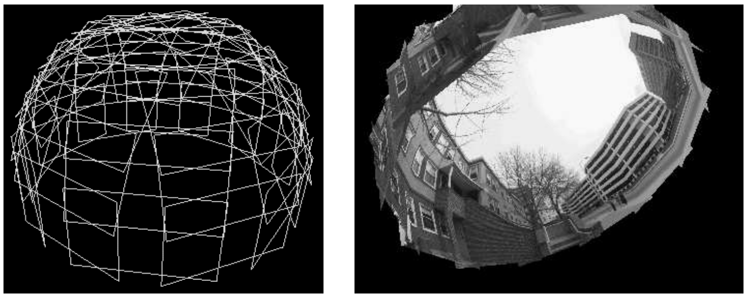


Figure 1-4: The hemispherical tiling comprising a node of the dataset.

At each node, the camera was rotated through a predetermined "tiling" of 50-70 orientations, yielding a roughly hemispherical field of view (Figure 1-4). The wide field of view generated by this tiling aids both pose refinement and extraction algorithms developed in this thesis.

Apart from avoiding inclement weather and darkness, no other restriction (e.g., selection of diffuse lighting conditions [DTM96]) was made on the days/times during which the dataset was acquired. It is thus important for the algorithms to be robust with respect to the effects of illumination variation in the images.

1.2.3 Pose Estimation

Figure 1-5: Orientation of a node is determined by pointing the camera directly at a known position (e.g., another node).

To provide camera pose for each node, initial position estimates were obtained with surveying instruments. Initial orientation estimates for each node were obtained by manual pointing of the pan-tilt head at some other node (marked by a second tripod and a ball; see Figure 1-5). The errors associated with these estimates (about a degree for rotation and a meter for position) are an order of magnitude greater than pixel level accuracy demanded by typical model extraction algorithms; it is therefore crucial to employ the pose refinement algorithms described in this thesis.

1.2.4 Image Features

The raw images acquired by the camera are processed to generate features (edges and corners) for use in model extraction. It is difficult to design a "perfect" feature detector (e.g., one that identifies *all* visible edges of buildings) due to complex lighting and shading effects present in the images. Fortunately, as the proposed algorithms use large sets of images, it is sufficient for the feature detector to perform reasonably well (e.g., detect an interesting feature in a majority of images), permitting the use of the following, relatively straightforward feature detection algorithm.

(a) (b) (c)

Figure 1-6: Part (a) shows an image input to the feature detector. Part (b) shows gradient direction ϕ of each pixel represented as pixel intensity. Part (c) shows Canny edges derived from the image.

In this work, edge pixels are detected using the Canny edge detector [Can86]. The edge detector convolves the image with the derivative of a gaussian filter, and searches for maxima of gradient magnitude (along the gradient direction). Edge line segments are identified by connecting adjacent pixels that possess the same gradient direction [BHR87]. Corners are determined by intersecting adjacent line segments, i.e., whose end points are within a threshold distance (e.g., one pixel). Figure 1-6 shows feature detection performed on a single image. The features detected are stored along with their corresponding images, so that they can be accessed easily by reconstruction algorithms.

1.3 Summary

This thesis describes a novel approach to 3-D modeling based on images annotated with camera pose. The algorithms proposed include pose refinement algorithms that report accurate pose and geometry/texture extraction algorithms that recover 3-D information from pose imagery. The large dataset used in this thesis consists of four thousand calibrated images of an urban office complex annotated with pose obtained by surveying instruments.

Chapter 2

Related Research

In this chapter, I discuss previous research in 3-D model extraction, with emphasis on techniques closely related to those proposed in this thesis. Some of the major areas described are photogrammetry (Section 2.1), interactive modeling systems (Section 2.2), computer vision techniques to recover structure from calibrated (Section 2.3) and uncalibrated (Section 2.4) imagery, and finally, image-based rendering (Section 2.5).

2.1 Photogrammetry

This field originated with the process of measuring topographical geometry using aerial images. A good overview of the major techniques can be found in Wolf [Wol74] and Slama et. al [Sla80]. Classically, methods in photogrammetry involve significant human input, e.g., manual procedures for alignment of aerial photographs. It is only recently that focus has shifted to automated methods [Gre97]. However, most work is still focused on long range aerial images, in contrast to the use of close range ground-based images employed in this thesis.

As in 3-D modeling, photogrammetry also involves determining accurate 3-D positions of relevant features. Thus there is significant emphasis on good camera calibration, for both interior parameters (e.g., camera focal length) and exterior parameters (position and orientation of the cameras). Traditionally, exterior parameters for a camera were estimated based on "tie-points", i.e., points whose 3-D positions are somehow known. Recent methods employ GPS and leveling instrumentation to obtain pose estimates [Gre97], similar to the methods employed in this work.

Computing accurate pose from estimates is also important in photogrammetry, as physical measurements are usually inaccurate. Two different classes of techniques are used in practice. First, a variety of closed-form analytic solutions compute camera poses from a few correspondences (e.g., five point correspondences for relative pose [Hor90]). While such analytic solutions do not require initial estimates, the solutions tend to be numerically unstable and sensitive to data errors. Alternatively, there are "bundle-adjustment" techniques that utilize many correspondences to generate accurate pose by global optimization. In this thesis, the correspondence-based global optimization employs a similar technique for locating spherical mosaics (Chapter 4).

2.2 Interactive Modeling Systems

These systems are widely used to perform 3-D model extraction from images (e.g., commercial systems such as FotoG [Vex] and research projects such as RADIUS [CJS+95]). In such systems, a human designer performs model extraction by identifying features and correspondences across several images. The system computes 3-D information based on user supplied correspondences. While, in theory, such systems can be used to model any environment, they quickly become tedious to use for large image sets.

One way to reduce the extent of user input in these systems is to use geometric constraints to aid the modeling process as in Debevec et al.'s Facade system [DTM96] and Becker's visually assisted modeling system [BB95]. The constraints arise from the rich geometric structure inherent in buildings of typical urban environments. These constraints are usually encoded in the system as high-level modeling primitives such as *blocks* in the Facade system (described below) or *parallel and perpendicular lines* in Becker's system. Recently, similar constraints were used to interactively design 3-D models from *mosaics* [SHS98].

In the Facade system, the user employs blocks (cuboids, pyramids, etc.) to model buildings in an urban environment. As many urban structures can be modeled by a few well-chosen blocks, their 3-D models can be predetermined by a human designer and encoded in the system. Crucially, however, the blocks are not specified in an absolute coordinate system, but are parametrized by a few important attribute variables (e.g., size, orientation, etc.). A suitable block can thus be subjected to transformations such as scaling and rotation to match a modeled entity in the environment.

The geometry of the environment is determined by computing these transformations using user-supplied correspondences between block and image edges. Once geometry is known, textures are computed by back-projecting input images onto the generated model. However, the user must also assist the process by identifying pixels arising from occluders (e.g., trees) so that they are not assigned to the model.

Debevec et al. demonstrate that this block-based approach reduces the number of variables to be determined by a factor of hundred, leading to robust solutions even with very few correspondences. In addition, Euclidean properties of the environment (right angles, verticality etc.) are *directly* encoded in blocks, enhancing the quality of the model generated. However, despite these advantages, the human user must process each image (and each structure) individually to produce the 3-D model. It therefore appears difficult to extend this system for very large sets of images, and most 3-D models built using the Facade system employ only a small set (a few tens) of images.

This thesis, especially the algorithm to extract vertical facades (Chapter 5), is partly inspired by such interactive modeling systems. However, the focus is on performing model extraction automatically (given pose estimates), making the process scalable for larger sets of images.

2.3 Calibrated Imagery

Classically, techniques in computer vision that compute 3-D structure from a few (two or three) calibrated images have relied on *stereo* (Section 2.3.1). Newer techniques, discussed in Section 2.3.2, extend classical stereo to recover structure from multiple calibrated images.

2.3.1 Stereo Vision

Stereo [DA89] involves matching corresponding *features* (usually pixels or edges) across two images and recovering 3-D positions by triangulation. The central focus of much of stereo vision research has been designing good algorithms to solve the correspondence problem.

At first glance, matching a feature between two registered images (i.e., images with known camera pose) might appear to involve a two-dimensional search over the entire image. However, this search can be simplified to a single dimension by using the *epipolar constraint* [Fau93] that guarantees that the corresponding feature can lie only on a specific line.

Given this simplification, various strategies are known in the vision literature to match "similar" features. Correlation-based approaches measure similarity in image brightness directly (e.g., linear correlation [Gen77]). Attribute-based approaches use similarity in derived image information (e.g., edge orientation and length [Aya91]).

In practice, however, neither of these methods alone yield correct matches, and other constraints are imposed on the problem to choose a "good" set of matches. Examples of such constraints are the disparity gradient constraint that enforces continuity in pixel depths across an epipolar line [PMF85], and the ordering constraint that assumes little change in occlusion between images [OK85]. Another method is to prune the set of matches using a third image and a prediction–verification scheme [Aya91].

Stereo techniques have also been extended for a pair of *uncalibrated* images [Fau92]. The techniques rely on computing the *fundamental matrix* [Har95] between the images, which serves as a proxy for inter-camera pose. In such cases, the structure of scene can only be determined up to a (projective) transformation [Fau92].

The techniques currently proposed for stereo vision work fairly well for images taken from nearby camera positions, especially for applications (e.g., obstacle detection) for which only qualitative properties of the 3-D world are important. However, for 3-D modeling, where quantitative accuracy is necessary for simulation and rendering, stereo vision has several disadvantages:

- First, it tends to be less effective on general image pairs due to the inherent tradeoff between inter-camera distance (*baseline*) and matching. Images produced with short baselines are easy to correspond, but 3-D estimates obtained by triangulation are not very reliable. Longer baselines provide accurate 3-D information, but matching across disparate images is an extremely hard problem.

- Second, by definition, stereo vision can only provide information about a small portion of the 3-D environment (i.e., the part visible in the two images). Large scale environments thus require many stereo pairs. However, the technique itself does not provide a seamless method to integrate information provided by the various stereo pairs.

Due to these reasons, newer stereo algorithms extend basic stereo to recover 3-D structure from several images simultaneously; this is described in the next section.

2.3.2 Multi-baseline Stereo

The motivation behind multi-baseline stereo is that information present in multiple images *overconstrain* the 3-D reconstruction problem, yielding a more robust solution. It also allows camera configurations with larger baselines to be used for reconstruction.

Early techniques [BBM87, OK93] assumed specific camera configurations (e.g., camera centers lying on a single line). The advantage is that the epipolar constraint described above naturally generalizes to more than two cameras and can be used to restrict the matching search, as in classical stereo. However, this assumption becomes restrictive when modeling large objects such as buildings, where it is advantageous to position cameras at arbitrary locations in the environment.

Recent techniques [KS96, KWZK95] use multiple images taken from generic camera positions. However, these algorithms still rely on image-space correlation-based matching, implying that the cameras should be closely-spaced, and the images should be taken under stable illumination conditions. This is difficult to achieve in extended outdoor environments due to the effects of perspective, occlusion, and illumination variation. To overcome this drawback, several researchers have proposed algorithms that perform matching directly in 3-D space, avoiding image-space correlation; these are described below.

Three-Dimensional Matching

There are several advantages to performing matching directly in 3-D. First, as 3-D space is common to all the images, it naturally incorporates several cameras (and associated geometric constraints) in the algorithm. Second, perspective and occlusion can be handled correctly by operating in 3-D, thus improving the matching process.

Collins [Col96] uses this idea to design a space-sweep approach to perform matching of edgels across several calibrated images. The idea, related to the technique used in this thesis, is that spatial correlation itself provides important cues for matching, especially if the features being matched are sparse in the images. However, Collins' approach uses a *single* sweep-direction, and thus seems more suited to aerial imagery, where the z-direction (height) is a natural choice. Also, his technique does not produce a CAD model that can be used for graphics workstation rendering.

Seitz [KS98, SD97] proposes a color-based correlation sweep technique on 3-D voxels to identify their likely locations and colors. However, his algorithm assumes that no extracted feature can be within the convex hull of camera positions. Thus it is inapplicable for more general datasets where the cameras are situated at arbitrary places in the environment. Also, a raw color-based approach is more sensitive to effects of illumination variation. In contrast, the sweep technique proposed in this thesis uses geometric information (horizontal line segments) that is generally invariant to illumination.

Recently, Faugeras and Keriven [FK] present a technique based on calclus of variation and level set evolution to recover 3-D structure from multiple calibrated images. The algorithm first approximates an object being modeled by a simple implicit function (e.g., a sphere), and modifies this function based on the correlation between the 3-D model and image space information. They present good results for images focusing on a single object under simple, diffuse lighting. It is unknown whether a similar technique can be used for outdoor environments.

2.4 Uncalibrated Imagery

Computer vision research has also addressed the problem of recovering 3-D structure from raw imagery (i.e., no camera calibration). The motivation for this research stems from the ease of acquiring such imagery (e.g., by using digital video cameras). In this section, I describe two classes of algorithms that analyze uncalibrated images. *Structure from motion* algorithms track features across the video sequence, and use the constraints imposed to recover 3-D structure (Section 2.4.1). *Direct motion estimation* algorithms attempt to approximate the apparent pixel motion in a video sequence by a set of 2-D transformations, yielding some information about 3-D structure (Section 2.4.2).

2.4.1 Structure From Motion (SFM)

The idea in this method is to track corresponding features (pixels or edges) across the image sequence and use the constraints generated to recover both 3-D structure (i.e., locations of features) and camera motion information. SFM algorithms can be classified into two different categories: *batch* methods that perform reconstruction using all the constraints at once, and *recursive* methods that incorporate a single image (the next image in the sequence) to the existing description of the 3-D scene. Recursive methods [AP95] have the advantage that they can be used in active vision (e.g., mobile robots), where images are only available incrementally. However, as they depend on the first few images to initialize the algorithm, they tend to be less accurate than batch methods.

One of the more successful applications of the batch technique has been to the special case of orthographic projection [TK92] where, due to the linearity of the constraints, standard techniques like singular value decomposition [PTVF92] can be used to solve constraints imposed by tracking. However, these so-called factorization methods suffer from limiting assumptions about camera model and difficulties in dealing with occlusion.

More general algorithms [MVQ93, SK94, TK95] address the problem of perspective projection. Mohr et. al. [MVQ93] describe a technique that recovers structure up to a *projective* transformation. Szeliski and Kang [SK94] use point correspondences to solve for Euclidean structure. Taylor and Kriegman [TK95] solve a similar problem for edge correspondences.

There are a few disadvantages in this approach to recover 3-D information. First, as *both* structure and motion are recovered from a sequence of images, it is essential to track a large set of features to produce enough constraints to solve the problem robustly. In practice, this implies that images in a sequence need to be spaced very closely so that tracking succeeds in corresponding many features across the image stream. Unfortunately, the cost of acquiring such an image set becomes prohibitive if large-scale models are being reconstructed. Second, this technique provides only *sparse* 3-D information which is difficult to convert to a 3-D model suitable for computer graphics rendering.

2.4.2 Direct Motion and Model Estimation

Another approach to extract 3-D information from uncalibrated images [AS95, AW96, IRP97, KAH94, SA96, Wei97] is to directly estimate camera motion and geometry parameters from image optical flow [Fau93], i.e., the apparent pixel motion across images due to change in the viewpoint and/or object motion. The estimation is performed by using one or more 2-D image transformations to capture both *dominant* geometry (e.g., planes) and camera motion. The transformations are derived by minimizing an objective function that describes pixel differences after accounting for pixel motion due to the transformation. Unlike structure from motion, this technique uses information present in *all* pixels to solve for the geometry. This leads to a more robust estimation, especially for images containing dominant planar regions (e.g., the ground plane in aerial images). However, this technique tends to be sensitive to illumination variation and occlusion, and is thus more suited for closely spaced (both in space and time) images, as in a video sequence.

The vertical facade extraction algorithm described in this thesis (Chapter 5) uses a similar idea of extracting dominant planar regions. However, the extraction is performed using geometric features (horizontal line segments), making it usable for images obtained with large camera baselines.

2.5 Image-Based Rendering

Images have been used extensively for enhancing realism of computer graphics models. Their simplest application, texture mapping [Hec86, Wil83], simply "pastes" images on to existing geometry (e.g., a texture on a building facade or a grassy texture on a ground plane). Other applications use images to modify the underlying geometry itself: perturbing normals using *bump* mapping, or perturbing positions using *displacement* mapping [FvDFH90].

However, it is only recently that images have been used to *directly* generate synthetic views, supplanting the use of geometric models. In QuickTime VR [Che95], multiple images are "stitched" together to yield cylindrical panoramas, and these panoramas are reprojected from various angles to provide synthetic views. Multiple panoramas are used to model an entire environment by allowing the user to teleport from one mosaic to another. While QuickTime VR provides an easy method to model an environment, the user has limited navigational flexibility. In particular, it is not possible for the user to be situated at points other than the input set of camera positions.

McMillan [MB95] generalized this approach by computing optical flow information across several panoramas. Much like actual geometry, the flow information (i.e., pixel disparities) can be used to compute images from viewpoints other than the input points. However, computing optical flow information is a difficult problem. To make this computation feasible, McMillan's research utilized cameras positioned on a single line. It is unknown whether such information can be robustly computed over images taken from widely separated, generic camera positions.

Another approach has been to avoid computing depth or disparity information, and synthesize novel views by image *interpolation* [GGSC96, LH96, SD96]. The idea is to sample the space of viewpoints densely, and use nearby images to produce an image for a novel viewpoint. The sampling and interpolation itself is performed in 4-D linespace [GGSC96, LH96] that concisely captures all possible viewing rays. By restricting the set of novel viewpoints, this method avoids the difficult problem of depth computation. However, a very large set of images are needed to provide the illusion of smooth, 3-D viewpoint motion. Thus current applications of this approach use synthetic models or small real-world objects where it is possible to acquire such a dense image set.

While image-based rendering is still relatively new, it holds promise for future graphics applications due to its several advantages. First, it avoids the difficult problem of (full blown) 3-D model extraction, while providing realistic synthetic images. Second, if, as is typical, only a few images are used for generating each novel image, the cost of computation is independent of the geometric complexity of the scene. However, image-based rendering systems do not generate an editable 3-D representation (such as a CAD model). This is a disadvantage in many applications, such as when a designer experiments with geometry and/or lighting conditions or applies simulation techniques like collision detection. A hybrid system (e.g., [DTM96]) is probably the best option for the future: extract simple polyhedral geometric structures (the focus of this thesis) and utilize image-based techniques for more complex objects (e.g., trees).

2.6 Summary

While the problem of recovering 3-D information from images has been extensively researched, there is at present no practical method to extract 3-D models from a large set of images of an outdoor environment, necessitating a new approach. This thesis uses a large set of pose-instrumented images and exploits geometric constraints to generate a model consisting of texture-mapped polygons. Unlike photogrammetry and interactive modeling systems, the algorithms involve minimal human-intervention in the reconstruction process. Unlike stereo, the algorithms use information contained in a large set of images simultaneously. Unlike structure from motion, initial estimates of each camera's pose are used to to perform robust reconstruction. Finally, unlike image-based rendering, the output is a classical texture-mapped 3-D model that can be used in a variety of graphics applications.

Chapter 3

Automatic Spherical Mosaicing

As mentioned earlier, physical instrumentation alone does not produce sufficiently accurate estimates for direct incorporation into 3-D reconstruction algorithms. For example, it is straightforward to generate orientation estimates (e.g., using a pan/tilt head) to about a degree of accuracy. This is one to two orders of magnitude less accurate than pixel level alignment required for typical reconstruction algorithms (including those described in this thesis). It is thus necessary to design *pose refinement* algorithms that recover accurate camera pose from (approximate) initial estimates.

The method proposed for pose refinement consists of two parts. The first part (described in this chapter) is an optimization that recovers pose estimates of images *relative* to a single node. As the viewpoint is fixed, this is equivalent to computing orientations of each image with respect to a canonical image (e.g., the first image of the node). The second part (described in the next chapter) is an optimization that determines the position and orientation of each node in a single global coordinate system.

There are several advantages in this approach. First, there are useful geometric constraints that enable robust, automated recovery of relative orientations. Second, this reduces (by a factor of 50) the number of unknowns during further optimization. Finally, as a side-effect, the algorithm produces *spherical mosaics* for each node. In addition to being useful as "first-class" data objects for reconstruction, such mosaics also enable compact visualization of nodes.

Overview

(a) (b) (c)

Figure 3-1: Part (a) shows one image of a hemispherical tiling blended with its adjacent images. Part (b) illustrates blurring due to incorrect pose estimates, and Part (c) shows the same view after optimization.

Figure 3-1 illustrates the basic idea behind the optimization technique. As depth and parallax effects do not occur across images taken from a single optical center [Har97], the apparent pixel motion between such images can be explained by a 2-D *projective* transformation that depends on camera orientations and internal parameters (described below). As shown in Part (a) of the figure, this transformation maps pixels from adjacent images into a common 2-D space. Incorrect transformations arising from inaccurate estimates of camera pose result in mismatches between pixels, causing the *blurring* and *ghosting* shown in Part (b). The optimization techniques described in this chapter use these pixel differences to refine pose estimates and eliminate the blurring artifacts (shown in Part (c)).

The rest of the chapter is organized as follows. Section 3.1 briefly describes the process of image formation via perspective projection. Section 3.2 reviews 2-D projective transformations and methods to compute them. Section 3.3 presents a closed-form solution to recover camera internal parameters (and hence, rotations) from projective transformations. While theoretically elegant, this technique is sensitive to errors in image formation and generally yields poor results for real imagery. To address this problem, Section 3.4 presents a global optimization technique that directly computes rotations and camera internal parameters. The results of this optimization presented in Section 3.5 demonstrate that it is well-suited for computing accurate pose (relative to a node) from estimates.

3.1 Perspective Projection

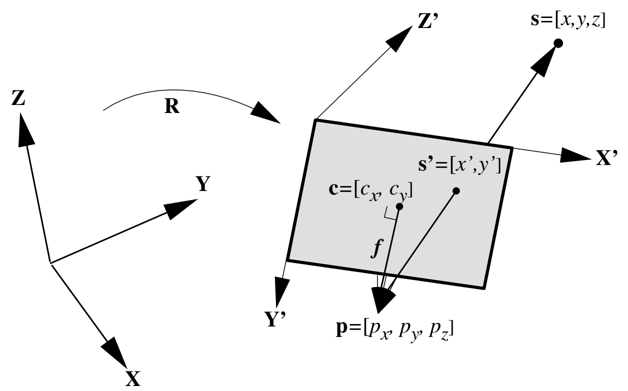


Figure 3-2: Overview of perspective projection.

Figure 3-2 shows the process of image formation by perspective projection, illustrated for a point $\mathbf{s} = [x, y, z]$ as viewed by a camera at position $\mathbf{p} = [p_x, p_y, p_z]$. The rotation from the global coordinate system $\mathbf{XYZ}$ to the camera coordinate system $\mathbf{X'Y'Z'}$ is specified by a 3×3 rotation matrix $\mathbf{R}$.

The first step is to compute $\mathbf{s_1} = [x_1, y_1, z_1]$, the coordinates of $\mathbf{s}$ in the camera's coordinate system:

$$\mathbf{s_1} = \mathbf{R}(\mathbf{s} - \mathbf{p})$$

Next, perspective projection scales the x and y coordinates by relative depth to yield normalized image coordinates $[x_2, y_2]$:

$$x_2 = \tfrac{x_1}{z_1} \quad y_2 = \tfrac{y_1}{z_1}$$

The normalized coordinates are converted to pixel values based on the focal length (expressed in pixels) and the coordinates of the principal point:

$$x' = fx_2 + c_x \quad y' = fy_2 + c_y$$

Note that it is not necessary that the principal point coincide with the image center; in practice, it is usually off by a few pixels.

It is sometimes convenient to represent the entire transformation as a chain of matrix transformations. This is done using projective geometry [Fau93]:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \cong \mathbf{KM} \begin{bmatrix} \mathbf{R} & -\mathbf{Rp} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.1}$$

where $\mathbf{K}$ is the 3×3 (upper-triangular) internal camera parameter matrix:

$$\begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

$\mathbf{M}$ is the 3×4 canonical perspective projection matrix:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The projective equality ($\cong$) is valid up to scaling.

Note that this discussion restricts the camera to be described by a three parameter model (instead of the standard five parameter model [Fau93]) by assuming that the pixels are square (i.e., the focal lengths are equal) and non-linear distortion is negligible. These assumptions are valid for the digital cameras used in this research; radial distortion is less than a pixel even near the edge of the image. It is straightforward to incorporate non-square pixels in the algorithms by using different focal lengths. However, any non-linear distortion has to be measured and corrected (e.g., by using the Stein's method [Ste95]) before using the images as input to the algorithms.

3.1.1 Rotations as Quaternions

There are several choices for expressing the rotation of the camera: 3×3 orthonormal matrices, Euler angles, quaternions etc. Horn [Hor87, Hor91] demonstrates that quaternions are a convenient representation, especially for problems involving numerical optimization. In this representation, a rotation $\mathbf{R}$ is represented as a four-dimensional unit vector $\mathbf{q} = [q_0, q_x, q_y, q_z]$ where $q_0^2 + q_x^2 + q_y^2 + q_z^2 = 1$.

The quaternion representation is compact when compared to the orthonormal representation (only four parameters instead of nine). Unlike the even more compact Euler angle representation (i.e., the three angles expressing rotations about the $\mathbf{X}, \mathbf{Y}, \mathbf{Z}$ axis), it is stable while representing large rotations, and has no singularities.

The following matrix converts a quaternion to a 3×3 orthonormal matrix:

$$\begin{bmatrix} (q_0^2 + q_x^2 - q_y^2 - q_z^2) & 2(q_x q_y - q_0 q_z) & 2(q_x q_z + q_0 q_y) \\ 2(q_y q_x + q_0 q_z) & (q_0^2 - q_x^2 + q_y^2 - q_z^2) & 2(q_y q_z - q_0 q_x) \\ 2(q_z q_x - q_0 q_y) & 2(q_z q_y + q_0 q_x) & (q_0^2 - q_x^2 - q_y^2 + q_z^2) \end{bmatrix}$$

Horn [Hor87] describes an algorithm to recover quaternion components from a rotation matrix.

Rotation-Matrix Derivative

A frequent expression used in this thesis is the derivative of the rotation matrix with respect to the quaternion parameters. This derivative is used in the various pose refinement algorithms that "update" a camera's rotation.

The derivative of a rotation matrix $\mathbf{R}^{-1}$ with respect to a quaternion $\mathbf{q} = [q_0, q_x, q_y, q_z]^T$ is a tensor of dimension $3 \times 4 \times 3$. Since the derivative is typically multiplied by a vector (say, $\mathbf{v} = [v_x, v_y, v_z]^T$), only its value at $\mathbf{v}$, a 3×4 matrix, is given below:

$$\begin{aligned} a &= +q_0 v_x + q_z v_y - q_y v_z \\ b &= -q_z v_x + q_0 v_y + q_x v_z \\ c &= +q_y v_x - q_x v_y + q_0 v_z \\ d &= +q_x v_x + q_y v_y + q_z v_z \\ \left(\frac{\partial \mathbf{R}^{-1}}{\partial \mathbf{q}}\right)_{\mathbf{v}} &= \begin{bmatrix} a & d & -c & b \\ b & c & d & -a \\ c & -b & a & d \end{bmatrix} \end{aligned}$$

Note the significant computational advantage of using quaternions over other representations (Euler angles, axis and angle) for rotations. In these other representations, derivatives contain sine and cosine terms, increasing the computational cost of estimating derivatives.

3.2 2-D Projective Transformations

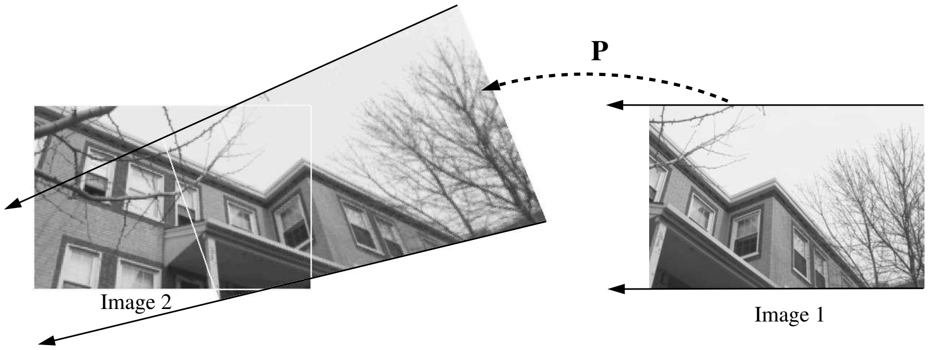


Figure 3-3: Transforming image 1's pixels to image 2's space.

Figure 3-3 illustrates the relation between two images taken from a fixed optical center, but with differing orientations. In such cases, pixels in one image can be mapped to the other image by a 2-D *projective* transformation [Har97]. Note that unlike simpler 2-D transformations (translation, rotation, affine), the projective transformation *does not* preserve parallel lines. This is evident in Figure 3-3, where the lines bounding image 1 intersect after the transformation.

As depth effects do not occur across two images taken from the same optical center [Har97, Sze96], the general perspective projection (Equation 3.1) simplifies to:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \cong \mathbf{KR} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.2}$$

Inverting Equation 3.2 yields:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \cong \mathbf{R}^{-1}\mathbf{K}^{-1} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} \tag{3.3}$$

Equation 3.3 provides a method to convert pixel positions in one image (say, image 1) to 3-D rays. Thus pixel coordinates in another image (say, image 2) can be obtained by projecting back into image 2's space using Equation 3.2:

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} \cong \mathbf{K}\mathbf{R}_2\mathbf{R}_1^{-1}\mathbf{K}^{-1} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \tag{3.4}$$

Thus the 2-D *projective* transformation that maps pixel (x_1, y_1) of image 1 to pixel (x_2, y_2) of image 2 is:

$$\mathbf{P} = \mathbf{K}\mathbf{R}_2\mathbf{R}_1^{-1}\mathbf{K}^{-1} \tag{3.5}$$

Note that, as $\cong$ denotes projective equality, Equation 3.4 is valid only up to a linear scaling of $[x_2, y_2, 1]^T$. As a consequence, only eight parameters are needed to describe the matrix $\mathbf{P}$. Thus 2-D projective transformations are also known as *8-parameter warps*. Typically, the unknown scale factor of the transformation is determined by fixing either $\det \mathbf{P} = 1$ or $\mathbf{P}_{33} = 1$.

3.2.1 Computing Warps

The simplest method to compute such warps is by using four point correspondences between the two images [Fau93]. Several algorithms (e.g., [ZFD97]) use this approach. However, identifying suitable features and correspondences is a difficult problem, and known methods yield good results only for images with significant overlap and minimal projective distortion.

An alternate method would be to use color or luminance information present in images to compute the warp by nonlinear optimization. Below, I briefly review an optimization due to Szeliski [Sze96] that computes 8-parameter warps using this technique. This also introduces the Levenberg-Marquardt (LM) optimization [PTVF92], which is used in several places in this thesis.

The idea is to compute a warp that (locally) minimizes image-space error by using nonlinear optimization. The error function for this optimization simply measures the difference in brightness between two images 1 and 2 (in the overlap region), after pixels in image 1 are mapped to image 2's space. The difference in brightness is measured by sum-of-squared difference (SSD) error metric:

$$E_{12} = \sum_{x_1,y_1} (L_1(x_1, y_1) - L_2(\mathbf{P}(x_1, y_1)))^2 \tag{3.6}$$

The SSD form is well suited for numerical optimization, as only first order derivatives can be used to compute update values for the iteration [PTVF92].

The optimization consists of analytically determining derivatives of a single error term of the form:

$$e^2_{x_1,y_1} = (L_1(x_1, y_1) - L_2(x_2, y_2))^2$$

with respect to $\mathbf{P}$. The derivative $\frac{\partial e_{x_1,y_1}}{\partial \mathbf{P}}$ is expressed as a 8-component vector consisting of derivatives with respect to each (variable) parameter in $\mathbf{P}$.

In LM optimization, the overall gradient term $\mathbf{G}$ is computed by accumulating over all error terms [PTVF92]:

$$\mathbf{G} = \sum_{x_1,y_1} e_{x_1,y_1} \frac{\partial e_{x_1,y_1}}{\partial \mathbf{P}}$$

Similarly, the (approximate) Hessian term corresponding to two adjacent images i and j is:

$$\mathbf{H} = \sum_{x_1,y_1} \frac{\partial e_{x_1,y_1}}{\partial \mathbf{P}} (\frac{\partial e_{x,y}}{\partial \mathbf{P}})^T$$

The optimization proceeds by incrementing the value of $\mathbf{P}$ by

$$\mathbf{\Delta P} = -(\mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{G}$$

where $\mathbf{I}$ is the identity matrix, and λ is a varying stabilization parameter set initially to a high value, and reduced to 0 as the procedure converges [PTVF92].

Initialization

As in any nonlinear optimization, it is important to initialize the warp with a value that is close to the optimum. Several techniques have been proposed for this step. McMillan [McM97] computes a 2-D translation with minimum error, and uses it to initialize the optimization. However, such translations are difficult to determine if the effects of perspective are large. Szeliski and Shum [SS97b] perform the initialization interactively, with the help of a human operator.

In this case, the problem of initialization is solved by the availability of pose estimates (and camera calibration). It is thus straightforward to compute a good initial estimate using Equation 3.5 by using the internal parameter matrix determined by camera calibration and rotations supplied by physical instrumentation.

Warp Example

Figure 3-4 shows the results of this optimization for two images (sized 762×506 pixels). Note that the blurring, initially present, disappears after optimization. The projective $\mathbf{P}$ matrix computed (normalized such that the determinant is 1) is:

$$\mathbf{P} = \begin{bmatrix} 0.7423 & 0.0162 & 584.7688 \\ -0.0994 & 0.9983 & 14.2725 \\ -0.0003 & 0.0000 & 1.0463 \end{bmatrix} \tag{3.7}$$

I next describe a technique to recover camera internal parameters and relative rotations from such warps.

3.3 Pose Estimates from Warps

It is straightforward to recover rotation from warps using Equation 3.5 if the camera calibration is known accurately. If not, the following closed-form solution can be used to derive camera calibration from the warp itself.

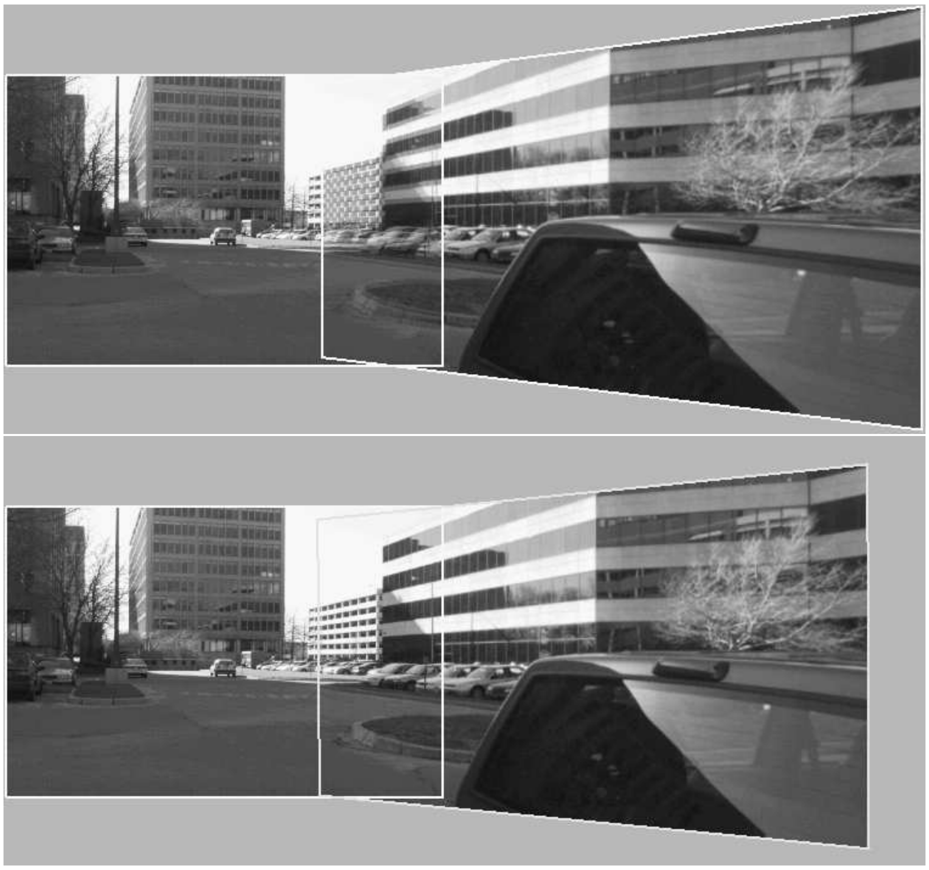


Figure 3-4: This figure shows the projective warp between two images, before and after optimization.

3.3.1 Closed-Form Solution for Internal Parameters

The technique proposed in this section is similar in spirit to closed-form solutions presented by Hartley [Har97] and McMillan [McM97], but much simpler to express. The basic idea is use Equation 3.5 to enforce the orthonormality constraint for valid 3×3 rotational matrices, and use eigen-vectors of the warp matrix to define the solution.

Rewriting Equation 3.5 in terms of $\mathbf{R} = \mathbf{R}_2\mathbf{R}_1^{-1}$, the relative rotation between the two images, and solving for $\mathbf{R}$:

$$\mathbf{R} = \mathbf{K}^{-1}\mathbf{P}\mathbf{K} \tag{3.8}$$

As $\mathbf{R} = \mathbf{R}^{-T}$ (the orthonormality condition for rotations),

$$\mathbf{K}^{-1}\mathbf{P}\mathbf{K} = \mathbf{K}^{T}\mathbf{P}^{-T}K^{-T}$$

Rearranging terms yields:

$$\mathbf{PC} = \mathbf{CP}^{-T} \tag{3.9}$$

where $\mathbf{C} = \mathbf{KK}^T$. Note that $\mathbf{C}$ is a symmetric 3×3 matrix given by components of the internal parameter matrix (i.e., focal length f and image center $[c_x, c_y]$):

$$\mathbf{C} = \begin{bmatrix} f^2 + c_x^2 & c_x c_y & c_x \\ c_x c_y & f^2 + c_y^2 & c_y \\ c_x & c_y & 1 \end{bmatrix} \tag{3.10}$$

The solution for $\mathbf{C}$ in Equation 3.9 can be expressed in terms of eigen-vectors of $\mathbf{P}$. The eigen-values of $\mathbf{P}$ are the same as that of $\mathbf{R}$ as they are related by *similarity transform* [Str88], i.e., left and right multiplication by a matrix and its inverse (Equation 3.8). The eigen values of the rotation $\mathbf{R}$ are 1, $e^{i\theta}$, and $e^{-i\theta}$, where θ is the angle of rotation. Let $\mathbf{e}_0$, $\mathbf{e}_1$ and $\mathbf{e}_2$ be the eigen-vectors of $\mathbf{P}$ corresponding to these three eigen values, respectively.

Note that $\mathbf{C} = \mathbf{e}_0\mathbf{e}_0^T$ is a solution to Equation 3.9:

$$\mathbf{PC} = \mathbf{Pe_0e}_0^T = \mathbf{e_0e}_0^T = \mathbf{e_0e}_0^T\mathbf{P}^{-T} = \mathbf{CP}^{-T}$$

This derivation uses the fact that if $\mathbf{e}$ is a right eigen-vector of $\mathbf{P}$, then $\mathbf{e}^T$ is a left eigen-vector of $\mathbf{P}^{-T}$.

Similarly, as the eigen-values corresponding to $\mathbf{e}_1$ and $\mathbf{e}_2$ are reciprocals of each other, it can be shown that $\mathbf{e}_1\mathbf{e}_2^T$ and $\mathbf{e}_2\mathbf{e}_1^T$ are also possible solutions of $\mathbf{C}$. The most general symmetric solution to $\mathbf{C}$ is therefore a linear combination of the three solutions, where the second two solutions are weighted equally:

$$\mathbf{C} = c_0\mathbf{e}_0\mathbf{e}_0^T + c_1(\mathbf{e}_1\mathbf{e}_2^T + \mathbf{e}_2\mathbf{e}_1^T)$$

The coefficients c_0 and c_1 are solved by enforcing the constraints that the $\mathbf{C}_{33} = 1$ and $\mathbf{C}_{12} = \mathbf{C}_{13}\mathbf{C}_{23}$ (again using the three parameter camera model). The matrix $\mathbf{K}$ can be recovered by "taking the square root" of matrix $\mathbf{C}$ by Cholesky Decomposition [PTVF92].

Using this technique for the warp described in Equation 3.7 gives values of $f = 1141$, $c_x = 397$, $c_y = 249$ (all in pixels). The rotation corresponding to these parameters is:

$$\begin{bmatrix} 0.8897 & 0.0090 & 0.4603 \\ -0.0004 & 0.9961 & 0.0010 \\ -0.4599 & 0.0080 & 0.8901 \end{bmatrix}$$

The rotation determined appears reasonable *qualitatively*, i.e., it is (close to) a rotation about the image's y axis; this agrees with Figure 3-4. However, as described in the next section, *quantitatively* there is a significant difference between this and the physically correct rotation.

3.3.2 Problems with Warps

The rotation computed in the previous section is approximately 27 degrees ($= \cos^{-1} 0.89$). However, the physical rotation was close to 30 degrees ($= \frac{360}{12}$ as twelve images formed a full circle). Using this rotation for the sequence of images would therefore leave a gap between the last and the first image.

At first glance, it might appear that the problem is due to incorrect internal parameters computed by this method. The computed focal length differs significantly from the focal length determined by Tsai's calibration algorithm (by around 150 pixels). However, even though a different set of internal parameters might yield other rotations, the *angle* of rotation is completely determined by the eigen-values of the projective warp $\mathbf{P}$. As this is unaffected by the method by which internal parameters are computed, this problem is inherent in the warp solution itself.

Figure 3-5 illustrates the problem by showing the warp computed by directly optimizing with respect to the internal parameters $\mathbf{K}$ and the relative rotation $\mathbf{R}$ (using the method described below). The internal parameters were computed to be $f = 1016$, $c_x = 393$, $c_y = 253$ (all in pixels), and the relative rotation and warp

Figure 3-5: This figure shows the projective warp between two images, after direct optimization.

between the two images are:

$$\mathbf{R} = \begin{bmatrix} 0.8637 & 0.0047 & 0.5040 \\ -0.0054 & 1.0001 & -0.0000 \\ -0.5043 & -0.0031 & 0.8638 \end{bmatrix} \quad \mathbf{P} = \begin{bmatrix} 0.669371 & 0.003488 & 592.614624 \\ -0.114616 & 0.999421 & 15.121033 \\ -0.000492 & -0.000003 & 1.058760 \end{bmatrix}$$

Note that the rotation ($= cos^{-1} 0.8637 = 30.27$ degrees) and internal parameters agree well with initial estimates. Also, visually, the overall warp computed differs from that described in Equation 3.7. This can be observed in the large "gap" between the right edge of the image and the border in Figure 3-4 (which is lesser in Figure 3-5). However, *within the region of overlap*, the two warps appear identical – there are no blurring artifacts visible in either case. Thus, there are several possible warps that yield low SSD error, but not all of them correspond to physically correct rotations. It is therefore essential to impose the rotational constraint *during* the optimization to obtain quantitatively correct results.

Similar results were also observed for warps corresponding to other image pairs. Thus, while 8-parameter warps generate visually consistent blending between adjacent images, they appear to be inadequate for recovering quantitative 3-D parameters, such as relative rotations. Also, more fundamentally, relying on local *pairwise* warps to compute global quantities can lead to inconsistencies in the computed internal parameters and rotations. In the next section, I present a method using global optimization that addresses these problems.

3.4 Spherical Mosaicing

The optimization described in this section directly solves for the set of "best" possible rotations for *all* images, given initial estimates. The advantage of this approach is that global consistency is *guaranteed* by computing a *unique* rotation for each image.

The proposed approach is related to computing cylindrical panoramas by McMillan [MB95], who solves for rotation angles of images taken with rotation around a single axis. In addition, he also enforces the constraint that angles computed for a cylindrical panorama should sum to 2π. A similar constraint is also used by Szeliski and Shum [SS97b] to "close the gap" between the first and last images. Unlike these results, I compute a spherical panoramic image from images with arbitrary rotations.

Shum and Szeliski in their recent paper [SS97a] also compute full-view spherical panoramas. However, their global alignment algorithm requires a combination of both correlation-based and feature-based optimization. In contrast, I directly optimize correlation to perform global alignment, avoiding the step of identifying suitable features.

The approach followed is to optimize a global correlation function defined for adjacent images with respect to all orientations (represented as quaternions). As a side effect, the algorithm computes a *spherical mosaic*, a composite of all images corresponding to a single node.

3.4.1 Optimization

The algorithm uses a global error function:

$$E = \sum_{i,j \text{ are adjacent}} E_{ij} + E_{ji}$$

where E_{ij} is the SSD error between luminance values of images i and j:

$$C_{ij} = \sum_{x_i, y_i} (L_i(x_i, y_i) - L_j(\mathbf{P}_{ij}(x_i, y_i)))^2$$

This correlation function is computed only for pairs of adjacent images in the spherical tiling, and only for pixels of image i that map to a valid pixel of image j. Even though no overlap between the two adjacent images gives zero error, the algorithm converges (in practice) to a value close to the initial estimates, with significant overlap between the images.

As in the pairwise warp estimation, this function is minimized by computing derivatives with respect to the orientations and using LM nonlinear optimization starting from the initial orientations. The various steps in the computation are described in greater detail below.

For a single error term of the form:

$$e_{x,y}^2 = (L(x,y) - L'(x'',y''))^2$$

where:

$$x'' = \tfrac{x'}{z'} \quad y'' = \tfrac{y'}{z'}$$

and

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \mathbf{v}' = \mathbf{P} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{K}\mathbf{R}'\mathbf{R}^{-1}\mathbf{K}^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3.11}$$

the derivatives are computed as follows (using the rotation-matrix derivative given in Section 3.1.1):

$$\frac{\partial \mathbf{v}'}{\partial \mathbf{q}} = \mathbf{K}\mathbf{R}'(\frac{\partial \mathbf{R}^{-1}}{\partial \mathbf{q}})_{\mathbf{v}}$$

where $\mathbf{v} = \mathbf{K}^{-1}[x,y,1]^T$. Then, the derivative of the term $e_{x,y}$ with respect to the quaternion $\mathbf{q}$ is given by:

$$\tfrac{\partial x''}{\partial \mathbf{q}} = \tfrac{\frac{\partial x'}{\partial \mathbf{q}} - x'' \frac{\partial z'}{\partial \mathbf{q}}}{z'} \quad \tfrac{\partial y''}{\partial \mathbf{q}} = \tfrac{\frac{\partial y'}{\partial \mathbf{q}} - y'' \frac{\partial z'}{\partial \mathbf{q}}}{z'} \tag{3.12}$$

$$\frac{\partial e_{x,y}}{\partial \mathbf{q}} = \frac{\partial L'}{\partial x''}\frac{\partial x''}{\partial \mathbf{q}} + \frac{\partial L'}{\partial y''}\frac{\partial y''}{\partial \mathbf{q}} \tag{3.13}$$

Note that Equation 3.13 involves image derivatives $\frac{\partial L'}{\partial x''}$ and $\frac{\partial L'}{\partial y''}$; these are approximated using the following convolution matrices applied at (x', y') [Hor86]:

$$\frac{\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}}{8} \quad \frac{\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}}{8}$$

The gradient term corresponding to the quaternion q_i is computed by accumulating over all terms that depend on $\mathbf{q}_i$:

$$\mathbf{G}_i = \sum_{x,y} e_{x,y} \frac{\partial e_{x,y}}{\partial \mathbf{q_i}}$$

Similarly, the Hessian term corresponding to two adjacent images i and j is:

$$\mathbf{H}_{ij} = \sum_{x,y} \frac{\partial e_{x,y}}{\partial \mathbf{q}_i} \left(\frac{\partial e_{x,y}}{\partial \mathbf{q}_j}\right)^T$$

For a spherical tiling consisting of n images, values of $\mathbf{G}_i$ and $\mathbf{H}_{ij}$ are concatenated to yield the global $1 \times 4n$ gradient $\mathbf{G}$ and the global $4n \times 4n$ Hessian $\mathbf{H}$.

In an unconstrained optimization, the increments would be computed as $-\mathbf{H}^{-1}\mathbf{G}$. However, as quaternions are constrained to be unit vectors, the following additional constraints involving the increments $\delta\mathbf{q}_i$ are necessary:

$$\forall i : \mathbf{q}_i \cdot \delta\mathbf{q}_i = 0$$

Using Lagrange multipliers λ_i to enforce the above constraints, the equation for computing the increments becomes:

$$\begin{bmatrix} \mathbf{H} & \mathbf{Q} \\ \mathbf{Q}^T & \mathbf{0} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{Q} \\ \Lambda \end{bmatrix} = - \begin{bmatrix} \mathbf{G} \\ \mathbf{0} \end{bmatrix} \tag{3.14}$$

where:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{q}_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{q}_n \end{bmatrix}, \ \Delta\mathbf{Q} = \begin{bmatrix} \delta\mathbf{q}_1 \\ \delta\mathbf{q}_2 \\ \vdots \\ \delta\mathbf{q}_n \end{bmatrix}, \ \Lambda = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix}$$

The optimization proceeds by solving Equation 3.14 for $\Delta\mathbf{Q}$ and Λ using a linear-solver[1], and then normalizing $\mathbf{q}_i$:

$$\frac{\mathbf{q}_i + \delta\mathbf{q}_i}{||\mathbf{q}_i + \delta\mathbf{q}_i||}$$

Convergence in the procedure is detected when the value of the objective function changes by less than some threshold (e.g., 0.1%).

3.4.2 Internal Camera Parameters

In addition to computing orientations, the algorithm also performs an optimization on the internal camera parameters. Even though the camera was calibrated offline, in practice, there could be small variations while actually collecting the nodes. These variations would result in incorrect warps between adjacent images, yielding misalignment artifacts similar to those arising from errors in rotation measurements.

The overall optimization alternates between a step that updates all rotations, and a step that updates internal parameters. The new parameters are computed using derivatives of $\mathbf{v}'$ in Equation 3.11 with respect to the camera focal length f and image center (c_x):

$$\begin{aligned}
\frac{\partial \mathbf{v}'}{\partial f} &= \left(\begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{R}'\mathbf{R}^{-1}\mathbf{K}^{-1} + \mathbf{K}\mathbf{R}'\mathbf{R}^{-1} \begin{bmatrix} -\frac{1}{f^2} & 1 & \frac{c_x}{f^2} \\ 0 & -\frac{1}{f^2} & \frac{c_y}{f^2} \\ 0 & 0 & 0 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \\
\frac{\partial \mathbf{v}'}{\partial c_x} &= \left(\begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathbf{R}'\mathbf{R}^{-1}\mathbf{K}^{-1} + \mathbf{K}\mathbf{R}'\mathbf{R}^{-1} \begin{bmatrix} 0 & 0 & -\frac{1}{f} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \right) \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\end{aligned}$$

The derivative with respect to c_y can be similarly determined. These are used to generate derivatives of the error term $e_{x,y}$ in a manner similar to Equations 3.12 and 3.13.

[1] Sparse matrix techniques can be used to improve the speed of the solver. However, as most of the computational effort is expended in computing the entries of $\mathbf{H}$, this optimization is not worthwhile.

Relative Importance of Camera Parameters

Are all the internal parameters equally important for this optimization? The following simplified 2-D analysis shows that determining the focal length accurately is much more crucial than determining the coordinates of the image center. This result is also confirmed empirically in Section 3.5. This enables the simpler calibration technique of Tsai [Tsa87] (which assumes that the image center is the same as the principal point) to be used instead of a more complex technique [LT87].

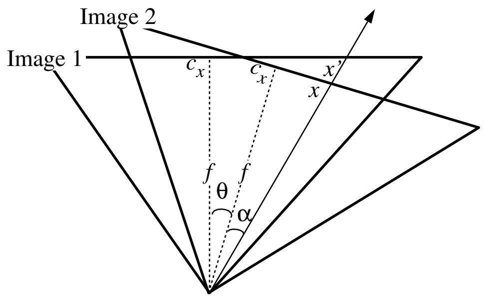


Figure 3-6: Rotation and camera parameters in 2-D.

Figure 3-6 shows two 1-D images rotated formed by rotating a "line" camera by an angle θ. In this figure, the transformation between pixel x (with offset angle α from the center) in image 2 to pixel x' in image 1 is given by:

$$x' = c_x + f\tan(\theta+\alpha) = c_x + f\frac{\tan\theta + \tan\alpha}{1 - \tan\theta\tan\alpha}$$

For small angles of rotation and small fields of view $\tan\theta\tan\alpha << 1$. Thus:

$$x' \approx c_x + f\tan\theta + f\tan\alpha = c_x + f\tan\theta + x - c_x = f\tan\theta + x$$

The mapping is thus fairly insensitive to the coordinates of the principal point, permitting the image center to be used as initial values for optimization.

3.4.3 Implementation

The implementation of the optimization contains several other features that improve its performance in practice. In this section, I briefly describe the use of band-pass values, a modification to deal with textureless images, and techniques for efficient implementation of the algorithm.

Band-Pass Values

Figure 3-7: An image and its band pass values.

In the implementation, the algorithm first performs the optimization on band-pass luminance values (see Figure 3-7) instead of luminance values, and then perform the optimization on luminance values. Band-passed luminance values are computed from image luminance values by convolving with the image with the derivative of a Gaussian (of 5 pixel radius). This results in elimination of both high-frequency components (by Gaussian convolution) and low-frequency components (by differentiation), resulting in a smoother optimization function. Section 3.5 demonstrates that use of band-pass values results in both faster convergence and increased avoidance of false minima.

Textureless Images

Several images in a node lack sufficient texture to determine their pose accurately (e.g., those corresponding to cameras oriented towards the sky). As the optimization is global, this may result in "corruption" of poses of other images. To avoid this problem, images with lack of texture are determined by thresholding on the fraction of high-frequency pixel information and are excluded from the optimization.

Efficiency

The dominant computing cost in the optimization is expended on traversing (and mapping) pixels in each image and accumulating global derivatives. Several techniques are used in the implementation to mitigate this cost. First, only pixels that actually map to a valid pixel in the adjacent image are traversed. This is achieved by computing a boundary for the overlap region from the warp, and traversing pixels only inside the boundary. Second, warped images and their derivatives are computed incrementally as an image is traversed, similar to texture mapping.

3.5 Mosaicing Results

In this section, I present both quantitative and qualitative results obtained on the dataset using the spherical optimization technique. Section 3.5.1 explores, using pairs of images, the sensitivity of the technique to initial values supplied to the optimization procedure. Section 3.5.2 presents results for full nodes in the form of several spherical mosaics computed using the algorithm.

3.5.1 Image Pairs

In these experiments, optimal calibration parameters and the rotation between two images were obtained by optimizing initial values provided by camera calibration and physical instrumentation. Then, each of these parameters was perturbed by some amount and the optimization was rerun. The metric used to measure the sensitivity is the number of iterations required to converge to the optimal (unperturbed) values. Optimizations that converged to a different minimum were discarded from the data. The results below show data obtained for two different adjacent image pairs of the dataset.

Figure 3-8 shows results obtained by perturbing the rotation angle (about the rotation axes) by a few degrees. Note that the optimization converges within a few iterations if the perturbation is on the order of a degree (the optimization fails to converge for larger angles with image pair B). In addition, using band-pass images reduces the number of iterations required for convergence.

Figure 3-9 shows results obtained by perturbing the focal length. The perturbation is expressed in terms of the fraction of the optimal value. Note that the algorithm is robust up to a few percent of error in the focal length. However, it fails to converge

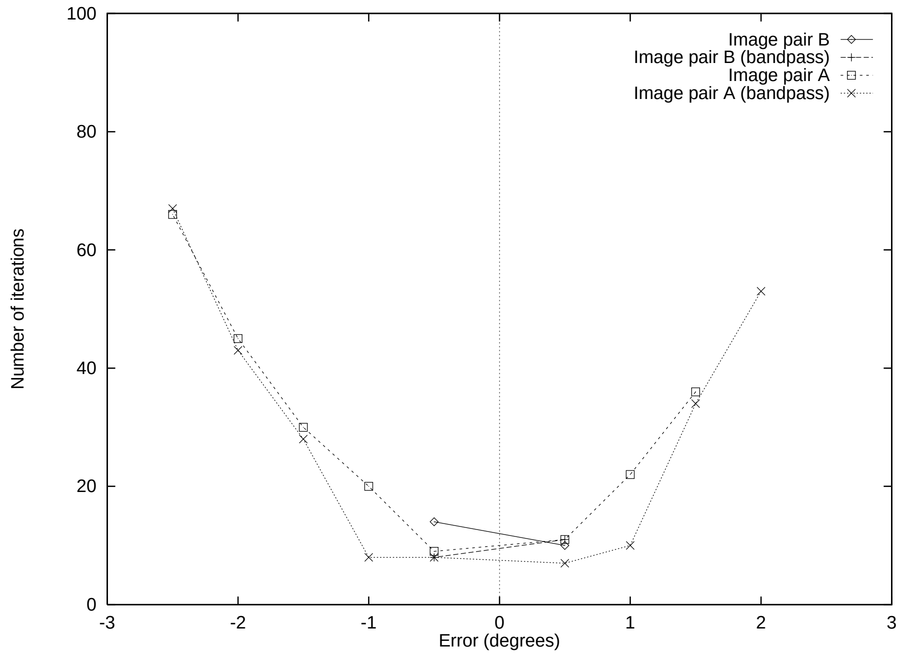


Figure 3-8: Convergence for different rotation errors.

for larger errors; fairly good camera calibration is thus required to provide initial focal length estimates. Note that a focal length estimate can also be provided by enforcing the 2π constraint [McM97].

Figure 3-10 shows results obtained by perturbing the image center (only results for the x coordinate is shown here). The optimization is very robust, converging to the optimal values even with errors of the order of a few tens of percentage points. This confirms the validity of the simple theoretical model described in Section 3.4.2. Also, note that the convergence behavior need not be symmetric between positive and negative errors: images are asymmetric in general, and shifts in the transformation by a few pixels to the "right" can yield very different results than shifts to the "left".

3.5.2 Node Optimization

Figure 3-11 shows convergence rates for images from two nodes. The graph plots total pixel luminance SSD error across all the images in a node across time. Note that the error decreases faster away from the optimum; this is consistent with the *quadratic* convergence rates guaranteed by LM optimization (near the minimum) [Sca85]. Also,

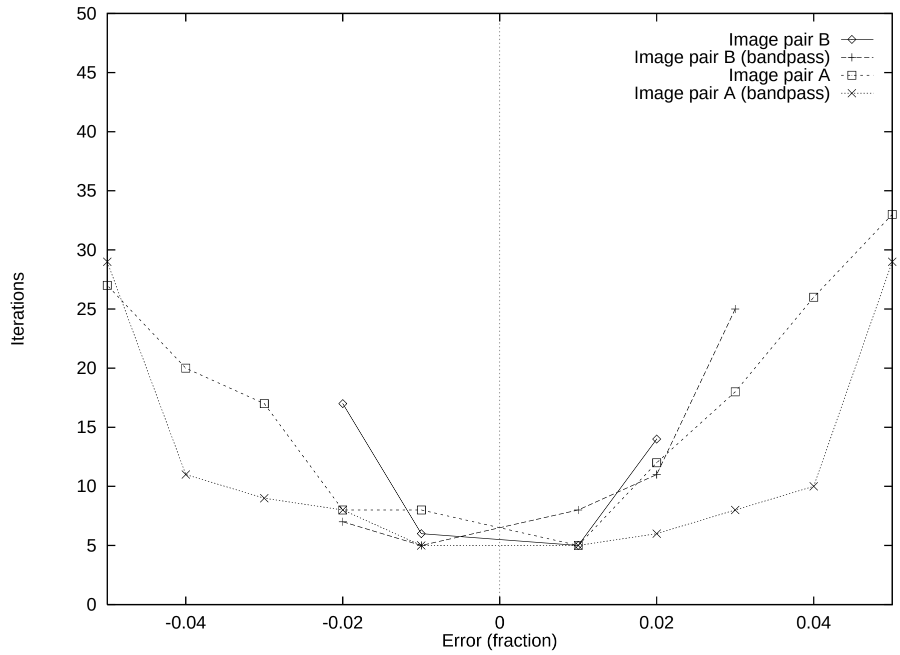


Figure 3-9: Convergence for different focal length errors.

due to slight errors in image formation and use of discrete sampling to estimate the optimum function, the optimization converges to a non-zero value. However, despite this, the mosaics generated using the optimal values do not exhibit any misalignment artifacts (see below), providing evidence that the images are registered accurately with respect to each other.

In a batch process, the algorithm was able to successfully process all (close to four thousand) input images[2] into eighty-one nodes, requiring about twenty minutes of processing per node for 381×253 pixel images and about two hours of processing for 762×506 pixel images (both on a 150MHz R10000 SGI O2 workstation). Internal camera parameters converged to values that differed by less than 1% across all nodes[3].

The relative rotations and camera parameters can be used to seamlessly blend all the images in a node to yield a single mosaiced image. Several mosaics resulting from the optimization are shown in Figures 3-12. Note that the input images are

[2]A few input images were unusable due to sun flare and/or CCD oversaturation.

[3]For 762×506 pixel images, focal length $f = 1015$ pixels; image center $c_x = 397$ and $c_y = 261$ pixels.

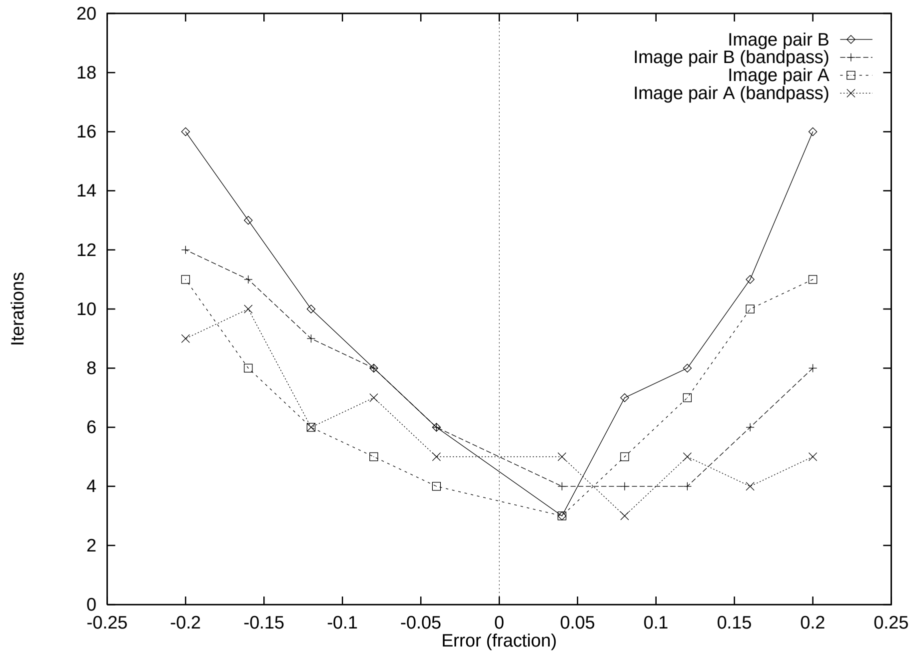


Figure 3-10: Convergence for different image center errors.

seamlessly blended[4], without any "blurring" or "ghosting" artifacts.

3.5.3 Mosaic Representations

Two alterate representations for the mosaics computed by the optimization algorithm are used in practice. The first (Figure 3-12) projects the sphere onto a flat surface by projecting a point (ϕ, θ) on a sphere to $(\phi, \sin\theta)$. As this projection preserves area of the sphere ([Gla90], page 319), it exhibits less distortion than the more commonly used (ϕ, θ) projection.

A disadvantage of the flat projection is that straight edges in images will map to curves in the projection (in general). This makes it difficult to detect straight edges in the mosaic (e.g., a building edge that spans several images) automatically. To ameliorate this problem, a second representation similar to cubical environment maps in computer graphics (see Figure 3-13) is used. The mosaic consists of six images corresponding to each face of the cube, and edge detection is performed independently

[4]During blending, pixels close to image centers are weighted more than pixels close to image boundaries.

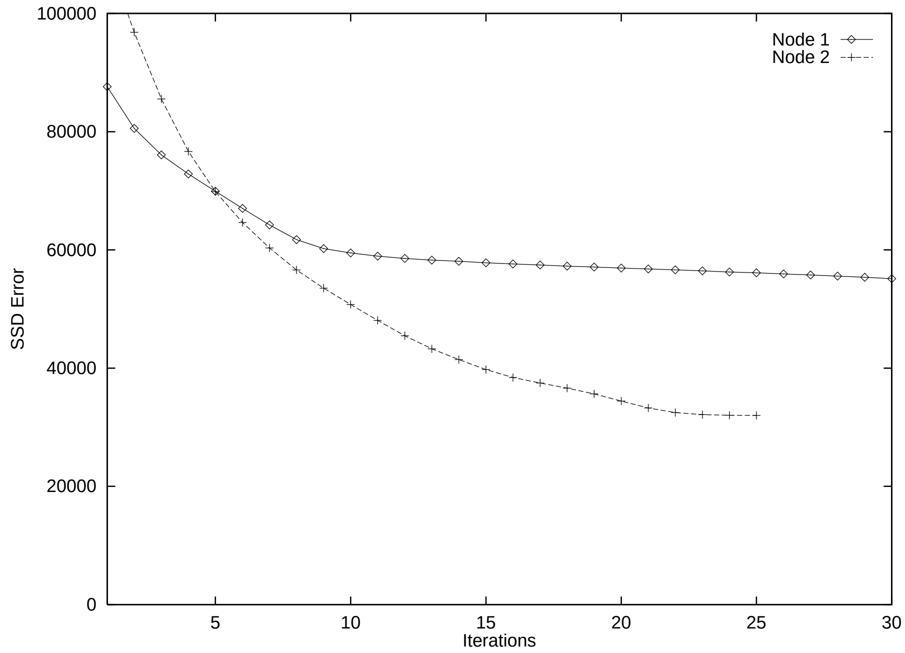


Figure 3-11: Convergence for different rotation errors.

on each face. The drawback of this representation, of course, is that edges spanning two cube faces are broken. However, as most edges are small relative to the 90 degree field-of-view provided by a cube face, this is not a problem in practice.

3.6 Summary

This chapter described two methods to recover relative rotations and internal camera parameters for the set of images in a single node. The first is a closed form solution using eigen vectors of 8-parameter warps. While this method is theoretically elegant, it yields quantitatively inaccurate results. The second method solves this problem by directly computing rotations and parameters from image-space correlation using a global optimization technique. The results (for over eighty nodes) demonstrate that this method is well suited for recovering pose relative to a node.

There are several benefits in performing the spherical mosaicing optimization. First, it allows robust automatic estimation of internal camera parameters. Second, it effectively produces an image with a spherical field of view, eliminating the ambiguity

Figure 3-12: Mosaicing Results

Figure 3-13: Four faces of a cubical projection for two nodes.

between camera motion and camera rotation found in narrow field-of-view images. Third, it reduces, by a factor of about fifty, the number of degrees of freedom when determining global positions and orientations by subsequent optimization (described in the next chapter).

Chapter 4

Global Mosaic Registration

In the previous chapter, I described a method to automatically compute relative rotations between images using a spherical mosaicing technique, generating mosaics for each node. The problem of computing accurate pose now reduces to computing a *single* position and orientation for each mosaic (rather than one for each image).

In this chapter, I present several techniques to perform *pose refinement*, i.e., determine accurate node poses from point correspondences across mosaics. The general strategy is to set up an appropriate objective function that captures errors in node pose and use iterative techniques to minimize this function. Correspondences for the minimization are supplied by a semi-automated system.

The minimization required to solve the pose refinement problem can be performed in two different ways. One possible method is to use projective space [MVQ93]: solve for 3×4 projective matrices and extract pose parameters from matrix values. Instead, the optimization is performed in Euclidean space due to the following reasons. First, this makes full use of available camera-calibration and pose information. Second, the use of the Euclidean framework decreases the complexity of the optimization by eliminating extra projective variables. Third, from a practical standpoint, it is much easier to visualize and debug algorithms operating in 3-D (using computer graphics) than algorithms that operate in projective space

For just two views, pose refinement is equivalent to computing relative orientation as in classical photogrammetry [Hor91]. While the photogrammetric technique performs well for pairs of images, a disadvantage of using this approach is that computing only pairwise pose may result in global inconsistency. Instead, I use global optimization to refine pose estimates, similar to the techniques proposed in structure-from-motion [SK94, TK95]. These approaches use correspondences (either manually specified [TK95] or obtained by tracking [SK94]) between image features to set up a global objective function, and perform an optimization using linear or nonlinear methods.

Formally, the pose refinement problem is as follows. Given:

- For $1 \leq i \leq m$, $\mathbf{p}'_i$ (position) and $\mathbf{q}'_i$ (orientation) estimates of the i^{th} node pose;
- For $1 \leq i \leq m$, $1 \leq j \leq n$, unit vectors $\mathbf{r}_{ij}$ (in node i's coordinate system) describing rays through the image feature corresponding to world-space point $\mathbf{s}_j$.

Compute:

- $\mathbf{p}_i$, $\mathbf{q}_i$ for $1 \leq i \leq m$ (the true node poses);
- $\mathbf{s}_j$ for $1 \leq j \leq n$ (the true 3-D feature positions).

The rest of the chapter describes various techniques to solve the pose refinement problem. Section 4.1 describes a linear optimization that solves for unknown node positions (but not orientations). Section 4.2 presents a nonlinear optimization that solves for both node positions and orientations. Finally, Section 4.3 describes the user aided system that generates point correspondences for optimization.

4.1 Position Estimation by Linear Optimization

In this section, I address the problem of recovering accurate node positions assuming known node orientations. This problem is important for the following reasons. First, in practice, it is possible to obtain fairly good orientation estimates without associated position estimates (e.g., by using only a compass and pan/tilt head without GPS

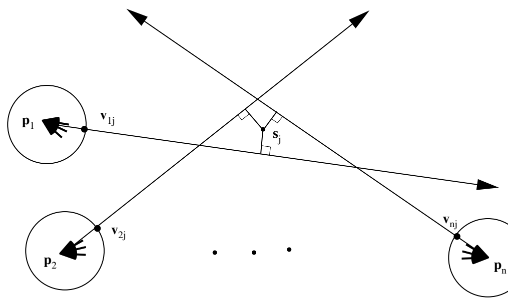


Figure 4-1: Reconstruction using least squares of distances.

instrumentation). Second, the problem can be formulated as a linear optimization, leading to a simple iterative solution.

Figure 4-1 illustrates (in 2-D) the idea behind the algorithm. Suppose all node poses are accurate, then all rays constructed by extruding image features would pass through the reconstructed 3-D point. Typically, due to error in the node pose estimates, they will diverge from the reconstructed point. This can be used to "correct" the node poses so that the rays are as close as possible to the 3-D reconstruction.

This idea can be formulated as a minimization of the following objective function:

$$O = \sum_{i=1}^{m} \sum_{j=1}^{n} \|(\mathbf{s}_j - \mathbf{p}_i) \times (\mathbf{R}_i^{-1}\mathbf{r}_{ij})\|^2 \tag{4.1}$$

Geometrically, this function represents the sums of the squared distances from reconstructed points to their corresponding rays (Figure 4-1). Note that this function is trivially minimized by setting all positions of 3-D points and nodes to the same value. This is due to the well-known rigid transformation ambiguity in computing structure-from-motion [TK95]. One way to avoid this problem is to fix one of the nodes as the origin and then enforce the following constraint:

$$\sum_{i=1}^{n} \|\mathbf{p}_i\|^2 + \sum_{j=1}^{m} \|\mathbf{s}_j\|^2 = 1 \tag{4.2}$$

which fixes the scale of the solution.

4.1.1 A Closed-Form Least-Squares Solution

Let $\mathbf{v}_{ij} = \mathbf{R}^{-1}\mathbf{r}_{ij}$ be the ray corresponding to node i and feature j in the global coordinate system. Then, a cross product with $\mathbf{v}_{ij}$ is equivalent to multiplication by the matrix $\mathbf{V}_{ij}$ determined by the components of the vector $\mathbf{v}_{ij}$:

$$\begin{bmatrix} 0 & \mathbf{v}_{ij\,z} & -\mathbf{v}_{ij\,y} \\ -\mathbf{v}_{ij\,z} & 0 & \mathbf{v}_{ij\,x} \\ \mathbf{v}_{ij\,y} & -\mathbf{v}_{ij\,x} & 0 \end{bmatrix}$$

Thus, the inner term of function O can be written as:

$$\|\mathbf{V}_{ij}(\mathbf{s}_j - \mathbf{p}_i)\|^2$$

Using the identity $\|\mathbf{a}\|^2 = \mathbf{a}^T\mathbf{a}$ yields:

$$(\mathbf{s}_j - \mathbf{p}_i)^T\mathbf{V}_{ij}^T\mathbf{V}_{ij}(\mathbf{s}_j - \mathbf{p}_i)$$

Now, consider a vector $\mathbf{B}$ of length $3 \times (n+m)$ formed by concatenating the vectors $\mathbf{p}_i$ for $1 \leq i \leq n$ and $\mathbf{s}_j$ for $1 \leq j \leq m$. Then, the optimization can be written as:

$$O = \mathbf{B}^T\mathbf{A}\mathbf{B}$$

where $\mathbf{A}$ is the $(3 \times (n+m)) \times (3 \times (n+m))$ matrix of the form:

$$\begin{array}{cccccc} & \ldots & i & \ldots & j & \ldots \\ \vdots & & & & & \\ i & & \sum_j \mathbf{V}_{ij}^T\mathbf{V}_{ij} & & \mathbf{V}_{ij}^T\mathbf{V}_{ij} & \\ \vdots & & & & & \\ j & & \mathbf{V}_{ij}^T\mathbf{V}_{ij} & & \sum_i \mathbf{V}_{ij}^T\mathbf{V}_{ij} & \\ \vdots & & & & & \end{array}$$

This is a quadratic form, whose minimization is given by $\mathbf{AB} = 0$. The constraint of Equation 4.2 is equivalent to the condition $\|\mathbf{B}\| = 1$. This to this constrained optimization is the eigen-vector of the matrix $\mathbf{A}$ corresponding to the least eigen-value [Str88]. Once this eigen-vector is determined, the components of the feature and node positions are directly given by the components of the eigen-vector.

While theoretically this procedure describes a closed form solution, all practical methods for computing eigen-vectors of large matrices use some form of iteration [PTVF92]. Also, these methods are quite expensive ($O(n^3)$ for a matrix of size n). To design a more practical solution, I exploit the special structure of the minimization and propose an iterative solution that has fast convergence in practice.

4.1.2 An Iterative Solution

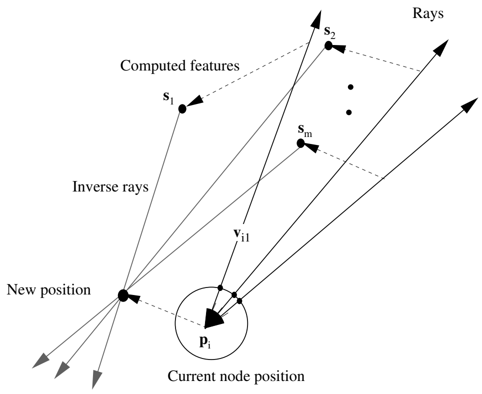


Figure 4-2: Position estimate using inverse rays.

The idea of the iterative method is to alternately compute and refine 3-D positions of the features and nodes. First, fixing the node positions decouples the optimization function into m functions, one for each feature. As in the previous section, a closed-form solution can be used to compute the location of point j by minimizing squared distances of $\mathbf{s}_j$ from the rays corresponding to each node. Also, as the matrix size for a *single* feature is small (3×3) the closed form solution can be implemented in practice to recover feature positions.

Similarly, as shown in Figure 4-2, it is also possible to do the inverse procedure, i.e., compute node positions assuming 3-D positions of the observed features are fixed. This is equivalent to finding the point that minimizes the squared distances from "inverse" rays through $\mathbf{s}_1 \ldots \mathbf{s}_m$.

The optimization consists of alternating these steps until convergence. Ordinarily, iterative procedures require good estimates for convergence. However, this optimization is linear. Hence, it is sufficient to initialize the iteration with arbitrary values; convergence to the unique minimum is guaranteed by linearity. In fact, this is analogous to using an iterative matrix solution approach (e.g., Jacobi iteration) to solve linear systems of equations, even though non-iterative solutions (e.g., Gaussian elimination) are available. This property is also illustrated in the empirical results shown below.

Empirical Results

In this experiment, ten randomly chosen points and nodes (in a box of size 10,000 units) were used to test the performance of the iterative procedure. Each point was projected onto each node, and correspondences between projections across different nodes were established. The node positions were then perturbed by different amounts producing new locations for each point. The procedure described was used to "update" node positions and point locations based on the generated correspondences.

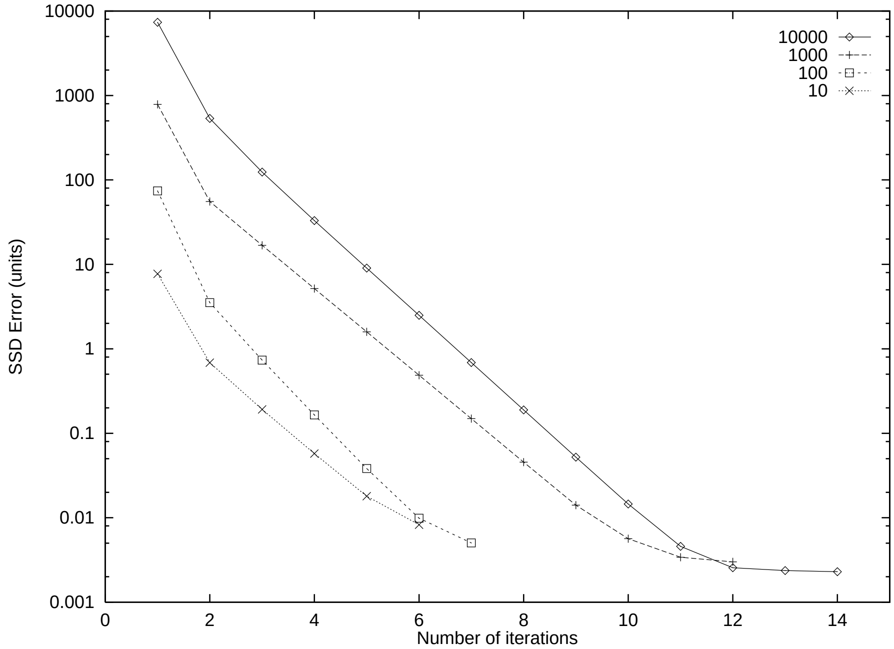


Figure 4-3: Convergence of iterative solution.

Figure 4-3 shows the results of this experiment. Note that convergence is extremely rapid. Even for node position errors of 10,000 units (i.e., initializing the optimization with completely random node positions), the algorithm converges within a few iterations. This algorithm is therefore effective in practice for recovering accurate node positions from correspondences (and known node orientations).

4.2 Position and Orientation Estimation

In this section, I describe a solution to the pose refinement problem that recovers *both* node position and orientation estimates. As this is a nonlinear optimization, I use Levenberg-Marquardt optimization to minimize the objective function described below.

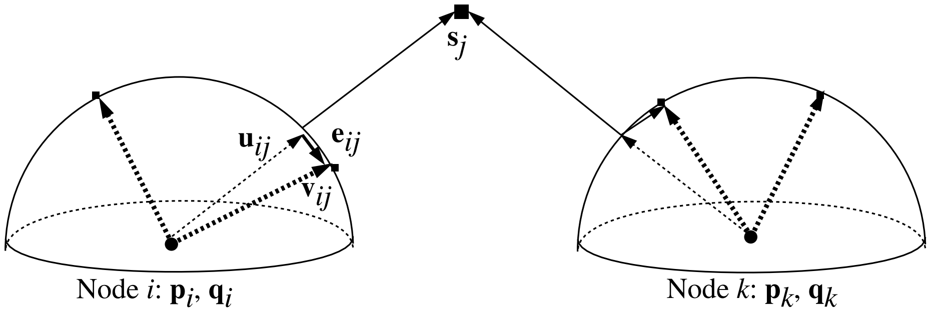


Figure 4-4: Parameters in the error function

Let $\mathbf{u}_{ij} = \frac{\mathbf{s}_j - \mathbf{p}_i}{\|\mathbf{s}_j - \mathbf{p}_i\|}$ represent the world-space unit vector directed from node i to point $\mathbf{s}_j$ (see Figure 4-4). The same vector predicted by using the image feature is $\mathbf{v}_{ij} = \mathbf{R}_i^{-1}\mathbf{r}_{ij}$, where $\mathbf{R}_i$ is the 3×3 rotation matrix equivalent to $\mathbf{q}_i$. Their difference in the spherical image, $\mathbf{e}_{ij} = \mathbf{u}_{ij} - \mathbf{v}_{ij}$, is the residual error vector of predicting ray $\mathbf{u}_{ij}$. The objective function O is simply the sum of the squared magnitudes of the residual vectors:

$$O = \sum_{i=1}^{m} \sum_{j=1}^{n} \|\mathbf{e}_{ij}\|^2$$

Note that, in contrast to the objective function used for position refinement (Equation 4.1), this function measures error in image space. As this measures error in the observed quantities (i.e., mosaic feature locations), this is a better choice for final refinement [Hor91].

To perform the optimization, I use an approach similar to "bundle-adjustment" in photogrammetry [Sla80] that alternately refines 3-D positions and pose estimates. The advantage of this method is that fixing the 3-D positions of point features decouples the optimization of the various node poses, each of which can be independently updated (and vice-versa). Thus, an efficient implementation of the optimization involves inverting only small constant-sized matrices. Experiments show that the optimization requires only a few iterations, making it usable in an interactive system.

The derivatives of a single residual term (omitting subscripts i and j) are computed as follows. Let $\mathbf{I}$ be the 3×3 identity matrix. Then, the gradient $\mathbf{G_s}$ and Hessian $\mathbf{H_s}$ with respect to a 3-D point $\mathbf{s}$ are:

$$\frac{\partial \mathbf{e}}{\partial \mathbf{s}} = \frac{\mathbf{I} - \mathbf{u}\mathbf{u}^T}{||\mathbf{s} - \mathbf{p}||}$$

$$\mathbf{G_s} = (\tfrac{\partial \mathbf{e}}{\partial \mathbf{s}})^T \mathbf{e} \quad \mathbf{H_s} = (\tfrac{\partial \mathbf{e}}{\partial \mathbf{s}})^T \tfrac{\partial \mathbf{e}}{\partial \mathbf{s}} \tag{4.3}$$

The gradient $\mathbf{G_{p,q}}$ and Hessian $\mathbf{H_{p,q}}$ with respect to pose parameters $\mathbf{p}$ and $\mathbf{q}$ are:

$$\tfrac{\partial \mathbf{e}}{\partial \mathbf{p}} = \tfrac{\mathbf{u}\mathbf{u}^T - \mathbf{I}}{||\mathbf{s} - \mathbf{p}||} \quad \tfrac{\partial \mathbf{e}}{\partial \mathbf{q}} = -(\tfrac{\partial \mathbf{R}^{-1}}{\partial \mathbf{q}})\mathbf{r} \quad \mathbf{D} = \begin{bmatrix} \tfrac{\partial \mathbf{e}}{\partial \mathbf{p}} & \tfrac{\partial \mathbf{e}}{\partial \mathbf{q}} \end{bmatrix}$$

$$\mathbf{G_{p,q}} = \mathbf{D}^T \mathbf{e} \quad \mathbf{H_{p,q}} = \mathbf{D}^T \mathbf{D} \tag{4.4}$$

To update the position of a 3-D point $\mathbf{s}_j$, the gradient $\mathbf{G}_{\mathbf{s}_j}$ and the Hessian $\mathbf{H}_{\mathbf{s}_j}$ are computed by accumulating Equation 4.3 over all nodes that "see" $\mathbf{s}_j$. To update the pose $\mathbf{p}_i, \mathbf{q}_i$ of node i, the gradient $\mathbf{G}_{\mathbf{p}_i,\mathbf{q}_i}$ and Hessian $\mathbf{H}_{\mathbf{p}_i,\mathbf{q}_i}$ are computed by accumulating Equation 4.4 over all points visible to node i. As in the spherical mosaicing optimization of Chapter 3, Lagrange multipliers are used to enforce the unitary constraint for quaternions.

4.2.1 Results

To test convergence of the optimization, I performed the experiment in Section 4.1.2, except that *both* orientations and positions of the node were perturbed by different quantities.

Figure 4-5 shows results of this experiment. The figure shows only convergence with respect to different orientation errors, as the convergence rate is insensitive to position errors. Note that the optimization converges to the correct error even for very significant errors in orientation (around 10 degrees), and reaches false minima only for very high errors (i.e., 27 degrees in Figure 4-5).

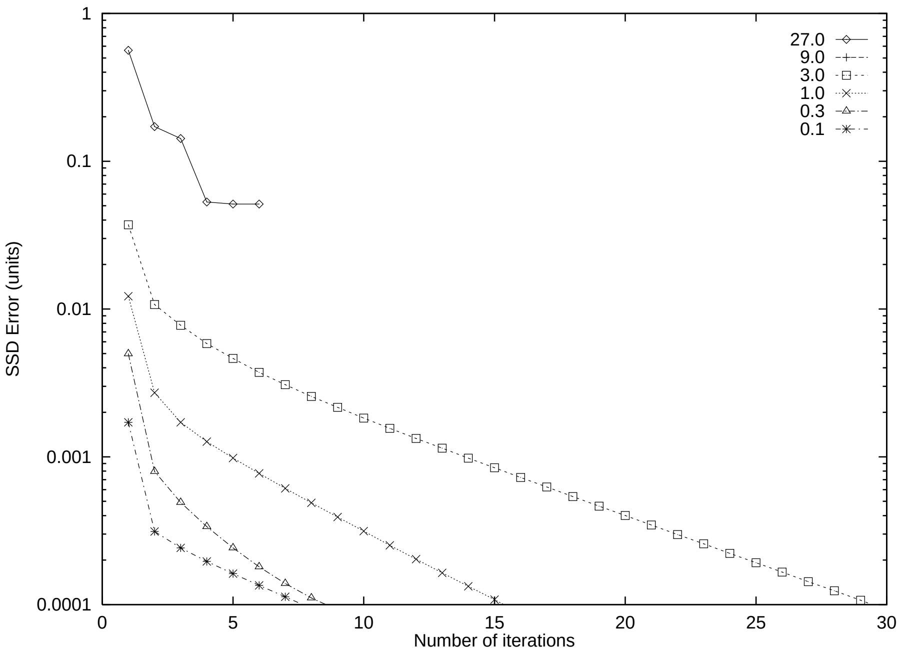


Figure 4-5: Convergence of pose refinement.

For the real dataset, typical residuals after optimization are about 0.001, equivalent to 0.05 degrees of error in predicting a ray, and to about 2.5 cm of error in predicting the location of a 3-D point about 25 meters from the node. This is about an order of magnitude more accurate than errors due to initial estimates. Also, as the optimization is initialized with pose estimates from instrumentation, convergence is fairly fast (a few tens of iterations).

4.2.2 Constrained Orientations

Figure 4-6: Vertical edges in a node.

In some cases, it is possible to derive estimates of node orientation from the information contained in the images themselves. This can be used to constrain the pose refinement optimization further. For example, as shown in Figure 4-6, there usually is an abundance of vertical edges in urban environment. Detecting them is straightforward with global pose estimates (such as those supplied by instrumentation). These vertical edges can be used to compute the vertical vanishing point [BO91] and align the mosaic. The only remaining degree of freedom in the orientation of a node is a rotation about the vertical axis. This can be used instead of a quaternion in the pose refinement optimization.

4.3 Correspondence Generation

All the techniques described thus far require point correspondences to perform pose refinement. While the problem of automatically generating correspondences has been extensively studied in computer vision, the process tends to be very fragile, especially across disparate images. The large inter-node distance in the dataset (with baselines of tens of meters) produces fairly dissimilar images due to perspective and occlusion effects. In addition, different nodes are acquired under different lighting conditions, further accentuating the dissimilarity.

Fortunately, as we are interested primarily in recovering accurate pose for each node, very few correspondences are necessary (five points per *mosaic* rather than per image, as in [Hor91]). Thus, the system allows a user to manually correspond points computed by intersecting adjacent straight edges obtained by using the Canny edge detector [Can86]. The user's task is further simplified by the system, which uses the available pose information to generate matches across adjacent nodes automatically in most cases.

4.3.1 User Interface

The system presents two views simultaneously to the user. In the 2-D view (Figure 4-7), the user can select a few mosaics (up to eight) to view. The user establishes correspondences across mosaics by either selecting a pixel in the image or by choosing a predetermined point feature. In addition to manual correspondence, the system uses a heuristic based on 3-D ray proximity to match selected features, when possible (as

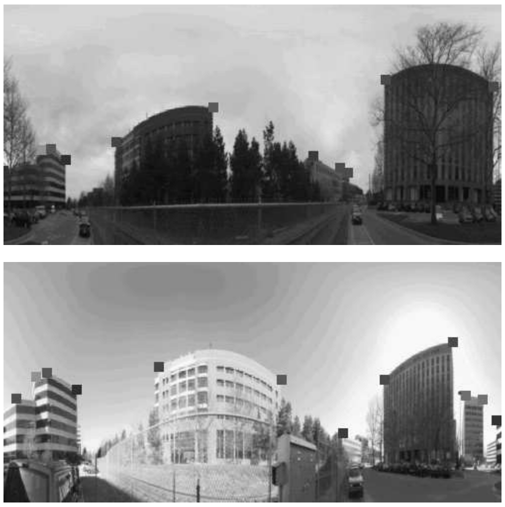


Figure 4-7: User selected mosaics and correspondences.

described in the next section).

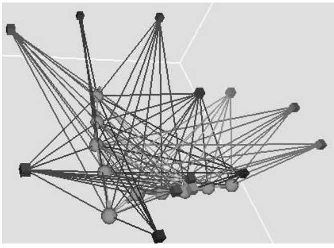


Figure 4-8: Three dimensional view of the optimization showing ten nodes (spheres) and computed feature positions (cubes).

The correspondences established are used to compute feature locations; these are displayed along with node positions in the 3-D view (Figure 4-8). The user then can interactively refine node positions.

4.3.2 Automatic Matching Heuristic

In this section, I describe a technique that exploits the availability of pose-information to aid the user in establishing correspondences across nodes. The basic idea, similar to that proposed by Collins [Col96], is as follows. If any *sparse* set of points in a set of nodes is projected into 3-D rays, regions with high *incidence* of rays correspond to likely locations of 3-D features. If a 3-D feature is present in multiple nodes, then rays through the corresponding 2-D points pass near the 3-D feature, increasing its incidence count. Conversely, as the set of points are sparse, it is rare that unrelated rays pass through the same 3-D region by chance.

The matching algorithm first defines a notion of adjacency using a 2-D Delaunay triangulation of node positions projected onto the ground plane[1]. Given a nearness threshold T, the match for a selected point $\mathbf{r}$ of node i is generated as follows:

For each $\mathbf{s}_j$ in decreasing order of incidence
    If $\|\mathbf{e}_{ij}\| < T$ associate $\mathbf{r}$ with point $\mathbf{s}_j$.
If $\mathbf{r}$ is not matched
    Let minimum error $e = T$, closest 3-D point $\mathbf{s} = \phi$.
    For each Delaunay neighbor node i' of i
        For each unmatched point $\mathbf{r}'$ of node i'
            Generate $\mathbf{s}_k$ closest to rays $\mathbf{r}$ and $\mathbf{r}'$.
            If $\|\mathbf{e}_{i'k}\| < e$ then $e = \|\mathbf{e}_{i'k}\|$; $\mathbf{s} = \mathbf{s}_k$.
    If $\mathbf{s} \neq \phi$ add $\mathbf{s}$ to the 3-D point set.

Informally, given a newly selected point $\mathbf{r}$, the algorithm first searches for an existing high incidence 3-D point $\mathbf{s}_j$ that projects close to $\mathbf{r}$. If such a point is not found, the algorithm matches $\mathbf{r}$ with a close unmatched point $\mathbf{r}'$ from a neighboring node i'.

It is possible for this heuristic to generate false positives, i.e., report a match between two rays where none exists. This occurs primarily due to errors in initial pose estimates. To solve this problem, the matching and optimizing steps are alternated with an *outlier detection* step [Hub81] that rejects matches that have high residual error (e.g., greater than 3 times the standard deviation error of all matches [SK94]).

To test the matching algorithm on the dataset, a user interactively selected five or more points (typically, building corners) in each mosaic. The algorithm automatically inferred sufficient number of matches to generate pose for over 95% of the nodes; for the remaining nodes, the user established correspondences manually. The user

[1] This spatial notion of adjacency is necessary as the usual temporal notion used in structure-from-motion is not applicable to the dataset, where nodes acquired at very different times can observe nearby 3-D regions.

required about one hour to process the entire dataset (eighty-one mosaics comprising nearly four thousand images), i.e., less than a second per image.

4.4 Summary

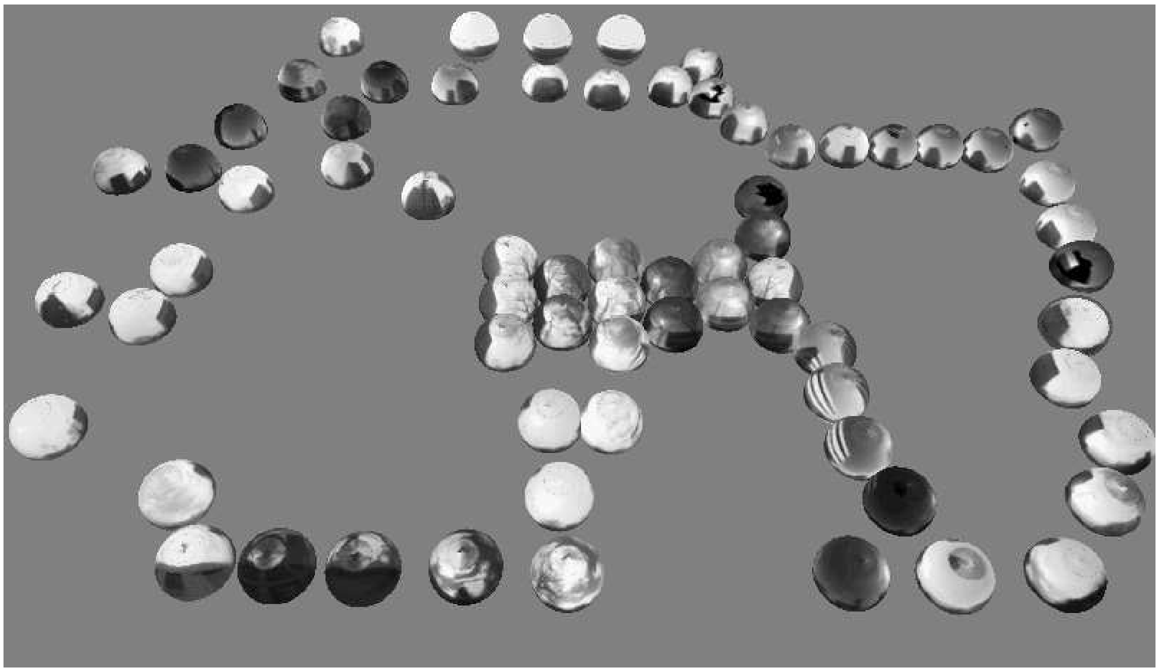


Figure 4-9: The (refined) pose mosaics comprising the dataset.

This chapter described optimization and correspondence techniques for recovering accurate pose of a large collection of mosaics. I first presented a linear optimization technique that recovers unknown node positions, assuming node orientations are given. I also described a more general optimization that recovers both node positions and orientations. This optimization, coupled with an automatic matching heuristic based on ray proximity, was successfully used to locate all eighty-one nodes in a global coordinate system. Figure 4-9 shows the generated pose mosaics of the dataset, which can now be used in the reconstruction algorithms described in the succeeding chapters.

Chapter 5

Vertical Facade Extraction

Thus far, I described techniques that accurately determine poses of mosaics. This chapter describes an algorithm that extracts vertical facade geometry observed in the various mosaics.

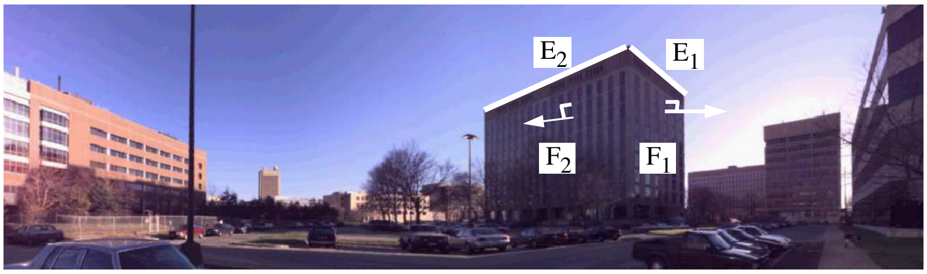


Figure 5-1: A single mosaic of the dataset showing dominant vertical facades.

Figure 5-1 illustrates the idea behind the approach followed here. First, as the figure demonstrates, vertical facades occur frequently in the dataset; this is typically true of other urban environments. While such facades do not constitute *all* the geometry in such environments, they provide a good initial estimate of the geometry, and thus can be used to initialize future, more general extraction algorithms.

Second, it is straightforward to determine the *orientation* of vertical facades (i.e., azimuth) from a *single* pose mosaic. To the human eye, it is obvious that the two facades in the center of Figure 5-1 are oriented differently. A major cue in providing this perception is that the images of *horizontal edges* from the two facades possess different (2-D) orientations in the image (assuming that the horizontal edges are parallel to the ground plane). In fact, as I demonstrate, a simple analytical expression can be used to derive 3-D facade orientations from such 2-D information. While this technique does not provide any information about facade locations, I show that a space sweep technique based on detected orientations that fuses information in several mosaics can recover vertical facade locations.

Overview

Figure 5-2 shows a high-level overview of the algorithm. The algorithm executes in communication with the acquired (and refined) pose image database, which stores raw data (e.g., pose and images) as well as derived data (e.g., detected edges).

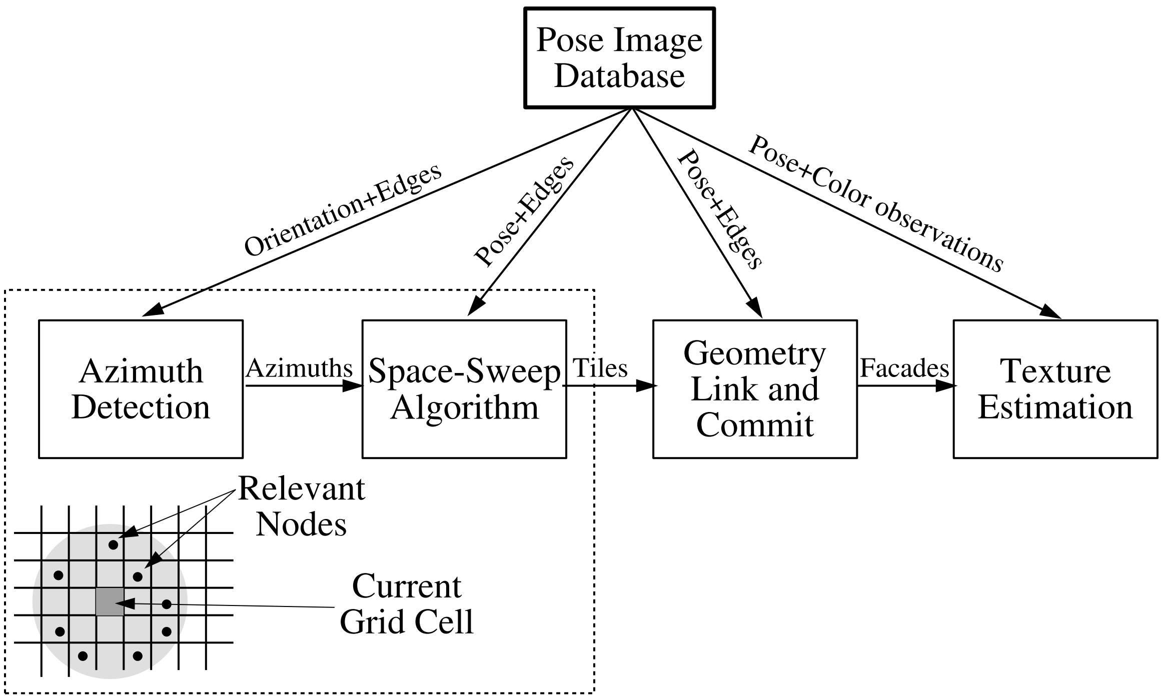


Figure 5-2: Overview of the Vertical Facade Extraction Algorithm.

The algorithm divides the 3-D region of interest into a 2-D XY grid based on a user-supplied grid size G. Grid-based subdivision enables restriction of subsequent processing to nodes that are *relevant* to the grid cell (e.g., within some world-space distance D of the cell). This subdivision also decouples different parts of the scene, making the reconstruction process more robust (see Section 5.2.2 for more details).

For each grid cell, the algorithm computes *tiles* – pieces of vertical facade – in a two step process. Likely tile azimuths are detected based on a histogram technique (Section 5.1). These are verified and located using a space-sweep technique (Section 5.2), populating the grid cell with only those 3-D tiles that are supported by sufficient image evidence. The tile geometry recovered is then linked to form complete facades; most spurious facades in the model are removed by a *facade commitment* process (Section 5.3).

Section 5.4 reports results of the algorithm on a large pose image dataset, and Section 5.5 discusses some of the limitations of the algorithm in its current form.

5.1 Azimuth Detection

This section describes a basic technique used to identify likely azimuths of tiles, using horizontal line segments. Such line segments arise often in urban environments, e.g., from windows and facade boundaries. First, I briefly review existing techniques in computer vision that recover facade azimuths from a single image. I then describe a simple technique to uniquely determine the azimuth of a tile using image-space edge information (Section 5.1.2). This technique is then applied to many *presumed* horizontal edges, and likely azimuths are identified by a histogram technique (Section 5.1.3).

5.1.1 Shape from Texture

Recovering 3-D shape from textured images is the topic of many papers in vision (see [Bar83, BM90, Kan84, Wit81]). Another related method is recovering 3-D shape using *vanishing points* [BO91] of sets of parallel lines.

Witkin's algorithm [Wit81] uses the distribution in the image of oriented textured elements (lines) to recover 3-D orientation based on the assumption that textured elements are isotropically distributed on a plane. Instead of a statistical method, Kanatani [Kan84] computes image attributes such as *moments* to recover 3-D shape. Blake et al. [BM90] design a hybrid method that uses both statistical- and momentbased techniques. Barnard [Bar83] computes orientations of planes assuming they contain regular structures (e.g., equilateral triangles and rectangles). However, his algorithms require large perspective effects to recover the plane orientations robustly.

A common theme in many approaches to shape from texture is the use of *aggregate* properties (i.e, statistics or moments or vanishing points) to recover 3-D shape. This has two consequences in practice. First, as the assumptions (e.g., isotropy) made to recover shape are only approximately true, the recovered orientation can have significant error (around 10–15 degrees in [BM90]). Second, if the image consists of several planar surfaces, it is necessary to segment the image into different regions corresponding to different planar surfaces. The size of the segment should be large enough to support meaningful aggregation, but small enough so that each segment is predominantly affected by a single planar region. Thus, it is difficult to choose the segment size and Witkin [Wit81] selects it manually. For these reasons, this approach has yielded only *qualitatively* correct results for 3-D orientation, and thus is not suited for use in a rendering system where quantitative properties are important.

In contrast, the method proposed here exploits 1) the availability of camera pose estimates and 2) the special nature of the planes/texture being analyzed to compute accurate 3-D orientation from a *single* (presumed horizontal) line segment. The accuracy of this method is limited only by the accuracy of 2-D line orientation estimation and of the estimated camera parameters. While this technique is applicable only for vertical facades, it should be effective for typical urban environments.

Recently, a similar technique has been used in photogrammetry to generate buildings from *monocular* (i.e., single) aerial images [MMC+97]; the novelties of this approach are 1) the use multiple nodes to discount non-correlated azimuths, and 2) robust azimuth verification, which are described in the succeeding sections.

5.1.2 Estimating Azimuth from a Horizontal Line Segment

The basic idea in azimuth estimation is that, when pose information (specifically, orientation) is known, the direction of a 3-D horizontal edge is completely determined by the image-space 2-D line equation of its projection. Figure 5-3 illustrates this idea.

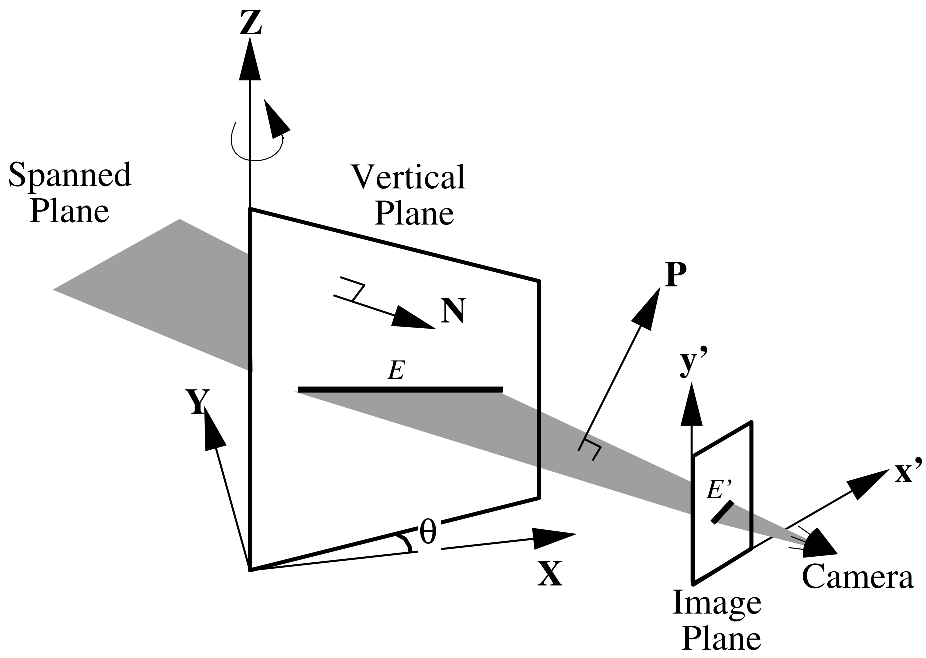


Figure 5-3: Deducing tile azimuth from a horizontal line segment.

Let a 3-D horizontal line segment E on a tile project to a 2-D edge E' in some pose image. In the figure, the vertical plane through E makes an angle θ with the $\mathbf{X}$ axis, i.e., its *azimuth* is θ. The normal $\mathbf{N}$ to this plane is $[\sin\theta, -\cos\theta, 0]^T$. Let $\mathbf{P} = [p_x, p_y, p_z]^T$ be the normal to the plane spanned by E' and the camera. $\mathbf{P}$ (in global coordinates) is determined by the orientation of the camera (a 3×3 rotation matrix $\mathbf{R}$) and the 2-D image-space line equation $ax' + by' + c = 0$ of E' (where $a^2 + b^2 + c^2 = 1$):

$$\mathbf{P} = \mathbf{R}^{-1} \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

Then, the direction of E is given by:

$$\mathbf{P} \times \mathbf{N} = [p_z \cos\theta, -p_z \sin\theta, (-p_x \cos\theta - p_y \sin\theta)]$$

As E is horizontal, the z component should vanish. This yields two solutions for the orientation (when either $p_x \neq 0$ or $p_y \neq 0$):

$$\theta = \begin{cases} \tan^{-1} \frac{-p_x}{p_y} & \text{or} \\ \tan^{-1} \frac{-p_x}{p_y} + \pi \end{cases} \tag{5.1}$$

Of these, the correct azimuth is the one that faces the observing camera. If $p_x = 0$ and $p_y = 0$, the camera has observed a horizontal line segment at the same height as itself, and no information can be determined. As most of the nodes in the dataset are taken from positions near ground level, this case arises very rarely.

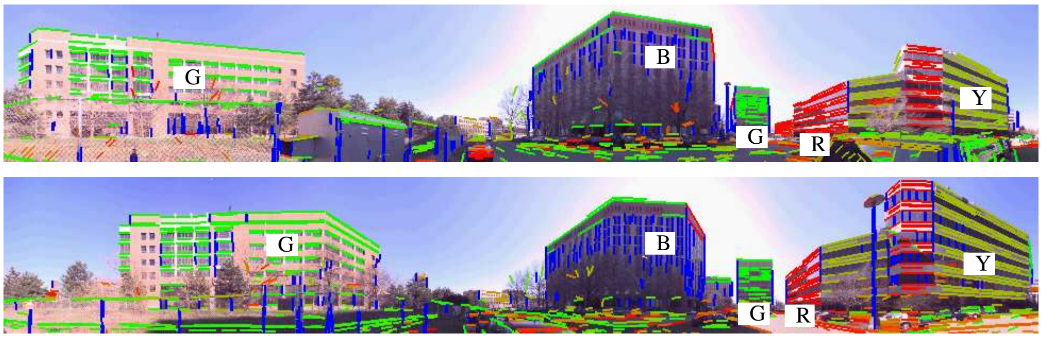


Figure 5-4: This figure shows two nodes (four faces of a cubical environment map) along with their (long) edges. Vertical edges are colored *blue* (B), and other edges are colored with (absolute values of) the tile normal derived from their azimuth (e.g., *red* (R) is $[1, 0, 0]^T$, *green* (G) is $[0, 1, 0]^T$, etc.).

Figure 5-4 shows the results of applying this technique to two nodes. In this figure, vertical edges were first identified by thresholding (rejecting those with $p_z < 0.01$). For the rest, an azimuth θ was estimated using Equation 5.1. Note that horizontal edges from the same facade are assigned the same azimuth, even across nodes. This property is not true of azimuths computed from non-horizontal edges (e.g., tree edges); thus they tend to be uncorrelated both within a node and across different nodes.

5.1.3 Azimuth Histograms

The technique to identify likely azimuths is based on the following idea: 2-D edges arising from truly horizontal line segments will be assigned identical azimuths in different nodes, whereas azimuths of non-horizontal edges will vary with node position. Thus computing histograms reinforces true azimuths; the rest tend to be unsupported, as they are uncorrelated across nodes.

This idea is used in the following algorithm, which reports a set of likely tile azimuths in a grid cell C, from edges in C's relevant nodes $1 \ldots k$:

Azimuths $A = \phi$.

for node $i \in 1 \ldots k$ **do**

Let C project to image-space region C' in node i.

Compute tile azimuths of each non-vertical edge in C' using Equation 5.1.

Add azimuths, weighted by image-space edge length, into histogram buckets[1].

Identify the most populated bucket and add its representative (median) azimuth to A.

endfor

Compute a histogram for the azimuths in A and report each bucket that contains at least three nodes.

Informally, the algorithm picks a dominant azimuth from each node, then reports azimuths that are supported by several nodes. These azimuths are then verified by the space-sweep algorithm described in the next section.

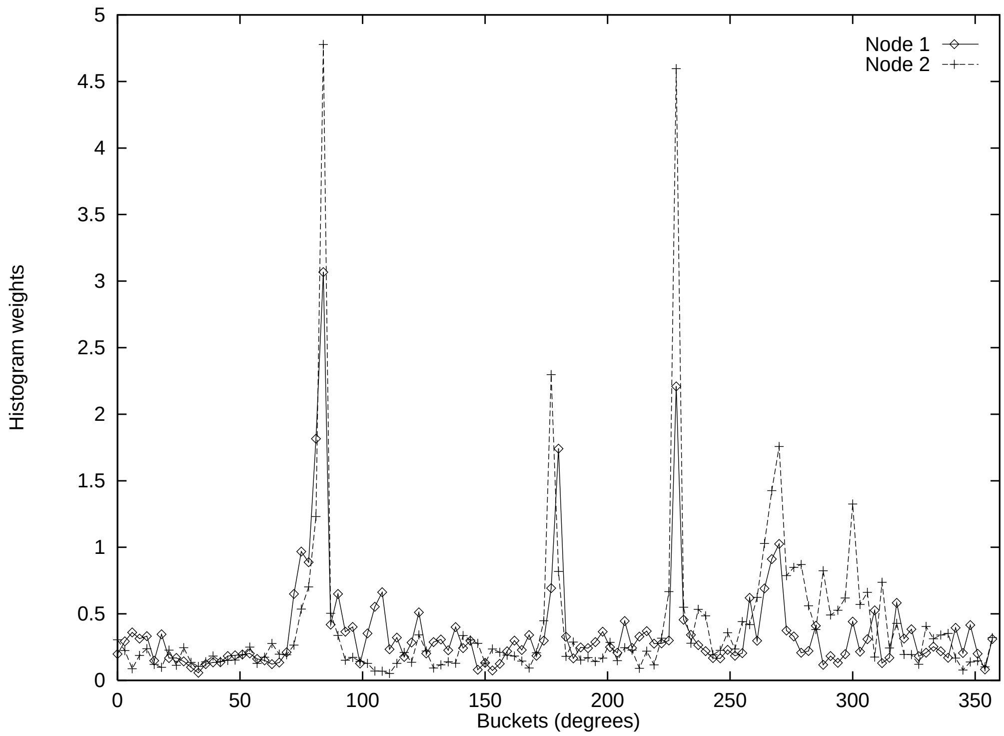


Figure 5-5: Histogram values for two nodes.

[1] I use overlapping θ buckets of size 3 degrees (the overlap is 50%).

Figure 5-5 shows histogram values for the two nodes in Figure 5-4. Note the significant correlation both within nodes (i.e., there are well defined peaks in the histogram) and across nodes (i.e., the peaks coincide).

5.2 The Space-Sweep Algorithm

The space-sweep algorithm to locate and verify tiles is based on an *incidence counting* idea, also used in generating correspondences (Chapter 4). The idea is to project a sparse set of features to 3-D, and search for spatial correlation among projections from different nodes. Unlike feature points used in Chapter 4 and edgels used by Collins [Col96], horizontal line segments are used in the space-sweep algorithm.

There are several advantages in using line segments, instead of points or edgels for extraction. First, as points are limited in spatial extent, they are hard to localize. Second, they are very sensitive to occlusion events, and may not be present in an image altogether. Edges have larger spatial extent and are less sensitive to occlusion and thus ameliorate both these problems. Also, edges (lines) carry additional information (orientation) that can be used effectively in the matching process.

Given an azimuth θ, the sparse features in the algorithm are edges that are likely to lie on vertical tiles with azimuth θ. Note that it is straightforward to identify such edges in each node; they are edges whose computed azimuth is θ. For example, in Figure 5-4, red edges are used for tiles with azimuth=$-\frac{\pi}{2}$ (i.e., normal = $[-1, 0, 0]^T$).

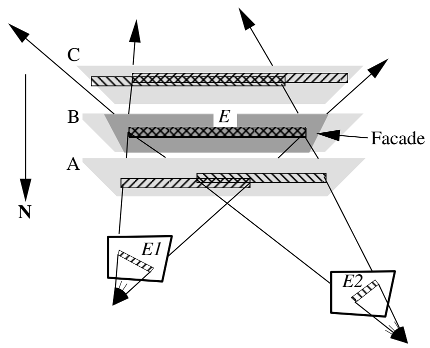
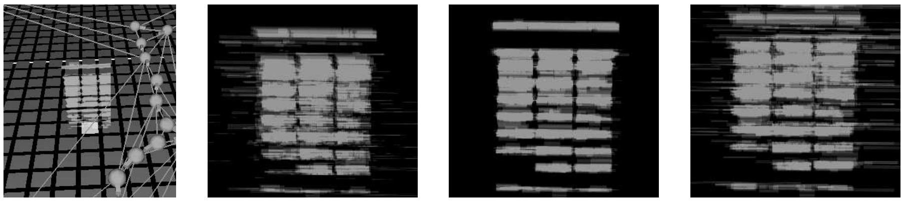


Figure 5-6: This schematic figure shows three different positions A, B, and C of a vertical plane with horizontal edge projections from two nodes.

Using such edges, Figure 5-6 shows the application of incidence counting to tile location. It shows three planes with common normal $\mathbf{N}$ and different offsets, with projected (and blurred) edges $E1$ and $E2$ from two nodes. Note that correlation between horizontal line segments is greatest at the position of the plane which corresponds to the tile location. Also, the segment endpoints need not coincide (e.g., due to occlusion) for the correlation to high – the function described below measures *overlap* between segments.



A B C

(a) (b)

Figure 5-7: Part (a) shows a plane positioned at its peak generated by line segments from a facade in the dataset (along with some of the nodes). Part (b) shows a close-up of this plane (B), and two additional planes positioned before (A) and after (C) the peak. Correlation between different line segments is indicated by brightness.

Figure 5-7 shows close-ups of three similar planes, but with several line segments from a real facade. In the figure, segments on the plane are blended together, so that regions with high incidence (and therefore high correlation) appear brighter. Note that the image of the plane appears brightest and sharpest at position B, i.e., when the plane is at the facade's actual location (offset). The technique to locate tiles is based on this idea. The implementation uses a correlation function defined in Section 5.2.1, and a space-sweep algorithm described in Section 5.2.2 that locates tiles using the maxima of this function.

5.2.1 Correlation Function

Given a plane and its associated horizontal line segments, the correlation function computes a measure of the extent of overlap between different line segments on the plane. Each line segment L is blurred into a *rectangle* of half-width σ_L to alleviate small errors in camera pose and smooth the peaks of the correlation function[2].

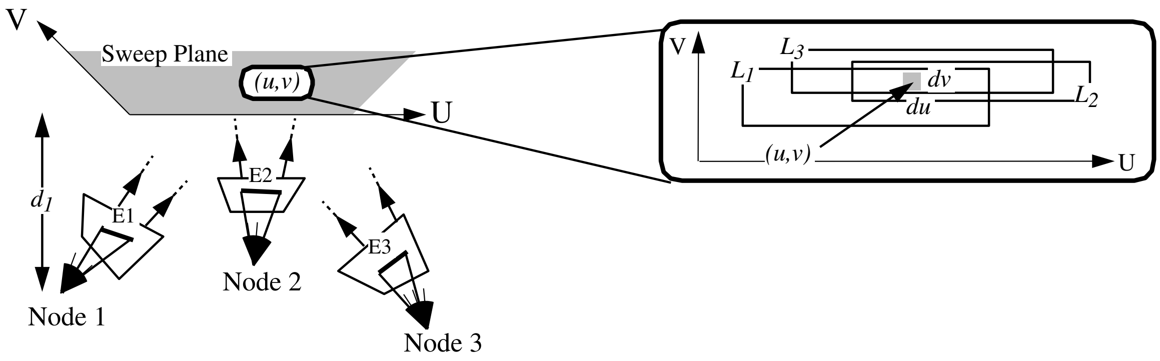


Figure 5-8: Parameters in the correlation function. The figure on the left shows edges $E1$, $E2$, and $E3$ observed (respectively) from nodes 1, 2, 3 and projected onto a common plane. The figure on the right is a blowup of a small section of the plane. It depicts overlap of the rectangles L_1, L_2, and L_3 corresponding to the edges.

$$\text{O} = \int\int \sum_{i=1}^{k} w_i(u,v) \sum_{i=1}^{k} \frac{1}{d_i^2} dudv \tag{5.2}$$

Consider the function defined in Equation 5.2 (see also Figure 5-8). In this equation, u and v range over the plane. The point (u, v) is inside rectangles $L_1 \ldots L_k$ arising from projections of edges observed at nodes $1 \ldots k$; the perpendicular distance from the plane to node i is given by d_i; and $w_i(u, v)$ is the weight contributed by L_i to plane element (u, v). The implementation uses a triangular filter for $w_i(u, v)$, as it is both efficient to evaluate and possesses a well-defined peak (Figure 5-9):

$$w_i(u,v) = \max(1 - \frac{|v - L_v|}{\sigma_L}, 0)$$

This correlation function has several properties that make it useful for locating vertical tiles. First, the function favors overlap between line segments arising from different nodes: "cross-terms" w_i/d_j^2 such that $i \neq j$ arise when L_i overlaps with L_j,

[2]I use $\sigma_l = 0.01d$, where d is the perpendicular distance between the node generating l and the vertical plane.

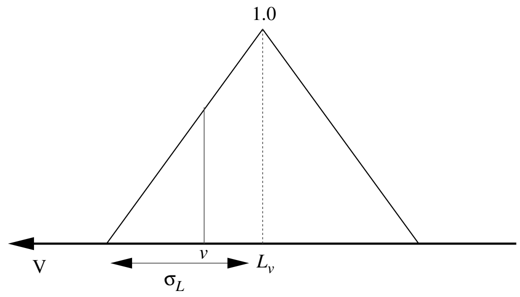


Figure 5-9: Triangular filter for edge overlap computation.

and contribute to O. An example is shown in Figure 5-10, in which Equation 5.2 was evaluated for different plane offsets using the line segments shown in Figure 5-7. Note that, due to the combined effect of many line segments, the function has a well-defined peak at the facade's location.

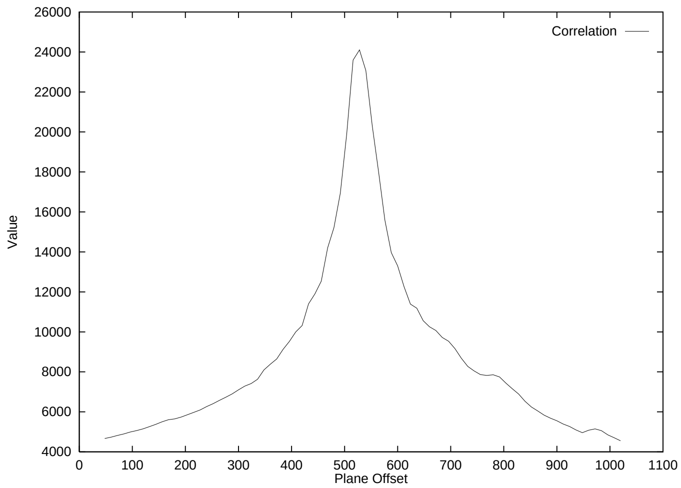


Figure 5-10: Correlation function values.

Second, there is no built-in bias toward planes of larger area; instead, the function downweights the area of each plane-element (u, v) by the squared-distance from the source node. This is advantageous, as such a bias would favor tiles farther away from the nodes. In effect, the function measures correlation in each node's image-space, and aggregates correlation to yield the total value.

Third, as shown below, the correlation function can be evaluated efficiently by exploiting the rectangular geometric structure inherent in Equation 5.2.

$w[s]$	$=$	$\frac{\int \sum_{i=1}^{k} w_i(u,v)dv}{l[s]}$	incremental weight
$d[s]$	$=$	$\sum_{i=1}^{k} \frac{1}{d_i^2}$	incremental distance factor
$W[s]$	$=$	$\sum_{s' \text{ is a descendant of } s} w[s']l[s']$	total subtree weight
$D[s]$	$=$	$\sum_{s' \text{ is a descendant of } s} d[s']l[s']$	total subtree distance factor
$O[s]$	$=$		subtree correlation function value

Table 5.1: Space-sweep variables.

Evaluation

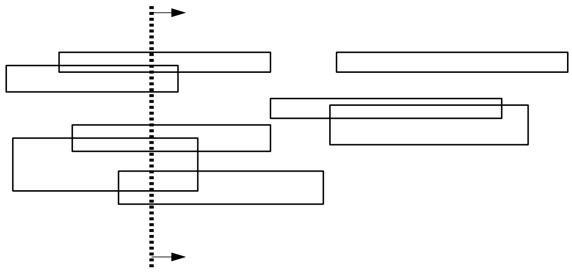


Figure 5-11: Sweeping a set of rectangles on a plane.

The evaluation technique is a modification of a segment-tree based plane-sweep algorithm that computes the total area of m rectangles. The segment tree [Ben75] is a data structure that efficiently represents a set of intervals. The intervals in this algorithm are intersections of a sweep line with the rectangles on the candidate plane (Figure 5-11). The events of the sweep are left and right abscissae of rectangles. The plane-sweep algorithm is completely specified by the set of variables maintained at each node of the segment tree and an *update* rule ([PS85], pages 322–324) that modifies these variables when intervals get added or deleted. The algorithm is described in more detail below.

Let u be the current position of the sweep line. Let $s = (b[s], e[s])$ be a node of the segment tree with begin and end points $b[s]$ and $e[s]$. Let $l[s] = e[s] - b[s]$ be its length, and LSON$[s]$ and RSON$[s]$ denote the left and right children of s, respectively. Let intervals $1 \ldots k$ reside at the node.

$$
\begin{array}{rclll}
W'[s] & = & W[\mathrm{LSON}[s]] + W[\mathrm{RSON}[s]] & \text{weight of children} \\
D'[s] & = & D[\mathrm{LSON}[s]] + D[\mathrm{RSON}[s]] & \text{distance factor of children} \\
O'[s] & = & O[\mathrm{LSON}[s]] + O[\mathrm{RSON}[s]] & \text{correlation value of children} \\
W[s] & = & W'[s] + w[s]l[s] & \text{accumulate weight} \\
D[s] & = & D'[s] + d[s]l[s] & \text{accumulate distance factor} \\
O[s] & = & O'[s] + w[s]d[s]l[s] + W'[s]d[s] + w[s]D'[s] & \text{update correlation function}
\end{array}
$$

Table 5.2: Update rules.

Table 5.1 lists the variables maintained during space-sweep. Updating variables $w[s]$ and $d[s]$ when an interval is inserted (deleted) to node s is straightforward: the contribution due the interval is just added (subtracted) to these intervals.

Table 5.2 lists update rules for the other variables. Note that, if s happens to be a leaf, the values corresponding to LSON[s] and RSON[s] are 0. The intuition behind the update rule for the correlation function is that its *quadratic* nature is *linearized* by maintaining individual contributions from the weight and distance factors ($W[s]$ and $D[s]$).

As the update operation takes constant time, the complexity of the algorithm is determined entirely by the cost of inserting and deleting intervals to and from the segment tree. This cost is $O(\log m)$ for m intervals [Ben75] and the overall complexity is $O(m \log m)$ time. This is optimal, as $O(m \log m)$ is a lower bound for computing the overlap area of m rectangles [PS85].

5.2.2 Grid-Based Maxima Location and Tile Generation

As mentioned earlier, the space-sweep algorithm is performed on a subdivision grid, generating tiles – pieces of vertical facades – for each grid cell. The reasons behind this approach are outlined below.

First, grid-based subdivision limits interaction between distinct facades during space-sweep. For example, in Figure 5-12, two facades with the same orientation

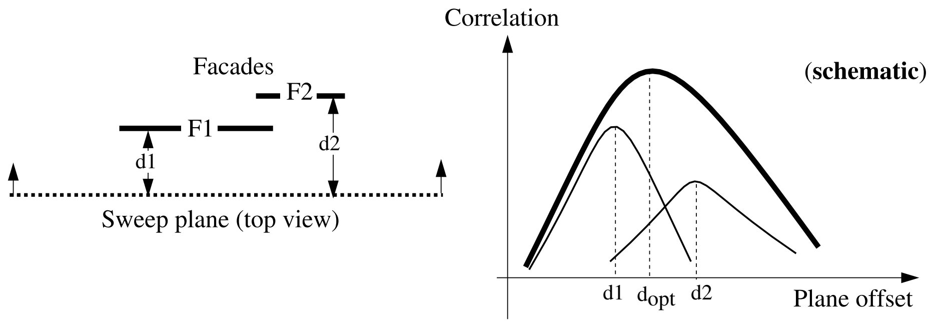


Figure 5-12: Interference between peaks corresponding to two distinct facades.

are positioned close to each other in 3-D. A space-sweep algorithm that considered the entire 3-D space would allow constructive *interference* in the correlation function between the two facades, leading to inaccurate results. However, a well-chosen grid size (e.g., the size of the smallest facade) would decouple the space-sweeps relevant to the two facades leading accurate extraction. Second, in an analogous manner, grid-based subdivision exploits spatial locality and disallows interaction between facades during azimuth estimation of the previous section. Finally, the grid-based approach limits the complexity of the reconstruction algorithm.

I now describe the space-sweep algorithm. Given a grid cell C, the azimuth θ of a tile, and a set of nodes $1 \ldots k$, the algorithm detects likely tile locations as follows:

/* *Project edges from each node onto canonical planes* */

for node $i \in 1 \ldots k$ **do**

  Let C project to image-space region C' in node i.

  Let $\mathbf{P}_i$ be a plane with azimuth θ and distance 1 from node i.

  Project each edge in C' with azimuth θ onto plane $\mathbf{P}_i$.

endfor

/* *Sweep plane through C locating and generating tiles* */

for offset d in C (with step size S) **do**

  Let plane $\mathbf{P} = (\theta, d)$.

  Reproject all edges in $\mathbf{P}_1 \ldots \mathbf{P}_k$ to $\mathbf{P}$.

  /* *Identify high-incidence regions and tiles corresponding to local maxima* */

  if $O(d) > O(d + S)$ and $O(d) > O(d - S)$ **then**

Identify regions with incidence $> K$ in $\mathbf{P}$ and corresponding node edges.
Extrude high-incidence regions vertically to a ground plane, producing tile rectangles.

endif

endfor

In the first phase, the algorithm identifies edges in each node that are likely to lie on the tile by comparing their azimuths with θ. For efficiency, such edges are projected to a canonical plane $\mathbf{P}_i$ (with azimuth θ) corresponding to node i. This intermediate projection permits a simple transformation – a scaling plus a 2-D translation – to be used to construct horizontal line segments on any other plane $\mathbf{P}$ with the same azimuth.

Next, the algorithm discretizes, using step size S, the set of all possible plane offsets corresponding to grid cell C. At each plane offset, it computes horizontal line segment positions and evaluates the correlation function. For each local maximum of the correlation function, it identifies regions on the plane that correspond to a tile by thresholding on incidence K, i.e., regions on the plane (if any) that overlap more than K rectangles (weighted using the triangular filter). In addition, it also identifies node edges that *support* (i.e., give rise to) the tile, by thresholding on the extent of overlap[3] with identified 3-D regions on the tile. Finally, the 3-D region information generated is converted to rectangles by extruding them onto a *ground plane*, and combining overlapping rectangles to produce tiles. The ground plane is estimated by using the z values of camera node positions.

The complexity of this algorithm for a single grid cell is $O(\frac{G}{S}ke(\log k + \log e))$, where G is the grid size, S is the step size, k is the number of nodes, and e is number of edges of a node that lie in the cell's projection. Thus, if the number of nodes (and edges) relevant to a grid cell is bounded by some constant, the complexity of the algorithm scales linearly with the XY area of the 3-D region of interest. Note that this property is typically not true of vision algorithms that employ pairwise matching.

[3] I use a threshold of $s_l = 0.8l$ for an edge with length l, i.e., if more than 80% of the edge overlaps with high incidence regions identified on the tile.

5.3 Geometry Link and Commit

The space-sweep algorithm of the previous section populates a set of grid cells with tiles likely to exist in the 3-D world. In this section, I describe two techniques to enhance the quality of the generated 3-D model. I first describe a technique to link tiles from different grid cells to form complete facades. Then, in Section 5.3.1, I use a priority ordering-based heuristic to eliminate spurious facades reported by the space-sweep algorithm.

The algorithm to form facades from tiles is straightforward: it first links tiles with similar azimuths and offsets across neighboring grid cells, then performs a depth-first-search to identify connected components of tiles. Connected tiles are then merged into a single facade, and the azimuth and location of the combined facade are recomputed, for greater accuracy, using the algorithms of Section 5.1 and Section 5.2.

5.3.1 Facade Commitment

In this section, I describe a technique to eliminate spurious facades present in the recovered 3-D model. The technique is based on the idea of facade *commitment*, which enforces the following constraint: a facade commited to the model *precludes* edges that gave rise to it from supporting other facades. Such enforcement can result in the removal of other facades, if the number of observations supporting them falls below the incidence threshold K of Section 5.2.

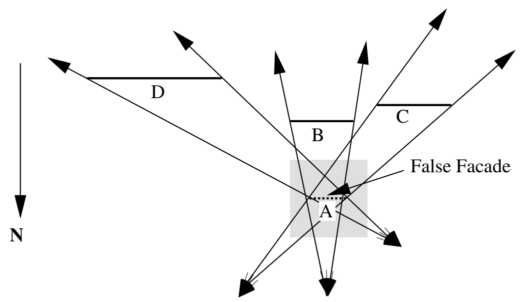


Figure 5-13: This figure shows one way in which a spurious facade can result by the interaction between unrelated facades of the same azimuth.

Figure 5-13 shows (in 2-D) an example of how a spurious facade might arise.

As the three real facades B, C, and D have the same normal $\mathbf{N}$, there is sufficient evidence to report (and verify) even the spurious facade A located in the shaded grid cell. Such spurious facades can arise even from a single facade, due to the interaction between repeated texture on the facade.

The spurious facade elimination technique consists of the following steps. First, the algorithm orders all facades using some criterion that favors real facades over spurious ones. Facades are then committed to the model in this order. Edges giving rise to facades earlier in the ordering are removed. For example, in Figure 5-13, if the ordering favors either B, C, or D over A, it would preclude at least one of A's observations from supporting it, resulting in A's removal.

I considered two possible facade orderings:

1. Order facades according to *occlusion*, similar to [SD97]. That is, if A and B are two facades such that $A < B$ in the ordering, only A can occlude B from any node, not vice-versa. Such an ordering exists for facades with the same normal – in Figure 5-13, it is simply the ordering opposite $\mathbf{N}$. Unfortunately, as this ordering favors locations closer to the nodes, it tends to select spurious facades. For example, in Figure 5-13, facade A would be selected, possibly suppressing the real facades.

2. Order facades according to *higher reported observations* (measured by total length of horizontal segments on the facade). This ordering heuristic favors facades that are larger in area and/or have more line segments. In Figure 5-13, each of the facades B, C, D is larger than A, resulting in A's removal after facade commitment.

In practice, I have observed that the first ordering tends to break up facades into several pieces before their actual location, whereas the second, by favoring larger facades, tends to retain real facades. The results in Section 5.4 are computed using the second ordering.

5.4 Results

The algorithm (and associated visualization) described in this chapter was implemented in about 5000 lines of C++ code. In addition to extracting all significant vertical facades in the office complex (the primary focus of the dataset), the algorithm surprised us by automatically extracting several neighboring facades. Details on the extraction process and algorithm execution times are provided below.

Nodes	81
Images	~ 4000
Resolution	762×506 pixels

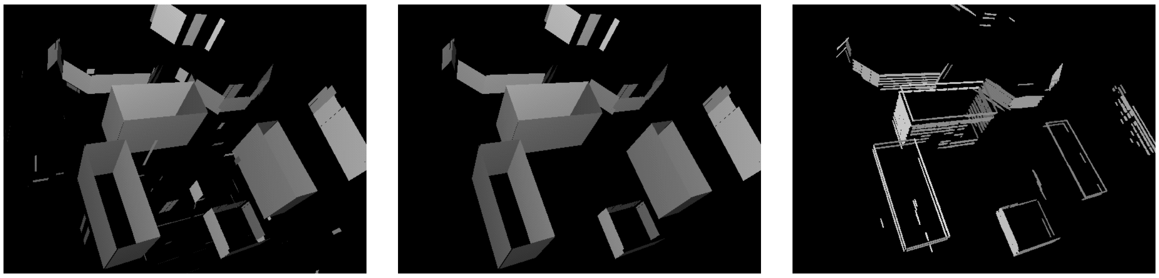
Table 5.3: Input characteristics.

Table 5.3 lists the characteristics of the input dataset. The facade extraction algorithm considered edges detected on six faces of a cubical environment map representing a node. Each face of the environment map was generated at 1024×1024 resolution by resampling the input images. Edge pixels were detected using the Canny edge detector [Can86] and converted to line segments by linking pixels with similar gradient orientation. Approximately 1000 edge segments were computed for each cube face (ignoring those less than 10 pixels in length).

Area of 3-D region of interest	$\sim 500m \times 500m$
Grid Size G	$\sim 10m$
Far Distance D	$\sim 100m$
Step Size S	$\sim 0.1m$
Minimum Weighted Incidence K	3.0

Table 5.4: Parameters supplied to the extraction algorithm. All lengths are given in meters.

Some of the important parameters supplied to the algorithm are listed in Table 5.4. The grid size G should be approximately equal to the size of the smallest facade, to avoid interaction between different facades during tile reconstruction. I use a grid size of 10 meters for this dataset. The minimum weighted incidence value of 3.0 usually implies that a facade must be observed by at least five nodes to be successfully extracted.



(a) (b) (c)

Figure 5-14: Part (a) shows the recovered vertical facades. Part (b) shows the model after removal of small facades (less than $100m^2$), and Part (c) shows the horizontal edges that generate this model.

The facade extraction algorithm took about seven hours on an SGI O2 workstation with one R10000 processor, most of which was spent in the space-sweep algorithm. The space-sweep algorithm recovered approximately 2000 tiles. The model consists of about 140 facades after tile linking and facade commitment (Figure 5-14-(a)). After removing facades with area less than $100m^2$, the model consists of about 40 facades (Figure 5-14-(b)). The horizontal lines used to generate this model are shown in Figure 5-14-(c).

Figure 5-15 shows the extracted facade geometry in wireframe, co-located with the input data (shown as texture-mapped spheres).

5.4.1 Facade Geometry Quality

The extracted geometry of the complex was compared with an independently acquired[4] 3-D CAD model. Table 5.5 compares several important lengths computed in the two models. Note the good agreement between the two models.

Table 5.6 shows azimuths of the vertical facades of two buildings. Note that the

[4]By using surveying and the Facade system [DTM96].

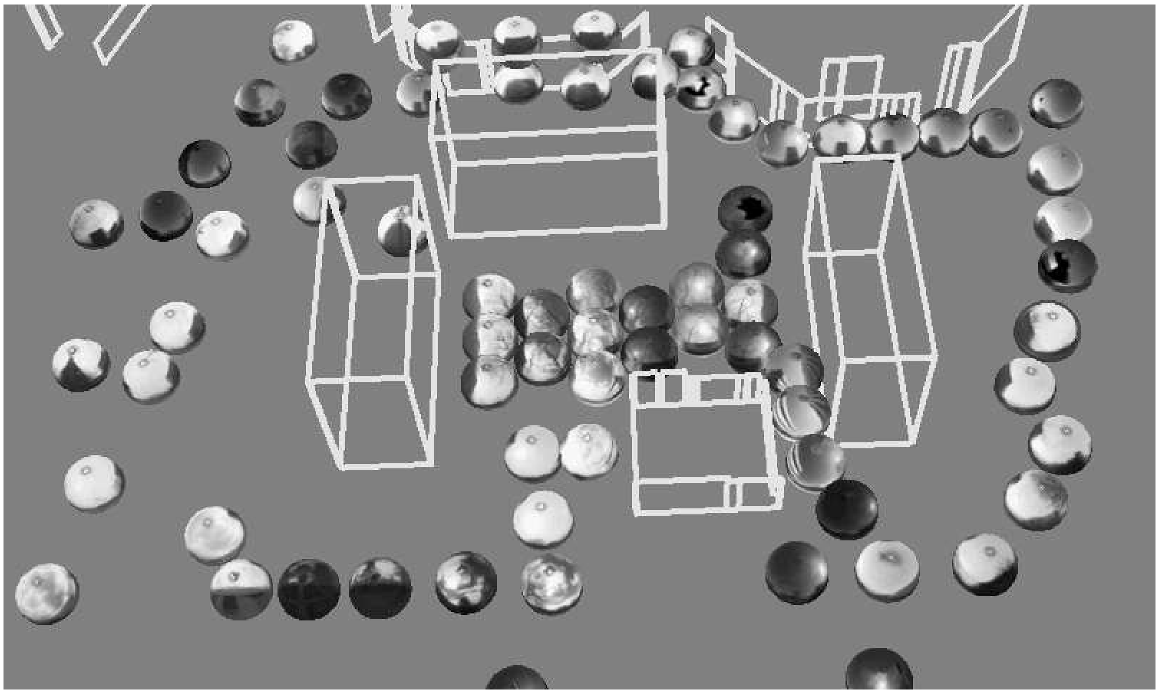


Figure 5-15: The extracted vertical facades (in wireframe) co-located with input pose mosaics (drawn as spheres).

azimuths of adjacent facades are very nearly perpendicular, even though the facades were determined independent of each other.

5.5 Discussion

In this section, I discuss some of the limitations of the space-sweep algorithm, and the sensitivity of the reconstruction results with respect to the various parameters.

	Bldg. 545 S	Bldg. 545 E	Distance between Bldg. 545 and 575 corners
CAD model	22.86	66.75	129.99
Space-sweep model	22.75	66.58	130.21

Table 5.5: Comparison between two models of the building complex. All lengths are in meters.

	Bldg. 545	Bldg. 565
South	270.1	269.89
East	179.97	179.78
North	89.78	89.95
West	0.15	359.94

Table 5.6: Normal azimuths (in degrees) of four vertical facades (shown for two buildings).

5.5.1 Limitations

Figure 5-16: Facades missing from reconstruction.

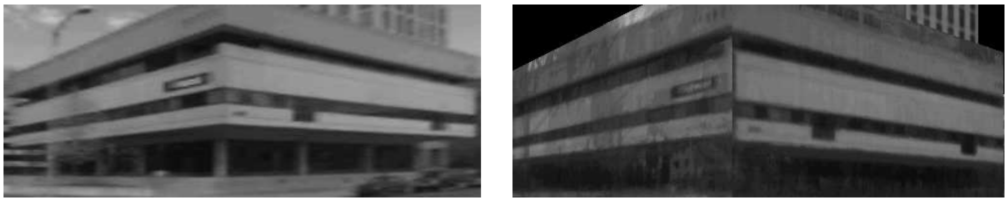


Figure 5-17: Incorrect facade extents.

While the space-sweep algorithm is able to robustly identify vertical facades (given sufficient horizontal edge observations), it is susceptible to the following errors in reconstruction:

- *Spurious facades*: As described in Section 5.3.1 the space-sweep could generate spurious facades due to interaction between horizontal edges from different facades. While the facade commitment technique reduces their occurrence significantly, it is possible that some spurious facades are retained as part of the model (see Figure 5-14-(a)). As shown in the next section, this problem is worse when low incidence thresholds are used.

In this thesis, I have adopted a simplistic approach of using thresholding (both incidence and area) to eliminate such facades. However, in the future, one could envision a method based on color information (e.g., correlation) that suppresses them from the final model.

- *Missing facades*: If a facade is observed by too few nodes, the space-sweep algorithm can fail to detect it (e.g., Figure 5-16). One possible method to address this is to design algorithms that group facades into higher level structures (i.e., buildings) and "fill-in" missing pieces. Another would be to perform model-driven data acquisition and obtain more observations near the missing facade.

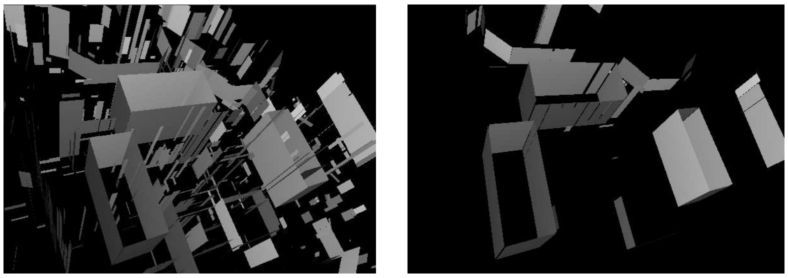
- *Incorrect facade extents*: Another limitation of the extraction algorithm is the assumption that a vertical facade is bounded by a horizontal edge at the top and the ground plane at the bottom. If this assumption is violated, the model generated does not correspond the real geometry in the environment (e.g., Figure 5-17). A possible solution is to analyze the texture computed for facade and compute extents more accurately.

5.5.2 Sensitivity to Parameters

The most important parameter provided to the algorithm is the incidence threshold K used to posit a 3-D edge from K overlapping node observations. A high threshold value implies that the algorithm is fairly conservative in identifying edges and facades. However, this could cause several facades to be undetected. A low threshold value detects a large number of facades, but could also generate several spurious ones.

Figure 5-18 shows geometry generated for different values of the incidence threshold. Note the relative abundance of spurious facades when the threshold is 2.0 and the omission of real facades when the threshold is 4.0. The intermediate value of 3.0 used to generate results of the algorithm was chosen manually after experimenting with a range of values. A future approach would be to tune this parameter automatically based on (say) the number of observations of a facade.

The algorithm is less sensitive to other parameters supplied, and works well within



$K = 2.0$ $K = 4.0$

Figure 5-18: Facades generated for different incidence threshold values.

"reasonable" ranges. The σ_L parameter used to blur edges into rectangles should be slightly larger than the errors in predicting a 3-D ray so that errors due to pose do not result in non-overlapping observations. The s_l parameter used to assign facades to edges should ideally be l (i.e., all of edge length l should be observed on the facade). However, to ameliorate the effects of errors in pose and occlusion, a value of $0.8l$ was used.

5.6 Summary

This chapter described an algorithm that extracts vertical facade geometry from pose mosaics. It uses a technique based on horizontal line segments to detect facade azimuths, and locates detected facades using a space-sweep technique. The algorithm has several advantages over traditional matching-based geometry extraction algorithms known in computer vision. First, it makes full use of constraints both due to camera pose and geometric structure present in the environment. Second, it operates in three-dimensions, ameliorating the effects of perspective (e.g., edges are rectified to the correct plane before overlap), and occlusion (e.g., even broken edges can contribute to facade location). Finally, it scales well to a large set of images, making it useful to model large scale urban environments. The results demonstrate that the algorithm successfully extracts vertical facades in an environment, provided they are present in a significant number of pose mosaic observations.

Chapter 6

Texture Estimation

In order to enhance realism of the generated vertical facade geometry of the previous chapter, I now describe an algorithm to synthesize a single (diffuse RGB) texture for a facade given its various observations. Note that the set of nodes that observe the facade are known, having been reported by the facade extraction algorithm.

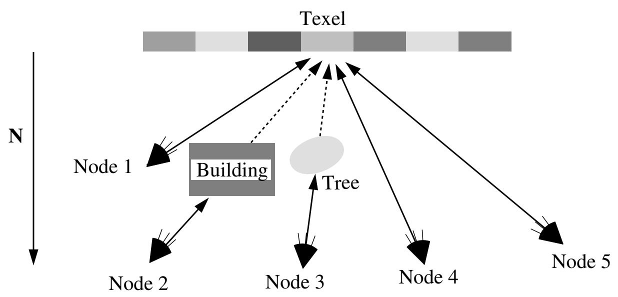


Figure 6-1: This figure illustrates the problem of texture estimation from various node observations.

Figure 6-1 illustrates the problems inherent in estimating a texture for a facade from many observations. Note that each observation can report very different colors for a point on the facade, due to occlusion, obliqueness, illumination variation, etc.

Figure 6-2 shows such effects in several nodes of the dataset which were used to identify and estimate texture for a facade in the office complex. The relevant pixels from the nodes are shown *rectified*, i.e., projected (and clipped) onto the facade. Note the significant variations in pixel color due to changes in illumination and the effects

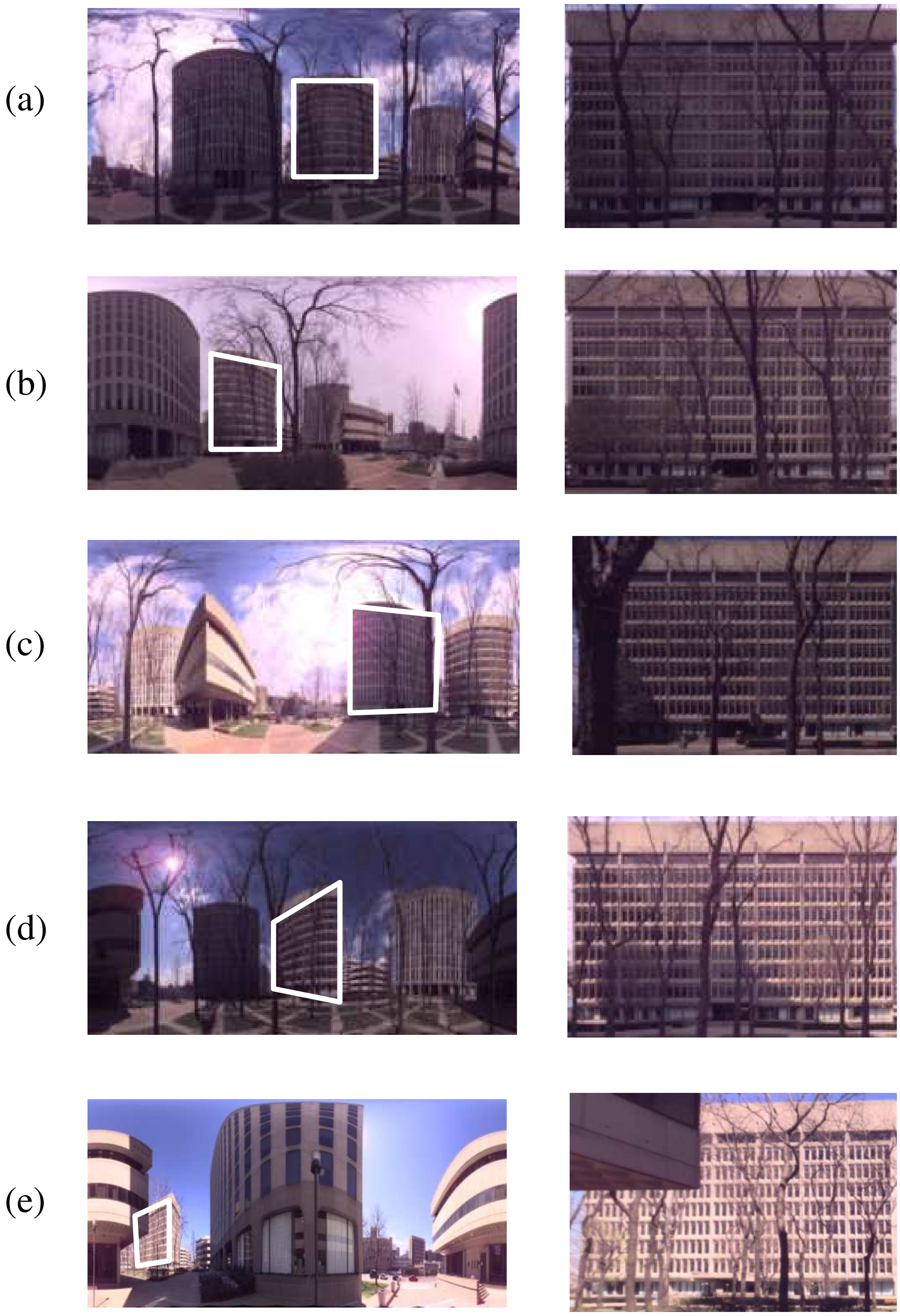


Figure 6-2: This figure shows (the relevant portions of) five nodes rectified to a facade. Note the significant differences in illumination, and the effects of shadows, reflections, and occlusion.

of shadows and reflections. In addition, parts of the facade are occluded by various objects (e.g., trees) in most nodes, making it difficult to determine (and use) a single "best" node, or even interpolate between various nodes based on viewing direction [DTM96]. Instead, it is necessary to combine the information present in all relevant nodes to compute a single facade texture. This is a formidable task due to the sheer amount of pixel data that has to be processed, especially for a human-assisted modeling system.

One possible automatic approach would be to fit the the various observations onto a standard reflectance model and recover coefficients of the model. An example of this technique is [SWI97], where the authors estimate specular and diffuse coefficients of a small object from many images. However, such algorithms tend to be less effective when there is significant occlusion and illumination variation. Instead, I employ a technique based on median statistics, making it robust with respect to such variations.

The rest of the chapter is organized as follows. Section 6.1 describes the simplified illumination model that forms the basis for texture estimation. Section 6.2 describes the median technique that recovers textures robustly. Section 6.3 addresses removal of blurring from the median texture. Section 6.4 presents results of the technique, including synthetic views of the final texture-mapped model.

6.1 Illumination Model

As exteriors of buildings (excluding windows) are typically dull, matte surfaces, their illumination can be usefully modeled by the Lambertian diffuse model [FvDFH90]. Under this model RGB color (I_r, I_g, I_b) of a point on a facade with normal $\mathbf{N}$ is given by:

$$\begin{aligned} I_r &= S_r r_d(\mathbf{N} \cdot \mathbf{L}) \\ I_g &= S_g g_d(\mathbf{N} \cdot \mathbf{L}) \\ I_b &= S_b b_d(\mathbf{N} \cdot \mathbf{L}) \end{aligned}$$

where (S_r, S_g, S_b) is the intensity of the light source (i.e., the sun); $\mathbf{L}$ is the light direction; and (r_d, g_d, b_d) are diffuse coefficients of the point.

Note that raw RGB color of the same point can vary significantly as it is dependent on the light source. However, making the assumption that the source is predominantly white $(S_r = S_g = S_b)$, the lighting equation reduces to:

$$\begin{aligned} I_r &= \alpha r_d \\ I_g &= \alpha g_d \\ I_b &= \alpha b_d \end{aligned}$$

where $\alpha = S(\mathbf{N} \cdot \mathbf{L})$. In other words, the RGB color components scale linearly in different lighting conditions. Therefore, in a color space that separates raw chromaticity from luminance, the chroma components will be the same; only brightness (or luminance) varies.

To exploit this property, I use the normalized CIE xyY color representation [FvDFH90, Poy], which decouples chromaticity x, y from luminance Y:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.412453 & 0.357580 & 0.180423 \\ 0.212671 & 0.715160 & 0.072169 \\ 0.019334 & 0.119193 & 0.950227 \end{bmatrix} \begin{bmatrix} A \\ B \\ C \end{bmatrix}$$

$$\begin{aligned} x &= \frac{X}{X+Y+Z} \\ y &= \frac{Y}{X+Y+Z} \end{aligned}$$

This color representation has several advantages. First, if the RGB components vary linearly, the x and y chromaticity values remain unchanged. Second, the Y component corresponds directly to luminance (brightness), unlike in other normalized representations (such as HSV).

6.2 The Median Texture

The technique divides the facade into a (area dependent) number of texels, and computes a single diffuse value for each texel from values of all its observations. The method consists of two steps: first, it converts RGB color values into the xyY color space; second, it picks the "best" color value for the texel, and converts the result back to RGB.

The *median* statistic is used to compute a single x and y value for each texel from various observed x and y values. The median possesses useful robustness properties, i.e., it is less sensitive to *outliers* than is simple averaging [Hub81]. This is important as outliers are fairly common, typically arising from occlusion by foliage and other structures. Fortunately, such outliers do not appear at the same texel on different rectified nodes (i.e., they exhibit *parallax* [KAH94]), allowing the median to generate a good estimate of the texel's chromaticity. Note that the median generates a good estimate of the chromaticity only if the texel is observed unoccluded in the *majority* of its observations.

The median is also used to select a single luminance value for the texel. Even though luminance values vary significantly, the median luminance tends to reasonably reflect the luminance of the texel under "average" lighting conditions. In any event, the differences in illumination are normalized away, so that the generated texture can be subjected to different lighting conditions.

6.2.1 Weighted Observations

Note that the observation *quality* of a texel varies across different nodes. For example, nodes that view a texel obliquely or from a distance generally yield poor observations. Thus the algorithm uses the *weighted* median. The median v_m of values $v_1 < v_2 \ldots < v_k$ weighted by $w_1 \ldots w_k$ is given by the index m, where:

$$m = \max\{j \text{ such that } \sum_{i=1}^{j} w_i < \frac{\sum_{i=1}^{k} w_i}{2}\}$$

For the weights w_i, the dot product $\mathbf{N} \cdot \mathbf{V}$ is used, where $\mathbf{N}$ is the facade's normal, and $\mathbf{V}$ is the vector direction from the texel center to the node. This downweights nodes that view the texel obliquely. I have observed that incorporating a distance term (e.g., weighting by the texel's solid angle), tends to pick the node that is closest to the texel, losing much of the robustness of the median. In any case, nodes that are far from the texel do not contribute observations, as they are excluded by the geometry extraction algorithm.

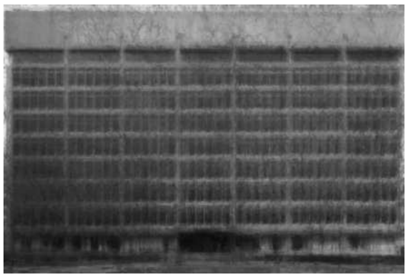


Figure 6-3: A synthetic image of the median texture.

Figure 6-3 shows the results of the median texture algorithm using these weights. Note the automatic removal of most occlusion from the the texture; the luminance pattern on the texture is also reasonably approximated. Compared to Figure 6-2, the median texture has less occlusion, fewer changes in luminance across the texture (e.g., due to shadows), and fewer view-dependent effects (e.g., reflection). The texture is slightly blurred due to small errors in camera pose that have persisted after pose refinement; the next section describes a technique to ameliorate this problem.

6.3 Texture Sharpening

The iterative technique for "sharpening" the median texture consists of two steps. The first step *rewarps* each rectified node to achieve a higher correlation with the median texture. The second step recomputes the median using the new values. These steps are repeated until the median values converge.

The rewarping technique is a correlation-optimization based on luminance values, identical to warp-based mosaicing optimization [Sze96, SS97b] described in Chapter 3. Briefly, the optimization uses an 8-parameter, 2-D projective transformation described by a 3×3 matrix $\mathbf{M}$:

$$u = \frac{\mathbf{M}_{00}x+\mathbf{M}_{01}y+\mathbf{M}_{02}}{\mathbf{M}_{20}x+\mathbf{M}_{21}y+1} \quad v = \frac{\mathbf{M}_{10}x+\mathbf{M}_{11}y+\mathbf{M}_{12}}{\mathbf{M}_{20}x+\mathbf{M}_{21}y+1}$$

In this algorithm, u, v are texture coordinates, and x, y coordinatize the rectified node. Starting from an identity matrix, the algorithm computes $\mathbf{M}_i$ for each node i by minimizing the sum-of-squared differences of luminances:

$$O = \sum_{x_i, y_i} [Y'(u, v) - Y_i(x_i, y_i)]^2 \qquad (6.1)$$

To compensate for illumination variation between a rectified node and the median texture, luminance values are normalized by equalizing the average and standard deviation for each 16×16 patch on the rectified node with the corresponding patch on the median texture[1].

6.3.1 Occlusion Maps

One disadvantage of directly using mosaicing techniques to perform the sharpening is that unlike in mosaicing, there could be significant occlusion between the rectified nodes (see Figure 6-2 (d) and (e), for example). Thus even with perfect alignment across nodes, it is possible for significant differences in luminance values to persist (e.g., due to tree pixels). To ameliorate this problem, pixel contributions are weighted with linear correlation [PTVF92] between a patch in a rectified node and the corresponding patch in the median texture. The idea is that pixels from occluding objects should contribute less to the optimization as they do not correlate with the median texture.

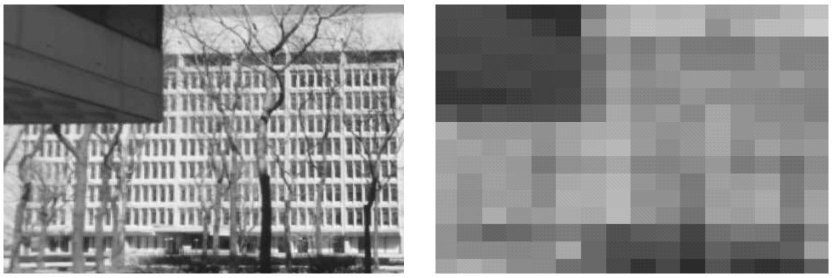


Figure 6-4: A single rectified node and its occlusion map. The map represents correlation by brightness.

Figure 6-4 shows such a map of the rectified node shown in Figure 6-2-(e). Note

[1]Normalizing across the entire texture would incorrectly handle occluded portions.

that pixels from occluding objects (i.e., the building at the top left corner) tend to be weighted less than unoccluded pixels.

6.4 Results

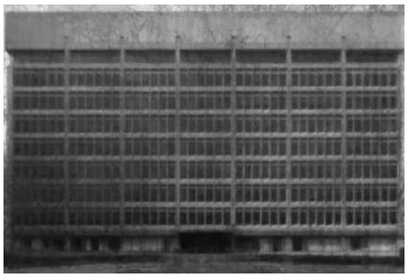


Figure 6-5: The sharpened texture.

Figure 6-5 shows final results for the facade. Note the significant improvement in the quality of the texture. Even though the color of each texel is computed independently, straight lines on the facade are clearly demarcated. Despite this optimization, it is possible for some blurring to persist in the texture. This is caused by parts of the facade extruding out of its plane. One possible solution would be to construct disparity maps to capture such extrusions, as in [DTM96].

Before estimating texture for the entire model, the facade geometry recovered was further processed using the following steps:

- Adjacent facades (those with end-points in the same grid cell) with different orientations were linked, modifying facade boundaries to the edge formed by intersecting adjacent facades. Also, a "roof" was added to each connected set of facades after equalizing facade heights to the maximum height in the set.

- A 2-D Delaunay triangulation of the camera XY positions was computed, and an approximation to the ground terrain was constructed using ground heights obtained from camera poses.

Textures were computed to $0.1m$ ($10cm$) resolution, with the largest texture containing 1024×512 pixels. Texture computation took about ten hours. Textures for non-vertical geometry (i.e., roof and ground polygons) were extracted from a single aerial pose image of the complex. These additional geometry and textures provide *context* information for the vertical facades, for more realistic visualization.

Figure 6-7 shows the final results of the algorithm: two frames of a real-time virtual flythrough of the final reconstructed model on an SGI O2 workstation. Also, Figure 6-8 shows a ground view of the final model along with (the relevant portion of) a node taken with the same viewpoint. Note the similarity between building facades in the two images. However, as the model does not contain occluders (e.g., cars, trees) and view-dependent effects (e.g., reflection), the images, in totality, appear significantly different.

6.4.1 Limitations

Figure 6-6: Facade texture illustrating limitations of the median-based algorithm.

If the simplifying assumptions made about facade illumination and occlusion are not valid, it is possible for the median technique to generate incorrect results. Figure 6-6 shows an example. Note that shadows are part of the texture; ideally, they should be removed. Also, due to insufficient observations of this facade, some of the occluded regions (e.g., parts of trees and the fence) are still present in the texture.

Several techniques could be used in the future to overcome these drawbacks. First, a more complex illumination model that captures effects such as shadows and reflections could be used. Second, a larger set of images could be used to observe all parts of the facade completely.

6.5 Summary

This chapter described a robust technique to extract facade textures from multiple observations. Median statistics are used to reject outliers, i.e., pixels arising from occluding objects. Pixels are represented in xyY color space to minimize sensitivity to illumination variation. A sharpening step corrects final misalignments in between nodes and generates high quality textures. The textures can be mapped onto the recovered geometry to generate a real-time visualization of the modeled building complex.

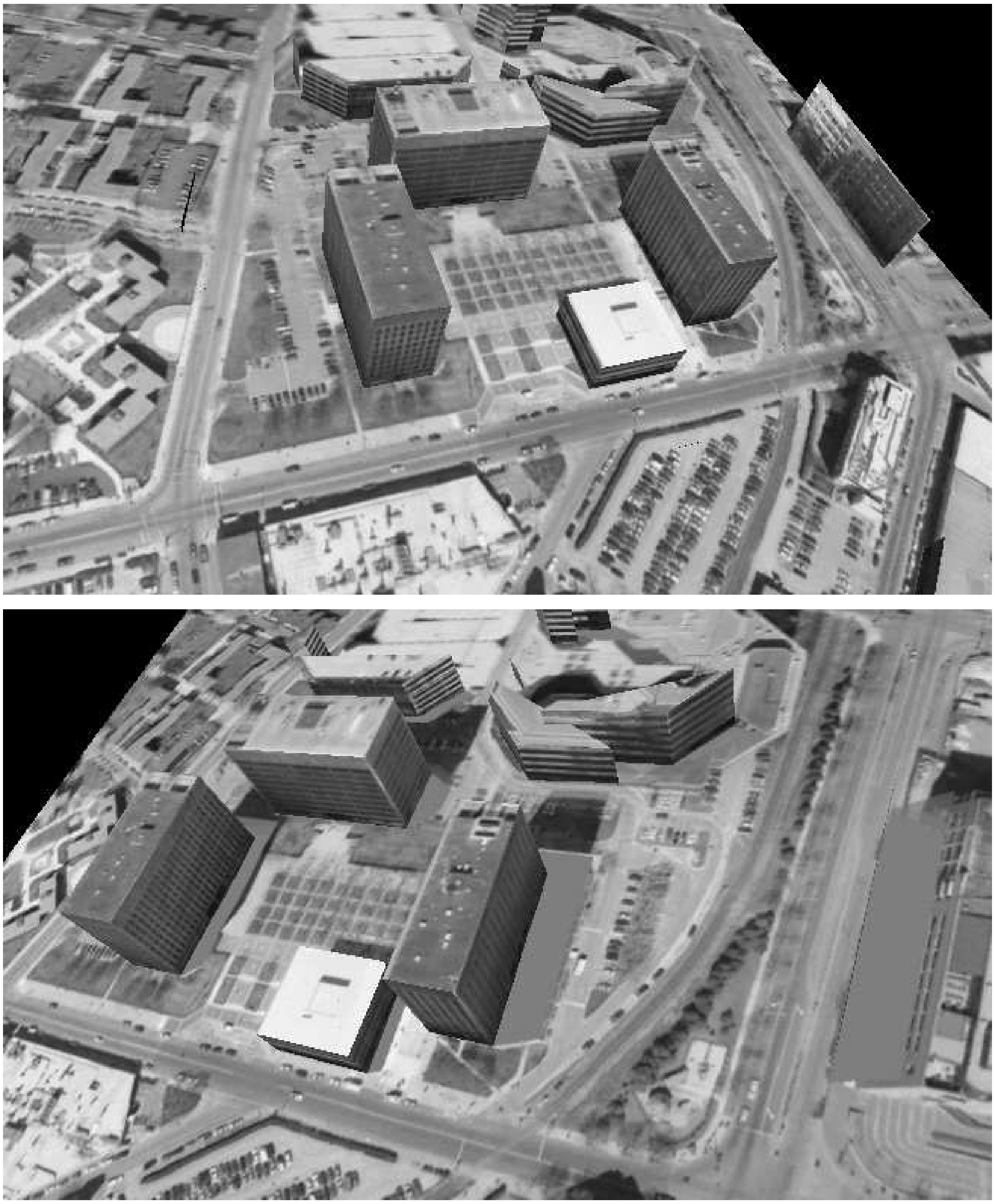


Figure 6-7: Two aerial views of the final model.

Figure 6-8: Synthetic (top) and real (bottom) ground views of the office complex.

Chapter 7

Conclusion

This thesis described a novel approach to the problem of extracting 3-D models of urban environments from pose imagery – images annotated with camera pose information supplied by physical instruments. While raw pose information is by itself not sufficient for direct incorporation in 3-D reconstruction algorithms, I demonstrated that they provide useful initial estimates for optimization techniques that recover accurate pose. These refined pose estimates formed the basis for the proposed reconstruction algorithms.

I described several novel algorithms in this thesis:

- An automatic spherical mosaicing algorithm that recovers relative rotations between images taken from a single node (Chapter 3).

- A quaternion-based registration algorithm that locates mosaics in a global coordinate system based on semi-automatically generated correspondences (Chapter 4).

- A geometry extraction algorithm that extracts dominant vertical facades (Chapter 5).

- A robust facade texture estimator based on median statistics (Chapter 6).

These algorithms possess several desirable properties for automatic model extraction. They use all information from relevant images, yet scale to an arbitrary number of images and model size. They exploit geometric constraints inherent in the 3-D environment. Finally, they are robust with respect to occlusion and changes in illumination. I believe that these properties will be crucial for future systems that perform practical, large-scale reconstruction.

The algorithms have been implemented and tested on a large pose image dataset consisting of about four thousand images. The results provided in this thesis demonstrate that these algorithms can successfully be used for 3-D reconstruction. To my knowledge, this is the first system to automatically analyze such a large set of images, and produce a realistic 3-D model suitable for computer graphics rendering.

7.1 Future Work

The research described in this thesis can be extended in several directions:

- Build platforms that acquire pose imagery rapidly and accurately (Section 7.1.1);
- Develop automatic matching techniques to eliminate user input (Section 7.1.2);
- Design techniques that enhance the quality of the extracted 3-D model (Section 7.1.3);
- Finally, design algorithms that recover generic geometry from pose imagery (Section 7.1.4).

7.1.1 Pose Image Acquisition Platform

As demonstrated in this thesis, pose imagery is very useful for 3-D reconstruction algorithms. It is thus important to be able to acquire such a dataset rapidly and reliably. One approach, currently being pursued as part of the MIT City Scanning project [Tel97], is to build a device that 1) rapidly acquires digital imagery of an urban environment and 2) integrates information from various physical instruments to report accurate pose.

Figure 7-1: Argus: a GPS-based pose imagery acquisition platform.

A prototype of such a device is Argus (see Figure 7-1) that combines input from GPS receivers, wheel encoders, and inertial navigation sensors using a Kalman filter to report camera pose in a global coordinate system [DeC98]. Preliminary results [DeC98] indicate that complete nodes can be acquired in about six minutes, a significant improvement over manual acquisition (around twenty minutes per node).

7.1.2 Automatic Matching

The pose refinement algorithm described in Chapter 4 requires point correspondences to perform the optimization. While the user input required for this algorithm is minimal, it could be a potential bottleneck for applying these techniques to very large image sets. Thus, it is fruitful to investigate alternate, fully automatic approaches to matching.

Though computer vision research has addressed problem of correspondence in the context of stereo vision [Fau93] and structure from motion [TK92], these algorithms require very specialized input images (i.e., short baseline, stable illumination). Extending such algorithms to general image pairs appears to be extremely hard.

However, I believe that it is possible to design automatic matching techniques that generate correspondences for pose refinement due to the following reasons. First, unlike the dense set of correspondences required for stereo and structure from motion,

Figure 7-2: Automatically detected skylines in two mosaics (by thresholding on RGB values).

only a few pairwise correspondences are required to generate accurate camera pose. Second, specific attributes of urban environments can be recognized and used to guide the matching process. For example, as Figure 7-2 illustrates, one approach might be to automatically identify *skylines* in the input nodes, and attempt to match only corners of the skylines. Note that the sky detector need not be perfect; it is sufficient for it to identify five or more matchable corners.

7.1.3 Enhancing Model Quality

Several improvements are possible for the generated model. The algorithm typically overestimates the extent of vertical facades (e.g., by extruding to the ground or choosing the maximum possible height for the facade). This could be corrected by clipping the model to the edges of computed textures. Also, textures could be augmented with disparity maps to model small extrusions, as in [DTM96].

Other areas for future work include designing pose imagery-based extraction algorithms to recover more general reflectance properties (e.g., specular coefficients, BRDFS) of the model. Note that this would probably require a denser image set of the environment, captured under many different lighting conditions (such as a time series).

7.1.4 Generic Geometry

One approach to extract more general geometry would be to use the incidence counting idea to extract *general* line segments, instead of just horizontal line segments (Chapter 5). Instead of performing the count in 3-D, the algorithm needs to operate in a higher dimensional *linespace* that captures the geometry of lines.

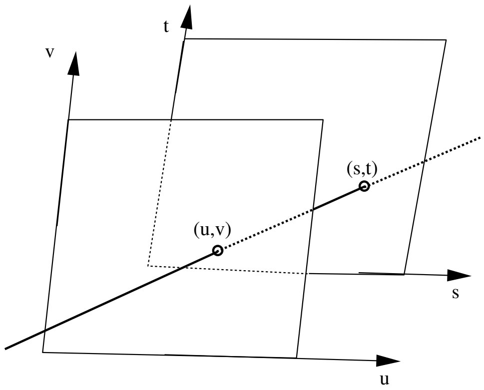


Figure 7-3: 4-D linespace parameterized by two parallel planes.

A good candidate is the representation that parameterizes lines by their intercepts on two parallel planes *st* and *uv* [GGSC96, LH96] (Figure 7-3). Three pairs of planes corresponding to each axial direction are sufficient to parameterize the set of lines.

To perform incidence counting of lines, it is necessary to transform 2-D image edges to 4-D linespace. An example of this transformation is shown in Figure 7-4. First, image edges are extruded to form 3-D planes; these planes intersect the *uv* and *st* planes in two lines. All possible lines that could correspond to the image edge can be generated by picking two points, one on the *st*-intercept and the other on the *uv*-intercept. Thus, high incidence in 4-D corresponds to high incidence in *both* the *uv* and *st* planes.

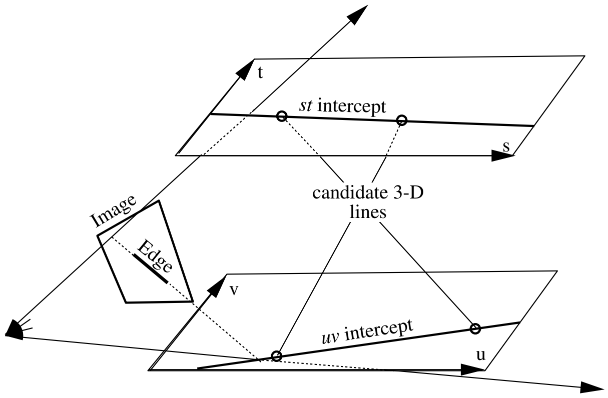


Figure 7-4: Projecting an image edge to 4-D.

While this algorithm is more general that the algorithm in Chapter 5, several issues need to be addressed before a practical implementation can be achieved. First, there is a higher likelihood of spurious matches being detected by the algorithm, as all edges are processed simultaneously (unlike in Chapter 5, where only edges with the same orientation were processed together). It is thus important to detect and suppress such matches. Second, an efficient technique (e.g., a space-sweep) needs to be designed for searching through linespace to detect regions of high incidence. Finally, a robust algorithm that extracts facades based on 3-D edge information is required.

Other, alternate approaches to recover 3-D information from pose imagery are also being pursued as part of the MIT City Scanning project:

- J. P. Mellor is investigating a multi-baseline technique based on *epipolar image* analysis and pixel correlation [MTLP96], a generalization of the epipolar constraint used in classical stereo.

- George Chou is investigating an approach based on matching edge segments with similar attributes [CT97a].

Finally, an interesting area of future research would be to design hybrid algorithms that blends these techniques with those developed in this thesis (e.g., using pixel correlation to directly recover facades).

Bibliography

[AJ88] Kurt Akeley and Tom Jermoluk. High-performance polygon rendering. In *SIGGRAPH '88 Conference Proceedings*, pages 239–246, August 1988.

[Ake93] Kurt Akeley. RealityEngine graphics. In *SIGGRAPH '93 Conference Proceedings*, pages 109–116, 1993.

[AP95] A. Azarbayejani and A.P. Pentland. Recursive estimation of motion, structure, and focal length. *PAMI*, 17(6):562–575, June 1995.

[AS95] S. Ayer and H. Sawhney. Layered representation of motion video using robust maximum-likelihood estimation of mixture models and MDL encoding. In *ICCV95*, pages 777–784, 1995.

[AW96] E.H. Adelson and Y. Weiss. A unified mixture framework for motion segmentation: Incorporating spatial coherence and estimating the number of models. In *CVPR96*, pages 321–326, 1996.

[Aya91] Nicholas Ayache. *Artificial Vision for Mobile Robots.* The MIT Press, Cambridge, MA, 1991.

[Bar83] S. T. Barnard. Interpreting perspective images. *Artificial Intelligence*, 21:435–462, 1983.

[BB95] Shawn Becker and V. Michael Bove. Semiautomatic 3-D model extraction from uncalibrated 2-D camera views. In *Proceedings of Visual Data Exploration and Analysis II, SPIE Vol. 2410*, pages 447–461, 1995.

[BBM87] R.C. Bolles, H.H. Baker, and D.H. Marimont. Epipolar-plane image analysis: An approach to determining structure from motion. *IJCV*, 1(1):7–56, 1987.

[Ben75] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[BHR87] J.B. Burns, A.R. Hanson, and E.M. Riseman. Extracting straight lines. In *Readings in Computer Vision*, pages 180–183, 1987.

[BM90] Andrew Blake and Constantinos Marinos. Shape from texture: Estimation, isotropy and moments. *Artificial Intelligence*, 45(3):323–380, October 1990.

[BO91] B. Brillault-O'Mahony. New method for vanishing point detection. *Image Understanding*, 54:289–300, 1991.

[BS97] Richard Bukowski and Carlo H. Séquin. Interactive simulation of fire in virtual building environments. In *SIGGRAPH '97 Conference Proceedings*, pages 35–44, August 1997.

[Can86] F. J. Canny. A computational approach to edge detection. *IEEE Trans PAMI*, 8(6):679–698, 1986.

[Che95] Shenchang Eric Chen. Quicktime VR – an image-based approach to virtual environment navigation. In *SIGGRAPH '95 Conference Proceedings*, pages 29–38, August 1995.

[CJS+95] R. Collins, C. Jaynes, F. Stolle, X. Wang, Y. Cheng, A. Hanson, and E. Riseman. A system for automated site model acquisition. In *SPIE Proceedings Vol. 7617*, 1995.

[Col96] R.T. Collins. A space-sweep approach to true multi-image matching. In *CVPR96*, pages 358–363, 1996.

[CT97a] George Chou and Seth Teller. Multi-image correspondence using geometric and structural constraints. In *DARPA Image Understanding Workshop*, May 1997.

[CT97b] Satyan Coorg and Seth Teller. Real-time occlusion culling for models with large occluders. In *Proc. 5^{th} ACM Symposium on Interactive 3D Graphics*, pages 83–90, 1997.

[DA89] U. R. Dhond and J. K. Aggarwal. Structure from stereo – A review. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):1489–1510, November-December 1989.

[DeC98] Doug DeCouto. Instrumentation for rapidly acquiring pose imagery. Master's thesis, Dept. of Electrical Engg. and Computer Science, MIT, 1998.

[DTM96] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH '96 Conference Proceedings*, pages 11–20, August 1996.

[Fau92] O. D. Faugeras. What can be seen in three dimensions with an uncalibrated stereo rig? In *Proceedings of Computer Vision (ECCV '92)*, pages 563–578, 1992.

[Fau93] O.D. Faugeras. *Three-Dimensional Computer Vision.* MIT Press, 1993.

[FK] O. Faugeras and R. Keriven. Variational principles, surface evolution, PDE's, level set methods and the stereo problem. *IEEE Trans. on Image Processing.* To appear.

[FvDFH90] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics, Principles and Practice, Second Edition.* Addison-Wesley, Reading, Massachusetts, 1990.

[Gen77] D. B. Gennery. A stereo vision system for an autonomous vehicle. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 576–582, August 1977.

[GGSC96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *SIGGRAPH '96 Conference Proceedings*, pages 43–54, August 1996.

[Gla90] Andrew Glassner, editor. *Graphics Gems.* Academic Press Professional, 1990.

[Gre97] C.W. Greeve. *Digital Photogrammetry: an Addendum to the Manual of Photogrammetry.* American Society of Photogrammetry and Remote Sensing, 1997.

[Har95] R. Hartley. In defence of the 8-point algorithm. In *ICCV95*, pages 1064–1070, 1995.

[Har97] Richard Hartley. Self-calibration of stationary cameras. *IJCV*, 22(1):5–23, 1997.

[Hec86] Paul S. Heckbert. Survey of texture mapping. *IEEE Computer Graphics and Applications*, 6(11):56–67, November 1986.

[Hor86] Berthold Klaus Paul Horn. *Robot Vision.* MIT Press, Cambridge, MA, 1986.

[Hor87] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4), April 1987.

[Hor90] Berthold K. P. Horn. Relative orientation. *IJCV*, 4(1):59–78, January 1990.

[Hor91] Berthold K. P. Horn. Relative orientation revisited. *Journal of the Optical Society of America A*, 8(10):1630–1638, October 1991.

[Hub81] Peter J. Huber. *Robust Statistics.* Wiley, 1981.

[IRP97] M. Irani, B. Rousso, and S. Peleg. Recovery of ego-motion using region alignment. *PAMI*, 19(3):268–272, March 1997.

[JLF95] William Jepson, Robin Liggett, and Scott Friedman. An environment for real-time urban simulation. In *Proc. 1995 Symposium on Interactive 3D Graphics*, pages 165–166, 1995.

[KAH94] R. Kumar, P. Anandan, and K. Hanna. Shape recovery from multiple views: a parallax based approach. In *ARPA Image Understanding Workshop, Monterey, CA*, November 1994.

[Kan84] K. I. Kanatani. Detection of surface orientation and motion from texture by a stereological technique. *Artificial Intelligence*, 23:213–237, 1984.

[KS96] Sing Bing Kang and Richard Szeliski. 3-D scene recovery using omnidirectional multibaseline stereo. In *International Conference on Computer Vision and Pattern Recognition*, pages 364–370, San Francisco, CA, June 1996.

[KS98] K.N. Kutulakos and S. M. Seitz. What do n photographs tell us about 3d shape? Technical Report TR680, Computer Science Dept., U. Rochester, 1998.

[KWZK95] Sing Bing Kang, Jon A. Webb, C. Lawrence Zitnick, and Takeo Kanade. An active multibaseline stereo system with active illumination and real-time image acquisition. In *International Conference on Computer Vision*, Cambridge, MA, 1995.

[LH96] Marc Levoy and Pat Hanrahan. Light field rendering. In *SIGGRAPH '96 Conference Proceedings*, pages 31–42, August 1996.

[LT87] R. Lenz and R. Tsai. Techniques for calibration of the scale factor and image center for high accuracy 3D machine vision metrology. In *Proc. IEEE International Conf. on Robotics and Automation*, 1987.

[MB95] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *SIGGRAPH '95 Conference Proceedings*, pages 39–46, August 1995.

[MBDM97] John S. Montrym, Daniel R. Baum, David L. Dignam, and Christopher J. Migdal. Infinitereality: A real-time graphics system. In *SIGGRAPH '97 Conference Proceedings*, pages 293–302, August 1997.

[McM97] L. McMillan. *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, Dept. of Computer Science, Univ. of North Carolina (Chapel Hill), 1997.

[MMC+97] D. M. McKeown, C. McGlone, S. D. Cochran, Y. C. Hsieh, M. Roux, and J. Shufelt. Automatic cartographic feature extraction using photogrammetric principles. In *Digital Photogrammetry*, pages 195–212. ASPRS, 1997.

[MTLP96] J. P. Mellor, Seth Teller, and Tomás Lozano-Pérez. Dense depth maps for epipolar images. Technical Report Technical Memo 1593, Artficial Intelligence Laboratory, MIT, 1996.

[MVQ93] R. Mohr, F. Veillon, and L. Quan. Relative 3D reconstruction using multiple uncalibrated images. In *CVPR93*, pages 543–548, 1993.

[OK85] Y. Ohta and T. Kanade. Stereo by two-level dynamic programming. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 7(2), April 1985.

[OK93] M. Okutomi and T. Kanade. A multiple-baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4):353–63, 1993.

[PMF85] S. Pollard, John Mayhew, and John Frisby. PMF: A stereo correspondence algorithm using a disparity gradient limit. *Perception*, 14:449–470, 1985.

[Poy] Charles A. Poynton. Frequently asked questions about color. Avaliable at http://home.inforamp.net/~poynton/ColorFAQ.html.

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry: an Introduction*. Springer-Verlag, 1985.

[PTVF92] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.)*. Cambridge University Press, Cambridge, 1992.

[SA96] H.S. Sawhney and S. Ayer. Compact representations of videos through dominant and multiple motion estimation. *PAMI*, 18(8):814–830, August 1996.

[Sca85] L.E. Scales. *Introduction to Non-Linear Optimization*. Springer-Verlag, 1985.

[SD96] Steven M. Seitz and Charles R. Dyer. View morphing: Synthesizing 3D metamorphoses using image transforms. In *SIGGRAPH '96 Conference Proceedings*. ACM SIGGRAPH, August 1996.

[SD97] S.M. Seitz and C.R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *CVPR97*, pages 1067–1073, 1997.

[SHS98] H. Shum, M. Han, and R. Szeliski. Interactive construction of 3-d models from panoramic mosaics. In *CVPR98*, pages 427–433, 1998.

[SK94] R. Szeliski and S. B. Kang. Recovering 3D shape and motion from image streams using non-linear least squares. *Journal of Visual Communication and Image Representation*, 5(1):10–28, 1994.

[Sla80] C.C. Slama. *Manual of Photogrammetry*. American Society of Photogrammetry and Remote Sensing, 1980.

[SLS+96] Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, and John Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH '96 Conference Proceedings*, pages 75–82, 1996.

[SS97a] Heung-Yeung Shum and Richard Szeliski. Panoramic image mosaics. Technical Report MSR-TR-97-23, Microsoft Research, 1997.

[SS97b] Richard Szeliski and Harry Shum. Creating full-view panoramic mosaics and texture-mapped 3D models. In *SIGGRAPH '97 Conference Proceedings*, pages 251–258, August 1997.

[Ste95] G.P. Stein. Accurate internal camera calibration using rotation, with analysis of sources of error. In *ICCV95*, pages 230–236, 1995.

[Str88] G. Strang. *Linear algebra and its applications*. Harcourt Brace Jovanovich, 1988.

[SWI97] Yoichi Sato, Mark D. Wheeler, and Katsushi Ikeuchi. Object shape and reflectance modeling from observation. In *SIGGRAPH '97 Conference Proceedings*, pages 379–388, August 1997.

[Sze96] Richard Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, 16(2):22–30, March 1996.

[Tel97] Seth Teller. Automatic acquisition of hierarchical, textured 3D geometric models of urban environments: Project plan. In *Proceedings of the Image Understanding Workshop*, 1997.

[TK92] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9:137–154, 1992.

[TK95] C. J. Taylor and D. J. Kriegman. Structure and motion from line segments in multiple images. *PAMI*, 17(11):1021–1032, November 1995.

[TK96] Jay Torborg and Jim Kajiya. Talisman: Commodity Real-time 3D graphics for the PC. In *SIGGRAPH '96 Conference Proceedings*, pages 353–364, August 1996.

[TS91] Seth Teller and Carlo H. Séquin. Visibility preprocessing for interactive walkthroughs. *SIGGRAPH '91 Conference Proceedings*, pages 61–69, 1991.

[Tsa87] R. Tsai. A versatile camera calibration technique for high accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4), August 1987.

[Vex] Vexcel. The FotoG family of products and services. Available at http://www.vexcel.com/fotog/product.html.

[Wei97] Y. Weiss. Smoothness in layers: Motion segmentation using nonparametric mixture estimation. In *CVPR97*, pages 520–526, 1997.

[Wil83] L. Williams. Pyramidal parametrics. *Computer Graphics*, 17(3):1–11, July 1983.

[Wit81] A. P. Witkin. Recovering Surface Shape and Orientation from Texture. *Artificial Intelligence*, 17(1-3):17–45, August 1981.

[Wol74] P.R. Wolf. *Elements of Photogrammetry.* McGraw-Hill, 1974.

[ZFD97] I. Zoghiami, O.P. Faugeras, and R. Deriche. Using geometric corners to build a 2D mosaic from a set of images. In *CVPR97*, pages 420–425, 1997.