SCREW-TRANSFORM MANIFOLDS FOR CAMERA SELF CALIBRATION

by

Russell Alan Manning

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

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2003

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

NOTATION v

ABSTRACT viii

1 Overview 1

1.1 Camera calibration 1
1.2 Applications of camera calibration 3
1.3 Calibration by manifold intersection 4
1.4 Contributions of the dissertation 6
1.5 Structure of the dissertation 7

2 History and Context 9

2.1 Image-based rendering: sampling versus simulation 9
2.2 Self calibration and scene reconstruction 18

3 Introduction to Camera Calibration and Multiview Geometry 27

3.1 Platonic labels and coordinate systems 28
3.2 Pinhole cameras 30
3.2.1 Basic mathematical definition 30
3.2.2 Physical interpretation and history 34
3.2.3 Physical derivation of the pinhole camera matrix 35
3.2.4 Meaning of internal calibration 38
3.3 Concepts from projective geometry 39
3.4 Epipolar geometry 46
3.5 Scene reconstruction 52
3.5.1 Hierarchy of scene reconstructions 53
3.5.2 Hierarchy of camera reconstructions 55
3.5.3 Triangulation 57
3.5.4 Projective reconstruction using a fundamental matrix 58
3.5.5 The absolute quadric and metric reconstruction 59

Page

3.6 Finding point correspondences in practice . . . 62
3.7 Finding fundamental matrices in practice . . . 63

4 Screw-Transform Manifolds . . . 66

4.1 The screw transformation between two views . . . 66
4.2 Upgrading weak calibration . . . 69
4.2.1 Parameterizing rising-turntable scenarios . . . 69
4.2.2 Parameterizing relative calibration . . . 70
4.3 Screw-transform manifolds and self calibration . . . 73
4.3.1 Kruppa-constraint manifold . . . 73
4.3.2 Self calibration using Kruppa-constraint manifolds . . . 74
4.3.3 Stratified self calibration and the modulus constraint . . . 75
4.3.4 Modulus-constraint manifolds . . . 79
4.4 Non-general camera motions . . . 80
4.4.1 Classifying pairwise camera motions . . . 81
4.4.2 Turntable motion . . . 82
4.4.3 Transfocal motion . . . 84
4.4.4 Tests for classifying pairwise camera motions . . . 86

5 Manifold Intersection Algorithms . . . 92

5.1 Surface fitting . . . 94
5.2 Voting-based algorithm . . . 97
5.3 Monte-Carlo Markov-Chain approach . . . 101

6 Experimental Evaluation of Self Calibration from Screw-Transform Manifolds . . . 107

6.1 Experimental results for the surface-fitting algorithm . . . 107
6.1.1 The synthetic data sets and their nomenclature . . . 108
6.1.2 Answer to question 1: Algorithm correctness . . . 113
6.1.3 Answer to question 2: Algorithm speed . . . 113
6.1.4 Answer to question 3: Advantage of extra views . . . 115
6.1.5 Answer to question 4: Number of mutual intersection points . . . 116
6.1.6 Answer to question 5: Performance with real cameras . . . 117
6.1.7 Implementation details . . . 124
6.2 Experimental results for the MCMC-based algorithm . . . 127
6.2.1 The synthetic data sets and their nomenclature . . . 128
6.2.2 Answer to question 1: Algorithm correctness . . . 130
6.2.3 Answer to question 2: Algorithm speed . . . 131

Appendix
Page

6.2.4 Answer to question 3: Advantage of extra views 132
6.2.5 Answer to question 4: Choosing MCMC parameters 132
6.2.6 Answer to question 5: Comparison with surface-fitting algorithm 134
6.3 Self calibration from three views 135

7 Calibration and Image-Based Rendering for Dynamic Scenes 138

7.1 Overview 138
7.2 Using screw-transform manifolds with dynamic scenes 140
7.3 Linear algorithm for affine self calibration from scene motion 141
7.3.1 Introduction 141
7.3.2 Notation and preliminary concepts 144
7.3.3 Motion-based affine calibration 146
7.3.4 Generalizing to multiple objects 147
7.3.5 Experiments with synthetic data 148
7.3.6 Experiments with real data 153
7.3.7 Conclusion 157
7.4 View interpolation for dynamic scenes with apparent linear motion 157
7.4.1 Introduction 157
7.4.2 Static view morphing 159
7.4.3 Dynamic view morphing 160
7.4.4 Finding relative camera calibration 167
7.4.5 Applications 167
7.4.6 Experimental results 167
7.4.7 Conclusion 169
7.5 View interpolation of turntable motion 170
7.5.1 Turntable sequences 170
7.5.2 Conclusion 178

8 Summary 179

8.1 Accomplishments 179
8.2 Conclusions 181

BIBLIOGRAPHY 182

APPENDICES

Appendix A: Mathematical Details 196
Appendix B: Implementation Details 208

Appendix

Page

INDEX 215

DISCARD THIS PAGE

NOTATION

$\mathbf{M}$	A 3×3 matrix. Capital Latin letter in math bold font.
$\boldsymbol{\Psi}$	A matrix with dimensions other than 3×3. Capital Greek letter in math bold font.
$\mathbf{v}$	A vector. Lower-case Latin letter in math bold font.
k, γ	A real number. Lower-case Latin or Greek letter in math italic font.
A	Reference to camera "A." Capital Latin letter in math italic font. Cameras are labeled with letters rather than numbers.
$\boldsymbol{\Pi}_A$	The 3×4 camera matrix for camera A.
$\mathbf{M}_{(ij)}$	The entry in row i, column j of matrix $\mathbf{M}$
$\cong$	Equality up to a scale. Two quantities are equal up to a nonzero scale factor.
$:=$	Assignment operator. The symbol on the left is being assigned the value on the right.
$\stackrel{\text{def}}{=}$	A definition. The symbol on the left is being defined equal to the value on the right.
$\stackrel{\text{def}}{\cong}$	Definition up to a scale. The symbol on the left is being defined equal to the value on the right up to a nonzero scale factor.
(x, y)	An n-tuple or a vector written horizontally without matrix notation.
$\langle \mathbf{u}, \mathbf{v} \rangle$	The space spanned by a set of vectors.
RANSAC	An algorithm referred to by a short title. Small caps font.
`psif`	A variable from a pseudocode example. Bold teletype font.
${}_nC_m$	The combination "n choose m."
$\neg(y)$	The assertion that statement number y cannot hold.
T, w	Platonic label for a concept. Math italic font.
$\hat{\mathbf{u}}$	Platonic direction vector.

$\{\mathbf{t}\}_w$	Platonic vector $\mathbf{t}$ measured in coordinate system w. Note this is not a set; it is a single vector.
$\mathbf{e}_A$	(Platonic) vector $\mathbf{e}$ measured in coordinate system A. Abbreviated form of $\{\mathbf{e}\}_A$ used in cases where the meaning should be clear.
$\widetilde{\mathbf{p}}$	Homogeneous representation of position $\mathbf{p}$.
ϕ	Camera projection function.
verOrigin	A variable assigned a descriptive name rather than a single letter, as might be done in a computer program. Slanted Latin letters.
frob	A function assigned a descriptive name rather than a single letter, as is standard practice with functions like $\cos$ and $\log$. Latin letters, math variation.
$\mathrm{cent}(i)$	Part of the n-tuple defining camera i, written as a function. Latin letters, math variation.

SCREW-TRANSFORM MANIFOLDS FOR CAMERA SELF CALIBRATION

Russell Alan Manning

Under the supervision of Professor Charles R. Dyer
At the University of Wisconsin-Madison

This dissertation concerns the mathematical theory of screw-transform manifolds and their use in camera self calibration. A camera's calibration is the function that maps 3D scene points to 2D image points, e.g., in photographs taken by the camera. Between every two photographs taken from different positions there exists a pairwise constraint called the "fundamental matrix" which can be computed directly from the images. When the two photographs are captured by the same camera, the fundamental matrix induces a surface in calibration space called a "screw-transform manifold." This manifold represents every possible internal calibration for the camera. By acquiring several different pairwise fundamental matrices, it is possible to compute several different screw-transform manifolds; however, the internal calibration of the camera must be a member of each such manifold and hence, by finding the intersection point of all the manifolds, the camera's calibration can be determined. The process of determining calibration directly from images taken by a camera is called "self calibration."

The contributions of this dissertation include the theory of screw-transform manifolds and three original algorithms for determining the mutual intersection points of a collection of manifolds. While many papers have been written on self calibration, almost all previous methods posed their solutions as the global minima of an objective function. However, performing global optimization is problematic; it is easy to locate a local minimum without finding the global minimum, and in some cases the attraction basin of the global minimum is so small that the algorithm must essentially "guess" the solution in order to find it. One of the new approaches created as part of this dissertation, called STM-SURFIT, avoids optimization altogether but nevertheless locates all global minima (i.e., solutions to the internal calibration problem) in a single pass, running in

under 1 second on a modern home computer. This general approach to avoiding the difficulties of optimization may have wider applicability beyond camera calibration.

A tutorial on multiview geometry that assumes only knowledge of linear algebra is included to provide the necessary mathematical background. The related history and previous work on self calibration and image-based rendering is also presented. As part of the theory of screw-transform manifolds, a theorem is introduced that partitions monocular view pairs into six categories based on the underlying screw displacement of the camera and provides a simple test for determining category. In addition, some methods for self calibration and image-based rendering for dynamic scenes are presented. These latter image-based rendering techniques do not require camera calibration but are limited in applicability, thus adding to the growing body of evidence that camera calibration is necessary for useful image-based rendering.

Charles R. Dyer

ABSTRACT

This dissertation concerns the mathematical theory of screw-transform manifolds and their use in camera self calibration. A camera's calibration is the function that maps 3D scene points to 2D image points, e.g., in photographs taken by the camera. Between every two photographs taken from different positions there exists a pairwise constraint called the "fundamental matrix" which can be computed directly from the images. When the two photographs are captured by the same camera, the fundamental matrix induces a surface in calibration space called a "screw-transform manifold." This manifold represents every possible internal calibration for the camera. By acquiring several different pairwise fundamental matrices, it is possible to compute several different screw-transform manifolds; however, the internal calibration of the camera must be a member of each such manifold and hence, by finding the intersection point of all the manifolds, the camera's calibration can be determined. The process of determining calibration directly from images taken by a camera is called "self calibration."

The contributions of this dissertation include the theory of screw-transform manifolds and three original algorithms for determining the mutual intersection points of a collection of manifolds. While many papers have been written on self calibration, almost all previous methods posed their solutions as the global minima of an objective function. However, performing global optimization is problematic; it is easy to locate a local minimum without finding the global minimum, and in some cases the attraction basin of the global minimum is so small that the algorithm must essentially "guess" the solution in order to find it. One of the new approaches created as part of this dissertation, called STM-SURFIT, avoids optimization altogether but nevertheless locates all global minima (i.e., solutions to the internal calibration problem) in a single pass, running in

under 1 second on a modern home computer. This general approach to avoiding the difficulties of optimization may have wider applicability beyond camera calibration.

A tutorial on multiview geometry that assumes only knowledge of linear algebra is included to provide the necessary mathematical background. The related history and previous work on self calibration and image-based rendering is also presented. As part of the theory of screw-transform manifolds, a theorem is introduced that partitions monocular view pairs into six categories based on the underlying screw displacement of the camera and provides a simple test for determining category. In addition, some methods for self calibration and image-based rendering for dynamic scenes are presented. These latter image-based rendering techniques do not require camera calibration but are limited in applicability, thus adding to the growing body of evidence that camera calibration is necessary for useful image-based rendering.

Chapter 1

Overview

Perspective is nothing else than the seeing of an object through a sheet of glass, on the surface of which may be marked all the things that are behind the glass.

— Leonardo da Vinci [103]

1.1 Camera calibration

Consider Fig. 1.1, which depicts two devices aimed at a Stonehenge-like menhir. Each device has a rectangular window and an eyepiece consisting of a small, open loop. Wires have been strung across each window to form a grid. As one looks through the eyepiece, gazing out the window at the menhir, positions on the menhir can be assigned 2D coordinates on the grid. This process is depicted in the figure by dashed lines drawn between the lower left corner of the menhir and the center of each eyepiece. Devices such as these first appeared in the Renaissance and are mathematically analogous to a simple type of modern camera called a "pinhole camera." They are also an important component in modeling more complicated cameras. The term "pinhole camera" will be used in this dissertation as a generic label for both these devices and real pinhole cameras.

When a pinhole camera is in a particular position and orientation, it defines a mapping, called the camera's "calibration," from 3D positions in the world to 2D positions on its local grid.[1] The fact that camera calibration has been studied continuously since the early 1850's, only a decade after the first Daguerreotypes and Calotypes were publicly exhibited, is an indicator of its usefulness and of the difficulty in determining it. The main subject of this dissertation is a particular

[1] In a real pinhole camera there is no "wire grid"; this is merely a physical analogy for measurements made on photographs.

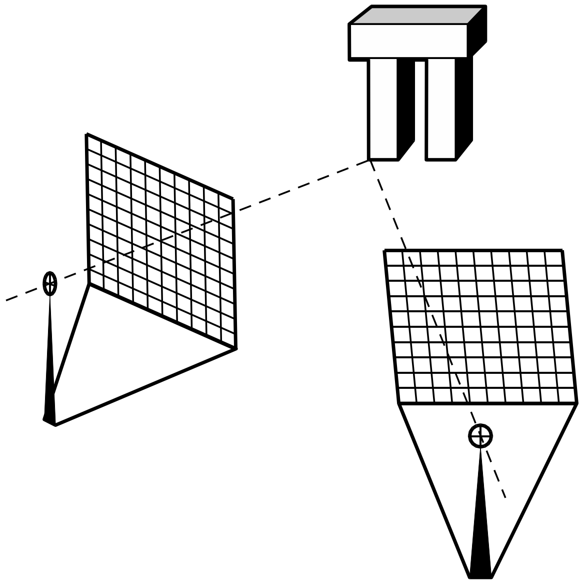

Figure 1.1 Example of pinhole cameras.

approach to determining camera calibration that I developed as a graduate student. My approach is classified as "self calibration" because calibration is determined directly from 2D measurements acquired using the "wire grid" (i.e., measurements in photographs); no 3D information from the world around the camera is used. This makes it possible to automatically calibrate a real camera using only the information contained in the photographs it takes. As an example, self-calibration techniques could be applied to a video stream captured with a hand-held camera to determine the orientation and position of the camera during every frame of the sequence. No information other than the video stream is necessary.

My approach differs significantly from earlier self-calibration algorithms and has several important advantages. Previous algorithms used geometric constraints called "epipolar geometry" to define a nonlinear objective function with a global minimum corresponding to the camera's calibration. This general approach is problematic because the global optimization of complicated objective functions can be very difficult; in particular, a global extremum might have an attraction basin so small that an optimization algorithm must essentially "guess" the correct answer. My approach does not define an objective function. Instead, the epipolar geometry constraints are used to define a collection of surfaces in camera-calibration space with each surface representing a set of legal camera calibrations. The mutual-intersection points of the collection of surfaces represent all possible legal calibrations for the camera. Thus the standard global optimization problem is exchanged for a more direct manifold-intersection problem, which I demonstrate can be solved efficiently and robustly. Along with presenting my algorithm and the accompanying mathematics of screw-transform manifolds, this dissertation emphasizes applications of camera calibration to image-based rendering and modeling and discusses larger issues concerning the true meaning of image-based rendering, the limitations of global optimization, and the upside of computation-intensive algorithms. See the summaries in Section 1.4 and Section 8 for a comprehensive list of contributions.

1.2 Applications of camera calibration

Once camera calibration is known, the primary application is determining 3D positions in the world through a process called "triangulation." The menhir example in Fig. 1.1 illustrates how triangulation works: First, the lower left corner of the menhir is sited through each device to determine 2D grid coordinates for the corner. Next, the calibration function of each device is used to sketch in 3D the dashed lines portrayed in the figure. By intersecting the two dashed lines in 3D, we determine the 3D location of the corner of the menhir. Scene reconstruction through triangulation appeared early in the history of "photogrammetry," the science of using photographs for measurement. It was used to measure buildings and construction sites, to survey battlefields and aim artillery, to measure the heights of mountains, and, in 1885 in its first archaeological application, to document the ruins of Persepolis. Scene reconstruction through triangulation is used in modern computer graphics in many ways, for example to acquire models of real objects (e.g., [120]), to perform motion capture, and to place synthetic objects into real scenes, as is done in movie special effects (e.g., [183]).

Under the Marr paradigm [114] in computer vision, acquiring full or partial 3D models of the objects being viewed was once considered a key step in object recognition and scene understanding. Three-dimensional models may still prove useful for scene understanding when they can be acquired, but there is significant evidence from psychology [165] that human beings understand what they are looking at without first estimating 3D structure. A further application of camera calibration in machine vision is the determination of "ego motion," meaning the camera's position and orientation in the world over time. This can be useful by itself in robotics, and is necessary for triangulation and plenoptic modeling (described below). Note that many of the areas that fall under the rubric of computer graphics are also actively investigated by machine vision researchers; we will distinguish between the two fields by saying computer vision is concerned with image understanding while computer graphics is concerned with image creation.

An important computer graphics concept that requires camera calibration is "plenoptic modeling," [1, 117] which refers to modeling the light that passes through a point in space from every direction. Once modeled, this information can be used to generate new images of the world *without constructing 3D models*. Several examples of plenoptic modeling from real photographs have been published (e.g., [96, 63, 115, 163]); camera calibration makes this possible by allowing rays of light that pass through the camera's grid to be tracked in 3D.

A major application in photogrammetry and computer graphics is techniques for combining photographs to make larger, synthetic photographs called "mosaics." The goal of mosaicing is to make synthetic photographs that are correct in the sense that they might have been obtained with a single wide-angle camera. Mosaicing has been used to create detailed maps from aerial views (e.g., terrestrial maps and maps of the surface of the moon) and to create panoramic views from a single viewing position (e.g., the surface of Mars as seen from the Viking probes). Since mosaicing is a limited form of plenoptic modeling, camera calibration can assist in creating high-quality mosaics. Mosaicing, plenoptic modeling, and scene reconstruction are all techniques used for "image-based rendering," a subject that is discussed in detail in Chapter 2 because of its importance to this dissertation.

1.3 Calibration by manifold intersection

As mentioned earlier, my camera-calibration method is a self-calibration method, meaning calibration is determined entirely from the information present in photographs captured by the camera. When two photographs of the same object are taken from different locations, there exists an "epipolar constraint" between the views, represented with a 3×3 "fundamental matrix"; detailed explanations of these terms are found in Chapter 3. Every time a fundamental matrix can be calculated from a pair of views, it places constraints on calibration. When no constraints are placed on a camera's calibration, the calibration can be located anywhere in "calibration space." However, by imposing constraints we are able to limit calibration to being a position on some manifold in calibration space. With enough constraints, meaning enough manifolds, we can determine camera calibration by finding the mutual-intersection point of all the manifolds.

In order to make this approach to calibration work in practice, efficient and robust methods for intersecting collections of manifolds had to be developed (see Chapter 5). These methods are computationally intensive but avoid the pitfalls of global optimization; in particular, the method called "surface fitting" will not become trapped in local minima. All alternative, general-camera self-calibration algorithms involve the global optimization of an objective function and will fail if the attraction basin of the global minimum is small enough. Of course, there are no shortcuts to optimizing complicated objective functions since global extrema can potentially be hidden anywhere; thus full solutions must inherently involve many computations.

Note that the term "computationally intensive" is relative to existing hardware. In the 18th Century, the square root of a number could be computed by hand using a few iterations of Newton's Method; thus, square root finding was a reasonable computation to perform. In the early 20th Century, analog computers were commonly used to perform computations that would not have been feasible by hand; for example, analog "torpedo data computers" [55] were developed in the 1930's for aiming torpedoes from submarines,[2] and banks of analog calculators operated by people were used to perform calculations for the Manhattan Project [48]. Shortly after digital computers were invented, the Metropolis algorithm [118] was published; this algorithm used randomness as part of its computation. Such an inefficient algorithm would not have been possible before digital computers, for it would have been simply too computationally intensive. Yet Monte Carlo methods have been used very successfully for many applications over the years. In 1959, Hough

[2] A torpedo is fired from one moving vehicle at another; the vehicles are moving different directions at different speeds. Furthermore, the torpedo is fired straight ahead or behind from the submarine and follows a curved trajectory to its target, guided by a gyroscope. The computations involved in aiming the device are significant.

[82] published a “voting method” for automatically locating the straight-line paths of subatomic particles moving through bubble chambers. Hough’s method, now one of the standard computer vision techniques, would have pushed to the limit the abilities of computers when it was created.

The larger point to be made is that, as computer hardware becomes more powerful, algorithms that may have been infeasible at one time become feasible, and thus it is important to design algorithms not just for today’s computers but for tomorrow’s. Directly intersecting manifolds to find camera calibration is a case in point: this approach would probably not have been feasible on typical hardware in the 1980’s, due to both its memory and computational requirements, but can run in under 1 second on a modern home computer. A related claim can also be made: Just as the algorithm designer should keep in mind the potential of new hardware and not be limited by what was feasible in the past, new hardware can be designed in combination with new algorithms. Biological vision systems are extremely reliant on specialized “hardware” (e.g., about 50% of the human brain is devoted to vision) and it seems likely that to achieve comparable performance with machine vision, radically new algorithms will have to be designed in combination with new hardware to make the algorithms feasible.

A final point to be made from the work in this dissertation is that global optimization is not a panacea and simply expressing the solution to a problem in terms of the global minima of an objective function does not mean the problem is solved. Unfortunately, a large number of machine vision papers are based on this approach. Global optimization can represent an ideal solution in cases where the global objective is provably easy to determine (e.g., as with a convex function), but few researchers attempt to prove, either empirically or analytically, that the objective functions they have created are actually feasible to solve. As is demonstrated in this dissertation, good alternatives to global optimization do exist for some problems without needing to weaken or modify the problem.

1.4 Contributions of the dissertation

This dissertation contains the following contributions to the theory and practice of camera self calibration, image-based rendering, and non-linear optimization:

- the mathematical theory of screw-transform manifolds;
- STM-SURFIT, an algorithm for the self calibration of a general pinhole camera (this algorithm, derived from the theory of screw-transform manifolds, does not use global optimization, finds all potential calibrations in a single pass, and runs quickly enough to be used in conjunction with RANSAC for great robustness to noise);
- a theorem partitioning all pairwise rigid displacements into 6 natural categories with a simple test for each category;
- three novel algorithms for manifold intersection;
- extensive experimental evidence showing the performance of STM-SURFIT and related algorithms (these experiments demonstrate, among other things, that high-quality scene reconstructions are possible from self calibration provided the input data is of high quality);
- an empirical demonstration that self calibration from 3 views probably has only 1 or 2 solutions (the lowest proven bound is 21 solutions);
- several algorithms for image-based rendering of dynamic scenes (these are among the earliest published methods on this topic);
- a linear algorithm for affine camera self calibration from dynamic scenes containing apparent linear motion;
- and a demonstration that the direct intersection of manifolds can form a practical and advantageous alternative to nonlinear optimization under the right conditions.

1.5 Structure of the dissertation

Following this overview chapter, there are two chapters that can be read without prior knowledge. Chapter 2 presents a brief history of image-based rendering and camera self calibration. The purpose is not just to present facts about past research, but to express my opinions about the course of development and ultimate significance of these fields. Chapter 3 provides a complete tutorial on multiview geometry from the ground up. The target audience is graduate students in machine vision and computer graphics who have never had exposure to multiview geometry. Only linear algebra is assumed; projective geometry in particular is avoided. Both Chapters 2 and 3 may be read independently of the dissertation and either may prove helpful to students interested in this area.

The heart of the dissertation is Chapters 4, 5, and 6. Chapter 4 presents the mathematics of screw-transform manifolds and Chapter 5 presents three general-purpose algorithms for finding the mutual intersection points of a collection of manifolds. Combining the ideas in Chapters 4 and 5 leads to a complete algorithm for self calibration; Chapter 6 demonstrates the performance of this algorithm on both simulated and real data sets.

Chapter 7 looks at calibration issues in relation to dynamic scenes. Two methods are presented for self-calibrating cameras using the information present in dynamic scenes, and then several algorithms are presented for performing image-based rendering from dynamic scenes with little or no calibration information. The latter results are seen as evidence, in a negative sense, that calibrated cameras are necessary for useful IBR techniques. Chapter 8 provides a detailed, technical summary of the contributions of the dissertation.

Two appendices are also provided. The first provides all the technical derivations for the mathematics of screw-transform manifolds, which were removed from Chapter 4 in order to make the chapter more accessible and readable. The second appendix provides some key implementation details for the manifold intersection algorithms of Chapter 5.

Chapter 2

History and Context

2.1 Image-based rendering: sampling versus simulation

According to conventional wisdom, a new paradigm for creating computer graphics called "image-based rendering" (IBR) appeared in the early 1990's, starting with the landmark paper of Chen and Williams [26]. In this chapter, the meaning and history of image-based rendering is presented and it is argued that, with the perspective of history, IBR existed before the 1990's and should be seen as part of a larger trend in engineering extending beyond the bounds of computer graphics.

In general, IBR concerns methods for converting pre-existing, highly-detailed or otherwise realistic images such as photographs into new, realistic images; the pre-existing images are called *reference views* and the new images are called *virtual views* or *synthetic views*. In the problem that Chen and Williams considered specifically, the reference views were not photographs but were instead highly-detailed computer graphic images generated with a computationally-intensive rendering algorithm. Their goal was to generate "interpolation sequences" between the pre-existing views in a computationally-efficient manner (e.g., efficient enough to run in real-time, as opposed to the algorithm that generated the reference views). An interpolation sequence is a series of images forming a smooth transition between two views, and the task of creating interpolation sequences is known as *view interpolation*; we will return to this subject later.

Image-based rendering had existed before Chen and Williams in various forms, but had never been given a formal designation or been seen has a paradigm. The classic example of using realistic, pre-existing images to create realistic computer graphics is "texture mapping," which originated in the 1970's [24]. Using texture mapping, a realistic-looking computer graphic image of a wood floor might be generated by mapping a photograph of the surface of a real wood floor onto a simple geometric model of the floor (i.e., a plane). Shortly after the invention of texture mapping by Catmull, a technique for "reflection mapping" (also known as "environment mapping") was published by Blinn and Newell [16]. In environment mapping, an image of the environment surrounding an object is used in rendering the object. If, for instance, the object has a mirror surface, then the surrounding environment would be seen reflected in the surface adding to the sense of realism [194, 29]. Diffuse surfaces are affected in the same way as mirrored surfaces, only they reflect the incoming light differently [119]. Environment mapping has been used with great success in movie special effects for inserting synthetic objects into real scenes [125, 187, 129]. By mapping the surrounding real environment (e.g., of the movie set) onto the surface of a computer-graphic model, the computer-generated object can be made to seem like it was part of the original scene; that is, it will better meet the perceptual expectations of viewers of the movie. In the original work of Blinn and Newell, the environment map was a hand-drawn computer graphic image; however, later work, for instance in movies, has used photographs of real environments [64].

What distinguished the work of Chen and Williams from earlier methods, enough to give impetus to a new graphics paradigm, was the extent to which the reference views were used. In texture and environment mapping, 3D models are created and rendered in the traditional manner (e.g., by specifying vertex coordinates, surface normals, etc.) and the reference views are used to add a level of detail that would otherwise be hard to model specifically. For example, a photograph of a wood floor is filled with details: the grain of the wood, variations in color, knots, scratches, dirt, and other patina effects. By covering a 3D model with texture drawn from the photograph, the model instantly has all the details present in the photograph without needing to specifically model dirt or scratches. In contrast, the paper of Chen and Williams dispensed with traditional 3D models entirely and rendered new views directly from the reference views. Although no models were used,[1] the synthetic views generated by Chen and Williams were *physically correct*, meaning

[1] Arguably, enough information existed to create a projective reconstruction and this projective reconstruction was used implicitly whether or not it was explicitly computed during execution of the algorithm.

they had the same shape properties (although not the same lighting properties) they would have had if generated using 3D models.

Another landmark paper, published a year earlier than the work of Chen and Williams, was the Beier-Neely morphing algorithm [14]. This algorithm could be used to generate computer graphic images portraying a smooth transition between two existing reference photographs. Generating an interpolation sequence from reference photographs of real scenes gave the sequence a high level of realistic-seeming detail, far more than would have been practical to create by hand following the traditional computer graphics paradigm. Furthermore, the Beier-Neely algorithm required relatively little computational power, especially in comparison to the apparent realism of its results. However, there was no guarantee that the interpolation sequences were physically correct with respect to some underlying 3D model of the scene as there was with the Chen and Williams work. It should be noted that, while Beier and Neely's algorithm became well known starting with its publication in 1992, image warping and morphing had been used for several years prior by private companies producing commercials and movie special effects [102, 195]. The private companies had used several different algorithms produced by various researchers working independently.

Thus by the early 1990's, graphics researchers started to realize that highly-detailed computer graphics could be created in a computationally-efficient manner by using existing, highly-detailed images (e.g., real photographs or computer graphics images generated using computationally expensive algorithms like ray tracing [3, 192] or radiosity [62]). Whether the synthetic imagery was physically correct (as in the paper of Chen and Williams) or simply had realistic-looking detail (as in morphing and warping), what made image-based rendering a new paradigm, distinct from traditional computer graphics, was the emphasis on acquiring detail from existing images rather than generating detail with elaborate models and physical simulation.

Over the years, researchers in several disparate fields have come to the same general conclusion, that *sampling* from nature is in many cases more efficient, simpler, and perhaps even more desirable than *simulating* nature. A prime example is electronic music devices. In the 1950's and 1960's, electronic organs with built-in synthetic drum machines started to appear for the home market. These machines could create a cymbal-like sound, for example, by passing random electronic white noise through a series of filters. Electronic organs themselves tried to emulate a wide range of musical instruments using electronic tricks. The effect, however, was always synthetic sounding.[2] In the 1980's, electronic instruments were made that sampled the sounds produced by real instruments and simply played them back, producing an effect essentially identical to the original instrument.

Other examples come from the areas of voice synthesis and motion capture. While there has been research performed on trying to physically simulate the human voice box or to artificially replicate phonemes (e.g., Dudley's work from 1939 [36]), the simplest and, it turns out, best method for speech synthesis is to sample phonemes produced by human speakers and then recombine them to produce arbitrary words (e.g., [86, 19]). In trying to make computer graphics containing realistic movement or facial expression, one could attempt to model muscles and the physics of motion, but it is far easier to record real movement from people and animals and then attach this data to computer models. The highly-realistic facial animation in the short movie "The Jester" [128] was achieved in this manner. Several researchers (e.g., [193, 144]) have worked on blending or reshaping captured motion data to create new motions in much the same way that voice synthesizers recombine phonemes to pronounce arbitrary words.

Following in the wake of Chen and Williams, a number of papers were published on image-based rendering (or sampling for rendering). Because research in IBR at this point branched out in different directions rather than following a linear path of development, we present below a brief survey of important results as a series of disjoint "micro-sections." Each micro-section includes a title indicating its content for convenient reference; the micro-sections are self sufficient and can be read in any order. This listing is not exhaustive but rather is intended to give the reader a sense of the wide variety of work that has been done in IBR and of the importance of this field.

DEVELOPMENT (Mosaicing). Mosaicing (see Szeliski [175, 176]) is the process of stitching together several reference photographs to make one large photograph. If all the reference views

[2] In some cases, the synthetic sound was sought after in its own right. Consider the success of the album "Switched-On Bach" [22] (released in 1968) and the genre of "music from space."

share a common optical center, then the mosaic will be a physically-correct view having that optical center; McMillan and Bishop [117] used this process to create cylindrical reference views for their seminal paper on plenoptic modeling. Alternatively, mosaics can be created from sequences of orthographic views to make a large orthographic view; this is a traditional application of photogrammetry used for making maps. Burt and Adelson [21] used image pyramids to help remove seams when piecing together images in a mosaic; they were specifically interested in creating seamless mosaics from space-exploration imagery. Sawhney et al. [149] studied the problem of correcting for noise in point correspondences when creating very-large-scale spherical mosaics. Szeliski and Shum [177] also studied the problems associated with large-scale mosaics, using local deformations to correct for ghosting artifacts. The "video brush" system [130] could create mosaics in real time by incrementally adding only the central 1D slice of each successive view to the growing mosaic. Rousso et al. [146] used projection onto a cylinder to create lateral mosaics from forward-looking views captured by a camera moving forward. Irani et al. [85] used image registration to increase image resolution.

Several authors have utilized scene motion with mosaics: for example, Irani et al. [83] did work on creating mosaics from video sequences in which the scene movement can be viewed in the context of one final, large-view mosaic; Davis [28] removed motion to create static mosaics from dynamic scenes; and Caspi and Irani [23] used motion to register views that had little or no visible overlap.

DEVELOPMENT (View interpolation and view synthesis). View interpolation is the process of creating a sequence of images representing a smooth, continuous transition between two reference views. Image morphing (e.g., [14]) is an example of view interpolation. Morphing has been used to create short special effects and stylistic imagery for movies, commercials, and music videos [102, 61, 56], and the creators of the commercial morphing product "Elastic Reality" were awarded an Oscar for technical achievement. Thus view interpolation had a profound impact on the establishment of image-based rendering as an important discipline. More recently, Kanade et al.

[88, 178] have used view interpolation as part of the "virtualized reality" project. Joint view triangulation [98] is a technique for approximately solving the dense correspondence problem implicit to view interpolation and has been used to create realistic-looking interpolation sequences.

Creating physically-correct view interpolation sequences is an extra challenge and most formal papers on interpolation have had this objective (e.g., [191]). Ullman's work [182], arguably the earliest on view interpolation, created physically-correct interpolation sequences from orthographic views. The seminal paper of Chen and Williams [26], which is commonly seen as the birth of image-based rendering, fits into this category. Seitz and Dyer [154] demonstrated a technique for making the Beier-Neely morphing algorithm physically correct. Manning and Dyer extended the "view morphing" technique to dynamic scenes with apparent linear motion [109] and to interpolating environment maps [111].

View synthesis is closely related to view interpolation, with the difference being that interpolation sequences are not necessarily the goal. Avidan and Shashua [6] demonstrated the ability to synthesize new, physically-correct views from arbitrary viewing positions using uncalibrated reference views and the trilinear tensor between them. Similar work, but based on fundamental matrices and thus less stable, was published by Laveau and Faugeras [95]. Irani and Anandan [84] demonstrated a novel view-synthesis technique that did not require multiview relationships like fundamental matrices; instead, a reference plane and many views are required, and ideally the scene should contain discrete objects separated by empty space.

DEVELOPMENT (Plenoptic modeling). Plenoptic modeling describes IBR techniques in which the rays of light entering a region from different directions are directly stored and later used to synthesize new views. McMillan and Bishop's seminal 1995 paper on plenoptic modeling [117] was one of the initial driving forces for IBR, although the idea of the plenoptic function was formalized at least as early as 1991 [1]. The important technique of light field rendering [96] (and the closely-related lumigraph technique [63]) appeared a year after McMillan and Bishop's paper. Light field rendering is most simply described as a computerized hologram. The technique allows the user to view a scene from any direction in a random-access manner, just as a person can tilt a hologram to see different views of a scene. It also provides for smoothly interpolating between closely-spaced views without the determination of point correspondences or optical flow. Drawbacks to the technique are the need for precise camera calibration and the large amount of data that must be stored. The great advantage is that scene reconstruction is unnecessary, and thus the difficult correspondence problem need not be solved. Light field rendering also produces highly-compelling results. Subsequent work has produced a Nyquist-like limit on the number of reference views required for light field rendering [25] and a demonstration that plenoptic modeling can be performed in conjunction with self calibration [75].

DEVELOPMENT (Image-based modeling). After the term "image-based rendering" had emerged, it became evident that many of the best image-based rendering techniques involved scene reconstruction. Thus the term "image-based rendering and modeling" (IBRM) came to replace the earlier phrase. Image-based modeling is the building of scene models from photographs (or other 2D acquisition devices like CCD's). The goal is high-quality models for use in computer graphics. The term "model" does not just refer to 3D shape, but also to light-reflectance properties and any other physical attribute one might wish to incorporate. Any engineering tool that can improve the model acquisition process can be utilized: the cameras can be calibrated, the motion of the cameras or of the scene objects can be controlled, and point correspondences can be precisely acquired with structured light. Perhaps the best tool for acquiring precise 3D models is a laser range scanner; e.g., Cybertron scanners are commonly used in IBRM research, such as the Digital Michaelangelo Project [97]. Of course, model acquisition should also be simple and use low-cost equipment if possible; this is where self-calibration can play a role.

The seminal "Facade" system by Debevec et al. [31], published in 1996, demonstrated the power of using models, even approximate models, for IBR. Scene reconstruction from reference views was not new, having been one of the first uses of photogrammetry in the mid 19th Century. However, the Facade system and accompanying demonstration movies convinced many people that IBRM could be a useful technique. The special effects designer for the highly-successful movie "The Matrix" [186] decided to use IBRM in the movie after seeing demonstrations produced with "Facade" [164]; he even hired one of the key researchers from the "Facade" project to help implement the special effects.

In the Facade system, a small number of calibrated reference views of a scene were captured and then a human user, with machine assistance, used simple geometric shapes like rectangular prisms, planes, and pyramids to approximately model what was seen in the reference views. The system was most appropriate for architecture, which tends to be made of simple geometric shapes. After the scene had been approximately modeled, new "virtual" views of the scene could be acquired from arbitrary positions using arbitrary virtual cameras. To complete the effect, texture was mapped onto the scene models by blending textures from the nearest reference views; thus the model's texture depended on viewing position, leading to "view-dependent texture." The reason this approach works well is that the human brain is very good at estimating scene shape from visual cues like shadows which are present in the textures. Thus, despite using only simple geometric models, the virtual views produced by Facade looked like they were generated from highly-detailed models because they retained the details from the original reference views.

Image-based modeling is a standard tool in movie special effects. Approximate scene models are constructed from camera views not to create new virtual views from different angles but to place synthetic objects into a scene. At one time, expensive survey crews were used to acquire precise but sparse models of film sets to assist with special effects. In addition to expense, this also required a lot of preplanning and prevented spur-of-the-moment changes. For the movie "Dinosaur" [188], which required outdoor shots of mesas located far from the movie studio, survey crews had to be flown to the site to acquire local geometry. Furthermore, large white markers had to be physically placed into the scene to tie the surveying information to the visuals, and later these markers had to be digitally removed from the film. In theory, self calibration could have been used to reconstruct the scene just from the images of the mesa without the need for surveying the area; this would also have made last-minute changes easy to accomplish.

DEVELOPMENT (Image-based lighting). Image-based lighting involves acquiring illumination information from real scenes and then using this information to light synthetic renderings (e.g., [29, 199]). Alternatively, it can concern the synthetic lighting of real objects before compositing in order to match the lighting in the destination scene (e.g., [202, 30]). Image-based lighting is not necessarily image-based rendering but fits into the larger paradigm of "sampling over simulation."

DEVELOPMENT (Multiple-center-of-projection images). Rademacher and Bishop [143] produced stylistic "multiple-center-of-projection" images that could be used in conjunction with compression of video streams. As a camera was moved around a computer-generated scene, information about camera calibration was stored along with the center strip of each view. The central strips, when pasted together in sequence, formed unusual and compelling images. Some mosaicing techniques also have multiple-center-of-projection properties, especially the pipe-projection mosaicing of Rousso et al. [146]. It is interesting to note that multiple-center-of-projection images have been in use for at least a century: they were used by museums to document as flat images the designs on ancient pottery (see Fig. 11 of [18]). This could be achieved, for instance, by placing a vase on a turntable and recording the vase with a special camera that only captures a thin vertical strip at each instant. As the vase is rotated on the turntable, the camera's film is translated horizontally to produce a final flat image depicting the design on the vase's curved surfaced.

DEVELOPMENT (Texture synthesis). In texture synthesis, a probability distribution (e.g., of colors and intensities) is acquired from a small sample of texture and then used to synthesize the texture over a large area. The key paper in this area is due to Effros and Leung [39] and is based on an observation by Shannon.

DEVELOPMENT (View augmentation and virtual reality). View augmentation (e.g., [9, 58, 94, 29]) is the placement of a synthetic object into a real view. This is a key method for special effects in movies, and can also be used to provide information to a person viewing a real scene through a head-mounted display (e.g., [116, 47, 138]). The key problems are keeping track of camera calibration (i.e., of the person's view), especially external calibration, and in some applications reconstructing the scene being viewed. Kanade et al. [122] developed a stereo system that could calculate scene depths in real-time and place synthetic objects into the scene with correct occlusions.

DEVELOPMENT (Hallucination and other synthesis techniques). Baker and Kanade [11, 10] created a technique they termed "hallucinating faces." In this technique, probabilistic information acquired from a large database of faces was used to fill in missing details in novel images of faces.

For example, if a face were partially occluded in a photograph, the database could be used to complete the missing portion. This technique was also applied to face recognition.

Other related IBR techniques exist for synthesizing classes of objects. Blanz and Vetter et al. [15] acquired 3D range-scan data for the faces of hundreds of subjects and then treated the data as a vector space spanned by a small number of significant basis vectors. 3D models of new faces, not already in the database, could be effectively synthesized as linear combinations of the basis vectors. This approach was demonstrated by creating 3D face models of famous celebrities using only a single photograph of each person: first the lighting of the scene in the photograph was estimated and then the face vector space was searched to find a good fit to the lighting and shadows evident in the image.

Using a similar approach but with a database of 2D face images, Cootes, Edwards, and Taylor [27, 38] were able to synthesize different expressions for a given face using only one 2D reference view. They could also make the face look older or younger or blend one face with another (e.g., blending a male and female face).

DEVELOPMENT (Time dilation). Shechtman et al. [162] demonstrated a technique for using several regular video cameras in orthographic configuration to create a slow-motion video sequence of a dynamic scene. For example, four cameras filming at 25 frames per second could produce a 100 frames-per-second slow-motion sequence.

2.2 Self calibration and scene reconstruction

Mathematicians and engineers have studied camera calibration since shortly after the birth of photography. Calibration was the key component in "photogrammetry," the science of measuring distances using photographs or related 2D phenomena, which are generically called "photogrammes." The mathematics of photogrammetry originated with the study of perspective painting in the Renaissance. Questions regarding perspective lead to the development of projective geometry with researchers like Girad Desargues and Blaise Pascal in the mid 17th Century. Poncelet formalized the study of projective geometry in the eary 19th Century. Although projective geometry can be used for photogrammetry, the two fields should not be confused. Horn [80] points out that real cameras have more constraints than projective geometry models allow for, and thus the more general nature of projective geometry can lead to undesirable and unnecessary ambiguity. Horn also states that projective geometry is rarely used in photogrammetry, citing Wolf [197] and Slama, et al. [166]. Mathematicians have published work specifically related to camera calibration and photogrammetry since the 19th century. For example, Hauck [74] in 1883 with "New Perspective and Photogrammetric Constructions" ("Neue Konstruktionen der Perspektive und Photogrammetrie") and Finsterwalder [50] in 1899 with "The Geometric Foundation of Photogrammetry" ("Die geometrischen Grundlagen der Photogrametrie"). Kruppa, working within the framework of projective geometry, made important contributions to photogrammetry in the early 20th Century [92]; the "Kruppa Equations" are still commonly used today.

Although specially-constructed equipment was always a part of photogrammetry, photogrammetrists started using more automated techniques called "analytic methods" beginning in the 1950's. Analytic methods made more intensive computations possible, which in turn opened up new approaches to photogrammetry as per the discussion at the end of Section 1.3. The era of analytic methods was succeeded by the digital era, in which digital images replaced photographs and algorithms were designed to utilize the ever-increasing computational power of digital computers. It was early in the digital era of photogrammetry that Longuet-Higgins [99] published his seminal paper on camera calibration, "A Computer Algorithm for Reconstructing a Scene from Two Projections." This paper introduced a linear algorithm for finding the external calibration of two cameras with known internal calibration using only information available in the two camera views. The method involved the determination of a 3×3 matrix later known as the "essential matrix" between two views.

The paper of Longuet-Higgins set in motion a chain of research leading ultimately to camera self calibration, from which this dissertation springs. Most important in the chain were two landmark papers published by Faugeras in 1992 at the European Conference on Computer Vision. One paper [44] introduced the "fundamental matrix" between two views, which is a more general form of the essential matrix that does not require knowledge of internal calibration. This paper also demonstrated that physically-valid new views of a scene could be created from existing, uncalibrated perspective reference views without needing to build a model of the scene. Using later parlance, it could be said that Faugeras actually performed “projective scene reconstruction” in order to create new views; the important point is that it was not necessary to calibrate the cameras or to build models resembling the original scene. Note that Ullman [182] had published a similar result in 1979 for uncalibrated orthographic views, which are simpler than uncalibrated perspective views.

The second paper [41], coauthored by Luong and Maybank, introduced the possibility of camera “self calibration.” As mentioned earlier, self calibration is the process of calibrating a camera using only images captured by the camera. Thus, using self calibration, it is possible to move a camera around a scene while taking pictures and then later determine the camera’s position, orientation, and internal properties directly from the resulting photographs. There are different kinds of self calibration algorithms, each having different requirements and producing different amounts of information. The algorithm of Faugeras et al. only produced internal camera properties and assumed (as do most self calibration papers) that the camera could be modeled as a pinhole camera. This algorithm was built on the “Kruppa constraints,” which were first published in 1913, and utilized the fundamental matrix.

The fundamental matrix is part of a larger group of multiview relationships that began to appear in the late 1980’s (e.g., [168, 167, 189]), from which the the “trilinear tensor” [159] (also called “trifocal tensor” [69]) ultimately emerged; Section 2.4 of Stein [171] relates the origins of trilinear relationships. A trilinear tensor relates three camera views; there exist relationships for higher numbers of views, such as the quadrifocal tensor, but these ultimately are equivalent to trilinear tensors. Both the trilinear tensor and fundamental matrix can be determined from point or line correspondences between views, or from combinations of point and line correspondences. Stein [170] performed some work on the direct determination (i.e., from optical flow) of trilinear tensors for closely-spaced views, and Brodsky et al. [20] performed related research on self calibration from optical flow. In general, except for closely-spaced views, it appears that multiview relationships require some knowledge of point or line correspondences.

Self calibration became a very active area of research following the paper of Faugeras et al.. There are three reasons for the interest in this topic. First is the long-held belief in the Marr paradigm, which held that scene reconstruction was an important part of image understanding; since camera calibration is necessary for scene reconstruction, self calibration could provide the missing step for converting images directly into models. Second is the growing importance of image-based rendering, which emerged as a separate discipline at almost the same time as self calibration. Again, self calibration makes model acquisition easier, and model building is important to IBR (see Section 2.1). The third reason is that self calibration is a very mathematical topic and as such results in self calibration are usually accompanied by proof of their correctness (for perfect input data). Thus papers on self calibration tend to produce "eternal" results that will be correct and part of scientific knowledge forever after. This contrasts with many papers in machine vision that tend to be heuristic in nature.

The following summary of developments in self calibration is meant to give the reader a sense of the wide variety of work that has been done on this topic; it is not exhaustive and some of the items listed are from prior to 1992 or are otherwise unrelated to Faugeras' work. See Pollefeys [131] for a more technical overview of developments in self calibration. Most of the terminology used in this summary is defined in Chapter 3. Some of the methods described below only involve finding external calibration (the position and orientation of a camera, perhaps relative to another camera) and as such could simply be considered structure-from-motion algorithms rather than self calibration algorithms. They have been included for various reasons involving their relationship to full self calibration algorithms.

DEVELOPMENT (Direct self calibration). Direct self calibration is the determination of metric internal calibration directly from pairs of fundamental matrices, without any intermediate steps such as projective scene reconstruction or affine calibration. Relative camera orientations are not determined. Direct self calibration is my own term, since none exists in the literature, created to contrast this style of self calibration with stratified self calibration. The original self calibration method of Faugeras et al. [41] is in this category, as is the method of Manning and Dyer [104].

Hartley and Zisserman [70] express pessimism that direct self calibration can be successful in practice.

DEVELOPMENT (Stratified self calibration). In contrast to direct self calibration, stratified self calibration [45, 35] proceeds in a series of stages, with calibration becoming better determined after each stage. First, projective reconstruction of the scene, including all the cameras, is performed. Then the projective reconstruction is upgraded to affine and finally to metric. The advantage of this approach is that all views will share the same measurement basis during each step of the algorithm; thus, for instance, the plane at infinity and the absolute conic will be located in the same position for each camera. A prime example of stratified self calibration is the modulus-constraint approach of Pollefeys and Van Gool [133]. Pollefeys [131] asserts that his stratified approach produces better calibration results than the direct method of Faugeras et al..

DEVELOPMENT (Projective or weak calibration). The first stage of stratified self calibration is projective reconstruction. Projective reconstruction of two camera views is possible whenever the fundamental matrix between the views is known, in which case the two views are said to be "weakly calibrated" relative to each other. Any research into determining fundamental matrices or epipolar geometry directly from camera views (e.g., [168, 67, 72, 160, 34, 145, 200, 147]) is research into projective self calibration.

DEVELOPMENT (Affine calibration). Affine reconstruction occurs when the reconstructed plane at infinity is coplanar with the true plane at infinity. This is the second phase of stratified reconstruction. The reconstructed cameras in an affine reconstruction are said to be affinely calibrated. Affine reconstruction is close to metric reconstruction, the most-complete reconstruction possible using camera views solely, and has a variety of uses [112]. For example, it is necessary for the linear interpolation of scene motion in view interpolation sequences [109]. There are many cases where affine calibration can be determined even when metric calibration cannot be, such as when two views are separated by a translational displacement [121]. Various authors (e.g., [33, 139, 12, 174, 142]) have done representative work on affine calibration.

DEVELOPMENT (Orthographic or affine views). Orthographic camera views are not affected by distance to the scene (i.e., they have no perspective effects). An example is views captured from a distance using a telephoto lens. The x- and y-coordinates in such views represent direct measurements of the scene. By assuming square pixels, Tomasi and Kanade [179] produced an elegant "factorization" method for self calibration and scene reconstruction from orthographic views. Shapiro [158] extended this approach to general affine cameras (with non-square pixels). Some authors have shown that affine views can be used directly without the need for calibration; for instance, Ullman [182] showed that new camera views can be created directly from orthographic views by interpolating point correspondences, and Koenderink and van Doorn [91] showed that affine scene reconstruction was possible from two affine views.

DEVELOPMENT (Absolute quadric). Triggs [181] introduced and studied the absolute quadric. The quadric is represented using a 4×4 matrix in the same way a conic is represented using a 3×3 matrix. The 4×4 matrix encodes information about internal camera calibration and the location of the plane at infinity in such a way that it can be used for self calibration. Triggs' self-calibration algorithm required nonlinear optimization.

The absolute quadric is discussed further in Section 3.5.5, where a detailed example is provided. As will be seen, using the absolute quadric for self calibration has two main advantages: it is mathematically simple and the cameras being calibrated do not need to be internally identical if other constraints are known about them. The latter property was the basis for a Marr-Prize-winning paper by Pollefeys, Koch, and Van Gool [132]; the property arises because the absolute quadric makes it possible to relate certain internal calibration parameters directly to projective camera matrices.

There are three drawbacks to using the absolute quadric. First is the need for nonlinear optimization, which we have argued against in Manning and Dyer [106]. However, note that if enough assumptions are made about internal calibration then a linear calibration algorithm is possible [132]. Second is the subtle relationships that the absolute quadric provides between internal calibration and projective camera matrices (see Section 3.5.5); the constraints thereby arising could be numerically unstable. Third is the introduction of bias towards one selected camera view, for which the absolute quadric is calculated. Such considerations are part of Horn's general complaint against using projective geometry for camera calibration [80]. Nonetheless, the absolute quadric has become very popular for self calibration.

DEVELOPMENT (Mosaics). Mosaics can theoretically be created from uncalibrated views, but knowing calibration can help stabilize errors due to noise and can lead to more natural-looking output. Hartley [68] pointed out that when a single camera with fixed internal parameters is used to create a mosaic with a single optical center, then the camera can be self calibrated if at least 3 reference views are utilized. McMillan [117] also performed a more limited form of self calibration in conjunction with mosaicing.

DEVELOPMENT (Critical motion sequences). Sturm [173] investigated critical motion sequences, which are collections of monocular camera views (i.e., all captured by the same camera) for which self calibration does not yield unique results. Pollefeys [131] expanded upon the list of critical motion sequences.

DEVELOPMENT (Variable focal length). Most self-calibration work assumes a camera with constant internal parameters. This is generally a good assumption for short time sequences (e.g., with closely-spaced views). The most likely internal parameter to change is the camera's focal length (e.g., due to zooming in and out). Pollefeys [132] studied the problem of self calibration with unknown focal length. Unfortunately, when a camera zooms there is usually a change of principal point as well as focal length. Fortunately, determining principal point incorrectly does not seem to affect scene reconstruction very much.

DEVELOPMENT (Radial lens distortion). Almost all self calibration research assumes a pinhole camera model, which does not account for the lens distortions effects that exist in real camera views. The most prominent lens distortion effect is "radial distortion" (also called "fish-eye" or "barrel" or "pincushion" distortion) in which projected light is moved away from the image center based on its distance from the image center. Stein [169] showed how a 3-view trilinear tensor could be used to determine radial distortion. Since self calibration requires at least 3 views, this means both radial distortion and internal calibration could be determined from 3 camera views.

Farid et al. [40] discussed the possibility of determining radial lens distortion directly from the power spectrum of an image, and Sawhney and Kumar [148] used mosaicing to correct for radial distortion.

DEVELOPMENT (Dynamic scenes). The use of multiple, independently-moving objects to perform self calibration has been investigated by Manning and Dyer [107, 112], Fitzgibbon and Zisserman [53], and Wolf and Shashua [196], among others. Having a single moving object present in a scene (with no static background) provides no more information than a single moving camera viewing a static scene, but having multiple moving objects introduces new information due to the motion of the objects relative to each other. Stein et al. [172] used the accumulation of motion information over time to provide point and curve correspondences between views, which could subsequently assist in camera calibration.

DEVELOPMENT (Stereo rigs). A pair (or more) of cameras rigidly connected to each other with unchanging relative orientation is called a *stereo rig*. Two cameras mounted rigidly on a robot is an example. Stereo rigs are of particular interest for recovery of dense stereo correspondence information (e.g., [126, 8]). Typically, a stereo rig will be moved through a scene and self calibration of all the cameras or determination of their relative orientations will be performed from the subsequent views. Many researchers have demonstrated various approaches to self calibration with stereo rigs: for example, Deriche et al. [34] investigated weak self calibration from stereo rigs; Quan [139], Sturm and Quan [174], and Zisserman et al. [201] studied affine self calibration from stereo rigs; Horaud and Csurka [78] investigated metric self calibration from stereo rigs; Stein [170] investigated self calibration of external parameters from optical flow using a 3-camera rig; and Manning and Dyer [112] demonstrated a linear algorithm for the affine self calibration of a stereo rig in the presence of apparent linear scene motion.

DEVELOPMENT (Nonstandard cameras). While most self calibration work has assumed a pinhole camera model, there has been some work involving alternative camera geometries. With catadioptric cameras, there are extra parameters (beyond the usual parameters of camera calibration) such as the curvature of the mirror surface; Geyer and Daniilidis [57] investigated this problem for certain kinds of catadioptric cameras. Also, several authors [141, 46] have investigated the potential benefits of using 1D cameras.

DEVELOPMENT (Horn's critique of using projective geometry for self calibration). Horn [79, 81] advocated the use of quaternions in solving the relative orientation problem (which is the problem of finding the angle of rotation and direction of displacement between two cameras with known internal calibration). Horn's approach is iterative and involves direct, physical modeling of the cameras. As stated earlier, projective geometry in some cases over generalizes calibration problems, leading to spurious solutions. For this reason Horn [80] has questioned its use when unnecessary, even in cases where it leads to linear algorithms. However, for unconstrained self calibration of internal parameters the use of projective geometry seems necessary because all potential solutions must be considered.

Chapter 3

Introduction to Camera Calibration and Multiview Geometry

This chapter presents the formal concepts and notation that will be used in the remainder of the dissertation. It also serves as a stand-alone introduction to multiview geometry. The goal is to be brief and to use basic rather than advanced mathematics; in particular, linear algebra will be used whenever possible in favor of projective geometry. Instead of providing a narrative or tutorial, the subject matter is presented as a series of disjoint vignettes (i.e., an outline) in the style of mathematical definitions plus examples and comments. This will make it easier for the reader to refer back to the meaning of individual concepts later in the dissertation, and will also allow the reader to skip topics with which he is already familiar.

For a complete discussion of concepts related to camera calibration and image-based scene reconstruction, see the excellent recent books by Hartley and Zisserman [70] or Faugeras and Luong [42], or the older book by Faugeras [43]. Many of the concepts presented here are sufficiently basic to be covered in general machine vision textbooks (e.g., [37, 54]). In addition, some recent Ph.D. dissertations relating to calibration and scene reconstruction (e.g., [171, 131, 115]) provide alternative presentations of the introductory material. Most of the resources listed above build their presentations around projective geometry or other mathematical concepts, such as tensors, that most scientists have never been exposed to; as was stated earlier, one major goal of this dissertation is to present the material in an alternative manner using basic mathematics and a minimum of formal projective geometry.

3.1 Platonic labels and coordinate systems

Consider the tip of your nose. The tip of your nose, at a particular instant in time, represents a position in space. The "tip of your nose" is also a Platonic concept; it exists in an abstract sense. We can think about this concept, and as such we may wish to assign a label to it. For example, we might denote the concept by the label T, which will be called a *Platonic label* in this dissertation. Here, the concept labeled T also represents a position in space so we will use the more specialized notation $\mathbf{t}$. The label $\mathbf{t}$ is a *Platonic vector*; it serves merely to denote a Platonic concept that also happens to be a position in space. To give $\mathbf{t}$ a more specific meaning, we need "coordinate systems."

Fig. 3.1 shows a position $\mathbf{t}$ in $\mathbb{R}^2$ as a Platonic concept, and then demonstrates some specific coordinate systems for measuring the position. Measuring a position with different coordinate systems is like expressing an idea in different languages: the idea remains the same but the words are different. This is another way to think about Platonic labels, as denoting an idea independently of representation. Note that a coordinate system is also a Platonic concept, and as such we can assign a Platonic label to it. In this chapter, lower case Latin letters in math font, such as w, will be used as Platonic labels for coordinate systems; this is expedient to typesetting as it produces formulas that are easier to read. The notation $\{\mathbf{t}\}_w$ will be used to denote the Platonic vector $\mathbf{t}$ as measured in coordinate system w. Note that in most later chapters, capital letters like W and A will be used for coordinate systems, and that at times (especially in Chapter 4) the simpler notation $\mathbf{t}_w$ will be used for the more explicit $\{\mathbf{t}\}_w$.

Platonic labels are a key notational device in this dissertation. Any concept can be assigned a Platonic label. For example, we will use Platonic labels for concepts such as "screw displacement scenario," "image plane," "pinhole camera," and "image coordinate system," as well as various lines, planes, positions, and directions. Computer scientists might appreciate the following analogies for Platonic labels. The first analogy is with "denotational semantics," in which a computer program in a particular language exists only abstractly until precise meaning is assigned to the

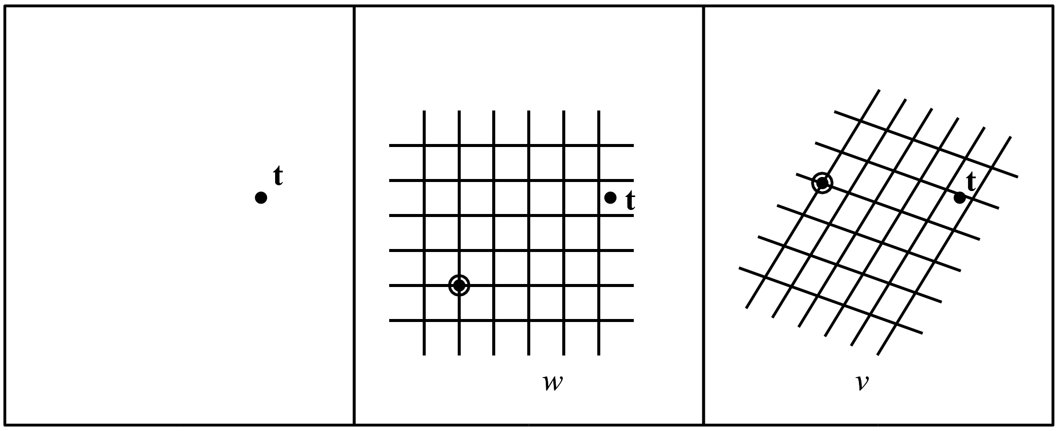

Figure 3.1 *(left)* A position $\mathbf{t}$ in $\mathbb{R}^2$; *(center)* a particular coordinate system w in which $\mathbf{t}$ can be measured; *(right)* an alternative coordinate system v.

language constructs. The program is the Platonic concept and applying denotational semantics is like assigning a coordinate system to a Platonic vector.

The second analogy, more specific to the use of Platonic labels in this dissertation, is with instances of "structures" in computer languages like C and Pascal. A coordinate system can be thought of as a data structure, with the origin and basis vectors of the coordinate system being data fields within the structure:

```
struct CoordinateSystem {
   Vector         origin;
   Vector         basisX, basisY, basisZ;
};
```

The Platonic label w for a coordinate system is like a variable representing a particular instance of the data structure `CoordinateSystem`. Alternative coordinate systems would be different variables and would be given different labels but would all have the same underlying structure.

We will use notation such as $\mathrm{orig}(w)$ to represent different parts of the Platonic concept denoted by w ; in this case, $\mathrm{orig}(w)$ might mean "the origin of coordinate system w." Note that the origin $\mathrm{orig}(w)$ itself is a Platonic vector, and thus we might use notation like $\{\mathrm{orig}(w)\}_i$ to mean "the origin of coordinate system w as measured in coordinate system i." Computer programmers may have preferred more full-blown functional notation like $\mathrm{OriginOf}(w)$ with its easy-to-remember mnemonic, or perhaps member-reference notation like `w.origin` or even `w.OriginOf()`. However, we have chosen a more mathematical-style notation analogous to the standard $\cos$ and $\log$ notation for cosine and logarithm. Thus the brevity of labels is for convenience in formulas and in keeping with mathematical customs.

Platonic labels will usually be uppercase Latin letters in math italics, but in some cases it will be expedient to use lowercase letters. As was already mentioned, lowercase letters will be used to denote coordinate systems in this chapter because they can be used well as subscripts. Furthermore, there will be certain Platonic concepts that are convenient to number; for example, if several cameras are viewing a scene it will be convenient to number the cameras and to refer to each camera by a Platonic label which is an index. Following standard mathematical usage, we will use the letters i, j, and k as indices, and thus these letters may be used as Platonic labels rather than just as integers. The following confusion can result, of which the reader should be aware: If $\mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbb{R}^3$ are n Platonic vectors and if we wish to express vector $\mathbf{p}_k$ in coordinate system i, the notation $\{\mathbf{p}_k\}_i$ is used. Here k is a number while i is a Platonic label; this notation is just like the notation $\{\mathbf{p}_k\}_w$ and does not represent, for instance, the i^{th} coordinate of vector $\mathbf{p}_k$. To reference the x-coordinate of this vector we will use the notation $\{\mathbf{p}_k\}_i^x$, and so on for other coordinates. In general, we will superscript x, y, and z to reference components of vectors; e.g., $\mathbf{p}_k^z$ is the z-coordinate of $\mathbf{p}_k$.

3.2 Pinhole cameras

We first present the abstract mathematical definition of a pinhole camera before discussing the history and physical meaning of these devices and their relationship to cameras with lenses.

3.2.1 Basic mathematical definition

A “pinhole camera” is a real device, but in this dissertation the term will usually refer to a mathematical abstraction of the device. In particular, it will refer to a function that maps $\mathbb{R}^3$ into $\mathbb{R}^2$. This section develops the abstract definition of a pinhole camera and introduces the important concepts of world coordinates, image coordinates, and camera coordinates.

DEFINITION (Pinhole camera). A *pinhole camera* is a triplet $(\mathbf{K}, \mathbf{R}, \mathbf{t})$ where $\mathbf{t} \in \mathbb{R}^3$, $\mathbf{R}$ is a 3×3 rotation matrix, and $\mathbf{K}$ is a 3×3 upper-triangular, non-singular matrix.

COMMENT (Intuitive meaning of the camera triplet). The vector $\mathbf{t}$ represents the position of the camera, the matrix $\mathbf{R}$ gives the "tilt" of the camera relative to a chosen coordinate system, and the matrix $\mathbf{K}$ describes internal characteristics of the camera.

DEFINITION (Optical center). The Platonic vector $\mathbf{t}$ labels the *optical center* of the camera.

DEFINITION (Scene). A *scene* is that part of the 3D universe that is currently under consideration.

DEFINITION (Camera view). The term *view* is a formal, generic expression for the light collected by a camera. Typically a modern camera uses photographic film or a charge-coupled device (CCD) to capture light, so a photograph or video image is a "camera view." Views will always be 2D in this dissertation, although not always flat. Some authors have considered lower-dimensional views (e.g., [46, 141]).

COMMENT (Views versus cameras). Since the same camera could capture several views of a scene from different positions, it might be logical to call the triple $(\mathbf{K}, \mathbf{R}, \mathbf{t})$ a "view" rather than a "camera," perhaps using the term "camera" only for $\mathbf{K}$. However, we will follow standard usage in this dissertation and the term "camera" will mean a particular camera at some instant in time at a particular location and orientation. The term "view" will refer only to what the camera "sees" and thus in particular means a 2D object.

DEFINITION (Camera projection function). Informally, the *projection function* of a camera is a function $\phi : \mathbb{R}^3 \to \mathbb{R}^2$ that maps 3D positions in a scene to 2D positions in the camera view. The mapping ϕ could also be called the *calibration* of the camera.

DEFINITION (World coordinates). The domain of ϕ, which is $\mathbb{R}^3$, is called *world coordinates*. The Platonic label w will be used for world coordinates. It will always be assumed that the world coordinate system is orthonormal; i.e., the basis vectors of the world coordinate system are all the same length and at right angles to each other. There are, of course, many possible orthnormal coordinate system for measuring the scene. If it is necessary to refer to an alternative world coordinate system, a second Platonic label such as v can be used.

DEFINITION (Image coordinates). The range of ϕ, which is $\mathbb{R}^2$, is called *image coordinates*. Intuitively, image coordinates represent a grid that might be superimposed on a photograph for measuring objects in the photograph. Thus it is easy to determine the image coordinates of a scene point (as opposed to denoting its 3D location), and this fact makes possible all the self calibration and scene reconstruction discussed later in this dissertation.

DEFINITION (Projection function of a pinhole camera). The basic projection function ϕ of a pinhole camera $(\mathbf{K}, \mathbf{R}, \mathbf{t})$ is defined using a 3×4 matrix $\mathbf{\Pi}$ and the algorithm of Fig. 3.2. The Platonic form of $\mathbf{\Pi}$ is

$$\mathbf{\Pi} = \mathbf{KR}\,[\,\mathbf{I}\,|\,-\mathbf{t}\,] = \mathbf{KR}\begin{bmatrix} 1 & 0 & 0 & -\mathbf{t}^x \\ 0 & 1 & 0 & -\mathbf{t}^y \\ 0 & 0 & 1 & -\mathbf{t}^z \end{bmatrix} \tag{3.1}$$

The form of $\mathbf{\Pi}$ in Fig. 3.2 is basis-specific rather than Platonic, as will be discussed presently.

DEFINITION (Camera matrix). The matrix $\mathbf{\Pi}$ is called the *camera matrix* of the camera.

COMMENT (Specifying the camera matrix). A specific camera and a specific world coordinate system are necessary to fully define the camera matrix. If i labels the pinhole camera and w labels the world coordinate system, then the camera matrix is written $\mathbf{\Pi}_{wi}$; this notation symbolizes the transformation from world coordinates w to "camera" coordinates i. With a specific world coordinate system, it becomes necessary to specify $\mathbf{R}$, $\mathbf{K}$, and $\mathbf{t}$ with respect to this coordinate system. Note that matrix $\mathbf{K}$ will only change by an overall scale factor for different orthonormal world coordinate systems. The notation $\mathbf{K}_w$ and $\mathbf{R}_w$ is used in Fig. 3.2 to make it clear that these matrices depend on the chosen world coordinate system.

DEFINITION (Internal and external calibration). The matrix $\mathbf{K}$ is called the *internal calibration* of the camera while the matrix $\mathbf{R}$ and the vector $\mathbf{t}$, taken together, are called the *external calibration*. Various alternative expressions have been used by other authors: "intrinsic" and "extrinsic" are sometimes used instead of "internal" and "external," and "orientation" is sometimes used instead of "calibration." These terms are used in any combination; for example, "intrinsic orientation" or "extrinsic calibration." The term "orientation" is often preferred in photogrammetry literature.

ALGORITHM (BASIC PINHOLE PROJECTION)

Let w denote world coordinates, let i denote the pinhole camera $(\mathbf{K}, \mathbf{R}, \mathbf{t})$, and let $\mathbf{\Pi}_{wi}$ be the camera matrix of i. If $\mathbf{q} \in \mathbb{R}^3$ denotes a scene position then the projection of $\mathbf{q}$ onto the image plane of camera i is $\phi_i(\mathbf{q})$ given by the following algorithm:

(1) Determine $x, y, z \in \mathbb{R}$ by the following:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} := \{\mathbf{q}\}_i = \mathbf{\Pi}_{wi} \begin{bmatrix} \{\mathbf{q}\}_w \\ 1 \end{bmatrix} = \mathbf{K}_w \mathbf{R}_w \left[\begin{array}{ccc|c} 1 & 0 & 0 & -\{\mathbf{t}\}_w^x \\ 0 & 1 & 0 & -\{\mathbf{t}\}_w^y \\ 0 & 0 & 1 & -\{\mathbf{t}\}_w^z \end{array}\right] \begin{bmatrix} \{\mathbf{q}\}_w^x \\ \{\mathbf{q}\}_w^y \\ \{\mathbf{q}\}_w^z \\ 1 \end{bmatrix}$$

$$= \mathbf{K}_w \mathbf{R}_w \{\mathbf{q} - \mathbf{t}\}_w$$

(2) If $z \leq 0$ then $\phi_i(\mathbf{q})$ is undefined. Otherwise, $\phi_i(\mathbf{q}) = (x/z, y/z)$.

Figure 3.2 Using a camera matrix $\mathbf{\Pi}_{wi}$ to define the projection function ϕ_i of pinhole camera i. The algorithm given here is a basic interpretation of pinhole projection.

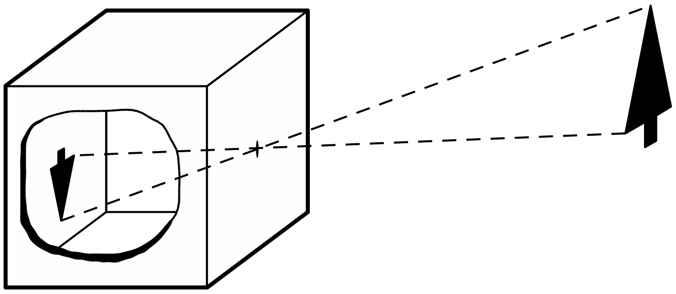

Figure 3.3 Diagram of a real pinhole camera imaging a Christmas-tree shape (i.e., an up arrow). Photographic film is on the back wall of the camera. Note that by having f (focal length) be a negative number (since the imaging plane is on the negative z-axis), the image is upside-down and mirror image. We will treat the image plane as if it is on the positive z-axis to avoid this.

DEFINITION (Camera coordinates). The world coordinate system after transformation by the affine mapping $\mathbf{\Pi}$ is called *camera coordinates*; that is, the origin of camera coordinates, expressed in world coordinates, is the point $\{\mathbf{t}\}_w$ and the basis is $(\mathbf{KR})^{-1}[1,0,0]^\top$, $(\mathbf{KR})^{-1}[0,1,0]^\top$, and $(\mathbf{KR})^{-1}[0,0,1]^\top$. A Platonic scene position $\mathbf{q}$ expressed in the coordinates of camera i is written $\{\mathbf{q}\}_i$ as in Fig. 3.2.

DEFINITION (Image plane). The *image plane* of the camera is the plane $z = 1$ in camera coordinates.

DEFINITION (Image of a point). For a position $\mathbf{q}$ in $\mathbb{R}^3$, the mapping $\phi(\mathbf{q})$ is called the *image* or *projection* of $\mathbf{q}$ onto the image plane of the camera.

3.2.2 Physical interpretation and history

A real pinhole camera is a closed, light-proof chamber with a small hole in one wall that allows light to enter (Fig. 3.3). Typically some means of recording the light is also associated with the chamber, although the device may be used simply for viewing. Recording is usually performed by photographic film or a charge-coupled device (CCD), onto which the light falls; alternatively, light can be recorded by hand (e.g., by drawing).

The small hole in the chamber is called an *iris*. Because the hole is small, the light entering the chamber becomes focused, leading, for example, to a sharp image on the photographic film. If the hole is too small, interference effects of light waves will become significant. An enjoyable, basic introduction to why light is focused by a small iris can be found in Feynman [49].

One may think it strange to consider using anything but an automatic method for recording the light entering the chamber, but the concept of pinhole cameras greatly predates photographic film and CCD's. Early pinhole cameras were used simply for viewing without recording (e.g., as a novelty) or as a mechanical aid to drawing. In this capacity, they were known as *camera obscura*. The term *camera* comes from the Latin for "chamber."

3.2.3 Physical derivation of the pinhole camera matrix

Consider the grid G depicted in Fig. 3.4(a), which serves as a 2D coordinate system. The origin of the system is marked q and the basis vectors are labeled u and v. Now consider embedding this grid in a plane M in $\mathbb{R}^3$ (Fig. 3.4(b)). Assume w is a world coordinate system in which

(1) M is the plane $z = f$ for some $0 \neq f \in \mathbb{R}$, and

(2) the u-axis of the grid G is parallel to the x-axis of world coordinates.

As a position on plane M, the point q now represents a 3D location as well as a 2D location on the plane; let $\mathbf{q} \in \mathbb{R}^3$ represent the 3D location given by q (Fig. 3.4(c)). Similarly, the 2D basis vectors u and v also represent 3D direction vectors as shown in Fig. 3.4(d) ; let $\mathbf{u}, \mathbf{v} \in \mathbb{R}^3$ denote these vectors. Note the three vectors, when represented in world coordinates, have the form of $\{\mathbf{q}\}_w = (*, *, f)$, $\{\mathbf{u}\}_w = (*, 0, 0)$, and $\{\mathbf{v}\}_w = (*, *, 0)$.

Thinking of the origin of world coordinates as being the iris of a pinhole camera, consider how light traveling in a straight line R through this iris would intersect the grid G. Let $\mathbf{r} \in \mathbb{R}^3$ indicate the direction of the ray of light in world coordinates. Since $\mathbf{q}$, $\mathbf{u}$, and $\mathbf{v}$ are linearly independent, $\mathbf{r}$ can be written as a linear combination

$$\mathbf{r} = a\mathbf{u} + b\mathbf{v} + c\mathbf{q}$$

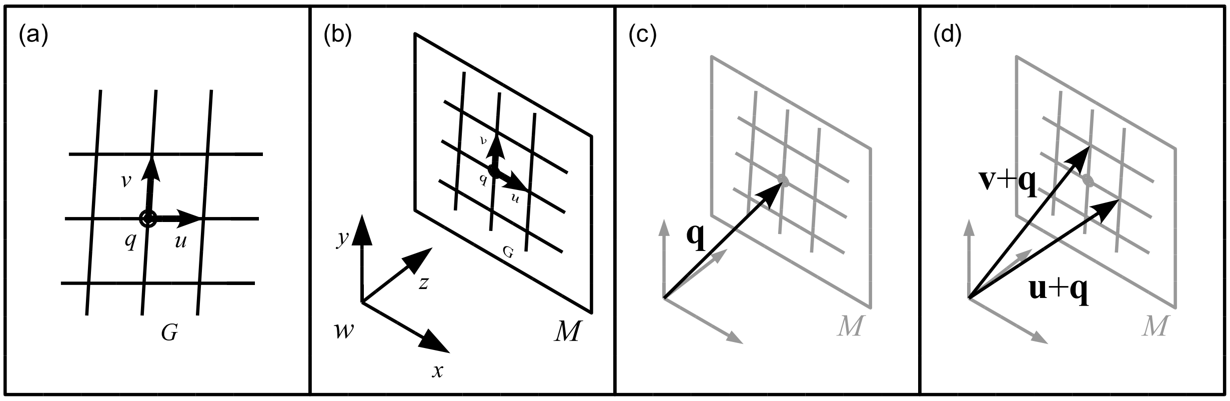

Figure 3.4 Image-plane coordinates in 3D. Note in (d) that $\mathbf{u} = (\mathbf{u} + \mathbf{q}) - \mathbf{q}$ and similarly for $\mathbf{v}$.

The line R intersects the grid G at $(a/c, b/c)$ in grid coordinates (to see this, notice that $\{\mathbf{u}\}_w^z = \{\mathbf{v}\}_w^z = 0$, so $\{\mathbf{r}\}_w^z = c\{\mathbf{q}\}_w^z$; thus $\frac{1}{c}\{\mathbf{r}\}_w$ has a z-component of f). Also observe that

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{q} \end{bmatrix}^{-1} \mathbf{r} \tag{3.2}$$

because $\mathbf{u}, \mathbf{v}, \mathbf{q} \in \mathbb{R}^3$ form a spanning basis for $\mathbb{R}^3$.

The relationship between abstract pinhole cameras (Section 3.2.1) and the discussion above about grid projection is as follows. The plane M in $\mathbb{R}^3$ which contains grid G represents the image plane of the camera (e.g., the photographic plate or CCD). We will use the notation $\mathrm{plan}(i)$ to denote the image plane M for camera i. The grid G itself represents how the image is measured or sampled; that is, grid G represents image coordinates, with q as the origin and u and v as the basis vectors. Conversion from world coordinates to camera coordinates is given by

$$\mathbf{K} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{q} \end{bmatrix}^{-1}$$

Note that $\mathbf{K}$ is an upper-triangular matrix because of the form of $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{q}$ mentioned earlier. A vector (a, b, c) in camera coordinates is mapped onto the image plane (and converted to image coordinates) by dividing by the z-component of the vector: $(a, b, c) \rightarrow (a/c, b/c)$. This is step (2) of the algorithm in Fig. 3.2. The complete pinhole camera matrix (Eq. 3.1) and projection algorithm (Fig. 3.2) together represent the following transformation steps:

(1) Translate world coordinates so that the origin is at the optical center of the camera.

(2) Rotate world coordinates so that (1) the (x, y)-plane of world coordinates is parallel to the camera's image plane (so any plane $z = f$ is parallel to the image plane) and (2) the world x-axis is parallel to the camera's u-axis.

(3) Map lines of light that travel through the world origin (i.e., the camera's optical center) to 2D positions on the camera's image plane by first multiplying by $\mathbf{K}$ to get camera coordinates than dividing by the resulting z-component to get image coordinates.

It remains to be discussed what f represents and how grid G specifically models the measurement of actual camera views. In a real pinhole camera, a scene is projected upside-down and mirror-reversed onto the image plane (Fig. 3.3). This can be represented by taking $f < 0$, which will result in x and y coordinates being multiplied by -1 during the conversion from camera coordinates to image coordinates. Alternatively, a pinhole camera can be thought of as portrayed in Fig. 1.1. Each device in this figure allows a person to view the scene through a grid, perhaps for drawing the scene with accurate perspective projection.[1] Since the grid is positioned between the viewer and the scene, this situation can be modeled by taking $f > 0$; we prefer this interpretation since the x and y coordinates will have their signs preserved.

We conclude this section with several physical analogies showing how grid G can model the measurement of positions in a real camera view. For a video or digital image captured with a CCD camera, grid G might represent the positions of sensors on the CCD. A properly-manufactured CCD camera will probably not have skew; that is, the u and v basis vectors should be very close to perpendicular in world coordinates. However, the position of sensors on a CCD may form rectangles rather than squares; i.e., the vectors u and v may have different lengths. For the pinhole-projection scenario depicted in Fig. 1.1, the grid has a very direct and literal interpretation: it is the system of string-lines stretched across the window. In the case of a real pinhole camera using photographic film, the grid represents the measurement system laid down on the photograph. For

[1] When used in this way, the device is called "Alberti's Grid" after a Renaissance artist who wrote about them.

example, measurement might be performed in pixels after the photograph has been scanned into a computer.

3.2.4 Meaning of internal calibration

We now consider the physical meaning of $\mathbf{K}$ with respect to a CCD camera. It is actually more intuitive to consider $\mathbf{K}^{-1}$ than $\mathbf{K}$ directly:

$$\mathbf{K}^{-1} = \begin{bmatrix} \mathbf{u}' & \mathbf{v}' & \mathbf{q}' \end{bmatrix} = \begin{bmatrix} \textit{horSpace} & \textit{horRowOffset} & \textit{horOrigin} \\ 0 & \textit{verRowOffset} & \textit{verOrigin} \\ 0 & 0 & f \end{bmatrix}$$

Vector $\mathbf{u}'$ has one nonzero component, *horSpace*, which is the horizontal distance between sensors as measured in units of the world coordinate system. The prefixes "hor" and "ver" mean "horizontal" and "vertical", respectively, where u and v are the horizontal and vertical directions on the image plane. The two nonzero components of $\mathbf{v}'$, *verRowOffset* and *horRowOffset*, measure how far shifted "up and to the right" each successive row of sensors is, again measured in units of the world coordinate system. The x and y components of $\mathbf{q}'$, *horOrigin* and *verOrigin*, give the position of the origin of image coordinates on the grid G, again measured in world coordinates. This is the offset, measured in world coordinates, of the image-coordinate origin from the z-axis. The z-component of $\mathbf{q}'$, denoted f, is called the *focal length* of the pinhole camera. In photography performed with a lens, the term "focal length" means the distance between the optical center (i.e., focal point) of the lens and the image plane when the camera is focused at infinity. Theoretically, pinhole cameras have infinite depth of field, so every distance (including infinite distance) is always in focus. Thus the focal length of a pinhole camera is taken to be the distance between the optical center and the image plane.

When grid G is thought of as being embedded in the image plane of the camera, which is the plane M in $\mathbb{R}^3$ as described in Section 3.2.3, then the z-axis of world coordinates intersects the image plane at a point $\mathbf{p} \in \mathbb{R}^3$ which corresponds to a 2D position p on grid G. The point p in the camera view is called the *principal point* of the camera. By Eq. 3.2, the location of p in image coordinates is given by the following multiplication after dividing by the resulting z-component:

$$\mathbf{K}\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top$$

When $\mathbf{K}$ has been normalized to make $\mathbf{K}_{(33)} = 1$ then the previous equation implies $(\mathbf{K}_{(13)}, \mathbf{K}_{(23)})$ is the position of the principal point on the image plane as measured in image coordinates. The remaining elements of $\mathbf{K}$ do not have such a simple or intuitively-meaningful interpretation, which is why $\mathbf{K}^{-1}$ was discussed instead.

3.3 Concepts from projective geometry

The goal of this section is to present a variety of concepts related to "projective geometry," which is a branch of mathematics devoted to dimensionality-reducing projective maps like ϕ. Particular emphasis is given to "homogeneous" coordinates and the duality of representations they induce. Define the following Platonic labels for mathematical objects: let i be a pinhole camera $(\mathbf{K}, \mathbf{R}, \mathbf{t})$, let $\mathbf{p}$ be a scene position, and let $q = \phi_i(p)$ be the image of $\mathbf{p}$ on the image plane of camera i. Then we have the following notation and concepts:

DEFINITION (Homogeneous coordinates for a position). The 4-vector

$$\widetilde{\mathbf{p}} \stackrel{\text{def}}{=} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} = [\mathbf{p}^x, \mathbf{p}^y, \mathbf{p}^z, 1]^\top$$

is said to represent position $\mathbf{p}$ in *homogeneous coordinates*. Any positive scalar multiple of $\widetilde{\mathbf{p}}$ is an equivalent representation of $\mathbf{p}$. This type of representation comes from projective geometry and is commonly used in machine vision and computer graphics. The vector $\widetilde{\mathbf{p}}$ measured in coordinate system v is expressed as

$$\{\widetilde{\mathbf{p}}\}_v \stackrel{\text{def}}{=} \widetilde{\{\mathbf{p}\}_v} = \begin{bmatrix} \{\mathbf{p}\}_v \\ 1 \end{bmatrix} = [\{\mathbf{p}\}_v^x, \{\mathbf{p}\}_v^y, \{\mathbf{p}\}_v^z, 1]^\top$$

Notice that homogeneous vectors can be transformed from world coordinates to camera i coordinates with a matrix multiplication:

$$\{\widetilde{\mathbf{p}}\}_i = \begin{bmatrix} \boldsymbol{\Pi}_{wi} \\ 0\ 0\ 0\ 1 \end{bmatrix} \{\widetilde{\mathbf{p}}\}_w$$

NOTATION (Equality up to a scale). The notation $\mathbf{u} \cong \mathbf{v}$ means vectors $\mathbf{u}$ and $\mathbf{v}$ are equal up to a nonzero scale factor.

COMMENT (Why the scale of a homogeneous vector does not matter). It was stated above that $\widetilde{\mathbf{p}}$ and $\lambda\widetilde{\mathbf{p}}$ for any $\lambda > 0$ are equivalent representations of position $\mathbf{p}$. There are two reasons for this equivalence. First, the original position $\mathbf{p}$ can be recovered from $\lambda\widetilde{\mathbf{p}}$ by dividing $\lambda\widetilde{\mathbf{p}}$ by its last component, which is traditionally labeled the w-coordinate (not to be confused with the use of w for world coordinates). Second, when used in the projection algorithm of Fig. 3.2 for determining $\phi_i(\mathbf{p})$, all $\lambda\widetilde{\mathbf{p}}$ will yield the same image position: Step (1) of the algorithm gives

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} := \mathbf{\Pi}_{wi}\{\lambda\widetilde{\mathbf{p}}\}_w = \lambda\mathbf{\Pi}_{wi}\begin{bmatrix} \{\mathbf{p}\}_w \\ 1 \end{bmatrix} \tag{3.3}$$

and step (2) removes the scale factor λ by dividing by the last component of the result. While not needed here, the restriction $\lambda > 0$ will be necessary later when considering the "image sphere" rather than the image plane.

COMMENT (Homogeneous representation of 2D image point). Notice that $\mathbf{\Pi}_{wi}\{\lambda\widetilde{\mathbf{p}}\}_w$ from Eq. 3.3 is a homogeneous 3-vector representing $\phi_i(\mathbf{p})$. Also notice that $\{\mathbf{p}\}_i$ is an equivalent homogeneous 3-vector representation of $\phi_i(\mathbf{p})$, as is any positive scalar multiple of $\{\mathbf{p}\}_i$. The set of positive scalar multiples of $\{\mathbf{p}\}_i$ is a ray that (1) has its vertex at the optical center of camera i and (2) pierces the point $\phi_i(\mathbf{p})$ on the image plane of the camera (when the image plane is thought of as the plane $\mathrm{plan}(i)$ in $\mathbb{R}^3$ described in Section 3.2.1). Fig. 3.5 illustrates the ray. This subject will be discussed again in relation to duality. Finally, notice that $\mathbf{\Pi}_{wi}$ maps the homogeneous 4-vector representation of $\mathbf{p}$ to the homogeneous 3-vector representation of $\phi_i(\mathbf{p})$.

DEFINITION (Direction vectors). While a vector in $\mathbb{R}^3$ can be thought of as a position in space, it can also be thought of as a *direction*. One can imagine a vector as an arrow pointing away from the origin; the direction the arrow is pointing is the direction represented by the vector. For example, the vector $(1, 0, 0)$ points in the direction of the x-axis. Directions as Platonic concepts will be

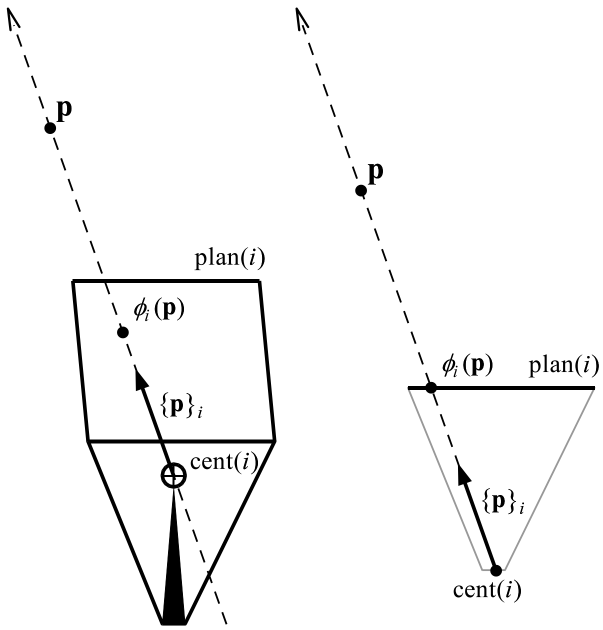

Figure 3.5 Homogeneous 2D coordinates. (*left*) Three-dimensional diagram of scene point $\mathbf{p}$ and its projection into camera i. The 3D vector $\{\mathbf{p}\}_i$ is a homogeneous representation of the 2D position $\phi_i(\mathbf{p})$. (*right*) The same diagram from overhead.

labeled with a "hat" as in $\hat{\mathbf{p}}$. It is important to realize that the "hat" is only a reminder that this vector is to be treated as a direction and not a position; the "hat" is not a function or transformation and $\hat{\mathbf{p}}$ is just a vector in $\mathbb{R}^3$.

EXAMPLE (The direction North as a Platonic direction). Suppose a magnetic compass is placed in a scene. The needle on the compass will point in the direction "North," and we can label this direction with a Platonic direction vector, for instance $\hat{\mathbf{n}}$. The vector $\hat{\mathbf{n}}$ represents where the compass needle is pointing. Under different world coordinate systems w and v the vector $\hat{\mathbf{n}}$ is likely to be represented differently; i.e., it is likely that $\{\hat{\mathbf{n}}\}_w \neq \{\hat{\mathbf{n}}\}_v$. But the meaning of $\hat{\mathbf{n}}$, as a particular direction, remains the same regardless of specific coordinate representation.

DEFINITION (Directions in homogeneous coordinates). In homogeneous coordinates, a direction is a vector of the form $(*, *, *, 0)$; that is, with a 0 instead of a 1 as the last component. Such a vector is called a *point on the plane at infinity* in projective geometry. We will use the notation

$$\widetilde{\hat{\mathbf{p}}} \stackrel{\text{def}}{=} \begin{bmatrix} \hat{\mathbf{p}} \\ 0 \end{bmatrix} = [(\hat{\mathbf{p}})^x, (\hat{\mathbf{p}})^y, (\hat{\mathbf{p}})^z, 0]^\top$$

to denote the direction represented by vector $\hat{\mathbf{p}}$ in homogeneous coordinates.

EXAMPLE (Camera coordinates revisited). Note the following:

$$\{\mathbf{p}\}_i = \mathbf{\Pi}_{wi}\{\widetilde{\mathbf{p}}\}_w \tag{3.4}$$

$$\{\hat{\mathbf{p}}\}_i = \mathbf{\Pi}_{wi}\{\widetilde{\hat{\mathbf{p}}}\}_w \tag{3.5}$$

COMMENT (Meaning of plane in plane at infinity). As always in homogeneous coordinates, any positive scalar multiple of the vector $(*, *, *, 0)$ represents the same direction. Hence, despite the presence of three "slots" in the vector that can contain arbitrary real numbers, there are only two degrees of freedom in this representation, leading the term "plane" in "plane at infinity."

COMMENT (Representing 3D planes using homogeneous coordinates). A more formal explanation for the term "plane" in "plane at infinity" comes from the following method for representing planes in $\mathbb{R}^3$ using vectors in $\mathbb{R}^4$. Let H be the plane in $\mathbb{R}^3$ that contains position $\mathbf{q} \in \mathbb{R}^3$ and has normal $\mathbf{n} \in \mathbb{R}^3$. Then a point $\mathbf{p}$ is on the plane H if and only if

$$\mathbf{n}^\top(\mathbf{p} - \mathbf{q}) = 0$$

Thus the 4-vector $\widetilde{\mathbf{m}} = [\mathbf{n}^\top, -\mathbf{n}^\top\mathbf{q}]^\top$ can be thought of as a normal to the plane H expressed in homogeneous coordinates:

$$\widetilde{\mathbf{m}}^\top\widetilde{\mathbf{p}} = [\mathbf{n}^\top, -\mathbf{n}^\top\mathbf{q}]\begin{bmatrix}\mathbf{p}\\ 1\end{bmatrix} = 0$$

The "plane at infinity" is the plane defined by the 4-vector $(0, 0, 0, 1)$, which happens to be the homogeneous representation of the origin.

COMMENT (Meaning of infinity in plane at infinity). Consider multiplying a point $(*, *, *, 0)$ on the plane at infinity by a camera matrix $\mathbf{\Pi}_{wi}$. The last column of the 3×4 camera matrix, which represents the camera position, is irrelevant. Thus if two cameras are viewing the same point on the plane at infinity, both cameras are essentially at the same location. This effect also happens with points that are very far away from the cameras relative to the distance between the cameras: the homogeneous vector $(\lambda a, \lambda b, \lambda c, 1)$ is the same as $(a, b, c, \frac{1}{\lambda})$, which becomes $(a, b, c, 0)$ as λ grows arbitrarily large. Hence the terminology "at infinity" in "plane at infinity."

DEFINITION (H-infinity matrix). Define the following Platonic matrix for camera i :

$$\mathrm{hmat}(i) \stackrel{\text{def}}{=} \mathbf{KR}$$

In relation to a specific world coordinate system, we will use the notation

$$\mathbf{H}_{wi}^{\infty} \stackrel{\text{def}}{=} \mathbf{K}_w\mathbf{R}_w$$

The matrix $\mathbf{H}_{wi}^{\infty}$ represents a transformation of directions from world coordinates to camera i coordinates. For example, if $\hat{\mathbf{u}} \in \mathbb{R}^3$ is a Platonic direction then

$$\{\hat{\mathbf{u}}\}_i = \mathbf{\Pi}_{wi}\widetilde{\hat{\mathbf{u}}} = \mathbf{\Pi}_{wi}\begin{bmatrix} \hat{\mathbf{u}} \\ 0 \end{bmatrix} = \mathbf{H}_{wi}^{\infty}\{\hat{\mathbf{u}}\}_w$$

In projective geometry terms, $\mathbf{H}_{wi}^{\infty}$ represents a planar homography induced by the plane at infinity, which explains the notation.

COMMENT (The duality of lines and planes). Let L be a line on the image plane of camera i and let H be the plane in $\mathbb{R}^3$ defined by L and the optical center of camera i. Note the 1-1 correspondence between lines on the image plane and planes through the optical center, such as exists with L and H; we refer to this as the duality of lines and planes. Since we already know a point on the plane H, namely the optical center of camera i, we can completely specify H with a vector $\mathbf{h} \in \mathbb{R}^3$ that is normal to the plane. Because of the duality of lines and planes, the vector $\mathbf{h}$ also specifies the line L. Note that any nonzero scalar multiple of $\mathbf{h}$ represents the same plane and line. Also note that $\mathbf{h}$ is a homogeneous representation of the line L exactly like the homogeneous representation of planes discussed earlier (see the comment "Representing 3D planes using homogeneous coordinates").

COMMENT (The duality of points and lines). Let q be a point on the image plane of camera i and let G be the line in $\mathbb{R}^3$ defined by q and the optical center of camera i. Note the 1-1 correspondence between points on the image plane and lines through the optical center, such as exists with q and G; we refer to this as the duality of points and lines. Since we already know a point on the line G, namely the optical center of camera i, we can completely specify G with a vector $\mathbf{g} \in \mathbb{R}^3$ that is parallel to the line. Because of the duality of points and lines, the vector $\mathbf{g}$ also specifies the point q.

Note that any nonzero scalar multiple of $\mathbf{g}$ represents the same point and line; $\mathbf{g}$ is a homogeneous representation of the point q. We will use the notation $\phi_i(\mathbf{p})$ and $\{\mathbf{p}\}_i$ interchangeably with the understanding that the latter is a homogeneous representation of the former.

COMMENT (Intersection of two lines). One advantage of representing a line on the image plane as a 3D vector is given in the following: Let U and V be two lines on a camera's image plane that intersect in a single point Q. Let $\mathbf{u}, \mathbf{v} \in \mathbb{R}^3$ define the two lines and let $\mathbf{q} \in \mathbb{R}^3$ define the point of intersection of the lines. Then

$$\mathbf{q} \cong \mathbf{u} \times \mathbf{v} \tag{3.6}$$

To derive this equation, equate the camera's optical center with the origin and consider $\mathbf{u}$ and $\mathbf{v}$ as normals to two planes through the origin. The two planes intersect in a line, which will also intersect the origin. The cross product of $\mathbf{u}$ and $\mathbf{v}$ is a nonzero vector contained within both planes, and hence within their intersection which is the line. The line intersects the image plane at a point that is contained within both lines U and V; i.e., the point Q. Hence the line is defined by $\mathbf{q}$.

Note that U and $\mathbf{u}$ are Platonic labels for the same object (a line on the image plane), but the notation $\mathbf{u}$ better emphasizes the vector representation of the object. In the future we will rarely use an additional Platonic label like "U" when using Platonic vectors like "$\mathbf{u}$"; the dual labeling was used here for emphasis.

COMMENT (Two points define a line). Let $\mathbf{p}$ and $\mathbf{q}$ be two distinct points on a camera's image plane and let $\mathbf{u}$ be the line (on the image plane) through the points. Then

$$\mathbf{u} \cong \mathbf{p} \times \mathbf{q} \tag{3.7}$$

To derive this equation, equate the camera's optical center with the origin and consider the plane through the origin that contains both $\mathbf{p}$ and $\mathbf{q}$. This plane also defines the line U; hence $\mathbf{u}$ is perpendicular to both $\mathbf{p}$ and $\mathbf{q}$.

COMMENT (Point on line test). Let $\mathbf{p}$ and $\mathbf{u}$ be a point and a line, respectively, on a camera's image plane. Then the point $\mathbf{p}$ is contained in the line $\mathbf{u}$ if and only if

$$\mathbf{p}^\top \mathbf{u} = 0 \tag{3.8}$$

To derive this equation, note that $\mathbf{u}$ is perpendicular to every vector in the plane (through the origin) defined by $\mathbf{u}$, and $\mathbf{p}$ is one such vector.

COMMENT (Care in specifying coordinate system). Equations like Eq. 3.7 and Eq. 3.8 hold for any coordinate system as long as all vectors are represented in the same coordinate system. The equations themselves are Platonic concepts in this sense.

COMMENT (Changing coordinate system for planes). Recall that the matrix $\mathbf{H}_{ij}^{\infty}$ represents a change of coordinate system (i.e., change of basis) for direction vectors. We now discuss how the transpose of the matrix $\mathbf{H}_{ij}^{\infty}$ can be used to change the coordinate system of a plane. Let $\mathbf{u}$ and the origin of camera i define a plane through $\mathrm{cent}(i)$. Note that $\mathbf{u}$ and the origin of camera j define a plane through $\mathrm{cent}(j)$ that is parallel to the first plane. The first plane can be thought of as the line $\{\mathbf{u}\}_i$ in view i; the second plane is the line $\{\mathbf{u}\}_j$ in view j. We have the following:

$$\{\mathbf{u}\}_i \cong (\mathbf{H}_{ij}^{\infty})^\top \{\mathbf{u}\}_j \tag{3.9}$$

This relation can be proven by applying Eq. 3.7 to any two distinct direction vectors that are visible on the line $\{\mathbf{u}\}_i$ in view i (and thus also on the line $\{\mathbf{u}\}_j$ in view j). Just as the direction vectors are points on the plane at infinity, the line they define is called a *line on the plane at infinity*; it is the "vanishing line" of a set of parallel planes in $\mathbb{R}^3$, like the two planes through $\mathrm{cent}(i)$ and $\mathrm{cent}(j)$ described above.

DEFINITION (Field-of-view). The *field-of-view* (fov) of a camera refers to what a camera can "see." A point in a scene is said to be in a camera's field-of-view if it will be projected into the camera's view. Field-of-view can also mean the specific angle measure of what a camera can view (e.g., the vertex angle of a cone, with vertex at the optical center, in which every scene point is visible), or can be a rough expression of the breadth of what a camera can see (e.g., "wide" or "narrow" field-of-view).

COMMENT (Image sphere). The concept of the "image plane" is meant to model the flat film or CCD or viewing plane that exists in a real camera. However, using a "viewing sphere" is more natural mathematically, and makes it possible to model cameras with a field-of-view larger than $180°$.

Recall that the image plane is the plane $z = 1$ in camera coordinates. The *image sphere* is the sphere centered at the origin with radius 1 in camera coordinates. In step (2) of the basic projection algorithm in Fig. 3.2, a ray of light (given in camera coordinates) is projected onto the image plane by dividing it by its z-coordinate, thus finding the point on the ray with z-coordinate equal to 1. Rays with a non-positive z-coordinate are considered to be from "behind" the image plane and ignored.

With an image sphere, no ray is ignored. Rays are projected onto the image sphere by scaling them to have length 1; this replaces step (2) of the basic projection algorithm. Thus an image sphere can be used to model cameras that take in light from all directions (e.g., omnicameras [123]). Furthermore, with image spheres the projection process does not grossly distort the original scene in regions that project onto the image plane far from the principal point. Finally, no two distinct lines on the image sphere are ever parallel — there are always two intersection points, both given by the expression in Eq. 3.6.

We note in passing that any 2D surface can be used for abstracting the light-collection process. For example, cylinders have been used to advantage by several authors [147, 131, 111].

3.4 Epipolar geometry

By observing point or line correspondences between two pinhole camera views, it is possible to determine the "epipolar geometry" between the views. Once determined, the epipolar geometry places constraints on the determination of future point correspondences (i.e., it makes it easier to find other point correspondences). As will be seen later in this dissertation, epipolar geometry can also be used to determine the internal calibration matrix $\mathbf{K}$ of a camera, in a process known as "self calibration." Knowledge of epipolar geometry has several other uses, such as enabling projective scene reconstruction.

In what follows, let i and j denote pinhole cameras $(\mathbf{K}, \mathbf{R}, \mathbf{t})$ and $(\mathbf{K}', \mathbf{R}', \mathbf{t}')$, respectively, with different optical centers.

DEFINITION (Epipoles). The *epipole* of view i (with respect to the pairing of views i and j) is the location of the optical center of camera j as seen in view i. Formally, let $\hat{\mathbf{e}} = \mathrm{cent}(j) -$

$\mathrm{cent}(i) = \mathbf{t}' - \mathbf{t}$ be a Platonic direction vector. Then any non-zero scalar multiple of the vector $\{\hat{\mathbf{e}}\}_i = \mathbf{\Pi}_{wi}\{\widetilde{\hat{\mathbf{e}}}\}_w = \{\mathrm{cent}(j)\}_i = \{\mathbf{t}'\}_i$ is called the epipole of view i. The 2D position $\phi_i(\mathbf{t}')$ in view i is also called the epipole of view i. Similarly, any non-zero scalar multiple of $\{\hat{\mathbf{e}}\}_j = \{\mathrm{cent}(i)\}_j = \{\mathbf{t}\}_j$ is the epipole of view j.

DEFINITION (Homography induced by the plane at infinity). Any plane viewed by cameras i and j induces a planar homography between the views. The "plane" at infinity also induces a planar homography, which is written $\mathbf{H}^{\infty}_{ij}$ and given by the following

$$\mathbf{H}^{\infty}_{ij} \stackrel{\text{def}}{=} \mathbf{K}'\mathbf{R}'\mathbf{R}^{-1}\mathbf{K}^{-1} = \mathrm{hmat}(j)\mathrm{hmat}(i)^{-1}$$

Note this is the homography from view i to view j; the reverse is written $\mathbf{H}^{\infty}_{ji}$. As with the homography $\mathbf{H}^{\infty}_{wi}$ discussed earlier, $\mathbf{H}^{\infty}_{ij}$ transforms direction vectors from one coordinate system to another:

$$\{\hat{\mathbf{p}}\}_j = \mathbf{H}^{\infty}_{ij}\{\hat{\mathbf{p}}\}_i$$

COMMENT (World camera). The world coordinate system can be thought of as the coordinate system of an abstract pinhole camera labeled w given by the triplet $(\mathbf{I}, \mathbf{I}, \mathbf{0})$. Then, without being careful about specifying basis,

$$\mathbf{H}^{\infty}_{wi} = \mathbf{K}\mathbf{R}\mathbf{I}^{-1}\mathbf{I}^{-1} = \mathbf{K}\mathbf{R} = \mathrm{hmat}(i)$$

This is the intuitive meaning of $\mathrm{hmat}(i)$, as a planar homography transforming direction vectors from world coordinates to camera i's coordinates. Furthermore,

$$\mathbf{H}^{\infty}_{ij} = \mathbf{H}^{\infty}_{wj}\mathbf{H}^{\infty}_{iw}$$

NOTATION (Cross-product matrix). The cross product of two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^3$ can be computed with the matrix-vector multiplication

$$\mathbf{u} \times \mathbf{v} = [\mathbf{u}]_{\times}\mathbf{v}$$

where, for $\mathbf{u} = (x, y, z)$, we define the *cross-product matrix*

$$[\mathbf{u}]_\times \stackrel{\text{def}}{=} \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

DEFINITION (Fundamental matrix). The *fundamental matrix* $\mathbf{F}_{ij}$ between camera view i and camera view j is [72]

$$\mathbf{F}_{ij} \stackrel{\text{def}}{\cong} [\{\hat{\mathbf{e}}\}_j]_\times \mathbf{H}^\infty_{ij} \tag{3.10}$$

This definition is only up to a nonzero scale factor. A fundamental matrix has rank 2 because $[\{\hat{\mathbf{e}}\}_j]_\times$ has rank 2 and $\mathbf{H}^\infty_{ij}$ is invertible.

COMMENT (Physical interpretation of the fundamental matrix). Let $\hat{\mathbf{q}}$ be a Platonic direction and let $\mathbf{t}$ be the optical center of camera i. The set $\{\mathbf{t} + \lambda\hat{\mathbf{q}} : \lambda \in \mathbb{R}\}$ is a line parallel to $\hat{\mathbf{q}}$ and containing $\mathbf{t}$; call this line Q. Assume $\hat{\mathbf{q}}$ is such that line Q does not contain $\mathbf{t}'$, the optical center of camera j; i.e., assume $\hat{\mathbf{q}}$ and $\hat{\mathbf{e}}$ are not parallel. For any $\mathbf{p} = \mathbf{t} + \lambda\hat{\mathbf{q}}$ we have

$$\{\mathbf{p}\}_i = \mathbf{\Pi}_{wi}\{\widetilde{\mathbf{p}}\}_w = \mathbf{\Pi}_{wi}(\{\widetilde{\mathbf{t}}\}_w + \lambda\{\widetilde{\hat{\mathbf{q}}}\}_w) = \lambda\mathbf{H}^\infty_{wi}\{\hat{\mathbf{q}}\}_w = \lambda\{\hat{\mathbf{q}}\}_i \tag{3.11}$$

$$\{\mathbf{p}\}_j = \mathbf{\Pi}_{wj}\{\widetilde{\mathbf{p}}\}_w = \mathbf{\Pi}_{wj}(\{\widetilde{\mathbf{t}}\}_w + \lambda\{\widetilde{\hat{\mathbf{q}}}\}_w) = \{\hat{\mathbf{e}}\}_j + \lambda\mathbf{H}^\infty_{wj}\{\hat{\mathbf{q}}\}_w = \{\hat{\mathbf{e}}\}_j + \lambda\{\hat{\mathbf{q}}\}_j \tag{3.12}$$

Thus

$$\{\mathbf{p}\}_j^\top \mathbf{F}_{ij}\{\mathbf{p}\}_i = (\{\hat{\mathbf{e}}\}_j + \lambda\{\hat{\mathbf{q}}\}_j)^\top [\{\hat{\mathbf{e}}\}_j]_\times \mathbf{H}^\infty_{ij}\,(\lambda\{\hat{\mathbf{q}}\}_i) \tag{3.13}$$

$$= \lambda^2\{\hat{\mathbf{q}}\}_j^\top [\{\hat{\mathbf{e}}\}_j]_\times \{\hat{\mathbf{q}}\}_j \tag{3.14}$$

$$= 0 \tag{3.15}$$

The equations above tell us that $\mathbf{F}_{ij}\{\mathbf{p}\}_i$ is a vector that is simultaneously perpendicular (in camera j coordinates) to both $\{\hat{\mathbf{e}}\}_j$ and $\{\mathbf{p}\}_j$. Therefore $\mathbf{F}_{ij}\{\mathbf{p}\}_i$ defines the line through $\{\hat{\mathbf{e}}\}_j$ and $\{\mathbf{p}\}_j$ on the image plane of camera j. Note that this line is the image of line Q as seen in view j. Furthermore, since λ was arbitrary, $\mathbf{F}_{ij}\{\mathbf{p}\}_i$ gives the image of Q using any point $\mathbf{p}$ on Q (except

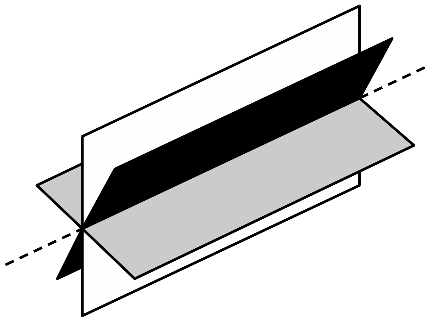

Figure 3.6 A pencil of planes.

for $\mathbf{t}$). As λ changes, the position $\{\mathbf{p}\}_j$ of $\mathbf{p}$ seen in view j will change but the position $\{\mathbf{p}\}_i$ of $\mathbf{p}$ seen in view i remains the same.

DEFINITION (Epipolar planes and epipolar lines). The vectors $\{\mathbf{p}\}_i \times \{\hat{\mathbf{e}}\}_i$ and $\{\mathbf{p}\}_j \times \{\hat{\mathbf{e}}\}_j$, together with the optical centers $\mathbf{t}$ and $\mathbf{t}'$, respectively, define planes in space called *epipolar planes*. These planes intersect the image planes of camera i and camera j at lines that are called *epipolar lines*. The first line is called the epipolar line of $\{\mathbf{p}\}_i$ and it is given, in the homogeneous sense, by the vector $\{\mathbf{p}\}_i \times \{\hat{\mathbf{e}}\}_i$ (see comments on the duality of lines and planes); similarly for $\{\mathbf{p}\}_j$ in view j. Note that the vectors $\{\mathbf{p}\}_i \times \{\hat{\mathbf{e}}\}_i$ and $\{\mathbf{p}\}_j \times \{\hat{\mathbf{e}}\}_j$ define the *same* plane in space, which is the plane containing both optical centers and the point $\mathbf{p}$.

The set of all possible epipolar planes for a pair of cameras is the set of all planes that contain both optical centers (and thus the line through the optical centers). The set of all planes that contain a particular line is called a *pencil* of planes in projective geometry (Fig. 3.6).

COMMENT (Why a fundamental matrix is not invertible). The physical interpretation of the fundamental matrix explains why its rank is 2 (and why the fundamental matrix is not invertible): For every point $\mathbf{p}$ on a particular epipolar plane, the fundamental matrix maps $\{\mathbf{p}\}_i$ to the same vector, up to a scale factor, in camera j's coordinates. That is, all the points in a plane get mapped to one line. The kernel of this mapping must have dimensionality 1; it is given by the line through the two optical centers.

DEFINITION (Point correspondence). If a scene position $\mathbf{p} \in \mathbb{R}^3$ is visible in both camera views, then $\phi_i(\mathbf{p})$ and $\phi_j(\mathbf{p})$ are said to be *corresponding points* or *conjugate points* or a *point correspondence* between views i and j. Using the point-line duality of homogeneous notation, we can also say $\{\mathbf{p}\}_i$ and $\{\mathbf{p}\}_j$ are point correspondences. See Section 3.6 discussing how point correspondences are determined in practice.

COMMENT (Determining the fundamental matrix from point correspondences). Let $\mathbf{p}_1, \ldots, \mathbf{p}_n \in \mathbb{R}^3$ be n scene points that are visible in both camera views. Then by Eq. 3.13,

$$\{\mathbf{p}_k\}_j^\top \mathbf{F}_{ij} \{\mathbf{p}_k\}_i = 0 \tag{3.16}$$

for $1 \leq k \leq n$. Each such equation places a linear constraint on the components of $\mathbf{F}_{ij}$. Naively, since $\mathbf{F}_{ij}$ is a 3×3 matrix, at most 9 constraints are needed to uniquely determine $\mathbf{F}_{ij}$. This is provided that $\mathbf{F}_{ij}$ exists and that only one matrix can satisfy 9 linear constraints of the form Eq. 3.16 simultaneously. By Eq. 3.10, the matrix $\mathbf{F}_{ij}$ always exists as long as the cameras have different optical centers. However, any scalar multiple of $\mathbf{F}_{ij}$ will also satisfy the constraints given by Eq. 3.16. This means that $\mathbf{F}_{ij}$ can only be determined up to a scale factor using these constraints, that $\mathbf{F}_{ij}$ has at most 8 degrees of freedom, and that at most 8 linear equations are necessary to determine it (and thus at most 8 point correspondences). It is because of these considerations that $\mathbf{F}_{ij}$ was only defined up to a nonzero scale factor in Eq. 3.10.

However, it is still possible that the set of matrices satisfying the constraints given by Eq. 3.16 is multidimensional; i.e., that $\mathbf{F}_{ij}$ and some non-multiple of $\mathbf{F}_{ij}$ both satisfy the system, and thus so does any linear combination of these two matrices. In fact, from certain sets of scene points $\mathbf{p}_k$ (e.g., all $\mathbf{p}_k$ lying on the same plane) $\mathbf{F}_{ij}$ cannot be uniquely identified. The conditions on the set of $\mathbf{p}_k$ under which $\mathbf{F}_{ij}$ can and cannot be determined uniquely (up to a scale factor) have been given by other authors (e.g., [99, 70]). In general, a randomly-chosen set of 8 scene points $\mathbf{p}_k$ will allow $\mathbf{F}_{ij}$ to be determined uniquely (up to a scale factor) by the linear system arising from the constraints in Eq. 3.16. In practice, the linear system must be properly normalized to handle noise in the point-correspondence data (see [73]).

The property that $\mathbf{F}_{ij}$ has rank 2 is another constraint not incorporated into the linear constraints of Eq. 3.16. By utilizing this extra constraint, the number of points correspondences needed to find $\mathbf{F}_{ij}$ is reduced to 7; see Hartley and Zisserman [70]. It has been shown [76, 140] that 6 is the smallest number of correspondences from which $\mathbf{F}_{ij}$ can be determined; however, 3 views are required rather than 2. The smallest number of line correspondences is 9 [127].

Because of the importance of fundamental matrices to this dissertation, methods for calculating $\mathbf{F}_{ij}$ are discussed in greater detail in Section 3.7.

COMMENT (Transpose of a fundamental matrix). By switching the labels of cameras i and j, we get the fundamental matrix

$$\mathbf{F}_{j\,i} \cong [\{\hat{\mathrm{e}}\}_i]_\times \mathbf{H}^\infty_{ji} \tag{3.17}$$

It is not obviously true, but $\mathbf{F}_{j\,i}$ is the transpose of $\mathbf{F}_{ij}$:

$$\mathbf{F}_{j\,i} = (\mathbf{F}_{ij})^\top \tag{3.18}$$

One way to prove Eq. 3.18 is to use the physical interpretation of the fundamental matrix as a transformation between positions measured in one coordinate system to corresponding epipolar planes measured in the other. Such a proof requires the fact that $(\mathbf{H}^\infty_{ij})^\top$ is a change of basis for planes.

The simplest way to prove Eq. 3.18 is to consider Eq. 3.16 for $\mathbf{F}_{j\,i}$ versus the transpose of Eq. 3.16 for $\mathbf{F}_{ij}$,

$$\{\mathbf{p}_k\}_i^\top \mathbf{F}_{j\,i} \{\mathbf{p}_k\}_j = 0 = (\{\mathbf{p}_k\}_j^\top \mathbf{F}_{ij} \{\mathbf{p}_k\}_i)^\top = \{\mathbf{p}_k\}_i^\top \mathbf{F}_{ij}^\top \{\mathbf{p}_k\}_j$$

and use the fact that fundamental matrices can be uniquely determined by using enough constraints of the form in Eq. 3.16.

COMMENT (Extracting epipoles from a fundamental matrix). It is immediately obvious from Eq. 3.10 that $\{\hat{\mathrm{e}}\}_j$ is a left null eigenvector of $\mathbf{F}_{ij}$; that is

$$\{\hat{\mathrm{e}}\}_j^\top \mathbf{F}_{ij} = 0$$

Since $\mathbf{F}_{ij}$ has rank 2, $\{\hat{\mathrm{e}}\}_j$ is the unique left null eigenvector. Furthermore, using $\mathbf{F}_{ij} = \mathbf{F}_{j\,i}^{\top}$ and Eq. 3.17 we see that $\{\hat{\mathrm{e}}\}_i$ is the unique right null eigenvector of $\mathbf{F}_{ij}$:

$$\mathbf{F}_{ij}\{\hat{\mathrm{e}}\}_i = 0$$

Null eigenvectors can be reliably extracted from a matrix using singular-value decomposition (see any standard reference on numerical linear algebra, such as Golub and Van Loan [59]; an excellent implementation is given in Golub and Reinsch [60]). Thus a fundamental matrix can be determined directly from point correspondences between two views, and then epipoles can be extracted from the fundamental matrix. The eigenvectors $\{\hat{\mathrm{e}}\}_i$ and $\{\hat{\mathrm{e}}\}_j$ will often be referred to as the *right epipole* and *left epipole* of $\mathbf{F}_{ij}$, respectively.

COMMENT (Equivalence of epipolar geometry and the fundamental matrix). Knowledge of the *epipolar geometry* between two camera views means knowledge of the location of both epipoles and of the correspondence between epipolar lines (i.e., which pairs of lines in the two views are induced by the same epipolar plane). This is exactly the information provided by the fundamental matrix.

3.5 Scene reconstruction

Scene reconstruction is the process of building a 3D model of a scene using 2D camera views. This process is also called "image-based modeling." While the focus of this dissertation is on camera self calibration, an important side effect of self calibration is the ability to reconstruct scenes. Note that scene reconstruction includes recovering the location, orientation, and characteristics of the cameras viewing the scene; in fact, under some circumstances this is all that scene reconstruction means. Also note that scene reconstruction has a broad meaning that includes a wide variety of unintuitive reconstructions, that is, reconstructions in which measurements and angles are not like what we experience in our usual interactions with a scene. We begin by discussing this "hierarchy" of scene reconstructions.

3.5.1 Hierarchy of scene reconstructions

The hierarchy of scene reconstructions (e.g., [45, 33, 133]) has three levels; listed from most general to most specific they are "projective," "affine," and "metric." The term "Euclidean" is sometimes used instead of "metric" (e.g., [35]). The best way to understand this hierarchy is in reverse, by considering ways of distorting the original scene. Let W be a set of position vectors in homogeneous notation representing the original scene. Assume the vectors of W have been measured with an orthonormal coordinate system so that they meet our intuitive expectations about scene shape. For a concrete example, if a square was in the original scene then W might contain 4 vectors representing the vertices of the square, and these vertices might have the form $(0, 0, 0, 1)$, $(1, 0, 0, 1)$, $(1, 1, 0, 1)$, and $(0, 1, 0, 1)$. Now consider transforming W using an arbitrary, invertible 4×4 matrix $\mathbf{\Psi}$ to get W':

$$W' = \mathbf{\Psi} W$$

The set of vectors W' is called a *projective reconstruction* of the scene W. This expression comes from the term *projective transformation* for matrices like $\mathbf{\Psi}$. Some versions of what W' might look like for the square example are shown in Fig. 3.7. In particular, note in the figure that the plane at infinity can be mapped to a real plane in $\mathbb{R}^3$ (meaning points of the form $(*, *, *, 0)$ can be mapped to a set of coplanar points of the form $(*, *, *, 1)$); this demonstrates how unintuitive projective scene reconstructions can be.

Now consider restricting $\mathbf{\Psi}$ so that the bottom row of the matrix is $[0\ \ 0\ \ 0\ \ 1]$ up to an arbitrary non-zero scale factor. The transformation represented by such a matrix is called an *affine transformation*, and the matrix itself is called an *affine matrix*. Intuitively, an affine transformation of $\mathbb{R}^3$ is a linear transformation plus a translation; this is, in fact, the usual mathematical definition.[2] When $\mathbf{\Psi}$ is an affine transformation, the vectors in W' form what is called an *affine reconstruction* of W. See Fig. 3.7 for an example of an affine reconstruction of a square. Affine transformations have the important property that they fix the plane at infinity, meaning vectors get mapped to the plane

[2] We are able to represent such a transformation with a single 4×4 matrix through the use of homogeneous coordinates; normally, one thinks of matrices as only representing linear transformations (without added translations).

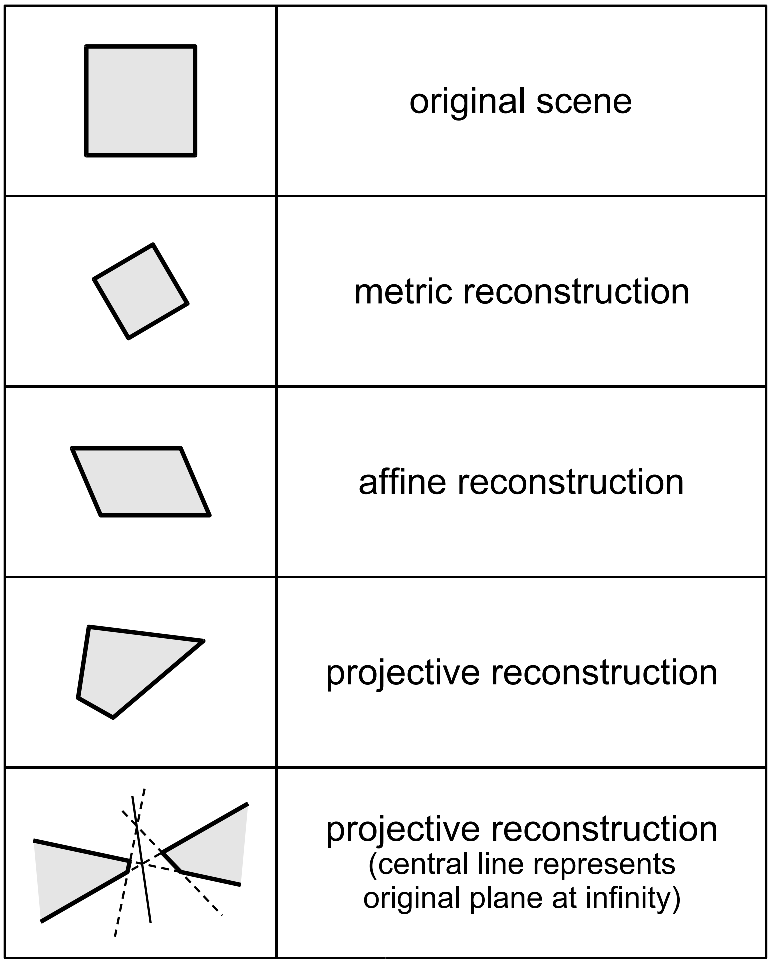

Figure 3.7 Hierarchy of scene reconstructions. In this example, the original scene is a square.

at infinity if and only if they are already on the plane at infinity. This property can also be taken as the definition of affine transformations.

Restricting $\mathbf{\Psi}$ even further, let $\mathbf{\Psi}$ be an affine transformation with a scaled rotation matrix in the upper left 3×3 block. Such a transformation is called a *Euclidean transformation*, and the reconstruction W' resulting from this transformation is called either a *Euclidean* or a *metric reconstruction* of W; in this dissertation, we use the term "metric reconstruction." Fig. 3.7 gives a sample metric reconstruction of a square.

Note that all Euclidean transformations are affine and all affine transformations are projective. The reverse is not true, making this is a proper hierarchy. The hierarchy of transformations defines the hierarchy of scene reconstructions; see Fig. 3.7 for an intuitive look at the hierarchy and Fig. 3.8 for a Venn-diagram representation of the hierarchy. As will be seen, projective scene reconstructions are the easiest to create from camera views but are the least intuitive and thus the least useful. Affine reconstructions are the next easiest to create, but are still not very intuitive or useful. Metric reconstructions are the hardest to create but are very useful; they represent the original scene up to an arbitrary rotation and overall scaling and thus meet our intuitive expectations about what a scene should look like. We have "intuitive" expectations about this kind of reconstruction because in our everyday interactions with the world we deal with rigid objects that can be picked up, rotated, and translated, and every time we move an object in this manner (i.e., by a Euclidean displacement) the object becomes a metric reconstruction of its "true" form. As will be seen, metric reconstruction is the best that can be achieved through self calibration; to reconstruct a scene at its original scale requires some knowledge of specific measurements in the scene (e.g., that the side of the square is 1 meter in length).

3.5.2 Hierarchy of camera reconstructions

In the preceding section, a set W of measurements of scene positions, acquired using an orthnormal world coordinate system, was transformed by various 4×4 matrices to create alternative representations of the scene called "reconstructions." In a similar way, the cameras viewing the scene can be measured using an orthonormal world coordinate system and the resulting camera

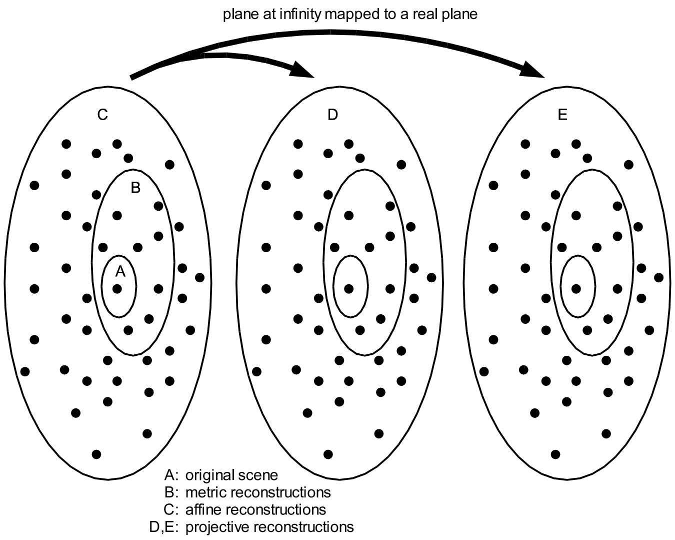

Figure 3.8 Hierarchy of scene reconstructions portrayed with a Venn diagram.

matrices can be transformed by various 4×4 matrices to create a hierarchy of "reconstructions" of the cameras.

To express this formally, let n cameras capture views of a scene and let $\mathbf{\Pi}_k$ for $1 \leq k \leq n$ be the camera matrices for these views. If W is transformed by $\mathbf{\Psi}$ then each camera matrix needs to be pre-multiplied by $\mathbf{\Psi}^{-1}$ to ensure the camera views will remain the same:

$$(\mathbf{\Pi}_k \mathbf{\Psi}^{-1})(\mathbf{\Psi} W) = \mathbf{\Pi}_k W$$

The set of cameras $\mathbf{\Pi}_k \mathbf{\Psi}^{-1}$ is called a *reconstruction* of the original cameras. Note that the inverse of a projective, affine, or Euclidean transformation is, respectively, a projective, affine, or Euclidean transformation; i.e., the class of a transformation is unchanged by inversion. Thus we can classify a camera reconstruction by the class of the transformation $\mathbf{\Psi}$ used to create it, just as with scene reconstructions.

As will be seen, the significance of this stratification comes from the fact that a projective reconstruction of the cameras viewing a scene can be created using the fundamental matrices between camera views. Since fundamental matrices can be calculated directly from point correspondences, this means projective camera reconstructions can be created directly from information available in the 2D camera views. Furthermore, it will be shown that projective camera reconstructions can be upgraded to affine and metric reconstructions under certain circumstances with no more information than is available in the 2D camera views. This is the process of "self calibration," which is the central topic of this dissertation.

3.5.3 Triangulation

Triangulation in its simplest form is illustrated in Fig. 1.1: the direction towards a target position $\mathbf{p}$ in space is determined from two different locations, and this information induces two 3D lines that, when intersected, yield the 3D coordinates of $\mathbf{p}$. Camera views can provide the necessary triangulation information provided the following is known: (1) the 2D projection $\phi_i(\mathbf{p})$ of $\mathbf{p}$ into each camera view, (2) the inverse projection function ϕ_i^{-1} of each view, and (3) the location of the optical center of each camera. Note that ϕ_i^{-1} is a function mapping 2D points to 3D rays, and that (1) and (2) combine to provide the direction $\phi_i^{-1}(\phi_i(\mathbf{p}))$ from the optical center of camera i towards the target.

Thus the process of triangulation can be used reconstruct a scene from point correspondences, but only after the set of camera views has been calibrated. Finding the calibration functions of a set of camera views for a shared coordinate system means reconstructing the camera views. If only a projective reconstruction of the camera views can be determined, then only a projective reconstruction of the scene can be created (see Section 3.5.4), and so on for the other levels of reconstruction.

More elaborate forms of triangulation are available. The most important is the simultaneous use of more than two camera views to triangulate a target position. This creates an over-constrained problem that greatly stabilizes the triangulation process relative to noise in the 2D positions $\phi_i(\mathbf{p})$. It is only this stabilization process that makes scene reconstruction possible from real data, because even slight errors in the 2D positions $\phi_i(\mathbf{p})$ and in the inverse projection functions ϕ_i^{-1} can lead to large errors in the triangulated 3D position otherwise. See Section 11.2 of Hartley and Zisserman [70] for a linear solution to the over-constrained problem. Another form of triangulation, presented by Seitz and Anandan [153], involves placing a 2D probability distribution around each measured 2D projection $\phi_i(\mathbf{p})$ to represent the likelihood that the true projection position is somewhere near the measured one. Each 2D distribution induces a distribution cone in 3D, and the "intersection" of these cones gives a distribution for the position of $\mathbf{p}$ in 3D.

3.5.4 Projective reconstruction using a fundamental matrix

Once the fundamental matrix $\mathbf{F}_{ij}$ has been determined, it is immediately possible to create a projective scene reconstruction. Let $\mathbf{e}_j$ denote the left epipole of $\mathbf{F}_{ij}$ and let $\mathbf{M}$ be any invertible 3×3 matrix such that

$$\mathbf{F}_{ij} \cong [\mathbf{e}_j]_\times \mathbf{M}$$

Note that $\mathbf{e}_j$ is found directly from $\mathbf{F}_{ij}$ but is only determined up to a scale factor; thus we can write $\mathbf{e}_j = \beta_1 \{\hat{\mathbf{e}}\}_j$ where $\beta_1 \in \mathbb{R}$ and $\{\hat{\mathbf{e}}\}_j$ is from Eq. 3.10. Next, create the following two 3×4 matrices:

$$\begin{aligned} \breve{\mathbf{\Pi}}_{wi} &:= \begin{bmatrix} \mathbf{I}, & \mathbf{0} \end{bmatrix} \\ \breve{\mathbf{\Pi}}_{wj} &:= \begin{bmatrix} \mathbf{M}, & -\mathbf{e}_j \end{bmatrix} \end{aligned} \tag{3.19}$$

Then camera matrices $\breve{\mathbf{\Pi}}_{wi}$ and $\breve{\mathbf{\Pi}}_{wj}$ are a projective reconstruction of the original cameras which has been derived entirely from the fundamental matrix $\mathbf{F}_{ij}$.

To prove the claim above, we must find a projective transformation $\mathbf{\Psi}$ such that

$$\breve{\mathbf{\Pi}}_{wi} \cong \mathbf{\Pi}_{wi}\mathbf{\Psi} \quad \text{and} \quad \breve{\mathbf{\Pi}}_{wj} \cong \mathbf{\Pi}_{wj}\mathbf{\Psi}$$

Here recall that $\mathbf{\Pi}_{wi}$ and $\mathbf{\Pi}_{wj}$ are the original metric camera matrices (in the chosen world coordinate system w), which have the specific form

$$\mathbf{\Pi}_{wi} = \mathbf{K}_i \mathbf{R}_i \, [\, \mathbf{I}, \;\; -\mathbf{t}_i \,] \tag{3.20}$$

The key to finding $\mathbf{\Psi}$ is to realize that

$$\mathbf{M} = \beta_2 \mathbf{H}_{ij}^{\infty} + \mathbf{e}_j \mathbf{a}^\top \tag{3.21}$$

for some vector $\mathbf{a} \in \mathbb{R}^3$ and $\beta_2 \in \mathbb{R}$. The fact that $\mathbf{M}$ must have the form given by Eq. 3.21 is proven in [70]; although $\mathbf{M}$ is determined from $\mathbf{F}_{ij}$, which has unknown scale, the unknown scale is handled by β_2 and $\mathbf{a}$. $\mathbf{\Psi}$ can now be set as

$$\mathbf{\Psi}^{-1} := \frac{1}{\beta_2}\begin{bmatrix} \mathbf{H}^{\infty}_{wi} & -\mathbf{H}^{\infty}_{wi}\mathbf{t}_i \\ \mathbf{a}^{\top}\mathbf{H}^{\infty}_{wi} & \alpha \end{bmatrix} = \frac{1}{\beta_2}\begin{bmatrix} \mathbf{K}_i\mathbf{R}_i & -\mathbf{K}_i\mathbf{R}_i\mathbf{t}_i \\ \mathbf{a}^{\top}\mathbf{H}^{\infty}_{wi} & \alpha \end{bmatrix} \tag{3.22}$$

where $\alpha \in \mathbb{R}$ is determined below. As a test that $\mathbf{\Psi}$ is correct and to determine α, observe

$$\begin{aligned}
\breve{\mathbf{\Pi}}_{wj}\mathbf{\Psi}^{-1} &= \tfrac{1}{\beta_2}\begin{bmatrix} \mathbf{M}, & -\mathbf{e}_j \end{bmatrix}\begin{bmatrix} \mathbf{H}^{\infty}_{wi} & -\mathbf{H}^{\infty}_{wi}\mathbf{t}_i \\ \mathbf{a}^{\top}\mathbf{H}^{\infty}_{wi} & \alpha \end{bmatrix} \\
&= \tfrac{1}{\beta_2}\begin{bmatrix} \beta_2\mathbf{H}^{\infty}_{ij}\mathbf{H}^{\infty}_{wi} + \mathbf{e}_j\mathbf{a}^{\top}\mathbf{H}^{\infty}_{wi} - \mathbf{e}_j\mathbf{a}^{\top}\mathbf{H}^{\infty}_{wi}\,, & -\beta_2\mathbf{H}^{\infty}_{ij}\mathbf{H}^{\infty}_{wi}\mathbf{t}_i - \mathbf{e}_j\mathbf{a}^{\top}\mathbf{H}^{\infty}_{wi}\mathbf{t}_i - \alpha\mathbf{e}_j \end{bmatrix} \\
&= \tfrac{1}{\beta_2}\begin{bmatrix} \beta_2\mathbf{H}^{\infty}_{wj}\,, & -\beta_2\mathbf{H}^{\infty}_{wj}\mathbf{t}_i - \beta_3\mathbf{e}_j \end{bmatrix} \\
&= \tfrac{1}{\beta_2}\begin{bmatrix} \beta_2\mathbf{H}^{\infty}_{wj}\,, & -\beta_2\mathbf{H}^{\infty}_{wj}\mathbf{t}_i - \beta_3\beta_1\mathbf{H}^{\infty}_{wj}(\mathbf{t}'_i - \mathbf{t}_i) \end{bmatrix} \\
&= \tfrac{1}{\beta_2}\begin{bmatrix} \beta_2\mathbf{H}^{\infty}_{wj}\,, & -\beta_2\mathbf{H}^{\infty}_{wj}\mathbf{t}'_i \end{bmatrix} \\
&= \mathbf{H}^{\infty}_{wj}\begin{bmatrix} \mathbf{I}, & -\mathbf{t}'_i \end{bmatrix} \\
&= \mathbf{\Pi}_{wj}
\end{aligned}$$

where we set $\beta_3 := \beta_2/\beta_1$ and $\alpha := \beta_3 - \mathbf{a}^{\top}\mathbf{H}^{\infty}_{wi}\mathbf{t}_i$.

Note that if world coordinates w are chosen to make $\mathbf{t}_i = \mathbf{0}$ (so that the world origin is at the optical center of camera i) and $\mathbf{R}_i = \mathbf{I}$, then by Eq. 3.22

$$\mathbf{\Psi}^{-1} = \frac{1}{\beta_2}\begin{bmatrix} \mathbf{K}_i & \mathbf{0} \\ \mathbf{a}^{\top}\mathbf{H}^{\infty}_{wi} & \alpha \end{bmatrix} \cong \begin{bmatrix} \frac{1}{\alpha}\mathbf{K}_i & \mathbf{0} \\ \mathbf{a}^{\top}(\frac{1}{\alpha}\mathbf{K}_i) & 1 \end{bmatrix} \tag{3.23}$$

Eq. 3.23 reappears in Eq. 4.6 of Section 4.3.3.

Once a projective camera reconstruction has been created, a projective scene reconstruction can be created by triangulating corresponding points on the camera's image planes. Note that the material presented in this section was first discussed by Faugeras [44], albeit in a significantly different form involving choice of projective basis.

3.5.5 The absolute quadric and metric reconstruction

It is convenient at this juncture to discuss the absolute quadric further and to provide a concrete example. The absolute quadric was previously mentioned in Section 2.2 as part of the history and context of camera self calibration. The absolute quadric has become very popular for self calibration because of its relative simplicity and the ability it provides to directly utilize known constraints on internal calibration (i.e., the matrix $\mathbf{K}$).

Let $\mathbf{\Pi}_{wi}$ for $1 \leq i \leq n$ be a series of camera matrices in a metric reference frame and let

$$\breve{\mathbf{\Pi}}_{wi} \stackrel{\text{def}}{=} \mathbf{\Pi}_{wi}\mathbf{\Psi}$$

be a projective reconstruction of these cameras, where $\mathbf{\Psi}$ is an invertible 4×4 matrix. By choosing world coordinates so that $\mathbf{R}_1 = \mathbf{I}$ and $\mathbf{t}_1 = \mathbf{0}$ (see Eq. 3.20), $\mathbf{\Psi}$ has the form given by Eq. 3.23 (after letting $\mathbf{K}_1$ absorb the $\frac{1}{\alpha}$ term):

$$\mathbf{\Psi} = \begin{bmatrix} \mathbf{K}_1^{-1} & \mathbf{0} \\ -\mathbf{a}^\top & 1 \end{bmatrix}$$

Next, define the 4×4 matrix

$$\mathbf{\Omega}^* \stackrel{\text{def}}{\cong} \mathbf{\Psi}^{-1} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} (\mathbf{\Psi}^{-1})^\top = \begin{bmatrix} \mathbf{K}_1\mathbf{K}_1^\top & \mathbf{K}_1\mathbf{K}_1^\top\mathbf{a} \\ \mathbf{a}^\top\mathbf{K}_1\mathbf{K}_1^\top & \mathbf{a}^\top\mathbf{K}_1\mathbf{K}_1^\top\mathbf{a} \end{bmatrix} \tag{3.24}$$

The matrix $\mathbf{\Omega}^*$ is known as the *dual image of the absolute quadric*; the terminology is technical and understanding its meaning is not necessary for this discussion.

Now consider what happens when we form the following matrix product:

$$\begin{aligned}
\breve{\mathbf{\Pi}}_{wi}\mathbf{\Omega}^*\breve{\mathbf{\Pi}}_{wi}^\top &\cong \mathbf{K}_i\mathbf{R}_i \begin{bmatrix} \mathbf{I} & -\mathbf{t}_i \end{bmatrix} \mathbf{\Psi}\mathbf{\Omega}^*\mathbf{\Psi}^\top \begin{bmatrix} \mathbf{I} \\ -\mathbf{t}_i^\top \end{bmatrix} \mathbf{R}_i^\top\mathbf{K}_i^\top \\
&= \mathbf{K}_i\mathbf{R}_i \begin{bmatrix} \mathbf{I} & -\mathbf{t}_i \end{bmatrix} \begin{bmatrix} \mathbf{K}_1^\top & \mathbf{K}_1^\top\mathbf{a} \\ \mathbf{0}^\top & 0 \end{bmatrix} \mathbf{\Psi}^\top \begin{bmatrix} \mathbf{I} \\ -\mathbf{t}_i^\top \end{bmatrix} \mathbf{R}_i^\top\mathbf{K}_i^\top \\
&= \mathbf{K}_i\mathbf{R}_i \begin{bmatrix} \mathbf{I} & -\mathbf{t}_i \end{bmatrix} \begin{bmatrix} \mathbf{K}_1^\top & \mathbf{K}_1^\top\mathbf{a} \\ \mathbf{0}^\top & 0 \end{bmatrix} \begin{bmatrix} (\mathbf{K}_1^{-1})^\top & -\mathbf{a} \\ \mathbf{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \mathbf{I} \\ -\mathbf{t}_i^\top \end{bmatrix} \mathbf{R}_i^\top\mathbf{K}_i^\top \\
&= \mathbf{K}_i\mathbf{R}_i \begin{bmatrix} \mathbf{I} & -\mathbf{t}_i \end{bmatrix} \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{bmatrix} \begin{bmatrix} \mathbf{I} \\ -\mathbf{t}_i^\top \end{bmatrix} \mathbf{R}_i^\top\mathbf{K}_i^\top \\
&= \mathbf{K}_i\mathbf{R}_i\mathbf{I}\mathbf{R}_i^\top\mathbf{K}_i^\top = \mathbf{K}_i\mathbf{K}_i^\top \cong \omega_i^*
\end{aligned} \tag{3.25}$$

Note that the projective camera matrices $\breve{\mathbf{\Pi}}_{wi}$ can be determined directly from images taken by the cameras (e.g., using a method like that in Section 3.5.4 for two-camera projective reconstruction), and that in Eq. 3.25 there is no requirement that each camera have the same internal calibration $\mathbf{K}$. The significance of Eq. 3.25 is that it allows entries in the projective camera matrices $\breve{\mathbf{\Pi}}_{wi}$ to be directly related to entries in internal calibration matrices, *because* $\mathbf{\Omega}^*$ *is the same for each camera*. This can be especially useful when some properties of the internal calibration matrices (e.g., zero skew) are already known. For example, if we assume

$$\mathbf{K}_i = \begin{bmatrix} f_i & 0 & 0 \\ 0 & f_i & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(e.g., because we have predetermined the principal point) then

$$\mathbf{K}_i\mathbf{K}_i^\top = \begin{bmatrix} f_i^2 & 0 & 0 \\ 0 & f_i^2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Now letting

$$\mathbf{\Omega}^* = \begin{bmatrix} a & b & c & d \\ b & e & f & g \\ c & f & h & i \\ d & g & i & j \end{bmatrix}$$

we can use equation Eq. 3.25 to determine the entries $a, \ldots, j \in \mathbb{R}$ of $\mathbf{\Omega}^*$. Of course, it will be necessary to use the constraints arising from Eq. 3.25 for several camera views before enough constraints exist to determine $\mathbf{\Omega}^*$. See Pollefeys, Koch, and Van Gool [132, 131] for details. Note that once $\mathbf{\Omega}^*$ has been determined, the projective reconstruction can be upgraded immediately to metric because $\mathbf{K}$ and $\mathbf{a}$ have been determined.

One potential problem with this approach, besides the requirement that some assumptions be made about internal calibration, is that the entries of $\mathbf{\Omega}^*$ are not independent of each other, and yet they are treated as being independent when Eq. 3.25 is invoked. Another problem is the way in which $\mathbf{\Omega}^*$ is defined relative to a specific camera (camera 1 in the example above), thus introducing bias.

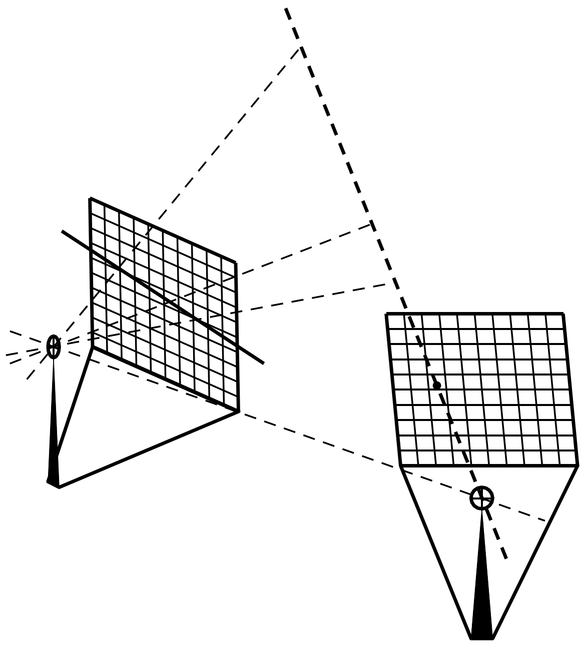

Figure 3.9 Physical explanation for why point correspondences are ambiguous and thus why the fundamental matrix is rank deficient (i.e., not invertible). Knowledge that a 3D scene point projects to a particular location in the right-hand view only reveals which epipolar line in the left-hand view the scene point will project onto.

3.6 Finding point correspondences in practice

If two cameras, i and j, view a scene point $\mathbf{p} \in \mathbb{R}^3$ then the pair of points $\phi_i(\mathbf{p})$ and $\phi_j(\mathbf{p})$ is called a *point correspondence* between the views. This concept can be generalized in the obvious way when more than two views are available.

Point correspondences are ambiguous, as depicted in Fig. 3.9 where a point in the right-hand view is shown to correspond to a line in the left-hand view. However, identifying point correspondences between views is the first step in most self-calibration and scene-reconstruction algorithms. Determining accurate point correspondences is paramount. For most of the history of photogrammetry, point correspondences were determined manually. Computers have opened the possibility of automatically determining point correspondences. When only a small number of point correspondences have been determined between two views, the correspondence is said to be *sparse*. When a large number of correspondences have been determined, the correspondence is *dense*. Because of the labor involved, a dense correspondence is only feasible when determined mechanically, either by a computer algorithm (e.g., [126, 89, 155]), or from available data (e.g., if the views are the output of a computer graphics algorithm rendering a synthetic scene [26, 157]), or from an engineered means (e.g., structured light [198]).

Software algorithms for determining point correspondences rely on the presence of *texture* in the scene, which means identifiable surface variations such as markings (e.g., writing, dirt, decorations, small holes) or shadows from a fixed light source. Apparent correspondences caused by occluding boundaries are often erroneous and often mislead algorithms. Having a reliable fundamental matrix between the views can be of great assistance in finding point correspondences: if $\mathbf{F}_{ij}$ is the fundamental matrix between views i and j and if $\{\mathbf{p}\}_i$ is a feature point in view i, then it is known that the corresponding point $\{\mathbf{p}\}_j$ in view j is on the epipolar line given by $\mathbf{F}_{ij}\{\mathbf{p}\}_i$. Thus it is only necessary to search for $\{\mathbf{p}\}_j$ on this line or nearby.

3.7 Finding fundamental matrices in practice

Fundamental matrices are notoriously difficult to determine reliably. The best way to find a fundamental matrix is to (1) use a very large field of view, (2) have reliable methods for eliminating lens distortion, (3) use many point correspondences, (4) locate point correspondences to sub-pixel accuracy (or equivalently, use high-resolution views), and (5) determine correspondences by hand to avoid outliers. In photogrammetry applications, special cameras are used that have no lens distortion and so suggestion (2) would not even be relevant. However, in many machine vision applications we would like to be able to use non-specialized cameras, what photogrammetrists derisively call "amateur equipment." Suggestion (5) is a traditional part of photogrammetry but is not a desirable option in machine vision, where algorithms should be automated as much as possible. To automate the process, the RANSAC method is typically used to cull outliers; this is discussed in greater detail below.

The following is an outline of the standard automatic technique for finding the fundamental matrix between two views [131, 70]:

(Step 1) Find feature points in both views.

(Step 2) Find potential conjugate pairs among the feature points.

(Step 3) Use RANSAC *to find a fundamental matrix that matches well with the greatest number of potential conjugate pairs.*

(Step 4) Use the fundamental matrix to eliminate potential conjugate pairs that are outliers.

(Step 5) (Optional) Return to step 2 with the reduced set of feature points.

(Step 6) Use all remaining conjugate pairs simultaneously to find a best-fit fundamental matrix.

(Step 7) (Optional) Use the calculated fundamental matrix to find more feature points and better potential conjugate pairs (i.e., ones that approximately match the epipolar geometry given by the fundamental matrix); return to step 3.

Step 1 is accomplished using corner detectors (e.g., [101, 65]; see also the comparison in [152]). The goal is to find positions that "stand out" and are likely to be detected in another nearby view as well. For example, a speck of dirt on a blank wall makes a good feature. Step 2 can be performed by comparing the immediate neighborhoods around each feature point using cross-differencing. Since a fundamental matrix can be calculated with 7 or 8 conjugate points, step 3 involves repeatedly choosing a small number of conjugate point pairs, finding the fundamental matrix for these points, and checking how well this matrix fits the other conjugate pairs. A fundamental matrix "fits" a conjugate pair if it generates the correct epipolar line for the pair. Correctness is measured using the distance from each point to its corresponding epipolar line; smaller distances are better. The RANSAC process continues until a fundamental matrix is found that fits a large number of the conjugate pairs.

By step 4, a reasonable fundamental matrix has been found. All potential conjugate pairs that are in disagreement with the fundamental matrix are considered bad and eliminated. The optional step 5 assumes that eliminating bad conjugate pairs has improved the chances of RANSAC randomly finding the correct fundamental matrix. Step 6 assumes that using all remaining "good" conjugate pairs at once will stabilize the fundamental matrix calculation. The optional step 7 is part of determining a "dense" correspondence between the two views [89]. When a feature point is determined in one view, it is only necessary to search along (or nearby) the corresponding epipolar line in the second view for the other half of the conjugate pair. The epipolar line is found using the estimated fundamental matrix. Both optional steps must be treated in a relaxation manner, gradually tightening the bounds over repeated iterations.

While this algorithm seems reasonable, it still has many problems in practice. *There is no substitute for good input data!* This is why suggestions (1), (2), (4), and (5) at the beginning of the section are of crucial importance.

Fundamental matrices can also be determined from line correspondences or combinations of line and points correspondences (e.g., [2, 66, 67]). Some researchers (e.g.,[180]) have experimented with determining trilinear/trifocal tensors [168, 159, 69], from which fundamental matrices can be extracted. The theory is that, since trilinear tensors are determined from 3 views and have stricter requirements than pairwise epipolar geometry, they can be determined more reliably. Thus determining a trilinear tensor and then extracting fundamental matrices from it should be a more reliable way to determine fundamental matrices. In practice, the trilinear tensor seems to be extremely sensitive to noise and can possibly lead to worse estimates of epipolar geometry than direct calculation of the fundamental matrix. Stein [170] discussed the possibility of determining trilinear tensors (and hence fundamental matrices) directly from optical flow.

Chapter 4

Screw-Transform Manifolds

4.1 The screw transformation between two views

It was shown by Chasles in 1837, and apparently earlier by both Cauchy and Mozzi [17], that if a rigid object is arbitrarily moved from one location to another, the movement $\mathbf{M}$ of the object can always be represented as a *screw transformation*. This means there exists a line in space called the *screw axis*, a rotation $\mathbf{U}$ around the screw axis, and a translation $\mathbf{V}$ parallel to the screw axis such that

$$\mathbf{M} = \mathbf{U}\,\mathbf{V} = \mathbf{V}\,\mathbf{U}$$

Although the mathematical-style notation used above is intended to convey the concept informally, $\mathbf{M}$, $\mathbf{U}$, and $\mathbf{V}$ could be thought of as 4×4 matrices representing rigid transformations in a homogeneous coordinate system. Fig. 4.1 gives a sample screw transformation. The term "screw transformation" refers to the way a screw both turns around and moves parallel to its axis.

If a camera with fixed internal parameters captures two views of a scene, then there exists a screw transformation that moves the camera from its first viewing location and orientation to its second. This screw transformation can be used when defining a world coordinate system for the cameras; in particular, a coordinate system can be chosen with its z-axis equal to the screw axis and the position $(1, 0, 0)^\top$ equal to the optical center of the first camera view. Letting upper-triangular, 3×3 matrix $\mathbf{K}$ denote the camera's fixed internal parameters, the camera matrices for the two

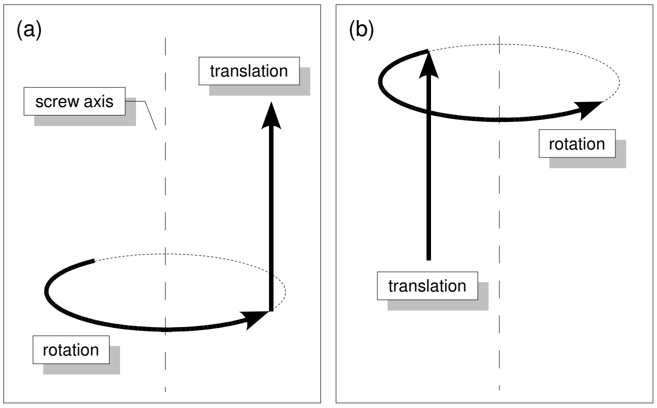

Figure 4.1 (a) A screw transformation. Every rigid-body motion can be decomposed into a purely rotational component and a purely translational component, which together form a screw transformation. (b) The two components can be composed in either order to recreate the original motion.

views written in the screw-transformation-based coordinate system are:

$$\mathbf{\Pi}_A = \mathbf{K}\,\mathbf{R}\left[\begin{array}{ccc|c} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{array}\right]$$

$$\mathbf{\Pi}_B = \mathbf{\Pi}_A \mathbf{S}(-\gamma, -\theta)$$

Here the letters A and B refer to the first and second view, respectively, the matrix $\mathbf{R}$ is a rotation matrix representing the "tilt" of the camera relative to the coordinate system, and the matrix $\mathbf{S}$ is given by

$$\mathbf{S}(\gamma, \theta) = \left[\begin{array}{ccc|c} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & \gamma \\ 0 & 0 & 0 & 1 \end{array}\right]$$

Note that γ and θ are parameters from the screw transformation: γ is the amount of translation parallel to the screw axis and θ is the amount of rotation around the screw axis. The distance from the optical center (of either view) to the screw axis is the basic unit of measurement in this coordinate system; the amount of screw translation γ is in terms of these units.

The two camera matrices given above will also arise from an alternative scenario in which the camera is motionless while the scene undergoes a transformation equal and opposite to the original camera motion, namely $\mathbf{S}(-\gamma, -\theta)$. Physically this occurs if the camera is mounted at position $(1,0,0)^\top$ while the scene being viewed rests on a "rising turntable" with rotation axis equal to the z-axis. A *rising turntable* is a turntable that both rotates around its axis and moves up and down parallel to its axis. Thus we refer to the alternative interpretation as the *rising-turntable formulation* of the viewing scenario. We refer to the duality of interpretation as *camera relativism*.

The fundamental matrix between the two views can be derived directly from the two camera matrices $\mathbf{\Pi}_A$ and $\mathbf{\Pi}_B$ (see Section A.1.2). The resulting fundamental matrix given in terms of the underlying screw transformation is

$$\mathbf{F} = \mathbf{F}^{\mathsf{A}} + \mathbf{F}^{\mathsf{S}} \tag{4.1}$$

$$\mathbf{F}^{\mathsf{A}} = [\sin\theta\mathbf{h}_2 + \gamma\cos\theta\mathbf{h}_3]_\times \tag{4.2}$$

$$\begin{aligned}\mathbf{F}^{\mathsf{S}} = {} & \frac{1-\cos\theta}{|\mathbf{H}|}\Big[(\mathbf{h}_1\times\mathbf{h}_3)(\mathbf{h}_1\times\mathbf{h}_2)^\top + (\mathbf{h}_1\times\mathbf{h}_2)(\mathbf{h}_1\times\mathbf{h}_3)^\top\Big] + \\ & \frac{\gamma\sin\theta}{|\mathbf{H}|}\Big[(\mathbf{h}_1\times\mathbf{h}_3)(\mathbf{h}_1\times\mathbf{h}_3)^\top + (\mathbf{h}_2\times\mathbf{h}_3)(\mathbf{h}_2\times\mathbf{h}_3)^\top\Big]\end{aligned} \tag{4.3}$$

where $\mathbf{H} = [\,\mathbf{h}_1\ \mathbf{h}_2\ \mathbf{h}_3\,] = \mathbf{KR}$.

We will use the 4-tuple $S = (\mathbf{K},\mathbf{R},\theta,\gamma)$ to conveniently and formally refer to the view-capturing scenario described above. Let $\mathrm{fma}(S)$ denote the fundamental matrix arising from scenario S and let $\mathrm{hin}(S)$ denote $\mathbf{H}^\infty = \mathbf{KRK}^{-1}$, which is called the *relative calibration* arising from scenario S or the *homography induced by the plane at infinity*. Note that both $\mathrm{fma}(S)$ and $\mathrm{hin}(S)$ can be calculated directly from S, the first through Eq. 4.1 and the second through knowledge of $\mathbf{K}$ and $\mathbf{R}$. The question we will consider in the next few sections is, what information does knowledge of $\mathbf{F}$ (without knowledge of the 4-tuple $(\mathbf{K},\mathbf{R},\theta,\gamma)$) provide us about $\mathbf{H}^\infty$ and $\mathbf{K}$. This question is of interest because $\mathbf{F}$ can be found directly from pairs of views by identifying feature-point correspondences [44, 73] and thus information about internal and relative calibration can be found directly from camera views.

4.2 Upgrading weak calibration

When the fundamental matrix between two views is known, the views are said to be *weakly calibrated*. Two stronger forms of calibration are *affine calibration* (i.e., knowledge of the relative calibration $\mathbf{H}^{\infty}$ between two views) and *metric calibration* (i.e., knowledge of the internal calibration $\mathbf{K}$ of the camera). We will use the term *monocular fundamental matrix* to describe a fundamental matrix arising from two views taken by a camera with fixed internal parameters. The goal of this section is to show how a monocular fundamental matrix can be upgraded to affine or metric calibration through the knowledge of 2 or 3 real numbers, respectively, that are related to the underlying screw decomposition of the camera motion. Since any choice of real numbers will lead to different, legal affine or metric calibrations, it is clear that weak calibration cannot be immediately upgraded without further information. More importantly, the parameterization given by these real numbers leads to a new method for self calibration (see Section 4.3).

4.2.1 Parameterizing rising-turntable scenarios

A given rising-turntable scenario $S = (\mathbf{K}, \mathbf{R}, \theta, \gamma)$ has a corresponding fundamental matrix $\mathbf{F} = \mathrm{fma}(S)$. There may be many other scenarios $S' = (\mathbf{K}', \mathbf{R}', \theta', \gamma')$ that give rise to the same fundamental matrix, so that $\mathrm{fma}(S') = \mathrm{fma}(S) = \mathbf{F}$. Define the set of scenarios consistent with fundamental matrix $\mathbf{F}$ as

$$\mathrm{consce}(\mathbf{F}) = \{S' : \mathrm{fma}(S') = \mathbf{F}\}.$$

Letting $\Phi_{\mathbf{F}}$ denote the mapping given by Algorithm A–1 (Fig. 4.2) for a fixed $\mathbf{F}$, we have the following:

CONJECTURE 1. *For every $S \in \mathrm{consce}(\mathbf{F})$, there exists a triplet of real numbers $(\kappa, \theta, \gamma) \in \mathbb{R}^3$ with $S = \Phi_{\mathbf{F}}(\kappa, \theta, \gamma)$.*

This statement is presented as a conjecture rather than a theorem because it is extremely difficult to prove formally. We have confidence that the conjecture is true for the following reasons:

- Each step in the algorithm is derived mathematically from Eq. 4.1 (see the step-by-step derivation in Appendix A.2.1).

- Computer experiments are consistent with the conjecture.

Assuming the conjecture is true (except perhaps for isolated special cases), the function $\Phi_{\mathbf{F}}$ represents a parameterization of the set $\mathrm{consce}(\mathbf{F})$. That is, each member of $\mathrm{consce}(\mathbf{F})$ can be associated with a triplet of real numbers. In Section 4.3 it is shown how this parameterization leads to "screw-transform manifolds" and a new method for self calibration.

Note that Algorithm A–1 only defines $\Phi_{\mathbf{F}}$ when the camera undergoes *general motion*, meaning $\gamma \neq 0$, $\theta \neq 0$, and the screw axis does not intersect the optical center. There are two additional, nontrivial motions that can occur and each leads to a different definition of $\Phi_{\mathbf{F}}$; these cases, called "turntable motion" and "transfocal motion," are discussed in Section 4.4.

4.2.2 Parameterizing relative calibration

The previous section introduced the set $\mathrm{consce}(\mathbf{F})$ of all rising-turntable scenarios sharing a given fundamental matrix $\mathbf{F}$. Since each scenario in $\mathrm{consce}(\mathbf{F})$ has a corresponding relative calibration, the set of all relative calibrations that are consistent with $\mathbf{F}$ can be defined as

$$\mathrm{conhin}(\mathbf{F}) = \{\mathrm{hin}(S') : S' \in \mathrm{consce}(\mathbf{F})\}$$

Letting $\Lambda_{\mathbf{F}}$ denote the mapping given by Algorithm B–1 (Fig. 4.2.1) for a fixed $\mathbf{F}$, we have the following:

CONJECTURE 2. *For every* $\mathbf{H} \in \mathrm{conhin}(\mathbf{F})$*, there exists a pair of real numbers* $(\kappa, \theta) \in \mathbb{R}^2$ *with* $\mathbf{H} = \Lambda_{\mathbf{F}}(\kappa, \theta)$.

Again, we present this statement as a conjecture rather than a theorem because it is extremely difficult to prove formally, and again we believe the conjecture is true because algorithm B–1 was deduced mathematically from Eq. 4.1 (see Appendix A.2.1 and Appendix A.2.4) and because computer experiments are consistent with its validity.

As with the function $\Phi_{\mathbf{F}}$, the function $\Lambda_{\mathbf{F}}$ represents a parameterization of the set $\mathrm{conhin}(\mathbf{F})$. In this case, each relative calibration that is consistent with a given fundamental matrix corresponds

ALGORITHM A–1

(1) Let $\mathbf{M}$ be any invertible 3×3 matrix such that $\mathbf{F} = [\mathbf{e}]_\times \mathbf{M}$, where $\mathbf{e}$ is the left epipole of $\mathbf{F}$ (i.e., $\mathbf{e}^\top \mathbf{F} = 0$).

(2) Let $\underline{\mathbf{h}}_3 = (\kappa \mathbf{I} - \mathbf{M})^{-1}\mathbf{e}$.

(3) Let $\underline{\mathbf{h}}_1 = (\mathbf{F}^{\mathsf{S}}\mathbf{m}) \times (\mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_3)$, where $[\mathbf{m}]_\times = \mathbf{F}^{\mathsf{A}}$.

(4) Find the unique null eigenvector $(\sigma_1, \sigma_2)^\top$ of $\left[\begin{array}{cc} \mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_1, & -\mathbf{F}^{\mathsf{A}}\underline{\mathbf{h}}_3 \end{array}\right]$. Note that $(\sigma_1, \sigma_2)^\top = \phi(1/s_1, \gamma/s_3)^\top$, where $s_1\mathbf{h}_1 = \underline{\mathbf{h}}_1$, $s_3\mathbf{h}_3 = \underline{\mathbf{h}}_3$, and ϕ is an unknown scalar determined in step (5).

(5) Solve the over-determined system $\phi\mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_3 = \sigma_1(1 - \cos\theta)(\underline{\mathbf{h}}_1 \times \underline{\mathbf{h}}_3)$ for ϕ.

(6) Let $\mathbf{h}_2 = (\phi\mathbf{m} - \sigma_2 \cos\theta \underline{\mathbf{h}}_3)/(\phi \sin\theta)$.

(7) Use γ (and ϕ) to extract s_3 from the eigenvector in step (4). Then $\mathbf{h}_3 = \underline{\mathbf{h}}_3/s_3$.

(8) Since $\mathbf{KR} = \mathbf{H} = [\mathbf{h}_1\mathbf{h}_2\mathbf{h}_3]$, perform QR decomposition on $\mathbf{H}$ to recover $\mathbf{K}$ and $\mathbf{R}$.

Figure 4.2 Algorithm for mapping $(\mathbf{F}, \kappa, \theta, \gamma)$ to a scenario $S = (\mathbf{K}, \mathbf{R}, \theta, \gamma) \in \mathrm{consce}(\mathbf{F})$ that is consistent with monocular fundamental matrix $\mathbf{F}$. The underlying motion is assumed to be general; alternative motions are described in Section 4.4, Fig. 4.6, and Fig. 4.8.

ALGORITHM B–1

(1) Perform steps (1)-(6) of Algorithm A–1 (Fig. 4.2) to find $\underline{\mathbf{h}}_3$ and $\underline{\mathbf{l}}_{12}$.

(2) Use Algorithm C (Fig. 4.4) to find $\mathbf{H}^\infty$.

Figure 4.3 Algorithm for mapping $(\mathbf{F}, \kappa, \theta)$ to $\mathbf{H}^\infty = \mathrm{hin}(S)$ for some scenario $S = (\mathbf{K}, \mathbf{R}, \theta, \gamma) \in \mathrm{consce}(\mathbf{F})$ that is consistent with monocular fundamental matrix $\mathbf{F}$. The underlying motion is assumed to be general.

ALGORITHM C

(1) Let $\mathbf{M}$ be any invertible 3×3 matrix such that $\mathbf{F} = [\mathbf{e}]_\times \mathbf{M}$, where $\mathbf{e}$ is the left epipole of $\mathbf{F}$ (i.e., $\mathbf{e}^\top \mathbf{F} = 0$).

(2) Let $\xi = \mathbf{e}^\top \mathbf{l}_{12}$ and $\mathbf{q} = \mathbf{M}^\top \mathbf{l}_{12}$.

(3) Solve the following 6×5 system to find the null eigenvector $(\lambda, \lambda\mathbf{a}, 1)^\top$:

$$\begin{bmatrix} \mathbf{M}_{(11)} + \mathbf{M}_{(22)} + \mathbf{M}_{(33)} & \mathbf{e}^\top & -(1 + 2\cos\theta) \\ \mathbf{M}\underline{\mathbf{h}}_3 & \mathbf{e}\underline{\mathbf{h}}_3^\top & -\underline{\mathbf{h}}_3 \\ (\mathbf{l}_{12})_x \mathbf{q}_y - (\mathbf{l}_{12})_y \mathbf{q}_x & (-(\mathbf{l}_{12})_y \xi, (\mathbf{l}_{12})_x \xi, 0) & 0 \\ (\mathbf{l}_{12})_x \mathbf{q}_z - (\mathbf{l}_{12})_z \mathbf{q}_x & (-(\mathbf{l}_{12})_z \xi, 0, (\mathbf{l}_{12})_x \xi) & 0 \end{bmatrix} \begin{bmatrix} \lambda \\ \lambda\mathbf{a} \\ 1 \end{bmatrix} = \mathbf{0}$$

(4) Having determined $\mathbf{a} \in \mathbb{R}^3$ in step (3), find $\mathbf{H}^\infty$ using

$$\mathbf{H}^\infty \cong \mathbf{M} + \mathbf{e}\mathbf{a}^\top \tag{4.4}$$

Figure 4.4 Algorithm for determining $\mathbf{H}^\infty$ from $\mathbf{F}$, θ, $\underline{\mathbf{l}}_{12}$, and $\underline{\mathbf{h}}_3$.

to a pair of real numbers. In Section 4.3 the function $\Lambda_{\mathbf{F}}$ is used to define the "modulus-constraint manifold" while the earlier function $\Phi_{\mathbf{F}}$ is used to define the "Kruppa-constraint manifold."

4.3 Screw-transform manifolds and self calibration

The fundamental matrix between two views contains certain limited information about camera calibration. In this section, we describe two existing self-calibration methods for converting this information into affine and metric calibration. We then show how the parameterizations given in Section 4.2 induce objects called "screw transform manifolds" that lead to new algorithms for self calibration.

4.3.1 Kruppa-constraint manifold

The first published self-calibration method, due to Faugeras, Luong, and Maybank [41], was based on the Kruppa constraints [92]. In particular, for a scenario $S = (\mathbf{K}, \mathbf{R}, \theta, \gamma)$ we have

$$\mathbf{F} \cong [\mathbf{e}_B]_\times \mathbf{H}^\infty$$

where $\mathbf{F} = \mathrm{fma}(S)$, $\mathbf{H}^\infty = \mathrm{hin}(S)$, and $\mathbf{e}_B$ is the epipole viewed in camera B, given by $\mathbf{e}_B \cong \Pi_B[1, 0, 0, 1]^\top$. It was observed that

$$\begin{aligned} \mathbf{F}\mathbf{K}\mathbf{K}^\top\mathbf{F}^\top &\cong [\mathbf{e}_B]_\times \mathbf{H}^\infty \mathbf{K}\mathbf{K}^\top (\mathbf{H}^\infty)^\top [\mathbf{e}_B]_\times \\ &= [\mathbf{e}_B]_\times (\mathbf{K}\mathbf{R}\mathbf{K}^{-1})\mathbf{K}\mathbf{K}^\top(\mathbf{K}\mathbf{R}\mathbf{K}^{-1})^\top [\mathbf{e}_B]_\times \\ &= [\mathbf{e}_B]_\times \mathbf{K}\mathbf{K}^\top [\mathbf{e}_B]_\times \end{aligned}$$

This equation places constraints on $\omega^* = \mathbf{K}\mathbf{K}^\top$. Because $\mathbf{F}$ can be determined directly from camera views and $\mathbf{e}_B$ can be determined from $\mathbf{F}$, with enough monocular fundamental matrices ω^* can be determined and then $\mathbf{K}$ can be found by Cholesky decomposition. Notice how in this technique metric calibration (i.e., the internal calibration matrix $\mathbf{K}$) is found immediately from weak calibration (i.e., fundamental matrices) without first determining affine calibration. We refer to this style of self calibration as *direct self calibration*, in contrast to the stratified approach of Section 4.3.3.

Define the notation $\mathrm{kma}(S) = \mathbf{K}/\mathrm{frob}(\mathbf{K})$, where $\mathrm{frob}(\mathbf{K})$ is the Frobenius norm of $\mathbf{K}$ (see Section 6.1.1 for the definition of Frobenius norm). We normalize the matrix to reduce the dimensionality of the internal-calibration search space; the matrix $\mathbf{K}$ can only be recovered up to a scale factor by any self-calibration algorithm anyway. Next define

$$\mathrm{conkma}(\mathbf{F}) = \{\mathrm{kma}(S) : S \in \mathrm{consce}(\mathbf{F})\}$$

and the mapping

$$\begin{aligned} \Omega_{\mathbf{F}} \quad : \quad & \mathbb{R}^3 \longrightarrow \mathrm{conkma}(\mathbf{F}) \subseteq \mathbb{R}^5 \\ & (\kappa, \theta, \gamma) \longmapsto \mathrm{kma}(\Phi_{\mathbf{F}}(\kappa, \theta, \gamma)) \end{aligned}$$

Note that although $\mathrm{conkma}(\mathbf{F})$ is not immediately a subset of $\mathbb{R}^5$, it is easy to create an injective mapping from $\mathrm{conkma}(\mathbf{F})$ to $\mathbb{R}^5$. For example, the matrix $\mathbf{K} = [a, b, c; 0, d, e; 0, 0, f] \in \mathrm{conkma}(\mathbf{F})$ can get mapped to $(a, b, c, d, e) \in \mathbb{R}^5$. The value f can later be recovered from the 5-vector by assuming $\mathrm{frob}(\mathbf{K}) = 1$ and thus $f = (1 - a^2 - b^2 - c^2 - d^2 - e^2)^{1/2}$.

The *Kruppa-constraint manifold* is defined as $\Omega_{\mathbf{F}}(\mathbb{R}^3)$, the image of $\mathbb{R}^3$ under the mapping $\Omega_{\mathbf{F}}$. Although we refer to this set and its parameterization as a "manifold," we will not attempt to formally prove the properties that define a manifold. Empirically, the term "manifold" is a good descriptor because the steps defining the algorithm $\Phi_{\mathbf{F}}$ are continuous in most places. The set $\mathbb{R}^3$ together with the mapping $\Omega_{\mathbf{F}}$ is referred to as the *coordinate system* of the manifold.

4.3.2 Self calibration using Kruppa-constraint manifolds

Let a camera with fixed internal calibration $\mathbf{K}$ capture a series of views, and assume that n pairs of these views overlap sufficiently to determine fundamental matrices $\mathbf{F}_1, ..., \mathbf{F}_n$. By camera relativism, each view pair i is equivalent to a rising turntable scenario S_i, so that $\mathbf{F}_i = \mathrm{fma}(S_i)$. By Conjecture 1, each manifold $\Omega_{\mathbf{F}_i}(\mathbb{R}^3)$ contains $\mathbf{K}$ and thus

$$\mathbf{K} \in \bigcap_i \Omega_{\mathbf{F}_i}(\mathbb{R}^3)$$

In other words, $\mathbf{K}$ can be found by intersecting the various Kruppa-constraint manifolds arising from the pairwise fundamental matrices.

Since the domain of $\Omega_{\mathbf{F}}$ is three-dimensional and the range of $\Omega_{\mathbf{F}}$ is five-dimensional, each Kruppa-constraint manifold is a three-dimensional manifold in a five-dimensional space. Since each manifold removes 2 degrees of uncertainty from the location of $\mathbf{K}$, the intersection of three manifolds yields a zero-dimensional space: a disjoint union of points. In their work concerning the modulus constraint, Pollefeys and Van Gool [133, 131] showed that when three fundamental matrices are used there are at most 64 points that satisfy the modulus constraints imposed by each fundamental matrix (see Section 4.3.3); Schaffalitzky [151] reduced this upper bound to 21. Thus since every point on a screw-transform manifold satisfies the modulus constraint imposed by the fundamental matrix generating the manifold, the size of the set of mutual intersection points is no more than 21. Experimental evidence suggests the size is actually much smaller at around 1 or 2 (see Section 6.1.5).

The observations just presented form the basis of a self-calibration algorithm: capture several views, find pairwise fundamental matrices between the views, determine Kruppa-constraint manifolds from the fundamental matrices, and find the mutual intersection point of the manifolds. The difficult part is efficiently finding the mutual intersection point. Some algorithms for this task are presented in Section 5.

4.3.3 Stratified self calibration and the modulus constraint

In this section we look at a *stratified* approach to self calibration in which the internal camera parameters are determined in several stages [33, 45, 35, 185, 133, 105]. Specifically, an initial weak calibration is determined which is then upgraded to affine calibration before finally being converted to metric calibration.

Let n views of a scene be captured by a camera with fixed internal parameters $\mathbf{K}$. As the camera is moved around the scene, the various views are related solely by rotations and translations. Thus the 3×4 camera matrices for the n views can be written

$$\breve{\mathbf{\Pi}}_i = \begin{bmatrix} \breve{\mathbf{H}}_i & \breve{\mathbf{e}}_i \end{bmatrix} = \begin{bmatrix} \mathbf{K}\mathbf{R}_i & \breve{\mathbf{e}}_i \end{bmatrix}$$

where $\breve{\mathbf{H}}_i$ is a 3×3 matrix and $\mathbf{R}_i$ is a rotation matrix. By choosing the appropriate metric coordinate system, we can assume $\breve{\mathbf{e}}_1 = \mathbf{0}$ and $\mathbf{R}_1 = \mathbf{I}$.

Under the stratified self-calibration paradigm, the first step in finding $\mathbf{K}$ and finding $\breve{\mathbf{\Pi}}_i$ is to place the camera matrices in a common projective basis. These initial projective camera matrices will be labeled $\mathbf{\Pi}_i$ and are related to the metric camera matrices by an invertible 4×4 matrix $\mathbf{\Gamma}$ representing a transformation of projective basis:

$$\mathbf{\Pi}_i = \begin{bmatrix} \mathbf{H}_i & \mathbf{e}_i \end{bmatrix} = \breve{\mathbf{\Pi}}_i \mathbf{\Gamma} \tag{4.5}$$

We will refer to the initial set of camera matrices $\mathbf{\Pi}_i$ in the common projective basis as a *projective reconstruction* of the scene. The usual approach to obtaining this projective reconstruction (e.g., [145, 13]) is to identify feature points in the scene that are visible in more than one camera, then reconstruct the features in a common projective basis (e.g., using fundamental matrices [44]), and then find the projective camera matrices by relating the reconstructed features to their viewed positions on the camera image planes. However, features are not an implicit part of the projective reconstruction and we will not refer to them further.

We can always choose the projective basis so that $\mathbf{\Pi}_1 = \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix}$. By Eq. 3.23 (using $\alpha = 1$ and $\mathbf{R}_w = \mathbf{I}$) and with the assumptions so far, $\mathbf{\Gamma}$ must have the form

$$\mathbf{\Gamma} = \begin{bmatrix} \mathbf{K}^{-1} & \mathbf{0} \\ -\breve{\mathbf{a}}^\top & a \end{bmatrix} \tag{4.6}$$

This form is commonly used (e.g., [131]). Different values for the scalar a simply lead to different overall scale factors for the final metric reconstruction. Since metric reconstruction only involves recovering the scene up to a scale factor, we can choose $a = 1$ for convenience.

The second stage in stratified self calibration is to upgrade the projective reconstruction to an affine reconstruction. This is equivalent to finding $\breve{\mathbf{a}}$ in Eq. 4.6. The final stage is to upgrade the affine reconstruction to metric by finding $\mathbf{K}$. The latter step can be performed in a simple way by solving a linear system [68]; the hard step is finding $\breve{\mathbf{a}}$ during the second stage.

Pollefeys et al. [137] introduced an elegant method for determining $\breve{\mathbf{a}}$ called the *modulus constraint*. Notice that from Eq. 4.5 and the assumption $a = 1$ we get

$$\begin{aligned} \mathbf{H}_i &= \breve{\mathbf{H}}_i \mathbf{K}^{-1} - \breve{\mathbf{e}}_i \breve{\mathbf{a}}^\top \\ \mathbf{e}_i &= \breve{\mathbf{e}}_i \end{aligned}$$

leading to

$$\mathbf{K}\mathbf{R}_i\mathbf{K}^{-1} = \mathbf{H}_i + \mathbf{e}_i\breve{\mathbf{a}}^\top \tag{4.7}$$

Since the left-hand side of Eq. 4.7 is conjugate to a rotation matrix, its eigenvalues must all have the same modulus (i.e., absolute value) and this leads to constraints on $\breve{\mathbf{a}}$. Similarly,

$$\mathbf{K}\mathbf{R}_j\mathbf{R}_i^{-1}\mathbf{K}^{-1} = \mathbf{H}_{1j}^\infty(\mathbf{H}_{1i}^\infty)^{-1} = (\mathbf{H}_j + \mathbf{e}_j\breve{\mathbf{a}}^\top)(\mathbf{H}_i + \mathbf{e}_i\breve{\mathbf{a}}^\top)^{-1} \tag{4.8}$$

Here the left-hand side is conjugate to the rotation matrix $\mathbf{R}_j\mathbf{R}_i^{-1}$ leading to additional constraints on $\breve{\mathbf{a}}$. Let $\mathbf{H}_{ij}^\infty(\mathbf{a})$ denote the right-hand side of Eq. 4.7 and Eq. 4.8, respectively, with $\breve{\mathbf{a}}$ replaced by a generic vector $\mathbf{a} \in \mathbb{R}^3$:

$$\mathbf{H}_{ij}^\infty(\mathbf{a}) = \begin{cases} \mathbf{H}_i + \mathbf{e}_i\mathbf{a}^\top & \text{if } i = 1 \\ (\mathbf{H}_j + \mathbf{e}_j\mathbf{a}^\top)(\mathbf{H}_i + \mathbf{e}_i\mathbf{a}^\top)^{-1} & \text{if } i > 1 \end{cases} \tag{4.9}$$

By expanding the characteristic equation for $\mathbf{H}_{ij}^\infty(\mathbf{a})$ and using the modulus constraint, each (i, j) pair with $i < j$ leads to a fourth-order multivariate polynomial ϕ_{ij} in the components of $\mathbf{a}$ that vanishes wherever $\mathbf{H}_{ij}^\infty(\mathbf{a})$ satisfies the modulus constraint (i.e., all its eigenvalues have the same modulus); see [134] for the specific form of ϕ_{ij}. Note that ϕ_{ij} represents a necessary but not sufficient condition on the legality of $\mathbf{a}$: if $\mathbf{H}_{ij}^\infty(\mathbf{a}) \in \mathrm{conhin}(\mathbf{F}_{ij})$ then $\phi_{ij}(\mathbf{a}) = 0$ but the reverse is not necessarily true.

Since $\breve{\mathbf{a}}$ has the property $\phi_{ij}(\breve{\mathbf{a}}) = 0$ for *all* pairs (i, j), $\breve{\mathbf{a}}$ is given by

$$\breve{\mathbf{a}} = \arg\min_{\mathbf{a}\in\mathbb{R}^3} \sum_{(i,j)} (\phi_{ij}(\mathbf{a}))^2 \tag{4.10}$$

If n is large enough ($n \geq 4$), Eq. 4.10 has a single solution except for some special collections of views called *critical motion sequences* (e.g., see [42, 173]). When $n = 3$ there are at most

<u>ALGORITHM D</u>

(1) If $i = 1$, then Eq. 4.7 can be used to find $\mathbf{a}$. For instance, let $\mathbf{E}_1 = [\mathbf{e}_j\ \mathbf{0}\ \mathbf{0}]$, $\mathbf{E}_2 = [\mathbf{0}\ \mathbf{e}_j\ \mathbf{0}]$, and $\mathbf{E}_3 = [\mathbf{0}\ \mathbf{0}\ \mathbf{e}_j]$, then solve

$$[\, \mathbf{H}^{\infty}_{1j}\ \mathbf{H}_j\ \mathbf{E}_1\ \mathbf{E}_2\ \mathbf{E}_3 \,][\, \sigma\ 1\ \mathbf{a}^x\ \mathbf{a}^y\ \mathbf{a}^z \,]^{\top} = \mathbf{0}.$$

where the 3×3 matrices ($\mathbf{H}^{\infty}_{1j}$, $\mathbf{H}_j$, etc.) are treated as column vectors in $\mathbb{R}^9$. Since the null eigenvector will be found up to a scale factor, divide by its *second* component to get the correct $\mathbf{a} = (\mathbf{a}^x, \mathbf{a}^y, \mathbf{a}^z)$. The algorithm is done.

(2) Otherwise $i \neq 1$ and Eq. 4.8 must be solved for $\mathbf{a}$; the remaining steps of the algorithm are for this task.

(3) Define $\mathbf{q}_i$ and $\mathbf{m}_i$ by $[\, \mathbf{q}_1\ \mathbf{q}_2\ \mathbf{q}_3 \,] = \mathbf{H}^{\infty}_{ij}\mathbf{H}_i$ and $[\, \mathbf{m}_1\ \mathbf{m}_2\ \mathbf{m}_3 \,] = \mathbf{H}_j$. Let $\mathbf{v}_1 = \mathbf{m}_1 \times \mathbf{m}_2$, $\mathbf{v}_2 = \mathbf{q}_1 \times \mathbf{m}_2 + \mathbf{m}_1 \times \mathbf{q}_2$, and $\mathbf{v}_3 = \mathbf{q}_1 \times \mathbf{q}_2$.

(4) Solve $\phi^2\mathbf{v}_1 + \phi\mathbf{v}_2 + \mathbf{v}_3 = \mathbf{0}$ for the scalar ϕ. One closed-form solution is $\phi = -(\mathbf{v}^x_3\mathbf{v}^y_1 - \mathbf{v}^y_3\mathbf{v}^x_1)/(\mathbf{v}^x_2\mathbf{v}^y_1 - \mathbf{v}^y_2\mathbf{v}^x_1)$. A least-squares approach is preferable.

(5) The equation to solve is now $\mathbf{U} + \mathbf{w}\mathbf{a}^{\top} = \mathbf{0}$, where $\mathbf{U} = \mathbf{H}^{\infty}_{ij}\mathbf{H}_i + \phi\mathbf{H}_j$ and $\mathbf{w} = \mathbf{H}^{\infty}_{ij}\mathbf{e}_i + \phi\mathbf{e}_j$. This can be solved as in step (1):

$$[\, \mathbf{U}\ \mathbf{W}_1\ \mathbf{W}_2\ \mathbf{W}_3 \,][\, 1\ \mathbf{a}^x\ \mathbf{a}^y\ \mathbf{a}^z \,]^{\top} = \mathbf{0}.$$

Figure 4.5 Algorithm for determining $\mathbf{a}$ from $\mathbf{H}^{\infty}_{ij}$ for view pair (i, j) with $i < j$.

21 solutions as mentioned in Section 4.3.2. Eq. 4.10 can be solved using nonlinear minimization techniques.

4.3.4 Modulus-constraint manifolds

When all the cameras can be placed in the same projective basis (the "initial projective reconstruction" of Section 4.3.3), a simpler form of screw-transform manifold can be used. The key advantage to this alternative form is that γ is no longer needed in the parameterization. Rather than being 3-dimensional manifolds in a 5-dimensional space like the Kruppa-constraint manifolds, these new manifolds are only 2-dimensional and exist in a 3-dimensional space.

Once all the cameras have been placed in the same projective basis, we can define the function $\mathbf{H}_{ij}^{\infty}(\mathbf{a})$ from the previous section. Note that if $\mathbf{H}_{ij}^{\infty}(\mathbf{a})$ is given for some unknown $\mathbf{a}$, then $\mathbf{a}$ can be determined using Algorithm D (Fig. 4.5); let $\mathrm{ave} : \mathbf{H}_{ij}^{\infty}(\mathbf{a}) \longmapsto \mathbf{a}$ denote this mapping. A step-by-step derivation of Algorithm D is given in Appendix A.2. Define the set of "a" vectors consistent with fundamental matrix $\mathbf{F}$ as

$$\mathrm{conave}(\mathbf{F}) = \{\mathrm{ave}(\mathbf{H}) : \mathbf{H} \in \mathrm{conhin}(\mathbf{F})\}$$

Remember that $\mathbf{H}_{ij}^{\infty}(\breve{\mathbf{a}})$ is the relative calibration between views i and j. That is, if the screw decomposition of the motion between views i and j is $S_{ij} = (\mathbf{K}_{ij}, \mathbf{R}_{ij}, \theta_{ij}, \gamma_{ij})$ then $\mathbf{H}_{ij}^{\infty}(\breve{\mathbf{a}}) = \mathrm{hin}(S_{ij})$ up to a scale factor. Thus $\mathbf{H}_{ij}^{\infty}(\breve{\mathbf{a}}) \in \mathrm{conhin}(\mathbf{F}_{ij})$ and $\breve{\mathbf{a}} \in \mathrm{conave}(\mathbf{F}_{ij})$. This provides a method for restricting the location of $\breve{\mathbf{a}}$: enumerate over all possible relative calibrations that are consistent with fundamental matrix $\mathbf{F}_{ij}$ and look at all corresponding vectors $\mathbf{a}$; $\breve{\mathbf{a}}$ must be among this set.

We now formally name the set introduced above. In Section 4.2 it was discussed how $\mathrm{conhin}(\mathbf{F})$ can be parameterized by two real numbers using the mapping $\Lambda_{\mathbf{F}}$. Define the mapping

$$\begin{aligned} \Psi_{\mathbf{F}} \ &: \ \mathbb{R}^2 \longrightarrow \mathrm{conave}(\mathbf{F}) \subseteq \mathbb{R}^3 \\ &\quad (\kappa, \theta) \longmapsto \mathrm{ave}(\Lambda_{\mathbf{F}}(\kappa, \theta)) \end{aligned}$$

CLASSIFICATION OF PAIRWISE CAMERA MOTIONS			
θ	γ	SCREW AXIS LOCATION	CLASSIFICATION
0	0		no motion
0	$\neq 0$		pure translation
$\neq 0$	0	through optical center	unifocal motion
$\neq 0$	0	not through optical center	turntable motion
$\neq 0$	$\neq 0$	through optical center	transfocal motion
$\neq 0$	$\neq 0$	not through optical center	general motion

Table 4.1 Classification of pairwise camera motions.

The *modulus-constraint manifold* is defined as the set $\Psi_{\mathbf{F}}(\mathbb{R}^2)$ together with the parametrization given by $\Psi_{\mathbf{F}}$. As with the Kruppa-constraint manifold, we will not attempt to formally prove that this set and its parameterization form a manifold in the mathematical sense but rather we use the term descriptively. See Fig. 5.1 for a visualization of several modulus-constraint manifolds; the term *a-space* refers to $\mathbb{R}^3$ as the range space of $\Psi_{\mathbf{F}}$.

Since $\breve{\mathbf{a}} \in \Psi_{\mathbf{F}_{ij}}(\mathbb{R}^2)$, self calibration with modulus-constraint manifolds can be achieved just like self calibration with Kruppa-constraint manifolds, by finding the mutual intersection point of a sufficient number of manifolds. Since each modulus-constraint manifold is 2-dimensional and exists in a 3-dimensional space, each manifold places 1 constraint on the location of $\breve{\mathbf{a}}$. Thus three or more manifolds are necessary to restrict $\breve{\mathbf{a}}$ to a 0-dimensional set (i.e., a disjoint set of points).

4.4 Non-general camera motions

In this section, we use screw decomposition to categorize every possible pairwise motion of a camera. Of the 6 categories that arise, the cases called “general motion,” “turntable motion,” and “transfocal motion” have screw-transform manifolds associated with them. The “general motion” case was presented in Section 4.3 and the remaining two cases are discussed here. We conclude this section with a theorem that provides a simple test for differentiating between each motion case.

4.4.1 Classifying pairwise camera motions

A *pairwise camera motion* is a rigid-body transformation[1] that takes a camera from one position and orientation to another; the camera's internal parameters are unchanged. Table 4.1 shows the partition of pairwise camera motions into categories based on screw decomposition. Three elements of the decomposition are used: the amount of rotation θ, the amount of translation γ, and the location of the screw axis. With each element, a binary question is resolved: First it is determined whether θ and γ are nonzero, indicating whether there is any rotation or translation. Then it is determined whether the screw axis intersects the optical center (of both camera views) or not.

Many of the categories given in Table 4.1 have been previously labeled and investigated in the context of self calibration by other authors, which indicates that the partition is natural. Translational motion was one of the first cases studied because of its simplicity and its use in affine reconstruction [121]. The case we call "unifocal motion," meaning rotation of the camera around its optical center, has been studied extensively because of its use in mosaicing. For example, Hartley [68] showed how metric self calibration can be achieved from unifocal motion, and several authors (e.g., McMillan [117] and Davis [28]) have used unifocal motion to self-calibrate simplified camera models when creating mosaics. Turntable motion has been investigated for self calibration in the context of actual turntables [52, 105] and arising from planar motion [12]. To the best of our knowledge, the case we call "transfocal motion" has not been specifically identified and investigated before.

Besides the trivial case of no motion, all remaining motions are termed "general." Most self-calibration papers (e.g., [41, 133, 181]) only require that the camera change location and rotate, and thus these papers simultaneously cover general motion, turntable motion, and transfocal motion. We show later in this section that transfocal motion contains less information for calibration

[1] Arbitrary rigid-body transformations are also called "displacements" in the terminology of kinematics [17].

than the other two cases, which may make it a degenerate case for some of the earlier calibration methods. Some collections of camera motions called *critical motion sequences* [173] do not contain enough information to allow a metric coordinate frame to be uniquely identified. Critical motion sequences exist outside of the motion categories given here; for example, a critical motion sequence might consist entirely of general motions or turntable motions or a combination of the two.

4.4.2 Turntable motion

Turntable motion arises when there is no translational component to the screw decomposition and the screw axis does not intersect the optical center. In this case, $\gamma = 0$ and $\theta \neq 0$ and the fundamental matrix formula from Eq. 4.1 becomes

$$\mathbf{F} = \mathbf{F}^{\mathsf{A}} + \mathbf{F}^{\mathsf{S}} \tag{4.11}$$

$$\mathbf{F}^{\mathsf{A}} = [\sin\theta\mathbf{h}_2]_\times \tag{4.12}$$

$$\mathbf{F}^{\mathsf{S}} = \frac{1-\cos\theta}{|\mathbf{H}|}\Big[(\mathbf{h}_1\times\mathbf{h}_3)(\mathbf{h}_1\times\mathbf{h}_2)^\top + (\mathbf{h}_1\times\mathbf{h}_2)(\mathbf{h}_1\times\mathbf{h}_3)^\top\Big] \tag{4.13}$$

Note that Eq. 4.1 is still applicable (i.e., the derivation given in Section A.1.2 is still applicable) because we are assuming that the screw axis does not intersect the optical center; this will not be true for the transfocal motion case discussed in Section 4.4.3. Also note that Eq. 4.11 was published earlier by Fitzgibbon and Zisserman [52] in the context of self-calibration from turntable motion, where it was given without proof, and this is the earliest reference to the equation of which we are aware.

The most obvious way for turntable motion to occur is when a stationary camera views an object on a rotating turntable. It will also occur when a camera is translated parallel to a plane while rotating around an axis perpendicular to the plane [12].

As in the case of general motion discussed in Section 4.3, two screw-transform manifolds arise from turntable motion, one suitable for direct self calibration and the other for stratified self calibration. Simply let $\Omega_{\mathbf{F}}$ be defined by Algorithm A–2 (Fig. 4.6) and let $\Lambda_{\mathbf{F}}$ be defined by Algorithm B–2 (Fig. 4.7). Note that γ is replaced by a symbolic variable γ' in these parameterizations because

<u>ALGORITHM A–2</u>

(1) Let $\mathbf{m}$ be given by $[\mathbf{m}]_\times = (\mathbf{F} - \mathbf{F}^\top)/2 = \mathbf{F}^{\mathsf{A}}$.

(2) Let $\mathbf{h}_2 = \mathbf{m}/\sin\theta$.

(3) Let $\mathbf{l}_{12} = -(\mathbf{F}^{\mathsf{S}}\mathbf{m})/((1-\cos\theta)\sin\theta)$.

(4) Use $\mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_1 = 0$ to find $\underline{\mathbf{h}}_1$ (i.e., find the null eigenvector of $\mathbf{F}^{\mathsf{S}}$).

(5) Let $\mathbf{h}_1 = (\|\mathbf{l}_{12}\| \,/\, \|\underline{\mathbf{h}}_1 \times \mathbf{h}_2\|)\underline{\mathbf{h}}_1$.

(6) Find $\mathbf{l}_{13}$ from the fact that $(\mathbf{F}^{\mathsf{S}} - (1-\cos\theta)[\mathbf{h}_1]_\times)$ is $[\, \mathbf{l}_{13}\; \mathbf{l}_{13}\; \mathbf{l}_{13} \,]^\top$ up to arbitrary scale factors on the rows.

(7) Let $\underline{\mathbf{h}}_3 = \mathbf{R}(\mathbf{l}_{13}, \kappa)\mathbf{h}_1$ where $\mathbf{R}(\mathbf{n}, k)$ denotes the rotation matrix with rotation axis $\mathbf{n}$ and angle of rotation $2\pi k$.

(8) Let $\mathbf{h}_3 = \gamma\underline{\mathbf{h}}_3$. Note that γ is not related to the amount of screw translation in this case.

(9) Since $\mathbf{KR} = \mathbf{H} = [\mathbf{h}_1\mathbf{h}_2\mathbf{h}_3]$, perform QR decomposition on $\mathbf{H}$ to recover $\mathbf{K}$ and $\mathbf{R}$.

Figure 4.6 Algorithm for mapping $(\mathbf{F}, \kappa, \theta, \gamma')$ to a scenario $S = (\mathbf{K}, \mathbf{R}, \theta, 0)$ that is consistent with monocular fundamental matrix $\mathbf{F}$ arising from turntable motion.

<u>ALGORITHM B–2</u>

(1) Perform steps (1)-(7) of Algorithm A–2 (Fig. 4.6) to find $\underline{\mathbf{h}}_3$ and $\underline{\mathbf{l}}_{12} \cong \underline{\mathbf{h}}_1 \times \underline{\mathbf{h}}_2$.

(2) Use Algorithm C (Fig. 4.4) to find $\mathbf{H}^\infty$.

Figure 4.7 Algorithm for mapping $(\mathbf{F}, \kappa, \theta)$ to $\mathbf{H}^\infty = \mathrm{hin}(S)$ for some scenario $S = (\mathbf{K}, \mathbf{R}, \theta, 0)$ that is consistent with the $\mathbf{F}$. Turntable motion is assumed.

γ is assumed to be 0. The new parameter γ' is no longer related to the screw decomposition; it simply provides the unknown scaling factor for $\underline{\mathbf{h}}_3$. Other than this, γ' can be treated like γ or any other parameter in the the algorithms of Section 4.3.

4.4.3 Transfocal motion

Transfocal motion arises when the screw axis intersects the optical center and $\gamma \neq 0$. The position of the screw axis means that the rising-turntable model must be changed. In particular, it can no longer be assumed that the first camera's optical center is at $(1, 0, 0)^\top$. Instead, we take the first camera to be at $(0, 0, 0)^\top$ and the second to be at $(0, 0, 1)^\top$. Note that we assume the amount of screw translation is 1. Any other value would simply lead to a different overall scaling factor for the internal calibration and scene reconstruction, and since self-calibration can only be achieved up to an overall scaling factor it is sufficient to assume the screw translation is 1. Thus γ will not play a role in the equations arising from transfocal motion.

We use the notation $S = (\mathbf{K}, \mathbf{R}, \theta)$ to describe a transfocal motion scenario S. This notation is just like the notation of Section 4.1 except without γ. The fundamental matrix $\mathrm{fma}(S)$ corresponding to S is

$$\begin{aligned} \mathbf{F} &= \mathbf{F}^{\mathsf{A}} + \mathbf{F}^{\mathsf{S}} && (4.14) \\ \mathbf{F}^{\mathsf{A}} &= [\cot\theta \mathbf{h}_3]_\times && (4.15) \\ \mathbf{F}^{\mathsf{S}} &= \frac{1}{|\mathbf{H}|}\Big[(\mathbf{h}_1 \times \mathbf{h}_3)(\mathbf{h}_1 \times \mathbf{h}_3)^\top + (\mathbf{h}_2 \times \mathbf{h}_3)(\mathbf{h}_2 \times \mathbf{h}_3)^\top\Big] && (4.16) \end{aligned}$$

One way to derive Eqs. 4.14–4.16 is by taking the limit of Eq. 4.1 scaled by $1/\gamma$ as $\gamma \to \infty$. This works because in a rising-turntable scenario γ measures the amount of screw translation as a multiple of the distance between the optical center and the screw axis. Thus making γ grow is equivalent to shrinking the distance between the optical center and the screw axis. An alternative, direct derivation is given in Section A.2.3.

The framework for describing the screw-transform manifolds associated with transfocal motion is just like that for general and turntable motion. However, an extra parameter is needed for

<u>ALGORITHM A–3</u>

(1) Let $\underline{\mathbf{h}}_3$ be given by $[\underline{\mathbf{h}}_3]_\times = \mathbf{F}^{\mathsf{A}}$.

(2) Deterministically pick an arbitrary line through $\underline{\mathbf{h}}_3$ to serve as the line $\underline{\mathbf{l}}_{13} \cong \mathbf{h}_1 \times \mathbf{h}_3$. For example, if $\underline{\mathbf{h}}_3 = [x, y, z]^\top$ then let $\underline{\mathbf{l}}_{13} = [-y, x, 0]^\top$.

(3) Let $\underline{\mathbf{h}}_1 = \mathbf{R}(\underline{\mathbf{l}}_{13}, \kappa_1)\underline{\mathbf{h}}_3$ where $\mathbf{R}(\mathbf{n}, k)$ denotes the rotation matrix with rotation axis $\mathbf{n}$ and angle of rotation $2\pi k$.

(4) Let $\underline{\mathbf{l}}_{23} = \mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_1$

(5) Let $\underline{\mathbf{h}}_2 = \mathbf{R}(\underline{\mathbf{l}}_{23}, \kappa_2)\underline{\mathbf{h}}_3$.

(6) Let $\mathbf{h}_3 = \tan\theta\underline{\mathbf{h}}_3$.

(7) Let $\phi = (\mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_1)^\top(\underline{\mathbf{h}}_2 \times \mathbf{h}_3)/\left\|\underline{\mathbf{h}}_2 \times \mathbf{h}_3\right\|^2$.

(8) Let $\mathbf{h}_1 = \gamma'\underline{\mathbf{h}}_1$.

(9) Let $\mathbf{h}_2 = \gamma'\phi\underline{\mathbf{h}}_2$.

Figure 4.8 Algorithm for mapping $(\mathbf{F}, \kappa_1, \kappa_2, \theta, \gamma')$ to a transfocal motion scenario $S = (\mathbf{K}, \mathbf{R}, \theta)$ that is consistent with $\mathbf{F}$.

<u>ALGORITHM B–3</u>

(1) Perform steps (1)-(5) of Algorithm A–3 (Fig. 4.8) to find $\underline{\mathbf{h}}_3$ and $\underline{\mathbf{l}}_{12} \cong \underline{\mathbf{h}}_1 \times \underline{\mathbf{h}}_2$.

(2) Use Algorithm C (Fig. 4.4) to find $\mathbf{H}^\infty$. Note θ is used only in this step.

Figure 4.9 Algorithm for mapping $(\mathbf{F}, \kappa_1, \kappa_2, \theta)$ to $\mathbf{H}^\infty = \operatorname{hin}(S)$ for some transfocal motion scenario $S = (\mathbf{K}, \mathbf{R}, \theta)$ that is consistent with $\mathbf{F}$.

each type of manifold. Let $\Omega_{\mathbf{F}}$ be defined by Algorithm A–3 (Fig. 4.8) and let $\Lambda_{\mathbf{F}}$ be defined by Algorithm B–3 (Fig. 4.9).

With these definitions, $\Omega_{\mathbf{F}}$ leads to a 4-dimensional manifold in the 5-dimensional search space for $\mathbf{K}$ and $\Lambda_{\mathbf{F}}$ leads to a 3-dimensional manifold in the 3-dimensional search space for $\mathbf{a}$. Thus it is still possible to perform direct self calibration from $\Omega_{\mathbf{F}}$ using transfocal motions, although 5 fundamental matrices are now needed. However, it is unclear whether transfocal motions can be used for stratified self calibration from $\Lambda_{\mathbf{F}}$ since the intersection of several screw-transform manifolds in a-space will still be 3-dimensional. Since each manifold will be smaller than the entire search space (because θ, κ_1, and κ_2 are restricted to the domain $[0, 1]$), the intersection of many manifolds may sufficiently restrict the location of $\breve{\mathbf{a}}$ because, in the presence of noise, $\breve{\mathbf{a}}$ can never be found with complete certainty anyway. The intersection process can be further improved if θ can be restricted to a more limited range.

The intersection algorithms given in Section 5 can still be applied after suitable modifications have been introduced to account for the increased number of parameters.

4.4.4 Tests for classifying pairwise camera motions

Table 4.2 provides a simple test for identifying each class of pairwise camera motion. Such tests are important because they allow a self-calibration algorithm to automatically use the correct screw-transform manifold or alternative calibration method in the cases of translational and unifocal motion. Note that every class of motion has either an $\mathbf{H}^\infty$ matrix or a fundamental matrix associated with it, and the tests are performed directly from these matrices.

Before proving the correctness of the tests (Theorem 1), the concept of *abstract feature points* must be introduced. An abstract feature point is the projected location of a position in space into both camera views as determined by each camera's matrix. Abstract features behave just like real feature points except they exist only as the information they provide; they are invisible and do not even need to be in either camera's field of view. When stating Theorem 1 and its proof, enough information from abstract feature points is assumed to be available (e.g., from an oracle) to perform the tests. The concept of abstract feature points is needed to avoid situations in which the tests might be fooled (e.g., by placing photographs of different views in front of the camera without actually moving the camera) or in which there simply is not enough information to perform the tests (e.g., the camera is in a featureless room, or the two views do not overlap at all). The theorem is thus made correct at all times regardless of what visual information is actually available. In practice, of course, performing the tests requires sufficient visual information and the tests could be fooled or could be impossible to perform.

LEMMA 1. *Let two views of a scene be captured by arbitrary cameras. Then exactly one of the following statements is true:*

(i) The two cameras share the same optical center, there exists a planar homography mapping one view onto the other (i.e., mapping every abstract feature point from one view onto the corresponding abstract feature point in the other), and the views do not induce a fundamental matrix.

(ii) The two cameras have different optical centers, the views induce a fundamental matrix, and there does not exist a planar homography mapping one view onto the other.

Proof: Either the cameras share the same optical center or they do not. In the former case, the planar homography mapping the first view onto the second is given by $\mathbf{H}^{\infty} = \mathbf{K}'\mathbf{R}\mathbf{K}^{-1}$, where $\mathbf{K}$ and $\mathbf{K}'$ are the internal calibration matrices for the two cameras and $\mathbf{R}$ is a rotation matrix (given by the screw decomposition). In the latter case, the fundamental matrix is defined as $[\mathbf{K}'\mathbf{e}]_{\times}\mathbf{H}^{\infty}$ where $\mathbf{e}$ is the displacement vector between the two optical centers expressed in the same metric coordinate system used to express $\mathbf{K}'$. If the optical centers are the same, $\mathbf{e} = 0$ and the fundamental matrix is not defined, proving *(i)* implies $\neg$*(ii)*. Now assume that the optical centers differ but there exists a planar homography $\mathbf{H}$ that maps all abstract feature points from one view into the other. Let A and B denote the optical centers of the two cameras and let C be the spacial location of some abstract feature point. Let D be another point on the line $\overline{AC}$, and use the notation p and p' to denote the projection of spacial point P into the first and second views, respectively. Note that $c = d$ in the first view but $c' \neq d'$ in the second view, which contradicts the existence of planar homography $\mathbf{H}$ that maps c to c' and d to d'. Thus the existence of $\mathbf{H}$ implies that both optical centers are at the same location, proving *(ii)* implies $\neg$*(i)*. $\square$

LEMMA 2. *Every fundamental matrix has rank 2.*

Proof: A fundamental matrix is defined as $[\mathrm{e}]_\times \mathbf{H}^\infty$ for some $\mathbf{e}$. $\mathbf{H}^\infty$ is invertible, so it has rank 3, and $[\mathbf{e}]_\times$ represents a cross-product operation, so it has rank 2. $\square$

LEMMA 3. *If two views are captured by a camera with fixed internal parameters undergoing either turntable or general motion and if the underlying screw rotation θ is not a multiple of π, then the motion is turntable if and only if* $\det(\mathbf{F}^{\mathsf{S}}) = 0$.

Proof: Call the two cameras involved camera A and camera B. Let $\mathbf{e}_A$ and $\mathbf{e}_B$ denote the epipole in camera A and camera B, respectively. Clearly if $\gamma = 0$ then $\det(\mathbf{F}^{\mathsf{S}}) = 0$ because $\mathbf{h}_1$ is a null eigenvector of $\mathbf{F}^{\mathsf{S}}$ (examine Eq. 4.3). Now assume $\det(\mathbf{F}^{\mathsf{S}}) = 0$ but $\gamma \neq 0$. In the logic below we will be using a fixed-camera, rising-turntable formulation [104] with the optical center at the origin of $\mathbb{R}^3$ (so there will only be one camera, fixed in position at the origin, viewing a scene that undergoes a screw transformation; camera A will denote what the camera views before the transformation and camera B will denote what the camera views after the transformation). Also, for vectors $\mathbf{f}, \mathbf{g} \in \mathbb{R}^3$, $\langle \mathbf{f}, \mathbf{g} \rangle$ will denote both the space spanned by $\mathbf{f}$ and $\mathbf{g}$ and the line on the image plane induced by this space. Let $\mathbf{u} \in \mathbb{R}^3$ be a scene point whose projection into camera B, denoted by $\mathbf{u}_B$, is in the null space of $\mathbf{F}^{\mathsf{S}}$ (i.e., $\mathbf{F}^{\mathsf{S}}\mathbf{u}_B = \mathbf{0}$). Using Eq. 4.3, $\mathbf{F}^{\mathsf{S}}\mathbf{h}_3 \cong \mathbf{h}_1 \times \mathbf{h}_3 \neq \mathbf{0}$ so $\mathbf{h}_3$ corresponds to a different position in view B than $\mathbf{u}_B$. Since the epipolar line for $\mathbf{u}_A$ in view A is represented by both $\langle \mathbf{u}_A, \mathbf{e}_A \rangle$ and by $\mathbf{u}_B^\top \mathbf{F} = \mathbf{u}_B^\top \mathbf{F}^{\mathsf{S}} + \mathbf{u}_B^\top \mathbf{F}^{\mathsf{A}} = \mathbf{u}_B^\top \mathbf{F}^{\mathsf{A}} = (\mathbf{m} \times \mathbf{u}_B)^\top$, we conclude $\mathbf{m}$, $\mathbf{u}_B$, $\mathbf{u}_A$, and $\mathbf{e}_A$ are all coplanar. Let $\mathbf{q} \in \mathbb{R}^3$ be an arbitrary scene point that projects to the line $\langle \mathbf{u}_B, \mathbf{e}_A \rangle$ in view A (which equals the line $\langle \mathbf{u}_A, \mathbf{e}_A \rangle$). So $\mathbf{q}_A$ is a linear combination of $\mathbf{u}_B$ and $\mathbf{e}_A$ and thus $\mathbf{F}\mathbf{q}_A \cong \mathbf{F}\mathbf{u}_B \cong \mathbf{m} \times \mathbf{u}_B$, implying $\mathbf{q}_B$ lies in the plane $\langle \mathbf{m}, \mathbf{u}_B \rangle$ which is also the plane $\langle \mathbf{u}_A, \mathbf{e}_A \rangle$. In other words, points that project to the line $\langle \mathbf{u}_A, \mathbf{e}_A \rangle$ in view A also project to that line in view B (after the screw transformation). If the planes $\langle \mathbf{u}_A, \mathbf{e}_A \rangle$ and $\langle \mathbf{h}_1, \mathbf{h}_3 \rangle$ are identical then $\mathbf{e}_A$ lies on the rotation axis $\langle \mathbf{h}_1, \mathbf{h}_3 \rangle$, which can only happen if θ is a multiple of π (by Eq. A.11).

Since we assume this is not the case, $\langle \mathbf{u}_A, \mathbf{e}_A \rangle$ and $\langle \mathbf{h}_1, \mathbf{h}_3 \rangle$ represent distinct lines in view A which intersect at a point $\mathbf{v}_A$. For some scale factor k, $\mathbf{v} = k\mathbf{v}_A$ is a point on the rotation axis in space. $\mathbf{v}_B$ is the projection of $\mathbf{v}$ after the screw motion; in this case, the screw motion translates $\mathbf{v}$ by γ along the screw axis, so under the fixed-camera formulation $\mathbf{v}_B$ and $\mathbf{v}_A$ are at different positions on the axis line $\langle \mathbf{h}_1, \mathbf{h}_3 \rangle$. But $\mathbf{v}_A$ is on the line $\langle \mathbf{u}_A, \mathbf{e}_A \rangle$ and so $\mathbf{v}_B$ must also be on this line, leading to the conclusion that $\mathbf{v}_B = \mathbf{v}_A$, a contradiction. □

THEOREM 1. *Let two views of a scene be captured by a camera with fixed internal parameters and assume $\theta \neq k\pi$, where θ is the amount of screw rotation and k is an integer. Let $\mathbf{H}$ denote the homography between the views if it exists and otherwise let $\mathbf{F}$ denote the fundamental matrix induced by the views and let $\mathbf{e}$ denote the left epipole of $\mathbf{F}$ (so that $\mathbf{e}^\top \mathbf{F} = 0$). Then exactly one of the following statements is true:*

(i) $\mathbf{H}$ *exists and is the identity matrix, the views are identical, and the camera underwent no motion.*

(ii) $\mathbf{H}$ *exists and is not equal to the identity matrix and the camera underwent unifocal motion (i.e., the camera rotated around its optical center).*

(iii) $\mathbf{F}$ *exists,* $\mathbf{F} \cong [\mathbf{e}]_\times$ *(i.e.,* $\mathbf{F}$ *is antisymmetric with 0's on the main diagonal), and the camera underwent translational motion.*

(iv) $\mathbf{F}$ *exists,* $\mathbf{F}^S \neq 0$ *and* $\mathbf{F}^S\mathbf{e} = 0$*, and the camera underwent transfocal motion.*

(v) $\mathbf{F}$ *exists,* $\mathbf{F}^S\mathbf{e} \neq 0$ *and* $\det(\mathbf{F}^S) = 0$*, and the camera underwent turntable motion.*

(vi) $\mathbf{F}$ *exists,* $\det(\mathbf{F}^S) \neq 0$*, and the camera underwent general motion.*

Proof: First observe that the statement of the theorem is well-defined because by Lemma 1 either $\mathbf{H}$ exists or $\mathbf{F}$ exists but not both. We know that the motion classes given in Table 4.1 partition the set of all pairwise camera motions since each class is defined by a series of binary tests (i.e., either $\theta = 0$ or $\theta \neq 0$, etc.). Thus only two claims need to be proven: (1) the conditions associated with motion class X hold when the underlying motion is in class X, and (2) the conditions associated with distinct motion classes X and Y cannot hold simultaneously.

Proof of claim (1): If there is no motion, then clearly the views are identical, the optical centers are equal so that $\mathbf{H}$ exists, and $\mathbf{H} = \mathbf{I}$. If there is unifocal motion, then $\mathbf{H}$ exists by Lemma 1 and $\mathbf{H} \neq \mathbf{I}$ because $\theta \neq 0$. If there is translational motion, then $\theta = 0$ and, by Eq. 4.1, $\mathbf{F} \cong [\mathbf{h}_3]_\times$. Here $\mathbf{h}_3$ equals the left epipole $\mathbf{e}$. If there is transfocal motion, Eq. 4.14 shows that $\mathbf{e} \cong \mathbf{h}_3$ and thus that $\mathbf{F}^{\mathsf{S}}\mathbf{e} = \mathbf{F}^{\mathsf{S}}\mathbf{h}_3 = 0$. Furthermore, $\mathbf{F}^{\mathsf{S}} \neq 0$ because $\mathbf{F}^{\mathsf{S}}\mathbf{h}_2 \cong \mathbf{h}_1 \times \mathbf{h}_3$ and $\mathbf{h}_1$ and $\mathbf{h}_3$ are linearly independent since the internal calibration matrix $\mathbf{K}$ is invertible. If there is turntable or general motion then Lemma 3 establishes that $\det(\mathbf{F}^{\mathsf{S}}) = 0$ for turntable motion and $\det(\mathbf{F}^{\mathsf{S}}) \neq 0$ for general motion (note that the condition on θ is necessary to use Lemma 3). Furthermore, if there is turntable motion then $\mathbf{e} = -(1 - \cos\theta)\mathbf{h}_1 - \sin\theta\mathbf{h}_2$ by Eq. A.11 with $\gamma = 0$, leading to $\mathbf{F}^{\mathsf{S}}\mathbf{e} = (1 - \cos\theta)\sin\theta|\mathbf{H}|^{-1}(\mathbf{h}_1 \times \mathbf{h}_2)$ by Eq. A.6 and Eq. A.7 with $\gamma = 0$. Thus $\mathbf{F}^{\mathsf{S}}\mathbf{e} \neq 0$ since $\mathbf{h}_1$ and $\mathbf{h}_2$ are linearly independent and θ is not a multiple of π.

Proof of claim (2): Let the notation $\neg$*(y)* indicate that statement *(y)* cannot hold; then it is sufficient to show that if the conditions of statement *(y)* are met then $\neg$*(z)* for every $z > y$. If $\mathbf{H}$ exists then $\neg$*(iii)*, $\neg$*(iv)*, $\neg$*(v)*, and $\neg$*(vi)*. Furthermore, if $\mathbf{H} = \mathbf{I}$ as in statement *(i)* then $\neg$*(ii)*. Now assume $\mathbf{F}$ exists. If $\mathbf{F} \cong [\mathbf{e}]_\times$ then $\mathbf{F}^{\mathsf{S}} = \mathbf{0}$; this follows because $[\mathbf{e}]_\times \cong \mathbf{F} = \mathbf{F}^{\mathsf{A}} + \mathbf{F}^{\mathsf{S}}$ where $\mathbf{F}^{\mathsf{S}}$ is symmetric and $\mathbf{F}^{\mathsf{A}}$ and $[\mathbf{e}]_\times$ are both antisymmetric with 0's on the main diagonal. Thus having $\mathbf{F} \cong [\mathbf{e}]_\times$ as in statement *(iii)* implies $\neg$*(iv)*, $\neg$*(v)*, and $\neg$*(vi)*. If $\mathbf{F}^{\mathsf{S}}\mathbf{e} = 0$ as in statement *(iv)*, then $\neg$*(v)* and $\neg$*(vi)* immediately follow (the latter because $\det(\mathbf{F}^{\mathsf{S}}) = 0$). Finally, if $\det(\mathbf{F}^{\mathsf{S}}) = 0$ as in statement *(v)* then $\neg$*(vi)*. $\square$

TESTS FOR CLASSIFYING PAIRWISE CAMERA MOTIONS				
CLASSIFICATION	TEST	DEGREES OF FREEDOM		NOTES
		AFFINE	METRIC	
no motion	views identical	0	5	$\mathbf{H}^{\infty} = \mathbf{I}$
pure translation	$\mathbf{F}$ is a cross-product matrix	0	5	$\mathbf{H}^{\infty} = \mathbf{I}$
unifocal motion	homography exists between views	0	2	$\mathbf{F}$ is not defined; $\mathbf{H}^{\infty}$ is the homography between views
turntable motion	$\det(\mathbf{F}^{\mathsf{S}}) = 0$ and $\mathbf{F}^{\mathsf{S}}\mathbf{e} \neq \mathbf{0}$	2	3	
transfocal motion	$\mathbf{F}^{\mathsf{S}} \neq \mathbf{0}$ and $\mathbf{F}^{\mathsf{S}}\mathbf{e} = \mathbf{0}$	3	4	
general motion	$\det(\mathbf{F}^{\mathsf{S}}) \neq 0$	2	3	

Table 4.2 Tests for determining in which category a given pairwise camera motion belongs, using only point correspondences between the views.

Chapter 5

Manifold Intersection Algorithms

As was discussed in Section 4.3.2 and Section 4.3.4, self-calibration can be achieved by finding the mutual intersection point of three or more screw-transform manifolds.[1] In the case of Kruppa-constraint manifolds, the mutual intersection point is the internal calibration matrix $\mathbf{K}$ up to an unknown scale factor. Finding the mutual intersection point of a set of Kruppa-constraint manifolds represents a direct self-calibration technique, meaning calibration is determined immediately from fundamental matrices generated by pairs of views without creating an initial projective reconstruction of the cameras or upgrading to an affine reconstruction.

In the case of modulus-constraint manifolds, the mutual intersection point is the vector $\breve{\mathbf{a}} \in \mathbb{R}^3$ which can be used to upgrade an initial projective reconstruction to affine. Once affine camera matrices have been determined, the pairwise relative calibrations are known and upgrading to metric reconstruction (i.e., finding $\mathbf{K}$) is a simple matter. In this way, modulus-constraint manifolds can be used in a stratified self-calibration technique. As with the Kruppa-constraint manifolds, pairwise fundamental matrices are required by this approach; however, an initial projective reconstruction of the cameras is also required (i.e., every camera matrix must be placed in the same projective basis).

In both self-calibration approaches, the mutual intersection of three screw-transform manifolds is a set of disjoint points (except when special, critical-motion sequences are involved). In some cases it is possible to find the set of all mutual intersection points and then eliminate incorrect points using other criteria. Otherwise, additional screw-transform manifolds can be utilized: as more manifolds are incorporated, the set of mutual intersection points grows smaller until only the

[1] It is coincidental that only three manifolds are necessary in both cases

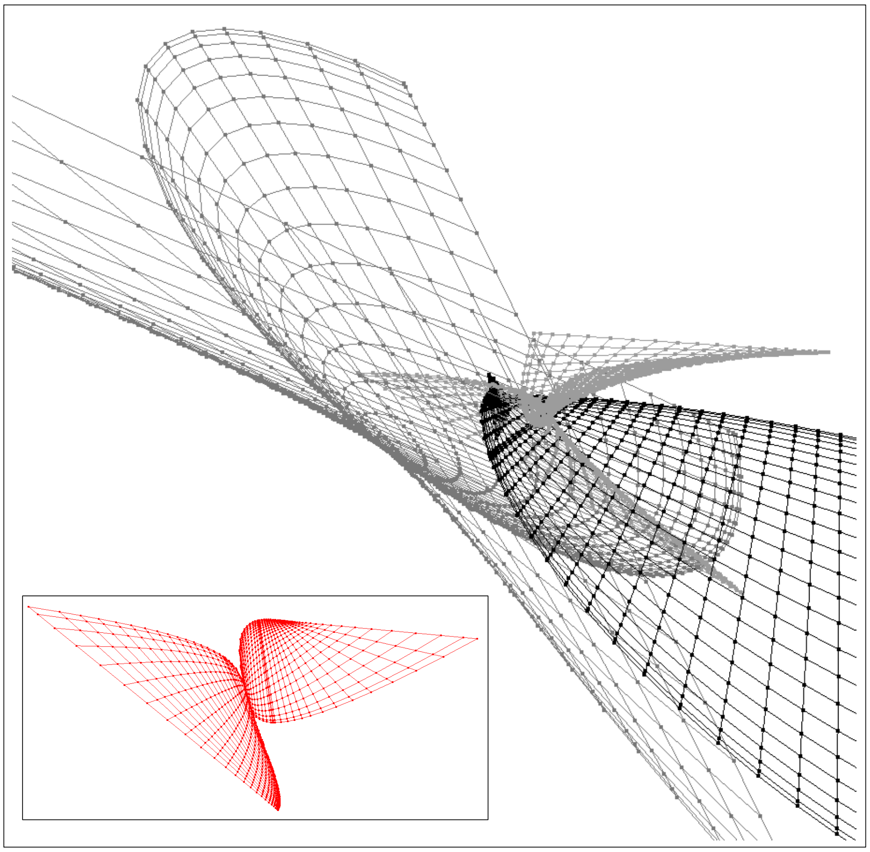

Figure 5.1 Piecewise-linear approximate surfaces fitted to three modulus-constraint manifolds in a-space. Determining the mutual-intersection point ǎ of the manifolds makes it possible to upgrade projective reconstruction to affine, after which metric reconstruction can easily be determined. The inset shows one modulus-constraint manifold by itself; notice the lone discontinuity. The two dimensions of the grid underlying each approximate surface are κ and θ.

correct self-calibration remains. Note, however, that our experiments suggest finding the mutual intersection point of just three modulus-constraint manifolds is usually sufficient to restrict $\breve{\mathbf{a}}$ to a set of one or two points (see Section 6.1.5).

The key task becomes creating an efficient, fast, and robust algorithm for finding the mutual intersection of several manifolds. We have experimented with three approaches: a surface-fitting technique [106], a voting algorithm [104], and a Monte-Carlo Markov-Chain scheme. Each approach is briefly summarized below; more details for the voting algorithm can be found in Appendix B.1.1. Note that each approach can be used with either Kruppa-constraint manifolds or modulus-constraint manifolds; these are general-purpose algorithms that can determine the mutual-intersection points of any set of manifolds. Thus it is irrelevant whether direct or stratified self calibration is being performed.

5.1 Surface fitting

One direct approach to solving the intersection problem is to fit an approximate, piecewise-linear surface to each screw-transform manifold and then find the mutual intersection points of the approximate surfaces. Our method involves imposing a coordinate grid onto each manifold; see Fig. 5.1 for a visualization. First, a feasible range is determined for each underlying screw-transform parameter. For the underlying parameter θ, which represents screw rotation, the range $[-\pi, \pi]$ can be used. For κ and γ, a range bounded by large positive and negative values should be sufficient. Once a finite range for each parameter has been decided, we can treat each parameter as being a real number in the range $[0, 1]$. A grid can then be imposed on each manifold in the obvious way, such as in the following pseudocode for the modulus-constraint manifold case:

```
grid : array[0..m][0..m] of 3-vectors
for i := 0 to m
   for j := 0 to m
      grid[i][j] := psif( theta(i/m), kappa(j/m) )
```

Here **psif** denotes the function $\Psi_{\mathbf{F}}$ and **theta** and **kappa** denote predetermined conversion functions that take $x \in [0, 1]$ to the desired range of θ and κ, respectively. For the Kruppa-constraint

ALGORITHM E (SURFACE FITTING)

Goal: Find all mutual-intersection points of a set $\breve{M}$ of manifolds.

Preconditions: There are only a finite number of mutual intersection points; a parameterization is known for each manifold in $\breve{M}$; a finite search range has been determined for each parameter.

(1) For each manifold in $\breve{M}$, find an approximate, piecewise-linear surface (see Section 5.1). Let M denote the set of approximate surfaces.

(2) Pick 3 surfaces (m_1, m_2, m_3) from M.

(3) Find the set P of mutual intersection points for (m_1, m_2, m_3) (see Section 5.1).

(4) For each $p \in P$, determine how well p fits with all surfaces in M.

(5) Repeat from step (2) a fixed number of times or until the fit is sufficiently good; use standard RANSAC protocols to determine how many loops are necessary [51].

(6) Return the mutual intersection point that fit best with the largest number of surfaces.

Figure 5.2 Manifold intersection algorithm using surface fitting and RANSAC.

manifold the grid array is 3-dimensional and contains 5-vectors, and an additional conversion function **`gamma`** and corresponding inner loop must be used, as well as $\Omega_{\mathbf{F}}$ instead of $\Psi_{\mathbf{F}}$.

Once the grid has been established, each grid square (or cube) can be divided into triangles (or tetrahedrons). For example, the square delimited by **`grid[i,j]`** and **`grid[i+1,j+1]`** becomes two triangles with vertex sets { **`grid[i,j]`**, **`grid[i+1,j]`**, **`grid[i,j+1]`** } and { **`grid[i+1,j+1]`**, **`grid[i+1,j]`**, **`grid[i,j+1]`** }, respectively. The set of triangles (tetrahedrons) forms the desired piecewise-linear surface approximation for the manifold.

With the surface approximations established, consider the case of intersecting just three manifolds. The process has two steps: first all pairs of intersecting triangles for two of the manifolds are determined, and then each of these triangle pairs is tested for intersection with each triangle of the third manifold. In pseudocode:

```
pairwise : set of triangle pairs
pairwise = []
foreach t1 in triangles1
   foreach t2 in triangles2
      if intersect(t1, t2) then
         pairwise := pairwise + [(t1,t2)]

mutual : set of points
mutual = []
foreach t3 in triangles3
   foreach (t1,t2) in pairwise
      if intersect(t1,t2,t3) then
         mutual := mutual + [intersection point of t1, t2, and t3]
```

Here **`trianglesi`** is the set of triangles forming the approximation of manifold **`i`** and **`intersect`** is a Boolean function indicating whether the given triangles intersect. This simple algorithm can be made more efficient with appropriate data structures and through the use of bounding boxes to quickly eliminate many cases where triangles do not intersect.

When more than three manifolds are available, successive loops could be used to further pare down the set **`mutual`**. However, in the presence of noise a set of 4 or more manifolds will not have a well-defined mutual intersection point. Note that if the noise level is small enough, a set of three manifolds will still have true mutual intersection points, albeit at incorrect locations.

Because of the last observation and because it is generally good practice when dealing with noisy data, we make use of extra manifolds (i.e., any manifolds above the minimum three) by combining RANSAC with the triangle-based intersection algorithm given above. The use of RANSAC is feasible because the surface-fitting intersection algorithm runs very quickly (see Section 6.1.3) and because only three manifolds need to be drawn from the set of all manifolds for every random sample, meaning only a small number of draws is necessary to be confident with the result. The complete RANSAC-style, surface-fitting, manifold-intersection algorithm is shown in Fig. 5.2.

In general, the approach outlined in this section works extremely well in practice, as the experiments of Section 6.1 demonstrate. Drawbacks to the algorithm stem from the process of "sketching" each manifold (i.e., fitting an approximate surface) because each grid takes a certain amount of memory to hold and a certain amount of time to sketch. The amount of memory and time required is proportional to the square of the resolution (i.e., the variable `m` from the first pseudocode) for modulus-constraint manifolds and the cube of the resolution for Kruppa-constraint manifolds. Experiments suggest a resolution of 20 or 30 is the bare minimum necessary for usable results; higher resolutions will, of course, produce finer results.

5.2 Voting-based algorithm

In this section, we present a voting scheme [104] for determining the intersection points of several manifolds. The voting scheme can be thought of as a Hough transform [82] (see also [87]) for self calibration. This approach has the same strengths in the presence of noisy data that the Hough transform has: large numbers of inliers will consistently vote for the same, correct answer while small numbers of outliers will vote for random, incorrect answers; thus the correct answer will receive much more support than any other answer. Furthermore, all the available data is used simultaneously, unlike in RANSAC-based approaches where only a minimal subset of the data is considered during any single iteration. One drawback is the large memory requirement for storing the votes. Another is the length of time the algorithm must be run for a reliable winner to emerge from the voting process. Perhaps the most crucial drawback is the coarse resolution for the voting grid, which can cause the wrong area of the search space to appear correct during early stages of the voting algorithm, thus misdirecting the entire search process.

For $1 \leq i \leq n$, let $F_i : \mathbb{R}^m \longrightarrow \mathbb{R}^k$ define an m-dimensional manifold $F_i(\mathbb{R}^m)$ in a k-dimensional space ($m < k$). Define the set of mutual intersection points $Q = \{q \in \mathbb{R}^k : \forall i \, \exists p \in \mathbb{R}^m [q = F_i(p)]\}$. Assume that all n manifolds contain at least one point in common, meaning $|Q| \geq 1$; our goal is to find at least one element of Q. Furthermore, assume that a hypercube $V \subset \mathbb{R}^k$ can be identified that contains at least one of the mutual intersection points. V is the *search region*.

The idea behind the voting scheme is to uniformly divide the search region V into equal-size hypercubes called *voting voxels* and then count how many manifolds intersect each voxel. Any voxel w that contains a member of Q will receive the maximum possible number of votes (i.e., n votes). Voxel w is then used as the new search region U and the entire voting process is repeated with U in place of V. We will refer to this reduction of search space size as a "zoom-in" step. With each iteration the search region gets smaller until a mutual intersection point can be directly determined (e.g., using the linear intersection method discussed later).

In practice, it is not possible to directly count how many manifolds intersect each voting voxel. Instead, the counting process is statistically approximated through voting. Points are randomly generated on each manifold (ideally, forming a uniform sampling across each manifold) and then each voxel that contains one or more points on manifold i receives a vote from manifold i. Note that each voxel is allowed at most one vote from each manifold. The voxel that receives the most votes is labeled w. To be safe, instead of using the volume of voxel w as the next search region, U is taken to be a region half the size of V and centered on w (or on the centroid of the votes received by w and its immediate neighbors). The full algorithm is given in Fig. 5.3.

Sample points are generated on each manifold by randomly selecting underlying coordinates (step (3) of the algorithm). For example, in the case of direct self calibration from Kruppa-constraint manifolds, three real numbers $\kappa, \theta, \gamma \in \mathbb{R}$ are chosen at random (within some pre-determined range) and the corresponding sample point becomes $\Omega_{\mathbf{F}}(\kappa, \gamma, \theta)$. As the algorithm

ALGORITHM F (VOTING)

Goal: Find a member of Q, where Q is the set of all mutual-intersection points of a set M of manifolds. The members of M are m-dimensional manifolds in $\mathbb{R}^k$.

Preconditions: $|Q| \geq 1$; a parameterization is known for each manifold in M; a finite search range has been determined for each parameter.

(1) Choose a volume $V \subset \mathbb{R}^k$ that contains members of Q.

(2) Partition V into equally-sized voting voxels (k-dimensional hypercubes). A finer subdivision slows down the algorithm but improves success. However, because real data contains noise the voxels must be large enough to allow for manifolds that only come close to intersecting.

(3) For each manifold i: Randomly select a point $\mathbf{c} \in \mathbb{R}^m$ to serve as coordinates, calculate the corresponding point $F_i(\mathbf{c}) \in \mathbb{R}^k$ on manifold i , and determine the voxel v containing $F_i(\mathbf{c})$. A vote is cast for v provided manifold i has not already voted for v.

(4) Repeat from step (3) until one voxel w receives enough votes (which will be some fraction of the maximum possible number of votes n).

(5) "Zoom in" step: Define a new volume U half the size of V and centered around the winning voxel w. Return to step (2) using the smaller volume U in place of V. Manifolds that have not yet generated any sample points within the new search region U are eliminated from future consideration.

(6) The algorithm continues until a sufficient resolution has been reached (e.g., enough manifolds are approximately linear within the current search region to use Eq. 5.1).

Figure 5.3 Voting-based manifold intersection algorithm: a Hough transform for self calibration.

continues, the search region becomes smaller and smaller and simply picking parameters at random will not be efficient because the corresponding sample point will probably lie outside the reduced search region. Instead, the underlying parameters are restricted to a range that is likely to correspond to points within the reduced search region. This range is determined from the set of sample points that already lie in the current search region (see the discussion of "range dithering" in Appendix B.1.2). Note that both the process of sampling each manifold based on underlying parameters and of sampling a small region of each manifold by restricting the range for the parameters is made possible by the very existence of the parameterization.

As the search region becomes smaller, the manifolds that continue to intersect it will appear more and more linear within the search region. This is an inherent property of manifolds stemming from the fact that manifolds are smooth and have derivatives. After the search region V has been reduced in size a few times, the search algorithm can attempt to fit a hyperplane to the sample points of each manifold that lie within it. If a good fit can be achieved for enough manifolds then the desired point of intersection can be determined in one step by intersecting the hyperplanes. For example, if t manifolds can be approximated by hyperplanes and if hyperplane i is defined by the normal vectors $\mathbf{n}_1^i, \ldots, \mathbf{n}_{k-m}^i \in \mathbb{R}^k$ and point $\mathbf{p}^i \in \mathbb{R}^k$ lying somewhere on the hyperplane, then the mutual intersection point $\mathbf{q} \in \mathbb{R}^k$ can be determined immediately because $\mathbf{n}_j^i \cdot (\mathbf{q} - \mathbf{p}^i) = 0$ for all i, j. The latter series of equations becomes the linear system:

$$\begin{aligned} &\left[\begin{array}{ccccccc} \mathbf{n}_1^1, & \mathbf{n}_2^1, & \cdots, & \mathbf{n}_{k-m}^1, & \mathbf{n}_1^2, & \cdots, & \mathbf{n}_{k-m}^t \end{array} \right]^\top \mathbf{q} \\ &\qquad = \left[\begin{array}{cccc} \mathbf{n}_1^1 \cdot \mathbf{p}^1, & \mathbf{n}_2^1 \cdot \mathbf{p}^1, & \cdots, & \mathbf{n}_{k-m}^n \cdot \mathbf{p}^t \end{array} \right]^\top \end{aligned} \tag{5.1}$$

During experiments with Kruppa-constraint manifolds, it typically took 4 or 5 zoom-in steps before enough manifolds were sufficiently linear to find $\mathbf{q}$ directly from Eq. 5.1 [104].

In the case of direct self calibration from Kruppa-constraint manifolds, choosing the initial search region V is easy. Every possible internal calibration is an upper-triangular 3×3 matrix. Since the overall scale factor is irrelevant, it can be assumed that the matrix has a Frobenius norm equal to 1. Thus the largest possible search space for internal calibration matrices can be identified with the hypercube $\{(x, y, z, w, u) \in \mathbb{R}^5 : x, y, z, w, u \in [-1, 1]\}$. Each point in this region corresponds to an upper-triangular matrix $[\, x\ y\ z;\ 0\ w\ u;\ 0\ 0\ (1 - x^2 - y^2 - z^2 - w^2 - u^2)^{1/2}\,]$.

When using modulus-constraint manifolds in a-space, empirical evidence suggests that $\breve{\mathbf{a}}$ will be close to $(0, 0, 0)^\top$ provided the initial projective reconstruction is chosen properly. Our implementation searches for an initial projective reconstruction that is fairly "rounded," meaning the standard deviation of the position of each scene point from the centroid of the scene in the direction of each principle component is about 1. Thus, assuming $\breve{\mathbf{a}}$ will be close to $(0, 0, 0)^\top$, the initial search region can be chosen as $\{(x, y, z) \in \mathbb{R}^3 : x, y, z \in [-\eta, +\eta]\}$ for some small η (e.g., $\eta = 20$). Of course, a larger η can be used if the initial choice fails.

In real applications the manifolds will not all intersect at a single, well-defined point because of noise. All self-calibration algorithms need to deal with this fact whether they explicitly calculate the manifolds or not. The voting algorithm has the advantage that it can find a small *region* that is intersected by all or most of the manifolds, and it can provide some measure of confidence that this region contains the true solution because the more manifolds a region contains and the smaller it is, the more reliable the solution is. More importantly, the voting algorithm is inherently robust to outliers. If any fundamental matrix $\mathbf{F}_i$ is severely incorrect (and thus an outlier), it will generate a manifold that only sporadically intersects the other screw-transform manifolds under consideration. Votes generated by this manifold will not coincide with votes generated by the other (inlying) manifolds and thus the erroneous manifold will not influence the zooming-in process of the algorithm. After a few zoom-in steps, the incorrect manifold will be outside the reduced search region and will be dropped from consideration.

5.3 Monte-Carlo Markov-Chain approach

Strictly speaking, Monte-Carlo Markov-Chain (MCMC) algorithms calculate estimates of expected values [124]; they do not search for global extrema.[2] In this section we will abuse the MCMC technique by using it to find the mutual-intersection point of three manifolds, a process

[2] However, the field of stochastic optimization uses many Monte Carlo methods, including MCMC-based techniques as well as simulated annealing and genetic algorithms.

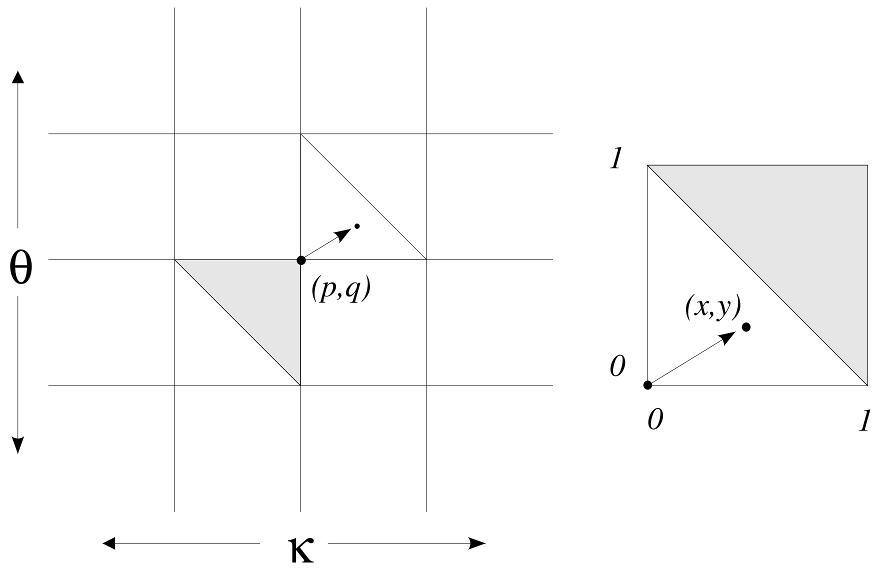

Figure 5.4 Each state in the MCMC search space represents a position on each of three manifolds, using 4 integers (p, q, x, y) per manifold. The figure shows how a quadruple (p, q, x, y) gets mapped to a position on a manifold. x and y are converted to the range $[0, 1]$, and if $x + y > 1$ the position is in the upper triangle of the lattice square (colored gray). The two triangles are handled separately since each lattice point lives in $\mathbb{R}^3$ and thus each lattice "square" is really a skew quadrilateral formed of two non-coplanar triangles.

that is equivalent to searching for the global maximum of a particular inverse-distance function. MCMC will efficiently search for this global maximum because the stochastic process underlying MCMC will spend most of its running time in states of high likelihood. In our case, the states of high likelihood will be those near the mutual intersection points of the three manifolds; thus the algorithm will be likely to "stumble across" the mutual-intersection points or at least come very close to them in a small amount of time.

We now present the algorithm in the context of finding mutual-intersection points of a set of modulus-constraint manifolds; how to use the algorithm with Kruppa-constraint manifolds is discussed later in this section. In our approach, a grid is first laid over each manifold as described in Section 5.1 and portrayed in Fig. 5.1. Let each grid have size $n \times n$. Each state in the state space will have the form

$$(p_1, q_1, x_1, y_1; p_2, q_2, x_2, y_2; p_3, q_3, x_3, y_3)$$

where all entries are integers and $p_i, q_i \in [0, n-1]$ and $x_i, y_i \in [0, m-1]$ for some m. Each state indicates three positions, one for each manifold we are trying to intersect. The entries p_i, q_i, x_i, and y_i taken together indicate a position on manifold i as shown in Fig. 5.4. The numbers p_i and q_i indicate a lattice point on the grid of manifold i and can be thought of as the *lattice position*. The numbers x_i and y_i indicate the *sub-lattice position*. Taken together, they represent a position in one of two triangles as shown in Fig. 5.4 and these two triangles together cover one lattice square.

We use the Metropolis algorithm [118, 124, 32] for exploring the state space. To use this algorithm, each state is assigned a likelihood with the highest likelihoods belonging to the mutual intersection points and the next highest likelihoods belonging to states that are close to mutual intersection points. This is accomplished easily by using inverse mutual distance for likelihood:

$$\mathrm{like}(s) = \frac{1}{(\mathrm{dist}_{12}(s) + \mathrm{dist}_{13}(s) + \mathrm{dist}_{23}(s) + \epsilon)^{\mu}}$$

Here $\mathrm{dist}_{ij}(s)$ means the Euclidean distance between the points on manifolds i and j that state s represents. ϵ is simply a very small number (e.g., 10^{-6}) to prevent division by 0. The power factor μ influences the search process: when μ is small there is little difference in the likelihood of different states, making the search process undirected; when μ is large, the search process can get "trapped" in local maxima, taking a long time to continue exploring other regions of the state space. Thus the choice of μ can be critical in making a usable algorithm (i.e., one that converges in a reasonable number of iterations). Note that, under the theory of MCMC algorithms, the global maximum will eventually be found regardless of the choice of μ because every state will eventually be visited.

The Metropolis algorithm proceeds as follows:

```
S := random starting state
best_state := S
best_like := like(S)
while (best_like not satisfactory) and
      (max iterations not performed) begin
   T := random new state to consider; see comments
   r := like(T)/like(S)
   x := random real number in the range [0,1)
   if (x<r) then begin
      S := T
      if (like(S)>best_like) then begin
         best_like := like(S)
         best_state := S
      end
   end
end
```

Our approach to picking the new state `T` allows for both coarse and fine search simultaneously, which is critical to making a working algorithm. We do the following:

```
x := random real number in the range [0,1)
i := random integer between 1 and 3
let u and v be integers chosen randomly from the set {-1,0,+1}
   so that either u or v is 0 and the other value is nonzero
T := S
if (x < 0.5) then begin
   T.p[i] := (T.p[i] + u) mod n
   T.q[i] := (T.q[i] + v) mod n
end
else begin
   T.x[i] := (T.x[i] + u) mod m
   T.y[i] := (T.y[i] + v) mod m
end
```

If the test **`(x < 0.5)`** is true then **`p[i]`** and **`q[i]`** are modified and coarse-level search is performed; otherwise **`x[i]`** and **`y[i]`** are modified and fine-level search is performed. To help avoid getting trapped in a local maximum, the new state **`T`** can occasionally be chosen from the state space at random; adding this step also makes it clear that the state space is fully connected as required by MCMC algorithms. In theory, of course, the resolution of each grid could be increased to make coarse-level search the equivalent of fine-level search; however, time and memory constraints prevent the manifold grids from having arbitrary resolution.

The main loop is repeated for a set number of iterations or until a state has been found that is sufficiently close to a mutual intersection point. By the end of the iterations, a "best state" has been determined. This state represents three positions, one on each manifold, and each position is within a particular triangle (Fig. 5.4). Thus to finish the algorithm, the mutual intersection point of the three triangles is found. If there is no mutual intersection point (i.e., none contained within all three triangles) then the algorithm has failed and can be restarted with a new random state. If the three triangles do have a mutual intersection point, then the algorithm has succeeded. To further refine the result, a "zoom-in step" could be performed (where the algorithm is repeated on a finer grid centered around the approximate mutual intersection point). In our experiments, such a zoom-in step has proven unnecessary; the piecewise-linear approximation to the screw-transform manifold that each grid represents is sufficiently good that make working at higher resolution is unnecessary.

The algorithm contains several parameters that must be chosen by experimentation. These factors are μ, n, m, the number of iterations, the likelihood of choosing **`T`** at random, and the balance factor between coarse and fine search. Our experiments typically have used values of $\mu = 1$, $n = 50$, $m = 100$, and an equal balance between coarse and fine search; see Section 6.2.5 for a discussion on the number of MCMC iterations to perform.

The MCMC approach can be easily modified to work with Kruppa-constraint manifolds. Since Kruppa-constraint manifolds are 3-dimensional, the state space requires two extra parameters per manifold (one for coarse-level search and the other for fine-level search). The new states have the form:

$$(p_1, q_1, r_1, x_1, y_1, z_1; p_2, q_2, r_2, x_2, y_2, z_2; p_3, q_3, r_3, x_3, y_3, z_3)$$

where all entries are integers and $p_i, q_i, r_i \in [1, n]$ and $x_i, y_i, z_i \in [0, m-1]$ for some m. The algorithm works in the same way as described above with the obvious modifications required by the extra parameters. The final "best state" found by the algorithm will represent one position on each manifold, and each position will lie within a tetrahedron rather than a triangle. In the 5-dimensional search space, the three tetrahedrons will intersect in a single point which will be one of the mutual intersection points of the three manifolds or a close approximation. If the tetrahedrons do not intersect, the algorithm is restarted with a new random starting state.

Note that if the manifolds intersect in more than one place the algorithm will only return one such mutual intersection point; this was also true of the voting algorithm (Section 5.2). Experimental evidence suggests that three modulus-constraint manifolds will usually intersect in 1 or 2 locations (see Section 6.1.5). Thus the algorithm may need to be rerun until two intersection points have been found. This represents an obvious inefficiency; a strength of the surface-fitting algorithm (Section 5.1) is that it finds all mutual intersection points in a single pass.

Chapter 6

Experimental Evaluation of Self Calibration from Screw-Transform Manifolds

6.1 Experimental results for the surface-fitting algorithm

Experiments using both real and simulated data were performed to answer the following questions about the surface-fitting algorithm (SURFIT) of Section 5.1:

(1) Does the algorithm work with noise-free data (i.e., is the mathematics of screw-transform manifolds correct and can surface fitting be used to find the required mutual-intersection point of the manifolds)? How does performance degrade as noise increases?

(2) How fast does the algorithm run?

(3) Does the use of more than three views improve results? By how much?

(4) How many points are contained in the mutual intersection of three screw-transform manifolds in a-space?

(5) Does the method work with views taken by a real camera (which does not necessarily follow a pinhole model)? How do the reconstructions look?

For an explanation of why the fourth question is important, see the discussion in Section 6.3.

The questions above are answered with respect to SURFIT in Sections 6.1.2–6.1.6. Similar questions with respect to the MCMC-based algorithm of Section 5.3 are answered in Section 6.2; for experimental results of the voting algorithm of Section 5.2, see Manning and Dyer [104].

Before presenting the SURFIT experimental results, we discuss in Section 6.1.1 how the synthetic data sets were generated, introduce the error measure used in the experiments, and introduce a nomenclature for describing each synthetic data set.

6.1.1 The synthetic data sets and their nomenclature

Many experiments were performed using randomly-generated, synthetic data sets. A typical data set was created as follows: First an internal calibration matrix $\mathbf{K}$ was generated randomly within realistic ranges for each parameter. The following Matlab-style pseudocode shows specifically how $\mathbf{K}$ was generated:

```
%% NOTE: randpm() returns a random number in the range [-1,1];
%% simulated_image_size contains dimensions of simulated image in pixels
K(1,1)=1+randpm()*0.2;
K(2,2)=K(1,1)+randpm()*1.0/20; %% make pixel height similar to width
%% lens_width_factor=0.2;       %% "wide lens"
lens_width_factor=0.6;          %% "medium lens"
K(1,1)*=lens_width_factor;
K(2,2)*=lens_width_factor;
K(1,2)=randpm()*1.0/25;         %% skew factor
K(1,3)=0.5+randpm()*0.15;       %% \ make principal point near middle
K(2,3)=0.5+randpm()*0.15;       %% / of image, +/-15% in each dimension
K(3,3)=1;
K(2,1)=K(3,1)=K(3,2)=0;
K=[simulated_image_size.x,0,0; 0,simulated_image_size.y,0; 0,0,1]*K;
```

Next, an object consisting of 2 perpendicular squares with feature points uniformly spread across the squares was created and a series of cameras, all with the same internal calibration $\mathbf{K}$, were placed randomly so that each was able to view every feature point on the object. See Fig. 6.1 for a sample scene and Fig. 6.2 for a sample camera view. All simulated views were 1000×1000 pixels.

To answer the first question, we must have a way of quantifying how well a self-calibration algorithm works. We do this by measuring the distance between the internal calibration matrix $\mathbf{K}'$ calculated by the algorithm and the true internal calibration matrix $\mathbf{K}$, which the algorithm should ideally have determined. For a distance metric, we use the Frobenius norm as is widely done in self-calibration literature:

$$\text{error}(\mathbf{K}, \mathbf{K}') = \text{frob}(\mathbf{K}/\text{frob}(\mathbf{K}) - \mathbf{K}'/\text{frob}(\mathbf{K}'))$$

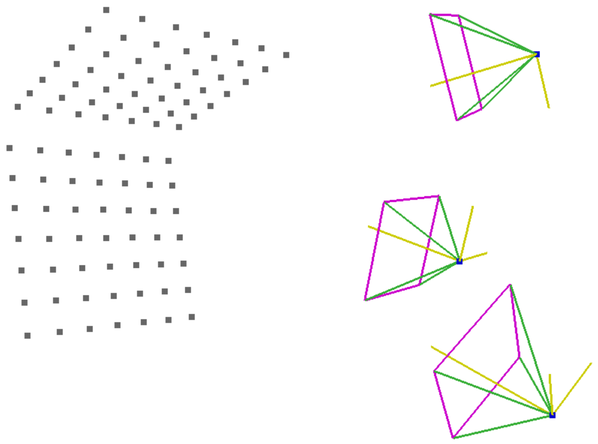

Figure 6.1 Example of a randomly-generated synthetic scene. On the left is the scene object and on the right are three cameras viewing the scene.

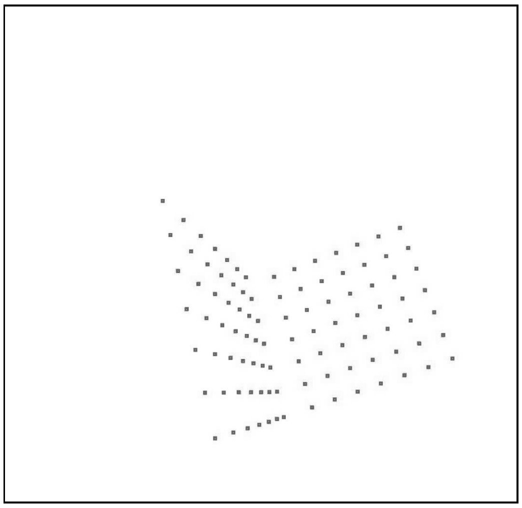

Figure 6.2 A typical camera view of a synthetic scene. View is 1000×1000 pixels; the box indicates the boundary of the view. Note that the object does not span the full view, making accurate calculation of the fundamental matrix more difficult.

SYNTHETIC DATA SETS	
FIGURE	SIGNATURE(S)
Fig. 6.3	$\langle$ 1160, 4, 2, 50, 50, 170, wide $\rangle$
Fig. 6.4	$\langle$ 591, 4, [0, 2], 50, 50, 170, wide $\rangle$ $\langle$ 611, 4, [0, 2], 50, 50, 240, medium $\rangle$ $\langle$ 434, 5, [0, 2], 50, 50, 170, wide $\rangle$
Fig. 6.5	$\langle$ 1160, 4, [0, 4], 50, 50, 170, wide $\rangle$ $\langle$ 1200, 4, [0, 4], 50, 50, 240, medium $\rangle$ $\langle$ 855, 5, [0, 4], 50, 50, 170, wide $\rangle$
Fig. 6.6	$\langle$ 1160, 4, [0, 4], 50, 50, 170, wide $\rangle$
Fig. 6.7	$\langle$ 204, 3, 0, 300, 100, 170, medium $\rangle$
Fig. 6.8	$\langle$ 204, 3, 0, 300, 100, 170, medium $\rangle$ $\langle$ 415, 3, 0, 20, 100, 170, medium $\rangle$ $\langle$ 378, 3, 0, 100, 10, 170, medium $\rangle$

Table 6.1 Description of each synthetic data set used in the experiments of Section 6.1.

where the Frobenius norm is defined as

$$\mathrm{frob}(\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}) = (a^2 + b^2 + c^2 + d^2 + e^2 + f^2 + g^2 + h^2 + i^2)^{1/2}$$

The exact significance of this error measure is difficult to specify. Clearly, if the error is 0 then perfect results have been achieved. However, it is unclear how to interpret small errors. For instance, if scene reconstruction is the ultimate goal then a qualitatively-good reconstruction can often be achieved even if the Frobenius error is large. This is because other factors are also important: the baseline distance between views, the amount of rotation between pairs of cameras, and the number of views being used (which stabilizes triangulation). As a rule of thumb, a Frobenius error of 0.001 usually means very good reconstruction results, although as was just indicated, larger errors in internal calibration can still yield excellent reconstructions.

Each data set is described by 7 terms

⟨ ntrials, nviews, noise, nka, nth, object width, { wide | medium } viewing angle ⟩

which have the following meaning:

ntrials: number of random trials contained in the data set

nviews: number of cameras used

noise: uniform noise, in pixels, added to each feature point as viewed on each camera's image plane; "d pixels of uniform" noise means each feature point was displaced in both the x and y direction by any amount between $-d$ and $+d$ pixels with equal likelihood; note that this means each feature might be displaced by up to $d\sqrt{2}$ pixels from its true location; sometimes noise will be given as a range $[m, M]$, which means that for each trial a random $r \in [m, M]$ was chosen and then uniform, random noise of radius r was added to each feature; thus different trials could have different levels of noise within the same data set

nka, nth: dimensions of the approximate surface (i.e., the grid) that is fit to each screw-transform manifold; in the pseudocode at the start of Section 5.1, these are the dimensions of the array `grid`; nka and nth correspond to the κ and θ directions, respectively

object width: measure of how much of the image plane was spanned by the object being viewed; when "object width" for a data set is w, this means the object had a "width" of at least w pixels on the image plane of each camera in the data set for each trial (but might have had a greater width); the specific measure of the object's width for a single view was the standard deviation of all feature points on the camera's image plane

viewing angle: our experiments used two rough measures of viewing angle, called "wide" and "medium"; the exact meaning of these terms is given by the pseudocode at the beginning of this section, but roughly the "wide" viewing angle was close to $150°$ while the "medium" viewing angle was close to $90°$

Table 6.1 gives the signatures for every synthetic data set used in the experiments of this section, listed by which figure they were used in. As an example, the data set used to generate Fig. 6.3 has the signature $\langle\ 1160,\ 4,\ 2,\ 50,\ 50,\ 170,\ \text{wide}\ \rangle$. This signature means 1160 randomly-generated trials were performed, each having 4 cameras with "wide" viewing angles, 2 pixels of uniform noise, and a medium-size (170 pixels) minimum retinal object width. Furthermore, the approximate surfaces that were fit to each manifold were 50×50 grids; the full range of both κ and θ were uniformly sampled at 50 locations each.

Since the self-calibration algorithms of this paper operate directly from fundamental matrices, the calculation of the fundamental matrices is crucial. We used the standard, normalized-linear method [73] so our results should be easy to reproduce. There are improved, nonlinear methods for calculating fundamental matrices (see [70]) and using these may improve the performance of our calibration algorithms.

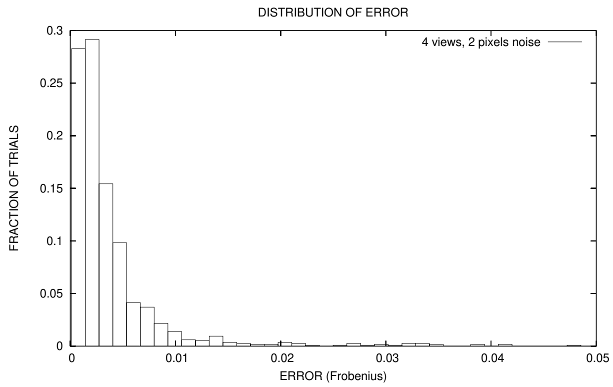

Figure 6.3 Error distribution for the surface-fitting algorithm applied to synthetic data with noise radius 2 pixels. See Table 6.1 for a full description of the data set.

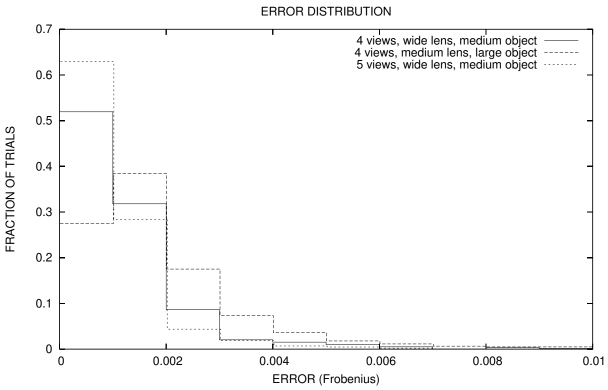

Figure 6.4 Comparison of error distributions for several different data sets (see Table 6.1 for detailed descriptions).

6.1.2 Answer to question 1: Algorithm correctness

The first question is answered by the graphs in Figs. 6.3–6.5. Fig. 6.5 shows error decreasing towards 0 as noise decreases towards 0, indicating that the surface-fitting algorithm would work perfectly in the absence of noise. Also from this graph we see that noise can reach almost 2.4 pixels before error rises above 0.001 in the 5 camera, wide-view case, indicating that the surface-fitting algorithm performs very well even in the presence of noise.

The other two figures were generated from noisy data sets; the noise radius for Fig. 6.3 was fixed at 2 pixels, while that for Fig. 6.4 varied from 0 to 2 pixels. These histograms show the vast majority of trials having very small error, again demonstrating that the surface-fitting algorithm performs well in the presence of noise.

6.1.3 Answer to question 2: Algorithm speed

The surface-fitting algorithm has two distinct phases: projective reconstruction followed by calibration. The projective-reconstruction phase is not a part of our research and we do not include its timing here. Our implementation performs a simple, brute-force search in order to find an initial projective reconstruction that is reasonably "round." The search process can be very slow (on the order of 30 seconds to 2 minutes). It is our understanding that closed-form solutions exist for this problem that execute almost instantaneously.

After the initial projective reconstruction has been found, the second phase of calibration involves upgrading the reconstruction to affine and then metric. Upgrading from affine to metric has a closed-form solution and can be performed instantaneously [68, 133]. The surface-fitting algorithm being tested in this section concerns the first part of the problem: upgrading the projective reconstruction to affine. The algorithm uses three fundamental matrices at a time in a RANSAC process (see Section 5.1). The total running time of the algorithm depends on how many iterations of RANSAC are performed. RANSAC can continue until: (1) a certain error level has been reached, (2) every possible triplet of fundamental matrices has been chosen from the initial set, or (3) enough triplets have been chosen for a particular probability of success to be achieved (meaning it is likely that a triplet of "good" or "inlying" fundamental matrices have been chosen, leading to a good calibration).

Since the RANSAC process involves an indeterminate number of iterations, we time the algorithm by timing how long each RANSAC iteration takes. A histogram of per-iteration run times is given in Fig. 6.6. This histogram shows that typical iterations take 0.25-0.75 seconds. When 4 camera views are used for calibration, there are at most ${}_4C_2 = 6$ fundamental matrices and at most ${}_6C_3 = 20$ iterations of RANSAC. One could thus expect a runtime of 5-15 seconds. This calculation does not take into account the fact that sometimes iterations take significantly more than 1 second to execute.

The surface-fitting algorithm also requires a preprocessing step. Namely, for each fundamental matrix there is a corresponding screw-transform manifold and each such manifold must have a surface fitted to it. The surface-fitting process runs in a fixed time dependent on the desired resolution of the surface, because each manifold sample point takes a fixed amount of time to calculate. Our implementation running on a Sun Ultra Sparc required 5.033×10^{-4} seconds mean time to compute each grid point, so approximate surfaces with a 50×50 resolution, which were used in most experiments, required 1.258 seconds per surface. This time could be improved: A surface is generated by fixing a value of κ then iterating over all θ before moving to the next value of κ. The

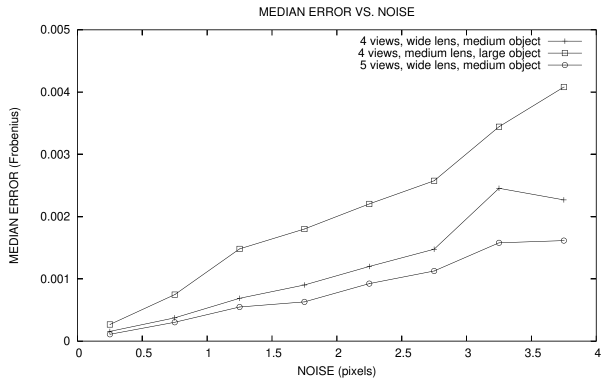

Figure 6.5 Relationship between noise and error in three different data sets.

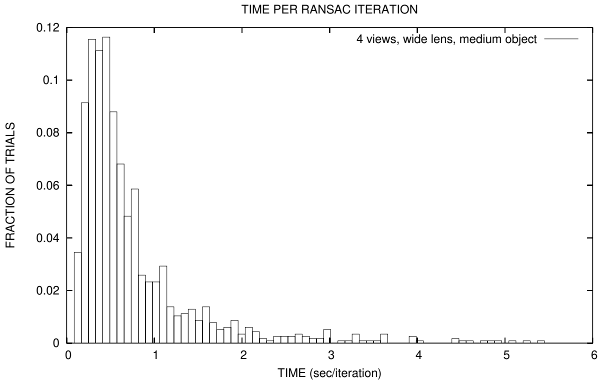

Figure 6.6 Distribution of run times.

calculations involving a particular κ could be performed once and then only those calculations involving θ would need to be performed during the inner loop. For example, steps (1)-(4) and much of step (5) in Algorithm A–1 (Fig. 4.2) can be performed without knowledge of θ. Our timings, however, do not include such efficiencies and are thus slower than they could be.

6.1.4 Answer to question 3: Advantage of extra views

Does the use of more views improve calibration? This question is answered by Fig. 6.5. To generate this graph, hundreds of trials were run each with a different noise radius chosen randomly between 0 and 4 pixels. The data sets were specifically:

$$\langle\, 1160,\ 4,\ [0,4],\ 50,\ 50,\ 170,\ \text{wide}\,\rangle$$
$$\langle\, 1200,\ 4,\ [0,4],\ 50,\ 50,\ 240,\ \text{medium}\,\rangle$$
$$\langle\, 855,\ 5,\ [0,4],\ 50,\ 50,\ 170,\ \text{wide}\,\rangle$$

Notice that 2 data sets used 4 views and 1 set used 5 views. The 4-view data sets differ in camera viewing angle and retinal object size. The data was then plotted as follows: for a given data set, every trial that had a noise radius between 0 and 0.5 pixels was identified and then the median error for this subset was calculated. This median error was plotted with an x-coordinate of $(0+0.5)/2 = 0.25$ pixels and the process was repeated for noise radii in the range 0.5 to 1.0 pixels, 1.0 to 1.5 pixels, and so on.

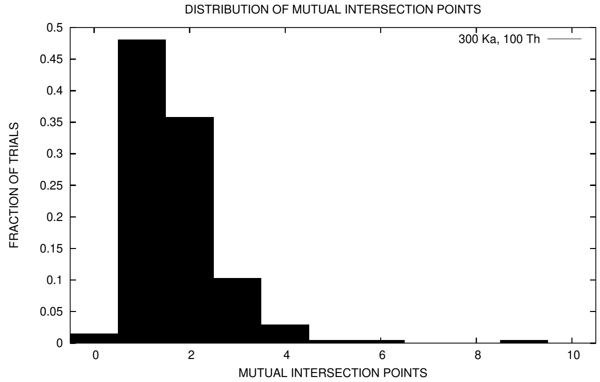

Figure 6.7 Distribution of the number of mutual intersection points of three modulus-constraint manifolds.

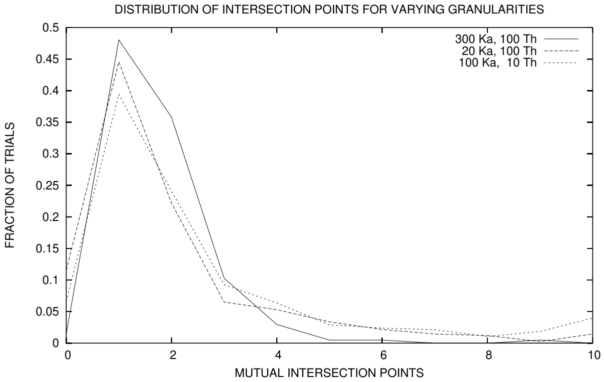

Figure 6.8 Distribution of mutual intersection points for modulus-constraint manifolds of differing granularities.

The resulting graph shows the following: First, as was already noted in Section 6.1.2, it shows that error decreases towards 0 as noise decreases towards 0. Second, it shows that using 5 views produces notably better results than using 4 views, ceteris paribus. Finally, it shows that using a wide field of view greatly lowers the error from using a medium-size field of view.

The last observation is significant because a wide field of view is rarely used in calibration and reconstruction in general (and yet apparently it should be). We believe the improvement arises because the fundamental matrix calculation becomes more stable as the field of view increases, and our calibration technique is entirely dependent upon fundamental matrices.

6.1.5 Answer to question 4: Number of mutual intersection points

Question 4 is answered by Fig. 6.7 and Fig. 6.8. The first figure shows the distribution of the number of mutual intersection points found during each trial of a data set with signature $\langle\ 204,\ 3,\ 0,\ 300,\ 100,\ 170,\ \text{medium}\ \rangle$. Approximately 85% of trials had either 1 or 2 mutual intersection points. Roughly 2% found no intersection points; this was probably due to the finite granularity of the surfaces that were fitted to each manifold. Few trials showed more than three mutual intersection points. These results suggest to us that the three-view self calibration problem has either 1 or 2 solutions for cameras in general arrangement. Under this interpretation, results showing 0 or 3 or more intersections are errors arising from the approximate surface-fitting technique.

The second graph shows how the granularity of the fitted surface affects the number of mutual intersection points found. The term “granularity” here means how many different values of κ and θ were used to generate the approximate surface; using more values means a better approximation, but also means a longer runtime and more memory consumption. For example, a granularity of “100 Th” means 100 values uniformly spaced in the domain $[-\pi, \pi]$ were used for θ. The domain of κ is $[0, 1]$ for the mapping described in Appendix B.2. In the figure, the solid line corresponds to the data used in the previous figure (Fig. 6.7) and represents the highest granularity tested. The other two lines show the effect of reducing the granularity of κ and θ independently. The results were somewhat worse when θ had a low granularity.

In general, the results are consistent regardless of the granularity, showing a spike at 1 and 2 mutual intersection points. For the two data sets with lower granularities, the spikes in the histograms are diminished as would be expected.

6.1.6 Answer to question 5: Performance with real cameras

In this subsection, we discuss the results of two experiments performed with real cameras. The goal of the first experiment was to acquire extremely high-quality data using a real camera in order to test how well our self-calibration and reconstruction algorithms would perform under ideal circumstances. The second experiment tested our algorithms with a more realistic and cluttered scene.

6.1.6.1 Experiment 1: High-quality data

To make a high-quality data set, we created a calibration object from a cardboard box by removing sides from the box until only three walls remained, all meeting at a single vertex. We covered the inside faces of these walls with a dot pattern printed on a laser printer (see Fig. 6.9). In order to establish a separate, 2-dimensional coordinate system for the dots on each face, three dots on each face were given a unique appearance; these dots served as the positions $(0, 0)$, $(1, 0)$, and $(0, 1)$ in local coordinates. The remaining dots on each face represented lattice points in the local coordinate system. A set of six photographs of the box was taken using a camera with fixed internal parameters and minimal radial lens distortion. The photographs were taken from different positions and orientations, but most of each face was visible in each photo.

Once the views were captured, the center of each dot was determined automatically. Because lighting was not uniform across the faces, a manual preprocessing step was performed to equalize the appearance of dark and light regions (i.e., the black dots and white paper) before extracting the dot centers. Also, the three faces were manually partitioned from each other in each view in order to identify which face each dot belonged to. The center of mass of each dot as it appeared in each view was used to approximate the projection of the dot's true center into that view; see [150] for another use of this approximation.

Thus every dot was assigned three identifying coordinates in each view. The first coordinate indicated which of the three faces the dot belonged to and the second and third coordinate were the dot's x and y positions on that face in the local coordinate system. Note that the same dot will be assigned the same three coordinates in each view. In this way, point correspondences were automatically determined between each of the six views. There were several hundred point correspondences between any two views and the correspondences covered the complete field-of-view of the camera; these two facts together made ideal circumstances for calculating pairwise fundamental matrices.

Once the fundamental matrices were calculated, self calibration and reconstruction could be performed. The following three calibration methods were tested on this data: direct self calibration using the voting scheme [104], stratified self calibration using the voting scheme [105], and stratified self calibration using the surface-fitting technique described in this paper. The results in each case were essentially identical, indicating the extreme high quality and low noise of this data set. The reconstructions appear to be near perfect, with each face perpendicular to the other two, each feature on each face lying in nearly the same plane, and features on each face forming orthonormal lattice grids (Fig. 6.10). For the voting algorithms, all six of the camera views were used to perform calibration; for the surface-fitting algorithm, only three were used.

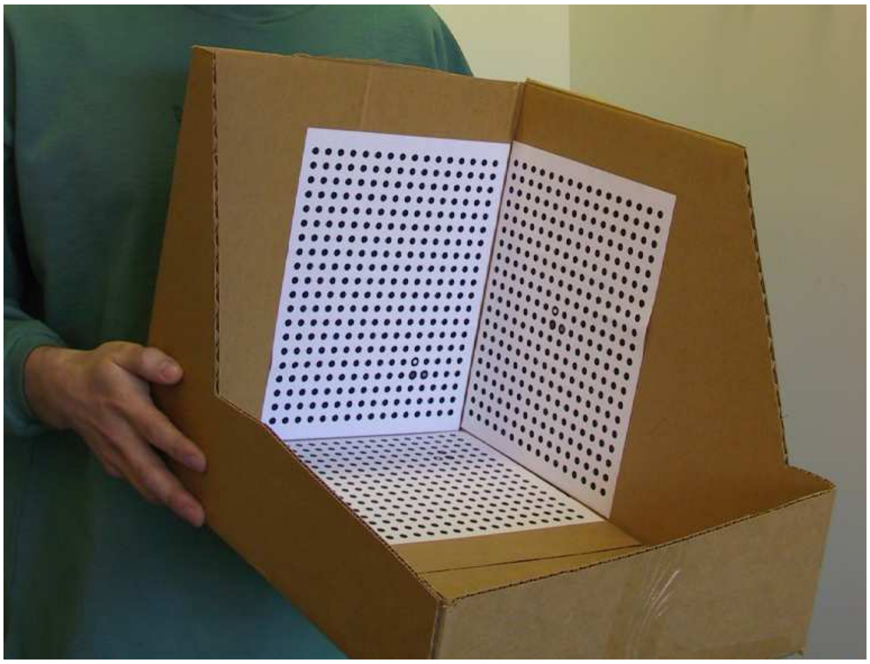

Figure 6.9 Photo of calibration box.

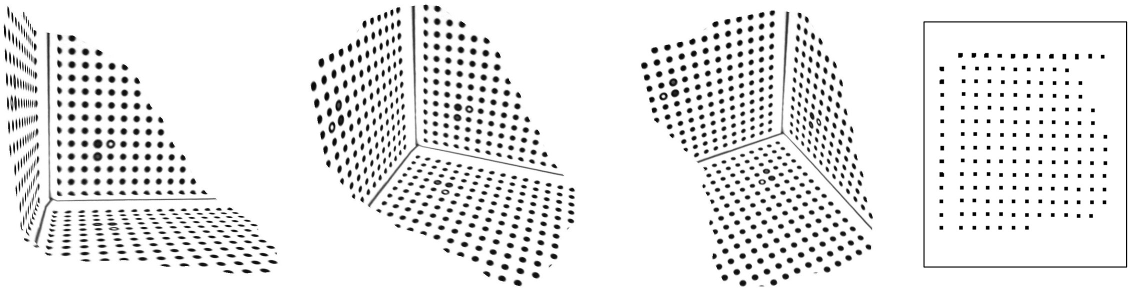

Figure 6.10 Reconstructed calibration box. On the right is an orthographic, overhead view showing the regularity of the dot pattern on the reconstructed surface.

Note that our analysis of the experimental results presented above is empirical rather than analytical. That is, we are interested in describing the apparent visual quality of the results rather than comparing the results to physical measurements of the original scene and camera. There are two reasons for this:

First, the only way to know the "true" internal calibration of the camera is to calibrate the camera using an alternative calibration technique. However, any alternative technique would still have to contend with noisy data and would ultimately produce only an approximation of the "true" internal calibration. Furthermore, the camera used in the experiments was not a pinhole camera and thus there was no "true" internal calibration matrix for this camera to begin with. At best, an analytical analysis of the self-calibration results would only compare one approximation with another.

Second and more importantly, our self-calibration algorithms have already been thoroughly tested with synthetic data to determine their capacity for finding correct internal calibrations. We know from these experiments that the algorithms can determine the true internal calibration of a pinhole camera to an arbitrary degree of precision provided the input data is sufficiently accurate. The data generated by the calibration box experiment is very good, verging on synthetic, and thus could only serve to reinforce the results from our synthetic-data experiments. In real applications, the input data would be of much lower quality: the feature points would not be determined with such subpixel accuracy, there would be fewer feature points, and the features would not cover the complete camera view.

6.1.6.2 Experiment 2: Realistic scene

For the second experiment, we used a scene with more-natural objects than the calibration box in Experiment 1. The scene consisted of three objects placed on a mostly-flat, patterned rug on a flat floor, in front of a flat wall perpendicular to the floor (Fig. 6.11). The objects were chosen for their shape, color, and texture.

The first object was a cardboard box,[1] chosen because its two visible walls should appear perpendicular to each other and to the ground in the reconstruction. Also, the features on these walls, as on all the flat surfaces, should be coplanar in the final reconstruction; this is a measure of the quality of the feature correspondences independent of the correctness of the metric reconstruction since coplanar points will remain coplanar in *any* projective reconstruction. Note that the cardboard box has matte surface-reflectance properties and limited surface texture for point correspondences.

The second object was an inflatable plastic globe. The shape of the globe is mostly spherical; however, viewed from overhead (i.e., looking down on the North Pole) the globe has a rounded hexagonal cross section. The plastic surface of the globe is fairly reflective, potentially introducing correspondence problems.

The third object was a furry toy monster. This object has a complicated, anthropoid shape without being overly-detailed and thus was good for testing the capabilities of the reconstruction. Furthermore, the furry surface texture provided a challenge for determining point correspondences.

Only the SURFIT calibration algorithm was tested on this data set. The algorithm was applied using only three views (Fig. 6.11), and the three chosen views were closely positioned in space making the calibration and reconstruction task more difficult due to small baselines and small rotation angles. Reconstruction results are shown in Fig. 6.11. Our algorithm returned only a single internal calibration, and this was the calibration used for the reconstructions.

Note that the walls of the cardboard box are close to perpendicular and the sphere is spherical. The toy monster also has the correct anthropoid shape, although many more feature points would be necessary to produce a realistic reconstruction (e.g., the two horns are lacking). Note in particular the correct chin and brow ridges, and the distinct, rounded legs. The flat floor and back wall are very planar in the reconstruction, indicating the highly-quality of the feature point correspondences.

However, there are problems. The back wall is not perpendicular to the floor and the texture on the rug is not uniformly proportioned (i.e., the texture appears larger in some regions than others even though it is uniform on the original rug). These problems are almost certainly due to the close

[1] In fact, the box was the back of the box from in Experiment 1.

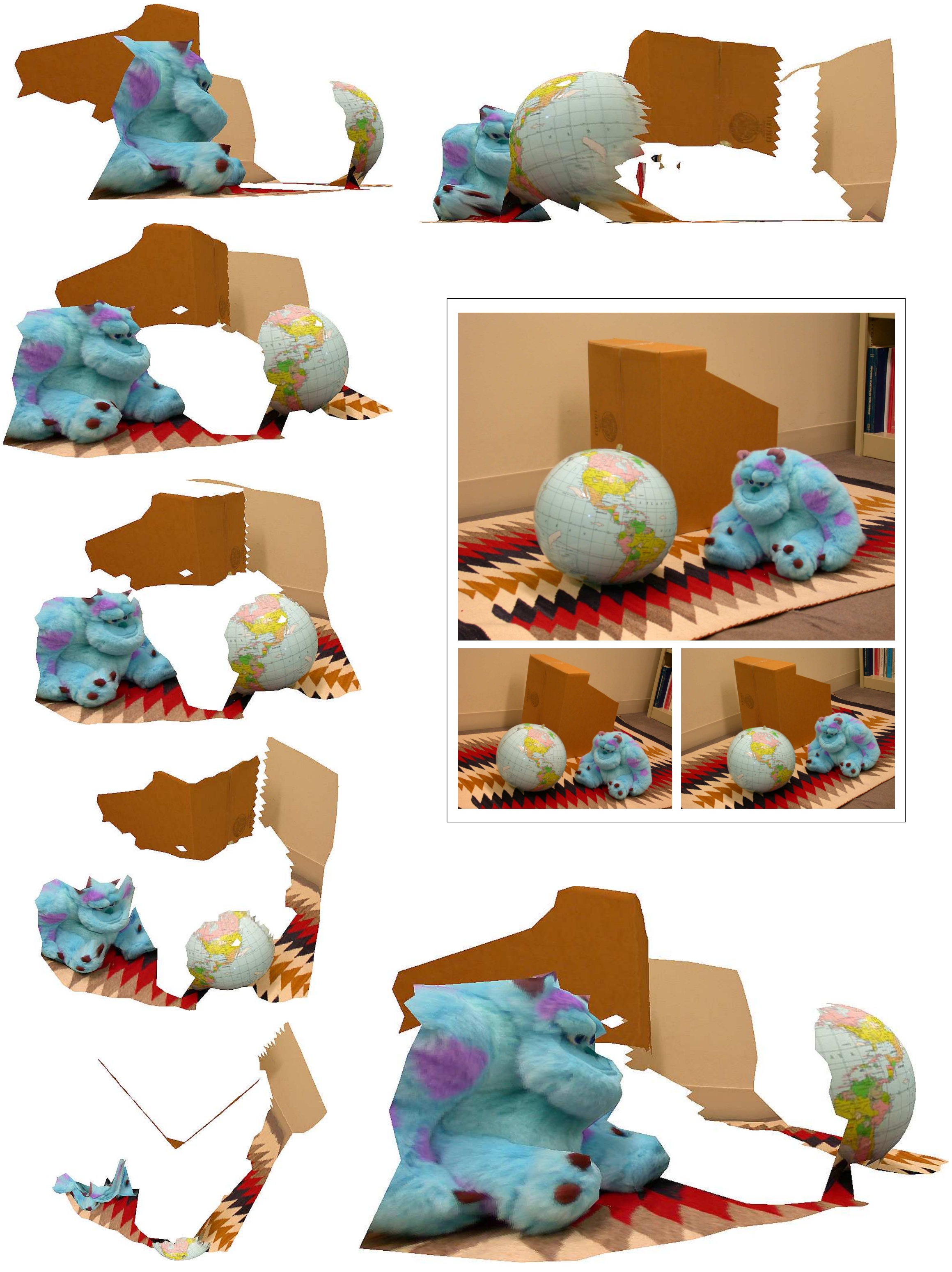

Figure 6.11 A scene reconstructed from three closely-spaced reference views (inset).

spacing of the original views. For example, a reconstruction in which the wall was perpendicular to the floor would probably have fit the given input data only as well as the reconstruction produced by our algorithm; from where the cameras are, it is impossible to determine if the back wall is slanted or perpendicular to the floor, given the noise in the data. On the other hand, if one of the cameras had been positioned to view the wall and floor more "edge on" then it would have been more evident to the algorithm (in the given data) that the wall and floor were perpendicular. This is an important, recurring problem in scene reconstruction: when there is noise in the data, incorrect reconstructions can fit the data as well as the correct reconstruction.

From a practical standpoint, a more important problem stems from the fact that the objects are viewed from only one side and thus can only be reconstructed in a very limited fashion. Views from many different angles would have been necessary to produce a complete scene reconstruction (and even then some parts of the scene would not be adequately visible, like parts of the objects near the floor). Trying to reconstruct scenes using widely-separated views introduces new problems due to accumulation of errors and the difficulty in finding point correspondences over wide baselines; see the SMILE workshops [90, 136], which were dedicated to this topic.

Note that, once the full camera calibrations (including position and orientation) have been determined by our algorithm, it is possible to use a wide variety of techniques to visualize the scene in a more-complete manner. An important class of visualization techniques besides scene reconstruction are light field [96] or lumigraph [63] techniques. Light-field rendering requires a dense set of views with full camera calibration for each view; no point correspondences are required (or recovered) and no scene reconstruction is performed. Starting with a dense set of views, self-calibration techniques can be used to recover camera calibration for each view after which light-field rendering becomes possible (see [75, 131] for an example of this approach).

Alternatively, knowledge of camera calibration can be used in conjunction with view interpolation techniques [182, 26, 191, 154, 111] to produce virtual camera views transitioning between original reference views under the control of an end user. Such interpolation techniques work by directly manipulating the original input images and thus can potentially produce more detailed-looking output than scene reconstruction (because all the detail visible in the original views is still more-or-less present in the interpolated views, up to the quality and density of the feature correspondences). Dynamic view morphing [109] in particular requires affine camera calibration when three or more moving objects are present.

Finally, knowledge of camera calibration makes scene reconstruction possible. Space carving [93] and voxel coloring [155] use full camera calibration to find dense correspondences between camera views, building a voxelized scene reconstruction as a side effect. The dense-correspondence-finding algorithm of Koch [89] also requires full camera calibration. Once dense correspondences have been determined, detailed scene reconstruction can be performed in a variety of ways, usually producing a triangle mesh as output. The "Facade" system [31] utilizes full camera calibration along with user interaction to create scene reconstructions. View interpolation techniques and texture-mapped scene reconstructions can both benefit from the view-dependent texture mapping used in Facade, and camera calibration is required for view-dependent texture mapping.

6.1.7 Implementation details

To complete our discussion of experimental results for the SURFIT algorithm, we need to discuss several important implementation details. These details are also relevant to the experimental results for the MCMC-based algorithm presented in Section 6.2.

6.1.7.1 Measuring the goodness of a mutual intersection point

Each experimental trial involves performing a series of RANSAC iterations. For each RANSAC iteration, a number of mutual intersection points are determined by the SURFIT algorithm; in the case of the MCMC-based algorithm, one or zero mutual intersection points are determined. Each mutual intersection point is converted into an internal calibration matrix $\mathbf{K}$, which is then assigned a number indicating how well $\mathbf{K}$ meets certain expected criteria for an internal calibration. We will call this the *goodness* of the mutual intersection point. As successive RANSAC iterations are performed, the calibration algorithm remembers which mutual intersection point had the highest goodness score so far. When all iterations have been performed, the intersection point with the highest goodness is considered the best answer and the calibration corresponding to this point is returned by the algorithm. Thus how the "goodness" of a mutual intersection point is measured is an important implementation detail. Note that one can think of goodness as an error measure; however, we are using the term "goodness" instead of "error" because error can only be measured if the correct solution is known a priori. In this sense, the relationship between error and goodness is like the relationship between probability and likelihood.

Our measure of goodness involved two components. Let $\mathbf{q}$ be a mutual intersection point and let $\mathbf{K}$ be the corresponding internal calibration. The first goodness criterion was that $\mathbf{K}$ must be reasonable for a real camera. Specifically, we required that

$$0.85 < \mathbf{K}_{(11)}/\mathbf{K}_{(22)} < 1.15 \tag{6.1}$$

$$0.35 < \mathbf{K}_{(13)}/w < 0.65 \qquad \text{and} \qquad 0.35 < \mathbf{K}_{(23)}/h < 0.65 \tag{6.2}$$

where h and w denote the height and width of the camera view in pixels. The first condition (Eq. 6.1) specifies that the camera must have somewhat rhomboid pixels (rather than arbitrary parallelogram-shaped pixels). The second condition (Eq. 6.2) indicates that the principal point must be somewhat central to the view. Both of these test are lenient; if more information were known about the expected internal calibration of the camera, these tests could be made tighter. When $\mathbf{K}$ did not meet these basic criteria, our implementation gave $\mathbf{q}$ a very poor goodness score (by adding a large penalty).

The second criterion of goodness was based on the following: The point $\mathbf{q}$ can be used to recover the $\mathbf{H}^{\infty}$ matrix between each pair of camera views using Eq. 4.9, with $\mathbf{q}$ in place of $\mathbf{a}$. By the left-hand sides of both Eq. 4.7 and Eq. 4.8,

$$\mathbf{K}^{-1}\mathbf{H}^{\infty}_{ij}(\mathbf{q})\mathbf{K} \cong \mathbf{R}_{ij} \tag{6.3}$$

for some rotation matrix $\mathbf{R}_{ij}$. If the correct internal calibration was found and all data was noise free, then the following holds:

$$0 = \sum_{ij} \mathrm{frob}(\mathbf{I} - \mathbf{R}_{ij}(\mathbf{R}_{ij})^{\top}) \tag{6.4}$$

where the sum is over all pairs of views i and j for which a fundamental matrix was provided. Eq. 6.4 uses the fact that rotation matrices are orthogonal. The left-hand side of Eq. 6.3 must be scaled properly before using Eq. 6.4; in particular, its determinant must become 1 after scaling.

How close the sum in Eq. 6.4 was to 0 was the second criteria of goodness we used. Essentially, the sum in Eq. 6.4 was used as the main error measure for each candidate internal calibration, and those calibrations that did not meet the first criteria had a severe penalty. In this way, a single number measuring the "goodness" of each mutual intersection point was produced.

Pollefeys [131] used a different measure of goodness. In his experiments, the goodness of a candidate internal calibration $\mathbf{K}$ was based on how well $\mathbf{K}$ met expectations about its form. In particular, $\mathbf{K}$ was expected to represent a camera with square pixels and a principal point in the center of the view. The candidate solution that was closest to the expected form would have the highest goodness and would be returned as the final answer. This contrasts with the technique given above, in which goodness is based on how well a candidate calibration induces rotation matrices between pairs of views, and the expected form of internal calibration is used only to eliminate solutions that do not represent reasonable cameras. Our approach allows recovery of cameras that have pixels of unknown (but reasonable) shape, for instance, without assuming the pixels have a particular aspect ratio.

6.1.7.2 Normalizing the search space

Screw-transform manifolds provide an explicit representation of the set of legal camera calibrations corresponding to particular fundamental matrices, in contrast to, for example, the Kruppa constraints or the modulus constraint which provide implicit representations. An important benefit of having an explicit representation is that it becomes possible to normalize the search space to provide maximum separation (in the metric of the search space) between alternative solutions to calibration.

Consider Fig. 5.1, which shows three modulus-constraint manifolds in a-space. Before trying to find the mutual intersection points of a triplet of manifolds like these, our implementation first fits a plane to each manifold and then transforms the search space so that the three fitted planes corresponded to the (x, y)-plane, the (x, z)-plane, and the (y, z)-plane. In this way, the three manifolds are made very distinct from each other. After transformation, the mutual intersection points lie in the vicinity of the origin while the outer reaches of each manifold are maximally separated. The basis for this idea is the empirical observation that, as in Fig. 5.1, each manifold consists of two broad, flat outer regions surrounding a small, curved inner region, and in general each manifold runs roughly in two directions rather than all three of a-space.

Our particular technique for transforming the search space is to create a cloud of points by uniformly sampling the lattice points from each of the three approximate manifolds and then find the principle components of this cloud. There will be three principle components because a-space has three dimensions. The search space is normalized by (1) translating the center of mass of the cloud of points to the origin, (2) mapping the three principle components to the unit x, y, and z vectors, and (3) stretching the space so that the standard deviation of the cloud of points in the direction of each principle component is 1. Because each manifold has an approximately planar shape, this simple technique will meet the objectives of normalization stated earlier.

Normalizing the search space as described in this section was performed for all experiments in both Section 6.1 and Section 6.2. For the SURFIT algorithm, normalization makes the use of bounding boxes meaningful and could help stabilize the triangle-intersection tests. For the MCMC algorithm, normalization is crucial because it ensures the distance between sample points that are not close to a mutual intersection point is large.

6.2 Experimental results for the MCMC-based algorithm

This section discusses the results of self-calibration experiments performed using the MCMC-based manifold-intersection algorithm (Section 5.3). The following questions were addressed by the experiments:

(1) Does the algorithm work with noise-free data (i.e., is the algorithm correct)? How does performance degrade as noise increases?

(2) How fast does the algorithm run?

(3) Does the use of more than three views improve results? By how much?

(4) How many MCMC-iterations and how many RANSAC-iterations should the algorithm perform?

(5) How does the MCMC-based algorithm compare to the surface-fitting algorithm?

When interpreting the results of this section, note that the implementation details given in Section 6.1.7 for the SURFIT algorithm also apply to the MCMC-based algorithm.

6.2.1 The synthetic data sets and their nomenclature

As in Section 6.1, many experiments using randomly-generated, synthetic data sets were performed. The nomenclature for these sets is the same as in Section 6.1.1 except that two additional data fields are required to store parameters specific to the MCMC-based algorithm:

mcmc iterations: number of iterations of the MCMC loop performed during each RANSAC iteration before returning the best answer found

ransac iterations: number of iterations of the RANSAC loop performed during each experimental trial

Thus each data set is described by 9 terms

$$\langle \text{ ntrials, nviews, noise, nka, nth, object width,} \\ \{ \text{ wide} \mid \text{medium } \} \text{ viewing angle, mcmc iterations, ransac iterations } \rangle$$

Table 6.2 gives the signatures for every synthetic data set used in the experiments of this section, listed by which figure they are used in. All other comments from Section 6.1.1 apply to the experiments in this section as well.

SYNTHETIC DATA SETS	
FIGURE	SIGNATURE(S)
Fig. 6.12	$\langle$ 1580, 3, [0, 4], 50, 50, 170, wide, 100000, 50 $\rangle$ $\langle$ 1020, 4, [0, 4], 50, 50, 170, wide, 100000, 50 $\rangle$ $\langle$ 1700, 5, [0, 4], 50, 50, 170, wide, 100000, 50 $\rangle$ $\langle$ 1680, 6, [0, 4], 50, 50, 170, wide, 100000, 50 $\rangle$
Fig. 6.13	$\langle$ 1020, 4, [0, 4], 50, 50, 170, wide, 100000, 50 $\rangle$ $\langle$ 1160, 4, [0, 4], 50, 50, 170, wide $\rangle$
Fig. 6.15	$\langle$ 360, 3, 0, 50, 50, 170, wide, 100000, 100 $\rangle$
Fig. 6.16	$\langle$ 432, 4, 0, 50, 50, 170, wide, 100000, 100 $\rangle$
Fig. 6.17	$\langle$ 260, 4, 0, 50, 50, 170, wide, 5000, 30 $\rangle$ $\langle$ 260, 4, 0, 50, 50, 170, wide, 10000, 30 $\rangle$ $\langle$ 260, 4, 0, 50, 50, 170, wide, 50000, 30 $\rangle$ $\langle$ 260, 4, 0, 50, 50, 170, wide, 100000, 30 $\rangle$ $\langle$ 260, 4, 0, 50, 50, 170, wide, 200000, 30 $\rangle$
Fig. 6.18	$\langle$ 260, 4, 0, 50, 50, 170, wide, 100000, 5 $\rangle$ $\langle$ 260, 4, 0, 50, 50, 170, wide, 100000, 10 $\rangle$ $\langle$ 260, 4, 0, 50, 50, 170, wide, 100000, 30 $\rangle$ $\langle$ 260, 4, 0, 50, 50, 170, wide, 100000, 50 $\rangle$ $\langle$ 260, 4, 0, 50, 50, 170, wide, 100000, 100 $\rangle$

Table 6.2 Description of each synthetic data set used in the experiments of Section 6.2.

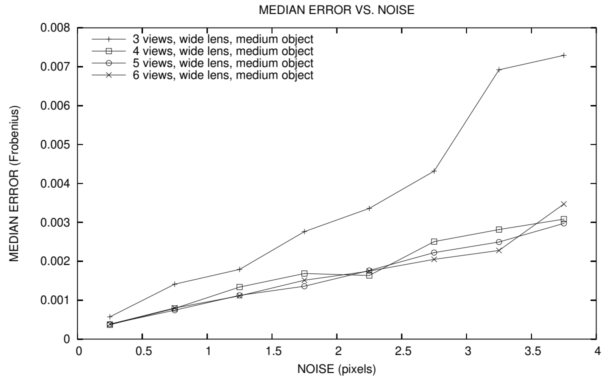

Figure 6.12 Relationship between noise and error in four different data sets

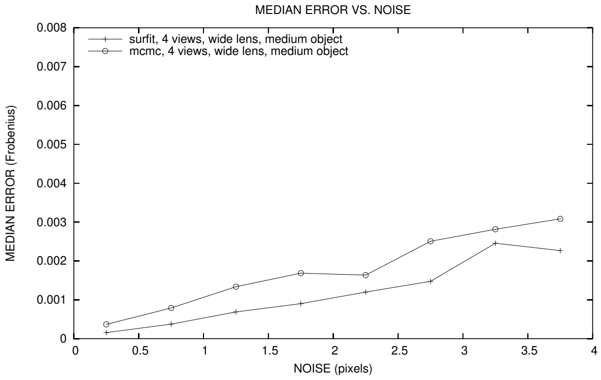

Figure 6.13 Direct comparison between SURFIT and MCMC-based algorithms.

6.2.2 Answer to question 1: Algorithm correctness

The question of whether the mathematics of screw-transform manifolds is correct and can be used for self calibration was answered by the experiments in Section 6.1.2. The graph in Fig. 6.12 showing error converging to 0 as noise goes to 0 also implies the correctness of the mathematical theory.

The question of whether the MCMC-based intersection algorithm can be used to locate the mutual-intersection point of several screw-transform manifolds is answered both by the convergence seen in Fig. 6.12 and also by the histograms in Figs. 6.15–6.16. The histograms demonstrate that, in the absence of noise, the mutual-intersection point can be found by the MCMC-based algorithm with a high-probability of success. Results from further noise-free experiments are shown in Fig. 6.14, which demonstrate that as the granularity of each approximate surface increases (making the surfaces less and less approximate) the algorithm finds the correct answer more often. With a high granularity, the algorithm finds the correct answer about 98% of the time (Fig. 6.14(right)).

Note that the histogram in Fig. 6.15 may seem to suggest that there is only one mutual-intersection point for three screw-transform manifolds, but this is misleading. If, for instance, there were 2 mutual-intersection points then the MCMC-based algorithm would only return 1 of the two intersection points and the success rate would be less than 50% (unless, due to some unknown property of the mathematics, the two mutual intersection points always happened to correspond

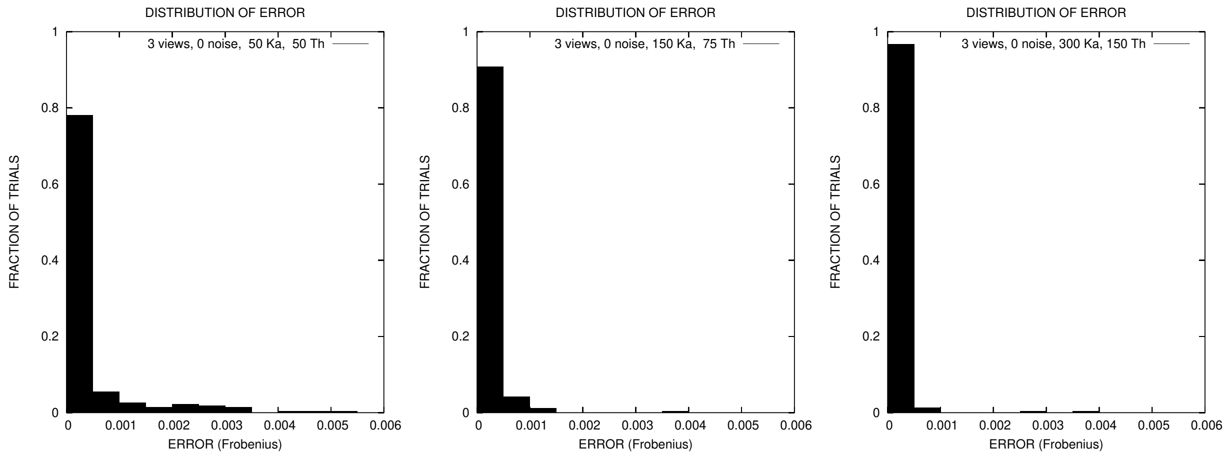

Figure 6.14 Increasing granularity (smoothness) of the approximate manifold surfaces leads to improved convergence for the MCMC-based intersection algorithm. From left to right the dimensions of the manifold sampling grids were 50×50, 150×75, and 300×150, where the first and second numbers are the dimensions of the κ and θ axii, respectively. Note that 50×50 grids were used for most experiments in the dissertation.

to similar internal calibrations). However, our implementation adds a severe penalty factor to any potential solution (i.e., mutual-intersection point) that does not represent a reasonable internal calibration. The criteria is fairly lenient; see the discussion in Section 6.1.7.1 about the added penalty factor. Thus it could be that other mutual-intersection points exist, but that they tend to produce unreasonable internal calibrations and are discarded by the algorithm. However, given the results of Section 6.1.5 that suggest there are between 1 and 3 mutual intersection points typically, the histogram in Fig. 6.15 seems to further imply a single mutual-intersection point.

6.2.3 Answer to question 2: Algorithm speed

Each RANSAC iteration of the MCMC-based algorithm runs at constant speed; this speed is directly proportional to the number of MCMC iterations performed. It takes 0.2728 seconds for our implementation to perform 100000 MCMC iterations on an 800 MHz Pentium III. The complete runtime is then

$$\left(\begin{array}{c} \textit{number of} \\ \textit{fundamental matrices} \end{array} \right) \times \left(\begin{array}{c} \textit{time to create} \\ \textit{manifold grid} \end{array} \right) + \left(\begin{array}{c} \textit{number of} \\ \text{RANSAC } \textit{iterations} \end{array} \right) \times \left(0.2728 \textit{ seconds} \right)$$

plus the time needed to construct the initial projective reconstruction, which is not part of our research. The RANSAC process can be short circuited as soon as an internal-calibration matrix with a sufficient goodness score (Section 6.1.7.1) has been found; however, the process is fast enough that we chose to use a fixed number of RANSAC iterations in the experiments.

For comparison with the timing results from Section 6.1.3, which were from experiments performed on a much slower computer, it took 1.261×10^{-4} seconds mean time to compute each manifold grid point on an 800 MHz Pentium III; thus a surface with resolution 50×50 took 0.315 seconds to compute per manifold. Based on this, run times for experiments in Section 6.1.3 should be divided by 4 (roughly) before comparison with the results in this section.

6.2.4 Answer to question 3: Advantage of extra views

This question is answered by the plot in Fig. 6.12. There is a very significant decrease in error when 4 camera views are used instead of the minimum 3. However, using 5 or 6 views does not seem to produce better results than using 4 views. Note that the same number of RANSAC iterations were performed in all trials and using more RANSAC iterations for the 5-view and 6-view cases might have produced better results (because these cases have more triplets of manifolds to explore); however, we ran no experiments to test this. Regardless, using more RANSAC iterations has the cost of a longer run time.

6.2.5 Answer to question 4: Choosing MCMC parameters

A common complaint against many machine-vision algorithms is the need to carefully tune the algorithm's parameters in order to achieve good performance. This is not the case with the two intersection algorithms we ran experiments on. The surface-fitting algorithm has no parameters beyond those used to create the approximate manifold surfaces (e.g., granularity) and to determine the "goodness" of potential solutions; performance of the algorithm does not hinge on choosing ideal parameters, and we have given some suggested values for these parameters throughout this dissertation. The only additional parameters that need to be set for the MCMC-based algorithm are

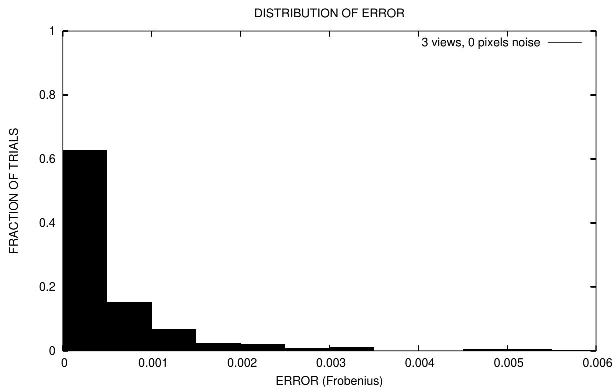

Figure 6.15 Histogram showing likelihood of successful calibration from 3 views and noiseless data.

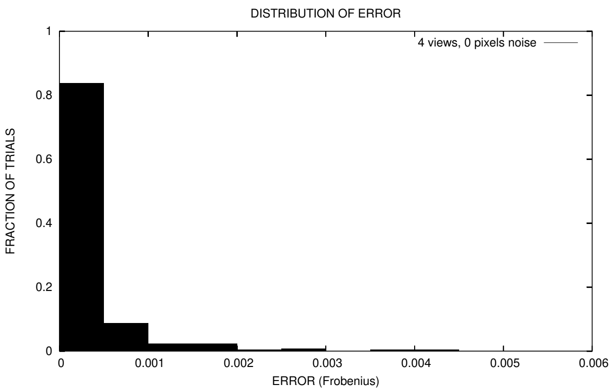

Figure 6.16 Histogram showing likelihood of successful calibration from 4 views and noiseless data.

the number of RANSAC iterations to perform and the number of MCMC iterations to perform per RANSAC iteration, as we now explain.

With the surface-fitting algorithm, the number of RANSAC iterations depended solely on the anticipated number of outlying fundamental matrices. This is because each iteration would yield all possible mutual-intersection points for the chosen triplet of manifolds and thus there was no need to re-explore that triplet. However, the MCMC-based algorithm produces at most one possible solution per triplet, and may not produce any solutions if it gets trapped in a local minimum of the error function. Thus there is a need to perform many extra RANSAC iterations to increase the probability of success.

We chose to perform a fixed number of RANSAC iterations, using the same number for each experimental trial. The graph in Fig. 6.18 shows the effect of using different numbers of RANSAC iterations; see Table 6.2 for a description of the data sets used. As expected, using more RANSAC iterations increased the likelihood of success; we chose to use 50 iterations for most of the experiments in this section.

Each RANSAC iteration consists of drawing three manifolds at random and then using an MCMC-based search process in an attempt to locate a mutual intersection point of the manifolds as described in Section 5.3. The search process starts at a random state representing three points

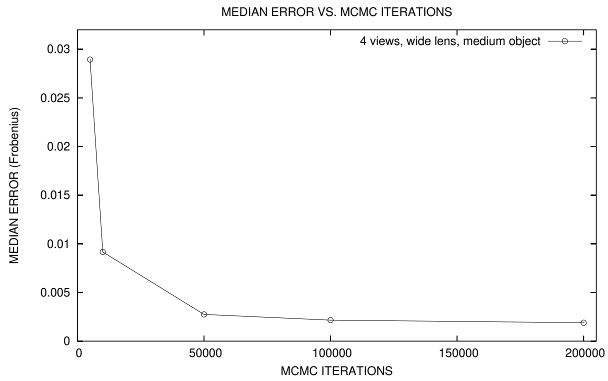

Figure 6.17 Median error for various amounts of iterations of the the MCMC loop.

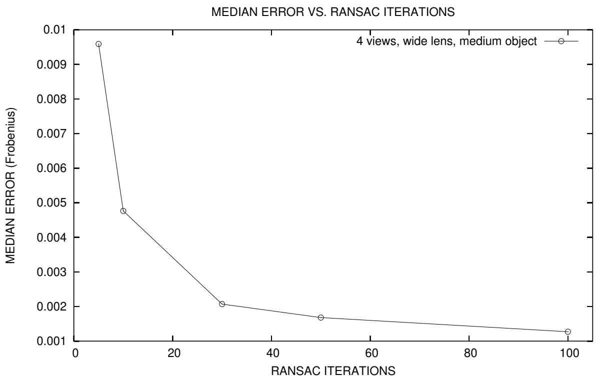

Figure 6.18 Median error for various amounts of iterations of the RANSAC loop.

in a-space (one position on each of the three manifolds). New states are explored stochastically with a tendency to explore states that have the three positions near each other in a-space (and thus potentially close to a mutual-intersection point). The number of MCMC iterations to perform is a parameter that must be tuned. Higher numbers are always better for reliability but slow down the algorithm.

The graph in Fig. 6.17 shows results from using different numbers of MCMC iterations (with a fixed number of RANSAC iterations). Most of our experiments used 100000 iterations; the graph suggests that using 200000 iterations, while doubling the runtime, would have produced little benefit in reducing error.

6.2.6 Answer to question 5: Comparison with surface-fitting algorithm

The surface-fitting algorithm will *always* produce better results than the MCMC-based algorithm, since both rely on the same underlying approximate surfaces but the surface-fitting algorithm finds all mutual intersection points while the MCMC-based algorithm may or may not find any mutual intersection points. However, the MCMC-based algorithm has several significant practical advantages and thus it is important to know how well it performs relative to the surface-fitting algorithm. The graph in Fig. 6.13 shows a direct comparison of the results of the two algorithms.

The two significant advantages of the MCMC-based algorithm are (1) it is very simple to implement regardless of the dimensionality of the manifolds and (2) it is not necessary to sample and store approximate surfaces for the manifolds since the manifolds can be sampled on the fly (albeit with a significantly slower runtime). The second property is extremely important when intersecting higher-dimensional manifolds, where storage and preprocessing time for creating approximate surfaces might become infeasible. It is also important for potential implementation of the algorithm on low-powered, portable platforms which may not have much memory. Potentially, the slow down in runtime could be offset by using many low-powered microprocessors working in parallel because the algorithm is arbitrarily scalable.

6.3 Self calibration from three views

In this section, we explain why Question 4 from Section 6.1 is important. Recall that each screw-transform manifold in a-space is 2-dimensional while a-space itself is 3-dimensional. Therefore the intersection of three or more manifolds will, in general, be a set of discrete points. One of these points represents the original internal calibration of the camera, while the others are either meaningless or provide alternative, legal internal calibrations for the camera. If it turns out that the intersection of three manifolds always yields a set with only one point, then self calibration can be achieved from just three camera views. Alternatively, if the number of mutual intersection points is large then it will probably be necessary to use more than three camera views in order to make this set smaller. If the number of mutual intersection points is always 2, for instance, it might suggest that every camera calibration has a "dual" calibration that is consistent with the observed data (i.e., the set of fundamental matrices). In other words, the self calibration problem may not have a unique solution until a sufficient number of camera views are incorporated.

The number of mutual intersection points (i.e., the number of legal camera calibrations) is critical when solving for calibration by minimizing an error function. This is because:

- The optimization algorithm (e.g., Levenberg-Marquardt or gradient descent) needs to be run at least one time for each global minimum, where each global minimum corresponds to a legal internal calibration.

- The optimization algorithm will probably need to be run more than one time, using different starting locations, to find some of the global minima. In particularly bad cases, such as when a global minimum has a small attraction basin, the optimization algorithm may need to be run hundreds or thousands of times and may still not succeed.

- Without knowing how many mutual intersection points exist, it is impossible to know when enough global minima have been found for the search process to stop.

- With noise-free data, all legal internal calibrations correspond to global minima of the error function. Because of noise in real data, however, some legal internal calibrations may correspond to local minima that are not global minima. Knowing how many legal calibrations are supposed to exist could help differentiate between meaningless local minima and local minima that correspond to legal internal calibrations. For example, if it is known that n legal calibrations are supposed to exist, the algorithm could return the n local minima with the smallest error measure.

- With RANSAC, the smaller the sample size the faster and more reliably the algorithm works. If using three camera views leads to only 1 or 2 possible internal calibrations then RANSAC can be used with a sample size of 3. Otherwise, it may be necessary to use 4 or more views in each sample to get a unique calibration and RANSAC may become infeasible (the algorithm may run too slowly or there may not be enough inlying fundamental matrices for a good sample).

These considerations are of particular importance because *almost all* existing self calibration algorithms rely on minimizing an error function to find calibration (e.g., [41, 71, 77, 181, 133, 132, 100, 53], to cite just a few; also, the MCMC-based algorithm of Section 5.3 falls into this category).

Note that the surface-fitting method of Section 5.1 is not an energy-minimization method and does not suffer from the drawbacks listed above. The surface-fitting method determines every mutual intersection point in one pass, which makes it ideal for experimentally determining the number of mutual intersection points in the three-view case.

Pollefeys [133] pointed out that, for three camera views, there is clearly an upper bound of 64 mutual-intersection points. By eliminating some impossible cases, Schaffalitzky [151] reduced the number to 21. Even this latter number is large enough to make self calibration from three views seem infeasible. However, our experiments using the surface-fitting method suggest that there are, in most instances, only 1 or 2 mutual intersection points (see Fig. 6.7 and the experiments of Section 6.1.5) and thus self calibration from three views is feasible. The experiment shown in Fig. 6.11 was performed using only three views.

Chapter 7

Calibration and Image-Based Rendering for Dynamic Scenes

7.1 Overview

Up until this point we have only been considering scenes that contain no motion, which are called *static scenes*. A scene can be considered static if all camera views are captured simultaneously; alternatively, a static scene is one that does not evolve over time. The opposite of a static scene is a *dynamic scene*, a scene that contains motion or changes over time. As will be seen, dynamic scenes present new challenges that researchers are just beginning to investigate. They also present new opportunities for gaining information about a scene or the cameras viewing the scene.

The standard term for scene reconstruction is "structure from motion," but this is not to be confused with the kind of motion we consider in this chapter. We can define "structure from motion" as follows:

Structure from motion: *Reconstruction of a rigid scene using changes in the way the scene looks in camera views due to either (1) a change in camera positions or orientations, (2) a displacement of the scene, or (3) a combination of (1) and (2).*

Thus the "structure from motion" problem can actually apply to static scenes as long as the camera moves (and thereby provides the "motion"). In constrast, the issues discussed in this chapter concern dynamic scenes almost exclusively. This is because we will consider dynamic scenes that have several different objects undergoing different motions, and thus there will be no way to substitute motion of the camera for motion of the objects.

Each moving object in a scene represents a separate, classical structure-from-motion problem. For instance, camera calibration might be determined separately from each moving object and then the results might be adjusted to arrive at a solution that is optimal for all the objects simultaneously. However, the movement of the scene objects *relative to each other* presents a new source of information for calibration or reconstruction. This extra information is used in Section 7.2 in conjunction with screw-transform manifolds and in Section 7.3 to produce a linear algorithm for affine self calibration.

One major use of camera calibration and scene reconstruction is to produce new views of a scene from novel vantage points (i.e., physically-correct views that are different from the input views). However, in some cases new, physically-correct views of a scene can be created without determining camera calibration (beyond the projective level) or performing scene reconstruction. In Section 7.4 and Section 7.5 we look at methods for interpolating between views of a dynamic scene; Section 7.5 addresses dynamic scenes that contain several moving objects following apparent straight-line trajectories, and Section 7.5 addresses dynamic scenes that contain rotating objects.

By examing view interpolation performed without camera calibration, we see how much more is possible when camera calibration can be determined. As will be seen, uncalibrated view interpolation severely limits control of the position and orientation of the virtual camera, and can even require the virtual camera to change over time, introducing undesirable distortion. With dynamic scenes, some calibration information may be necessary just to produce constant velocities in the moving objects. Furthermore, only a few special-case dynamic scenes can be interpolated without camera calibration, and specialized algorithms and preconditions are required for each case. In contrast, when camera calibration is known then scene reconstruction becomes possible, leading to unlimited control over camera and scene motion. These observations add further evidence that camera calibration is a necessity for useful IBR techniques.

7.2 Using screw-transform manifolds with dynamic scenes

We begin the discussion of dynamic scenes by showing one way to use the theory of screw-transform manifolds in conjunction with dynamic scene data. Let two fixed cameras A and B, possibly with different internal parameters, view a moving object. Assume the object has a rigid structure but can undergo any displacement. Let each camera capture a view of the moving object at time $t = 0$ and $t = 1$, leading to 4 reference views in all. Alternatively, we can think of cameras A and B as being in a stereo rig which is moved through a static scene.

We can think of the 4 views as corresponding to 4 separate cameras in fixed positions. Using the fundamental matrices between the 4 cameras, we can place all the cameras in the same projective basis. Note that there are only 5 distinct fundamental matrices induced by this system. Furthermore, two of the fundamental matrices are monocular (i.e., are generated by two views that share the same internal parameters). Each monocular fundamental matrix induces a screw-transform manifold in a-space. Note that it is legitimate to refer to a single a-space despite having two distinct internal parameters (i.e., from camera A and camera B) because all cameras have been placed in the same projective basis. The vector $\breve{\mathbf{a}}$ in a-space that will upgrade the projective reconstruction to affine lives in the intersection of the two screw-transform manifolds. Unfortunately, this intersection is 1-dimensional (because each manifold is 2D and a-space is 3D).

There is a clever way to further limit the location of $\breve{\mathbf{a}}$. Each screw-transform manifold is parameterized by 2 parameters: θ and κ. Consider creating a new search space called a,θ-space by adding an extra dimension to a-space that will correspond to the θ coordinate. For example, if (κ, θ) is mapped to position (x, y, z) in a-space, then (κ, θ) will be mapped to position (x, y, z, θ) in a,θ-space. The two screw-transform manifolds under consideration induce a pair of 2D manifolds in a,θ-space through this mapping. Note that the θ coordinate of $\breve{\mathbf{a}}$ on both manifolds in a-space must be the same because the moving scene object undergoes the same rotation as it is viewed by both cameras A and B.[1] This means that the intersection of the screw-transform manifolds in a,θ-space will contain a point corresponding to $\breve{\mathbf{a}}$; the θ coordinate of this point will be the rotation angle of

[1] This is not true for γ, the amount of screw displacement of the object, which is measured as a multiple of the distance between each camera and the axis of screw rotation and is thus camera dependent.

the moving object. Furthermore, note that the intersection of two 2D manifolds in 4D space is a 0-dimensional set (i.e., a set of discrete points) when the manifolds are in general configuration. Thus affine calibration can be determined by searching in a,θ-space rather than a-space. Given that the two manifolds under consideration intersect in a 1D manifold in a-space, it might be expected that their images in a,θ-space also intersect in a 1D manifold making the problem indeterminate. However, it has been shown that this problem, in the form a stereo-rig problem, has a unique solution [201]. A key difference between the technique of this section and that of Zisserman et al. is that the technique presented here works entirely from fundamental matrices.

7.3 Linear algorithm for affine self calibration from scene motion

7.3.1 Introduction

In this section, we present a linear algorithm that utilizes the relative motion of objects in a dynamic scene to determine the *affine calibration* between two cameras viewing the scene [112]. That is, the algorithm finds the homography induced by the plane at infinity between two views of the scene. Among other things, knowledge of affine calibration can be used for affine scene reconstruction and as an intermediate step in metric self-calibration.

The algorithm finds affine calibration directly from the fundamental matrices associated with moving objects. At least two fundamental matrices are required, but additional ones can be incorporated naturally into the linear system, providing greater numerical stability. If the two cameras have different optical centers, then the stationary background elements of the scene give rise to the standard fundamental matrix, which can also be incorporated into the linear system.

Although two views of a moving rigid-body object will usually give rise to a fundamental matrix, the matrix can only be used by our algorithm if the object's motion meets certain conditions. The simplest form of these conditions is that the object must undergo a rigid translational motion. However, since only two views of the scene are actually used by our algorithm, this basic condition can be generalized. First, notice that the two views must be captured at different times for the dynamic nature of the scene to be relevant. Consequently, there is a missing interval of time between when the views are captured. During this missing interval, the object can undergo *any* motion as long as the total change in the object and its location is equivalent to a single, rigid translational motion. When this condition is met, we say the scene has *apparent linear motion*.

The term *object* has a specific meaning in this section, defined by the general condition just given: An object is a group of particles in a scene for which there exists a fixed vector $\mathbf{u} \in \mathbb{R}^3$ such that each particle's total motion during the missing time interval is equal to $\mathbf{u}$. Throughout this section, objects will be assigned numbers and the notation $\mathbf{u}^i$ will represent the motion vector for object i.

The problem of finding the affine calibration between two views has been widely studied and is of great use in machine vision. For example, once the affine calibration has been recovered, affine scene reconstruction is immediately possible (e.g., by triangulation, or see [45]). Among other things, affine reconstruction can be used for affine, model-based object recognition, tracking, augmented reality, feature transfer, and novel view generation in image-based rendering. Finding affine calibration is also an essential intermediate step in the stratified approach to metric self-calibration [4, 201, 35, 135, 45]. For instance, if three views of a scene are available that have all been captured by the same camera with constant internal parameters and if affine calibration can be recovered for each pair of views, then the metric calibration of the camera can be determined [133, 131]. In the realm of pure image-based rendering, it has been shown [109] that affine calibration can be used to directly generate linear interpolation sequences for dynamic scenes having apparent linear motion without the need for scene reconstruction.

Various techniques for finding the affine calibration between pairs of views have been published. Several authors [184, 4] used the fact that if two views are captured by a fixed camera undergoing a rigid translational motion, then the infinity homography between the views is known to be the identity matrix. Faugeras [45] described an alternative approach to affine calibration that also involves pure translational motion. Other techniques [12, 5] have been developed for the restricted case of planar camera motion, that is, for when the camera's internal parameters do not change and the camera only undergoes translations and rotations that are parallel to a fixed plane. None of these techniques are directly related to dynamic scenes, and they all place restrictions on camera motion; our technique places restrictions on object motion but not camera motion.

The most direct method for finding affine calibration is to identify four conjugate directions (i.e., points on the plane at infinity) that are not all coplanar; like all planar homographies, the infinity homography is completely determined by its behavior on four points [45]. Pollefeys demonstrated that affine calibration between two views taken by the same camera can be determined from just two conjugate directions if the modulus constraint is utilized [133]. Since one conjugate direction can be determined from the motion of each moving object in the scene, these techniques might be applicable when two or more moving objects are present. However, the technique presented in this section is usable even when only one moving object is present (because the static background can provide the second necessary fundamental matrix); additionally, the cameras can be different in our approach.

The technique presented by Zisserman et al. [201] and later expanded upon by Horaud et al. [78] applies, in general, to a different class of problems than our technique and uses a completely different mathematical approach. Zisserman's algorithm is for a stereo rig viewing a static scene from two different locations and is mathematically based upon projective reconstruction of conjugate points. In contrast, our technique works directly from fundamental matrices without any need for reconstruction; thus additional errors introduced during projective reconstruction (e.g., errors introduced through triangulation) are avoided. Furthermore, in our technique it is not strictly necessary to identify conjugate points at all if the fundamental matrices can be determined by some other means. For example, Stein [171] presented a direct method for finding the trilinear tensor between three views using optical flow; the required fundamental matrices could be determined from such a trilinear tensor [70]. While our technique could be applied to the stereo rig problem for static scenes if the rig undergoes a rigid translation (see Section 7.3.6.2), it is not possible in general to apply Zisserman's technique to the dynamic scenes considered here.

Recently, several sections have been published concerning the use of dynamic scene information for various types of calibration. Fitzgibbon and Zisserman [53] studied the problem of metric self calibration from multiple moving objects. Their techniques, however, are presented as nonlinear minimization problems whereas the technique we present is linear. More closely related to the problem presented here, Shashua and Wolf [161] developed a technique for finding the *dual Htensor* between three views of a dynamic scene in which all the objects move along straight-line paths. Their algorithm is linear and, in principal, the dual Htensor could be used to find the relative calibration between any two views. However, the optical centers of the views must lie on the same line or alternatively the entire scene and all the scene motion must be in a single plane. Our technique only requires two views and has no restrictions other than the apparent linear motion requirement given earlier. Finally, Stein [172] presented a method for finding the weak calibration between two widely-separated views using statistics acquired from a dynamic scene over an extended period of time. His technique is unrelated to the present work and will not be discussed further.

7.3.2 Notation and preliminary concepts

Assume two camera views are captured at times $t = 0$ and $t = 1$ using pinhole cameras, which are denoted *camera* A and *camera* B, respectively. In this section, a *fixed-camera formulation* is used, meaning the two cameras are treated as if they are at the same location and the world is moving around them; this is accomplished by subtracting the displacement $\mathbf{e}$ between the two cameras from the motion vectors $\mathbf{v}^i$ of all objects in the scene. In the reformulated scene, object i moves by $\mathbf{u}^i = \mathbf{v}^i - \mathbf{e}$ and what had been the stationary background becomes an object that moves by $-\mathbf{e}$. Under the fixed-camera formulation, the camera matrices are just 3×3 and thus each camera represents a basis for $\mathbb{R}^3$. The basis induced by camera A will be called basis A, and so on. We reiterate that, although we choose to reinterpret the cameras as sharing the same optical center, in actuality the cameras can be at different locations and can be completely different internally.

Note that the vectors $\mathbf{e}$, $\mathbf{u}^i$, and $\mathbf{v}^i$ are Platonic. In this section, when we wish to express such a vector in a particular basis we will simply use a subscript rather than curly braces and a subscript; hence, for $\mathbf{e}$ in the basis of camera B we use $\mathbf{e}_B$ instead of $\{\mathbf{e}\}_B$. Also note that we will not be distinguishing between Platonic directions and Platonic positions.

If cameras A and B are at different locations in the original scene, then $\mathbf{e}$ is nonzero and there exists a fundamental matrix $\mathbf{F}$ for the cameras which has the following representation (by Eq.

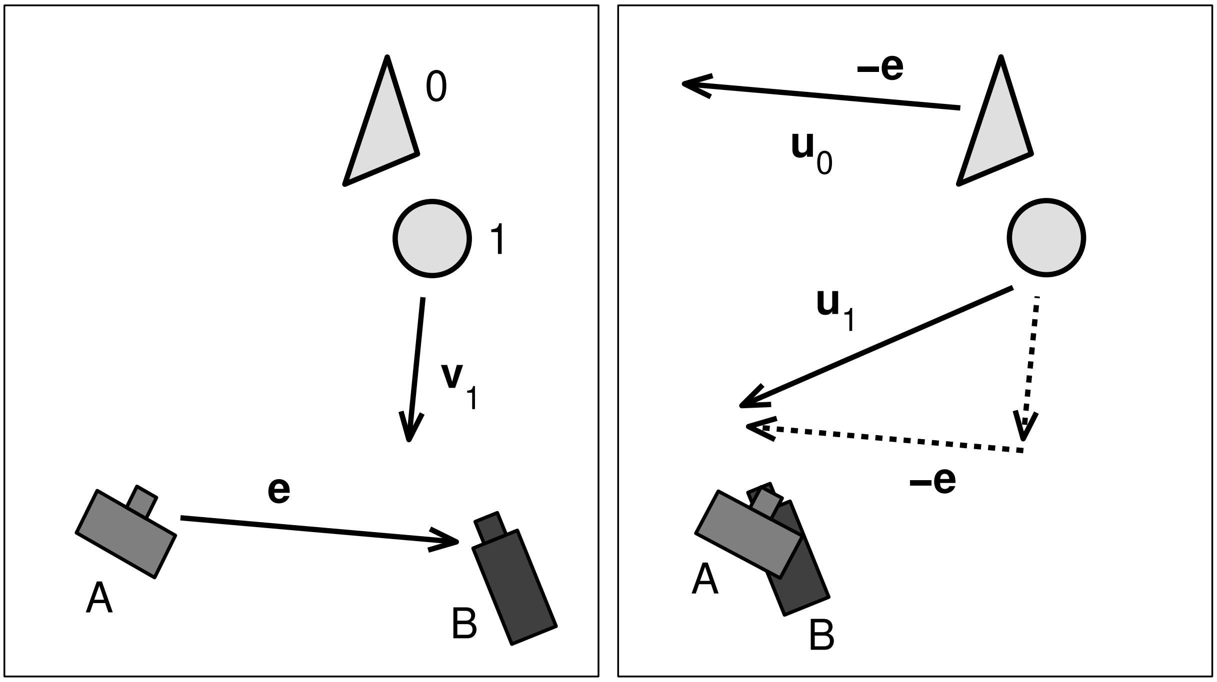

Figure 7.1 Example of the fixed-camera formulation. *(left)* Two different cameras A and B view a dynamic scene from different positions. Camera A captures a view at time $t = 0$; camera B at time $t = 1$. The scene has one moving object (labeled 1) and one stationary object (labeled 0). Object 1 translates by $\mathbf{v}_1$ between time $t = 0$ and $t = 1$. The displacement between the two optical centers is $\mathbf{e}$. *(right)* The same two views would have been captured under this alternative scenario: The two cameras share the same optical center, object 0 translates by $\mathbf{u}_0 = -\mathbf{e}$, and object 1 translates by $\mathbf{u}_1 = \mathbf{v}_1 - \mathbf{e}$.

3.10):

$$\mathbf{F} = [\mathbf{e}_B]_\times \mathbf{H}^\infty_{AB} \tag{7.1}$$

Here $[\cdot]_\times$ denotes the cross product matrix and $\mathbf{H}^\infty_{AB}$ is the homography induced by the plane at infinity, the quantity we seek to calculate. When the two cameras share the same optical center, the fundamental matrix is $\mathbf{0}$ and has no meaning. However, for each moving object i in the scene, we can define a new kind of fundamental matrix. If, after switching to the fixed-camera formulation, object i is moving in direction $\mathbf{u}^i$, then the fundamental matrix *for the object* is:

$$\mathbf{F}_i = [\mathbf{u}^i_B]_\times \mathbf{H}^\infty_{AB} \tag{7.2}$$

The epipoles of $\mathbf{F}_i$ are the vanishing points of object i as viewed from the two cameras, and the epipolar lines trace out trajectories for points on object i.

Notice that, under the fixed-camera formulation, the stationary background in the original scene becomes just another moving object (provided $\mathbf{e}$ is nonzero). Hence by using the fixed-camera formulation we are able to create a single mathematical theory that applies to pairs of cameras at different locations as well as to pairs of cameras that share the same optical center (e.g., two views from a single camera that is undergoing a zoom or rotating around its optical center).

7.3.3 Motion-based affine calibration

We now show how affine calibration can be computed directly from the motion of two scene objects that are not moving parallel to each other. Let the two objects be indexed by the set $\{0, 1\}$ and consider Eq. 7.2. Observe that $\mathbf{H}^\infty_{AB}$ is a rank three invertible matrix, but $[\mathbf{u}^i_B]_\times$ is rank two, and consequently $\mathbf{F}_i$ is also rank two. Because of the rank deficiency in $[\mathbf{u}^i_B]_\times$, the following arises: Let $S_i = \{\mathbf{M} \in \mathbb{R}^{3\times 3} : \mathbf{F}_i = [\mathbf{u}^i_B]_\times \mathbf{M}\}$. Then S_i is a 4-dimensional vector space over the real numbers; specifically, a basis for S_i is given by the matrices $\mathbf{p}^i_0, \mathbf{p}^i_1, \mathbf{p}^i_2, \mathbf{p}^i_3 \in \mathbb{R}^{3\times 3}$, where $\mathbf{p}^i_0 = \mathbf{H}^\infty_{AB}$ and

$$\mathbf{p}^i_1 = [\mathbf{u}^i_B, \mathbf{0}, \mathbf{0}],\ \ \mathbf{p}^i_2 = [\mathbf{0}, \mathbf{u}^i_B, \mathbf{0}],\ \ \mathbf{p}^i_3 = [\mathbf{0}, \mathbf{0}, \mathbf{u}^i_B]$$

Because $\mathbf{H}^\infty_{AB}$ is in the basis of both S_0 and S_1, and because $\mathbf{u}_0$ and $\mathbf{u}_1$ are not parallel, $S_0 \cap S_1 = \langle\, \mathbf{H}^\infty_{AB} \,\rangle$, where $\langle \cdot \rangle$ denotes the subspace generated by a set of vectors. Since we only need to find

$\mathbf{H}_{AB}^{\infty}$ up to a scalar, we only need to find one nonzero element in the intersection of S_0 and S_1. This is accomplished by first finding *any* two matrices $\mathbf{p}_4^i$ such that

$$\mathbf{F}_i = [\mathbf{u}_B^i]_\times \mathbf{p}_4^i \tag{7.3}$$

Next, notice that S_i is spanned by $\mathbf{p}_1^i$, $\mathbf{p}_2^i$, $\mathbf{p}_3^i$, and $\mathbf{p}_4^i$ (because if $\mathbf{p}_4^i$ is in $\langle\ \mathbf{p}_1^i, \mathbf{p}_2^i, \mathbf{p}_3^i\ \rangle$, then $[\mathbf{u}_B^i]_\times \mathbf{p}_4^i = 0$). Consequently, there exist scalars $k_1, \ldots, k_8$ such that

$$\begin{aligned} \mathbf{H}_{AB}^{\infty} &= -k_1\mathbf{p}_1^0 - k_2\mathbf{p}_2^0 - k_3\mathbf{p}_3^0 - k_4\mathbf{p}_4^0 \\ &= \quad k_5\mathbf{p}_1^1 + k_6\mathbf{p}_2^1 + k_7\mathbf{p}_3^1 + k_8\mathbf{p}_4^1 \end{aligned} \tag{7.4}$$

The second equality in Eq. 7.4 means that

$$\left[\ \mathbf{p}_1^0\ \ \mathbf{p}_2^0\ \ \mathbf{p}_3^0\ \ \mathbf{p}_4^0\ \ \mathbf{p}_1^1\ \ \mathbf{p}_2^1\ \ \mathbf{p}_3^1\ \ \mathbf{p}_4^1\ \right]\left[\ k_1\ \ k_2\ \ \ldots\ \ k_8\ \right]^\top = \mathbf{0} \tag{7.5}$$

Here we treat the matrices $\mathbf{p}_j^i$ as column vectors in $\mathbb{R}^9$. The above can be solved using standard techniques from linear algebra (e.g., singular value decomposition to find the eigenvector of eigenvalue 0). Once the k_i's are found, we can find $\mathbf{H}_{AB}^{\infty}$ (up to a scalar) using Eq. 7.4.

Formally, we must show that the left-most matrix in Eq. 7.5 has rank 7. The rank is less than 8 since Eq. 7.4 has a solution. The vectors $\mathbf{p}_1^0$, $\mathbf{p}_2^0$, $\mathbf{p}_3^0$, $\mathbf{p}_1^1$, $\mathbf{p}_2^1$, and $\mathbf{p}_3^1$ clearly form a linearly independent set because $\mathbf{u}_0$ and $\mathbf{u}_1$ are not parallel. If $\mathbf{p}_4^1 = h_1\mathbf{p}_1^0 + h_2\mathbf{p}_2^0 + h_3\mathbf{p}_3^0 + h_4\mathbf{p}_1^1 + h_5\mathbf{p}_2^1 + h_6\mathbf{p}_3^1$ for some scalars h_i, then by Eq. 7.3, $\mathbf{F}_1 = [h_1\mathbf{u}_2, h_2\mathbf{u}_2, h_3\mathbf{u}_2]$ where $\mathbf{u}_2 = \mathbf{u}_1 \times \mathbf{u}_0$. This is a contradiction since $\mathbf{F}_1$ has rank 2, not rank 1. Thus 7 of the column vectors are linearly independent.

Because of the reliance on the linear independence of the column vectors in Eq. 7.5, it is crucial that $\mathbf{u}_0$ and $\mathbf{u}_1$ be linearly independent; the algorithm becomes unstable as the two objects move in nearly parallel directions.

7.3.4 Generalizing to multiple objects

If more than two moving objects are present in the scene, then the mathematics presented above can be generalized to incorporate each object's fundamental matrix simultaneously into one large, linear system.

Let the objects be numbered 0 to $n-1$. Let $\mathbf{P}(i)$ denote the 9×4 matrix

$$[\mathbf{p}_1^i \; \mathbf{p}_2^i \; \mathbf{p}_3^i \; \mathbf{p}_4^i]$$

and let $\mathbf{0}_{9\times4}$ denote the 9×4 matrix filled entirely with 0's. We construct a matrix $\mathbf{M}$ by the following method:

Start with $\mathbf{M}$ *equal to the null matrix. For each* $i \in \{0, \ldots, n\text{-}2\}$ *and* $j \in \{i\text{+}1, \ldots, n\text{-}1\}$ *such that* $\mathbf{u}_i$ *and* $\mathbf{u}_j$ *are not parallel, enlarge the matrix* $\mathbf{M}$ *by appending the following* $9 \times n$ *matrix to its bottom:*

$$[\, \overbrace{\mathbf{0}_{9\times4}, \ldots, \mathbf{0}_{9\times4}}^{i-1}, \mathbf{P}(i), \overbrace{\mathbf{0}_{9\times4}, \ldots, \mathbf{0}_{9\times4}}^{j-i-1}, -\mathbf{P}(j), \overbrace{\mathbf{0}_{9\times4}, \ldots, \mathbf{0}_{9\times4}}^{n-j} \,] \tag{7.6}$$

Once $\mathbf{M}$ has been constructed, the following system is solved (e.g., by singular value decomposition):

$$\mathbf{M}\,[\, k_1 \; k_2 \; \ldots \; k_{4n} \,]^\top = \mathbf{0}$$

Affine calibration can now be determined from the following, which holds for every $i \in \{0, \ldots, n-1\}$:

$$\mathbf{H}_{AB}^{\infty} = k_{4i+1}\mathbf{p}_1^i + k_{4i+2}\mathbf{p}_2^i + k_{4i+3}\mathbf{p}_3^i + k_{4i+4}\mathbf{p}_4^i \tag{7.7}$$

7.3.5 Experiments with synthetic data

Extensive experiments with synthetic data were conducted to test the approach. In this section, we summarize the experimental method and present the results.

7.3.5.1 Experimental procedure

The general pattern for each trial run was as follows: Two or more objects were generated and a random translation was assigned to each object. Two cameras with random internal parameters were created and randomly positioned so that both objects at time $t = 0$ were visible in the first camera and both objects at time $t = 1$ were visible in the second camera. Next, noise was added to the projected points on each image plane and then the method described in Section 7.3.4 was used to recover the affine calibration between the cameras.

For different trials, the overall scale of each object was magnified or reduced, the distance that the objects moved was scaled by different amounts, and the amount of noise was varied. The error in the recovered $\mathbf{H}^{\infty}_{AB}$ was measured using the following error metric:

Error Metric: Treating the matrices as vectors in $\mathbb{R}^9$, with vectors $\mathbf{p}$ and $\mathbf{q}$ denoting the calculated $\mathbf{H}^{\infty}_{AB}$ and the true $\mathbf{H}^{\infty}_{AB}$, the error was calculated as:

$$1 - \frac{|\mathbf{p} \cdot \mathbf{q}|}{\|\mathbf{p}\| \, \|\mathbf{q}\|}$$

Note that this quantity is $1 - |\cos(\theta)|$, where θ is the angle between the vectors. An error metric based on the Frobenius norm would have represented the distance between the two matrices as points in $\mathbb{R}^9$ and thus two matrices that were almost equal except for an overall sign factor would have erroneously had a large error. We avoid this issue of overall sign by using the cosine of the angle between the matrices, thus measuring parallelness.

Each object consisted of up to 100 points selected randomly in a unit sphere such that the density of points was uniform throughout the sphere. The internal parameters of the cameras were randomly generated within ranges that are realistic for actual cameras. Each image was size 640×480 pixels; this fact is crucial for interpreting the results that follow since measurements (e.g., noise added) will often be given in pixels.

Within the framework outlined above, three different scenarios were created to simulate different conditions under which the algorithms of this section might be used in practice:

Scenario I: At time $t = 0$, the objects are near each other in space; by time $t = 1$ the objects have moved (each in an arbitrary direction) and are viewed by camera B, which is near camera A in space.

Scenario II: Objects are located arbitrarily within a circle and are only allowed to move parallel to the plane of the circle. Cameras are positioned randomly along the outside of the circle at a higher elevation than the objects.

Scenario III: Objects are located within a sphere of radius five units, and cameras are located in a larger, co-centered sphere but outside the sphere of the objects. Objects can move in any direction.

The positioning of cameras A and B close together in Scenario I simulates a hand-held camera, where the camera might not travel very far between views compared to how far the objects travel.

Scenario II simulates a “parking lot” where vehicles drive along the flat surface of the lot and surveillance cameras are positioned on buildings around the lot. Scenario III tests the algorithm under fully general conditions. Note that, when only two objects are used (as was usually the case), each scenario corresponds to the motion of vehicles over level terrain because two vectors are always mutually parallel to some plane.

One final detail should be noted: Noise was added to the projected points on the retinas in such a way that no outliers were created. Specifically, if the trial run called for an average of ν pixels of noise, then uniform noise with a radius of 2ν pixels was added to each point; thus no conjugate point was more than 2ν pixels from its true position on the retina. It is assumed that, in practice, outliers would be removed by earlier steps in processing. Because of the lack of outliers, accurate fundamental matrices could be found by a normalized linear method [73].

7.3.5.2 Results

Table 7.1 shows how calibration error was related to the number of conjugate points and to the average amount of noise added to each conjugate point. As would be expected, error decreases as the number of conjugate points increases and as the amount of noise decreases. The large standard deviations stem from occasional outliers; the scatter graphs in Fig. 7.2 give a visual indication of how the error values are distributed.

Recall that the algorithm becomes unstable as the objects move more parallel to each other in 3D when considered under the fixed-camera formulation. This instability is demonstrated in Fig. 7.2(a). Notice that there are few outliers for angles above approximately 20°. Thus for the remaining scatter graphs as well as for the table just presented, trials in which the angle between the object motion vectors was less than 20° were eliminated. For every trial in all the scatter graphs, 100 conjugate points were used per object and an average of 1.25 pixels of noise was added per point.

The scatter graph in Fig. 7.2(b) shows how error is reduced as noise is reduced. Notice that there are some outliers even at small noise levels, but the general trend is clear.

CALIBRATION ERROR					
	average noise added per point				
	5.003 pixels σ=0.261	2.500 pixels σ=0.130	1.250 pixels σ=0.065	0.500 pixels σ=0.026	0.250 pixels σ=0.013
100 points	error=0.0838 σ=0.153	error=0.0338 σ=0.0839	error=0.0207 σ=0.0777	error=0.00536 σ=0.0102	error=0.00277 σ=0.00400
60 points	0.103 σ=0.167	0.0470 σ=0.109	0.0200 σ=0.0497	0.00993 σ=0.0516	0.00276 σ=0.00413
30 points	0.142 σ=0.182	0.0632 σ=0.115	0.0295 σ=0.0726	0.0125 σ=0.0384	0.00764 σ=0.0225
10 points	0.381 σ=0.276	0.230 σ=0.236	0.115 σ=0.172	0.0494 σ=0.101	0.0313 σ=0.0911

Table 7.1 Calibration error for various amounts of object points and noise levels.

CALIBRATION ERROR			
	2 objects	3 objects	4 objects
100 points	0.0207	0.0144	0.0102
60 points	0.0200	0.0169	0.0124
30 points	0.0295	0.0235	0.0218
10 points	0.1154	0.0696	0.0651

Table 7.2 The positive effect of using more moving objects.

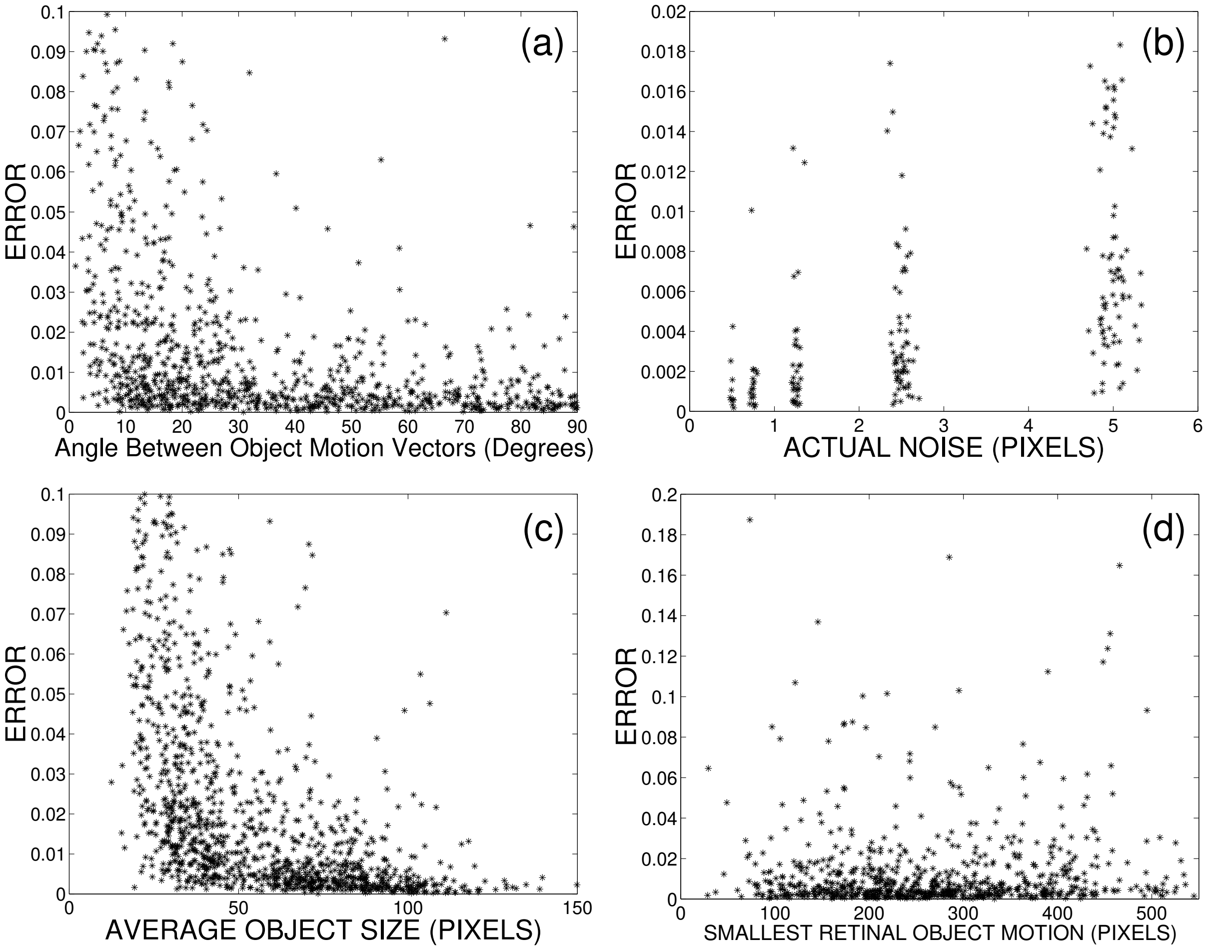

Figure 7.2 (a) Calibration error vs. angle (in degrees) between object motion vectors considered under the fixed-camera formulation; (b) calibration error vs. average noise added to each point; (c) calibration error vs. object area on the image plane; (d) calibration error vs. retinal object motion

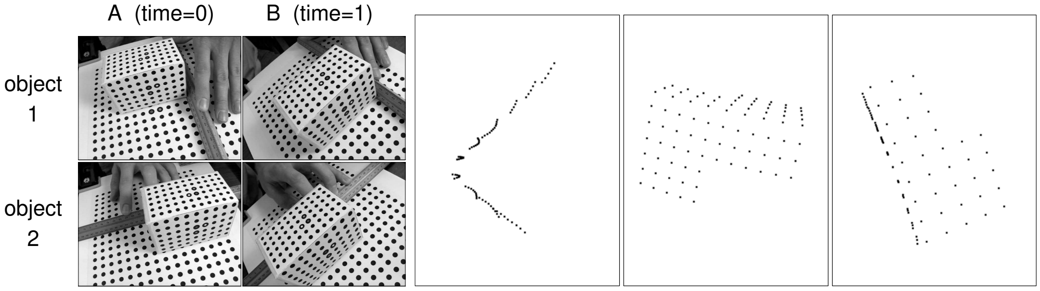

Figure 7.3 The four views on the left are the source views of the box that were used to find the two fundamental matrices for calibration. Views from camera A are on the left and views from camera B are on the right; the top pair shows object 0 moving towards the camera while the bottom pair shows object 1 moving laterally. The three rightmost views show the affine reconstruction of the box as seen from different angles.

Fig. 7.2(c) demonstrates how error is reduced as the objects appear larger on the image plane. Notice that when the average object size covers less than about 40 pixels in the image, error increases rapidly.

It might be hypothesized that the algorithm would be stabilized by greater projected object motion. However, Fig. 7.2(d) shows that error was not affected by the amount of apparent motion of the objects across the image plane, at least for the ranges tested. It would be expected, however, that as the amount of motion approached the noise level, the error would increase; this was not tested, however.

Finally, the table below shows how the result is stabilized by the use of more moving objects. Also note the improvement gained by using 30 conjugate points rather than 10; this could be due to increased stability brought on by using more conjugate points to compute the fundamental matrices.

7.3.6 Experiments with real data

In this section, we present the results from two experiments performed with real scenes.

7.3.6.1 Experiment I

The first experiment was designed to produce very reliable data. The object that was used in the experiment was covered with a regular dot pattern (see Fig. 7.3), and the center of each dot was determined to subpixel accuracy by an automatic algorithm that found the center of mass of each dot. The cameras were fixed in position throughout the experiment.

Only one actual object was used, but it was moved in two different directions and thus served as two different objects. This means the two objects were not visible at the same time, but that fact is irrelevant to the algorithm when the cameras are in fixed positions relative to each other (e.g., as on a stereo rig). This situation occurs, for example, when a pair of fixed cameras are monitoring the intersection of two roads. Occasionally, lone vehicles will cross the intersection, going in either direction. Each vehicle would give rise to a fundamental matrix, and over time the affine calibration could be accurately computed.

The ground truth affine calibration between the two views was acquired by using a three-dimensional calibration grid containing several hundred points at known positions. Each camera matrix was computed directly from the known 3D to 2D correspondences stemming from the calibration grid. Prior to this, radial distortion was corrected for as a separate step.

The ground truth affine calibration, as determined directly from the full camera matrices, was

$$\mathbf{H}^{\infty}_{AB} = \begin{bmatrix} 0.005270 & -0.002681 & 0.3752 \\ 0.002858 & 0.004966 & -0.9269 \\ 0.0000009253 & -0.0000000624 & 0.005347 \end{bmatrix}$$

while the affine calibration determined using the motion of the box was

$$\mathbf{H}^{\infty}_{AB} = \begin{bmatrix} 0.005127 & -0.002625 & 0.3773 \\ 0.002789 & 0.004809 & -0.9260 \\ 0.0000009684 & -0.0000001226 & 0.005186 \end{bmatrix}$$

The distance between the matrices, using the same error metric used for the synthetic experiments, was 2.71×10^{-6}, or about $0.13°$ when treating the matrices as vectors.

Figure 7.4 Views used in the second experiment. From left to right: the view from camera A at time $t = 0$, the view from camera B at time $t = 0$, and the view from camera A at time $t = 1$.

7.3.6.2 Experiment II

The second experiment utilized objects that had more natural texture so that fewer and less reliable point correspondences were obtained. In this experiment, several objects were placed on a piece of section such that the section could be slid across a table to simulate motion of the objects or the cameras. As before, the object was viewed by two cameras that were fixed in position throughout the experiment. The input images that were used for this experiment are shown in Fig. 7.4. Notice that the center view is zoomed in and has much less radial distortion than the left view. The left and center views, corresponding to camera A and camera B respectively, form one pair representing the object at time $t = 0$. From this pair, a fundamental matrix was recovered via standard techniques using about 30 point correspondences that were selected by hand. Next, the object was slid across the table in a manner approximating a pure translation. One final view was then captured from camera A only; this is shown as the rightmost view in Fig. 7.4. A second fundamental matrix was computed using the right and center views, again using about 30 point correspondences selected by hand.

Our algorithm was then applied to the two fundamental matrices, yielding the affine calibration

$$\mathbf{H}_{AB}^{\infty} = \begin{bmatrix} 0.351 & 0.153 & 0.196 \\ -0.433 & 0.505 & 0.151 \\ -0.222 & -0.053 & 0.546 \end{bmatrix}$$

The ground truth affine calibration was determined from vanishing points. In particular, a regular grid was viewed by both cameras as it was placed in various orientations in space. The vanishing

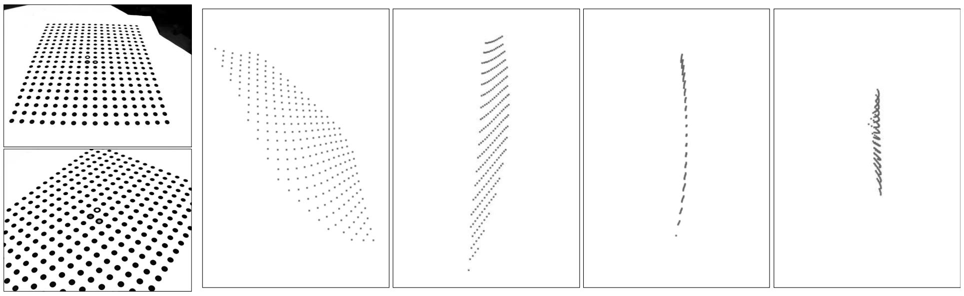

Figure 7.5 As an additional test of the affine calibration determined in the second experiment, affine reconstruction of a planar calibration grid was performed using two views of the grid *(left)*. Four views of the reconstructed surface are shown on the right.

points of this grid, found automatically by a separate program, represent conjugate directions in the two views; four such points at infinity are sufficient for finding the affine calibration and many more than four were actually used. The affine calibration thus determined was

$$\mathbf{H}_{AB}^{\infty} = \begin{bmatrix} 0.359 & 0.139 & 0.208 \\ -0.431 & 0.497 & 0.128 \\ -0.211 & -0.069 & 0.557 \end{bmatrix}$$

Again, agreement is very good despite the many potential sources of error in this experiment. The distance between the matrices, using the same error metric, was 0.000756, or about $2.2°$.

As an additional test of accuracy, the affine calibration determined by our algorithm was used to reconstruct a regular, planar grid of points that was viewed by both cameras (see Fig. 7.5). The reconstruction shows some curvature in the grid lines, probably resulting in part from residual lens distortion errors since reconstruction using the "ground truth" affine calibration yielded similar curvature artifacts. Radial distortion was prominent in camera A and less so in camera B; this distortion was corrected for as a separate preprocessing step using the same method as in the first experiment. However, some distortion seems to have remained. Despite this, we still see agreement in the two affine calibrations even though they were determined by distinct methods.

7.3.7 Conclusion

Dynamic scenes contain sources of information that are not present in static scenes, but not many methods exist to utilize this extra information. This section presented a linear algorithm that utilizes dynamic scene information to determine the affine calibration between two generally-positioned camera views. The algorithm has been shown to work on both synthetic and real data. Through experiments with synthetic data, it has been shown that the algorithm degrades gracefully with noise and the results improve as more moving objects are incorporated.

It remains to be investigated how the ideas of this section could be extended to utilize more than two views. The trilinear tensor arising from three views should stabilize the fundamental matrix calculation and improve results. Moreover, it may be possible to compute the affine calibration directly from pairs of trilinear tensors.

7.4 View interpolation for dynamic scenes with apparent linear motion

7.4.1 Introduction

View interpolation [26] involves creating a series of virtual views of a scene that, taken together, represent a continuous and physically-correct transition between two reference views of the scene. Previous work on view interpolation has been restricted to static scenes. Dynamic scenes change over time and, consequently, these changes will be evident in two reference views that are captured at different times. Therefore, view interpolation for dynamic scenes must portray a continuous change in viewpoint *and* a continuous change in the scene itself in order to transition smoothly between the reference views (Fig. 7.6).

Our approach to this problem is based upon an earlier technique called *view morphing* [154], which provides a method for interpolating between two widely-spaced views of a static scene. The technique has several strengths that make it suitable for practical applications. First, only two reference views are assumed. Second, it does not require that camera calibration be provided nor does it need to calculate the camera parameters. Third, the method works even when only a sparse set of correspondences between the reference views is known. If more information about

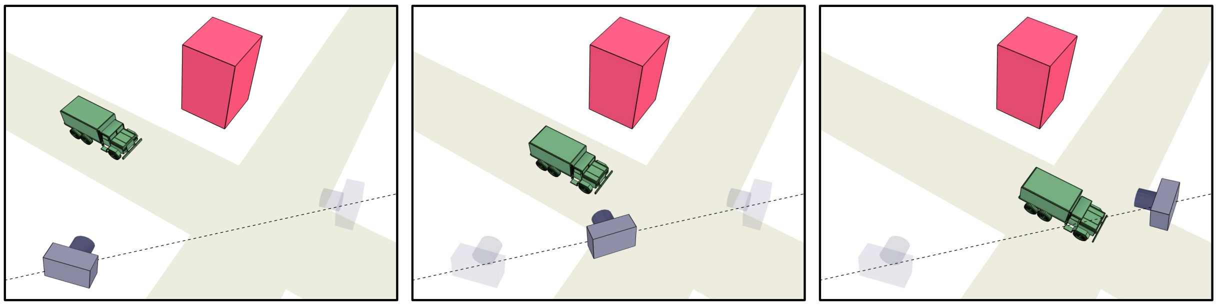

Figure 7.6 A dynamic scene at three different times. The goal of view interpolation for dynamic scenes is to synthesize the view from the camera in the middle frame starting with only the two reference views from the cameras in the left and right frames.

the reference views is available, this information can be used for added control over the output and for increased realism.

In addition to view morphing, numerous existing methods could be used to create view interpolations for static scenes [44, 117, 6, 156, 179]. However, none of these methods is directly applicable to dynamic scenes. Avidan and Sashua [7] provide a method for recovering the geometry of dynamic scenes in which the objects move along straight-line trajectories. Once the geometry is recovered, dynamic view interpolations could be created using the standard graphics pipeline. However, their algorithm does not apply to the problem discussed in this section because it assumes that five or more views are available and that the camera matrix for each view is known or can be recovered. There are several mosaicing techniques for dynamic scenes [83, 28], but mosaicing involves piecing together several small-field views to create a single large-field view, whereas view interpolation involves synthesizing new views from vantage points not in the reference set.

Because the original view morphing algorithm assumes a static scene, we refer to it as *static view morphing* to distinguish it from the *dynamic view morphing* technique presented in this section.

We seek to perform view interpolation directly from the reference views, without additional information about the scene. Consequently, there will be a missing interval of time between when the reference views were captured, and it will be impossible to know for certain what occurred during the missing interval. It is not our goal in this work to try and deduce the most likely manner in which the scene changed. Instead, we are interested in portraying *some* possible way in which the scene could have changed, and we want the portrayal to be physically correct and continuous.

Our method is for dynamic scenes that satisfy the apparent-linear-motion assumption: For each object in the scene, all of the changes that the object undergoes during the missing time interval, when taken together, are equivalent to a single, rigid translation.

The term *object* has a specific meaning in this section, defined by the condition given above: An object is a group of particles in a scene for which there exists a fixed vector $\hat{\mathbf{u}} \in \mathbb{R}^3$ such that each particle's total motion during the missing time interval is equal to $\hat{\mathbf{u}}$.

A method for dynamic view interpolation, even if it is physically accurate, may be unsatisfactory if it portrays objects moving along unreasonable trajectories. For instance, when portraying a car driving across a bridge, it is essential that the car stay on the bridge during the entire sequence. To address this problem, we have developed techniques for portraying both straight-line motion (in a camera-based coordinate frame) and straight-line, constant-velocity motion (in camera and world coordinate frames). For brevity, we refer to the latter style of portrayal as *linear motion*. Fig. 7.6 depicts a linear motion view interpolation.

If the reference cameras share the same position in world coordinates, then the virtual camera shares that position as well and straight-line motion relative to the virtual camera also implies straight-line motion in world coordinates. However, this may not be the case if the virtual camera moves during the view interpolation, as Fig. 7.7 demonstrates. It is easy to show that if all objects can be portrayed undergoing linear motion in camera coordinates, then the virtual camera can be considered undergoing linear motion in world coordinates, in which case all the objects will undergo linear motion in world coordinates as well.

7.4.2 Static view morphing

Static view morphing works by first prewarping the reference views to make their image planes parallel. After the prewarp, conjugate points in the two views are linearly interpolated to produce a physically-accurate new view of the scene. The new locations of the conjugate points are used to guide a morphing algorithm in filling the remainder of the virtual view. Only the interpolated

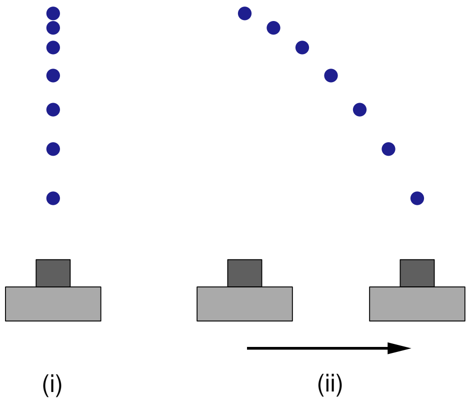

Figure 7.7 *(i)* A round object is filmed moving along a trajectory that is a straight line in the camera's frame of reference. The object is shown at equal time intervals and does not move at constant velocity. *(ii)* If the camera was in motion during the filming, then the object did not follow a straight-line trajectory in world coordinates.

conjugate points are guaranteed to be viewed in the correct, physically-accurate locations. By increasing the density of conjugate points, the virtual view can be made arbitrarily accurate. The prewarp is performed from information available in the fundamental matrix, which is calculated directly from the conjugate points. Complete details of the algorithm can be found in [154].

7.4.3 Dynamic view morphing

7.4.3.1 Preliminary concepts

We assume the two reference views are captured at time $t = 0$ and time $t = 1$ through pinhole cameras, which are denoted *camera* A and *camera* B, respectively.

We always use a *fixed-camera formulation*, meaning we assume that the two reference cameras are at the same location and that the world moves around them; this is accomplished by subtracting the actual displacement between the two cameras from the motion vectors of all objects in the scene (Fig. 7.1). Under this assumption, the camera matrices are just 3×3 and each camera is equivalent to a basis for $\mathbb{R}^3$. Note that no assumption is made about the cameras other than that they share the same optical center; the camera matrices can be completely different.

We let W denote the world coordinate frame and use the notation $\mathbf{H}^{\infty}_{WA}$ to mean the transformation between basis W and basis A. Hence $\mathbf{H}^{\infty}_{WA}$ is the camera matrix for A. Of particular interest to our work is the matrix $\mathbf{H}^{\infty}_{AB}$. Note that capital script letters will always represent 3×3 matrices; in particular, $\mathbf{I}$ is the identity matrix.

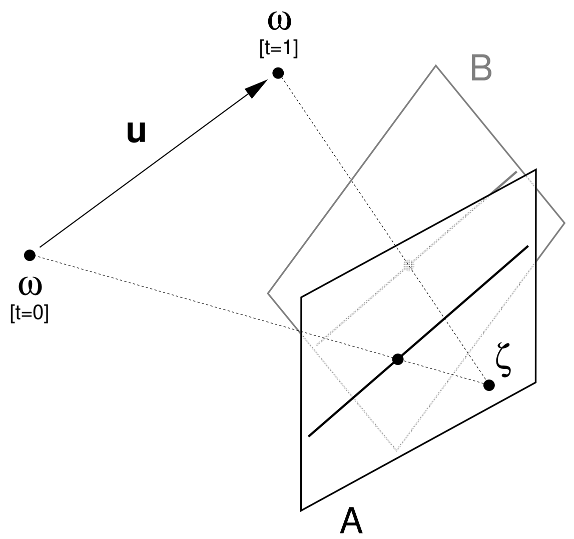

Figure 7.8 Cameras A and B share the same optical center ζ and are viewing a point on an object that translates by $\hat{\mathbf{u}}$. The image planes of the cameras are parallel to each other and to $\hat{\mathbf{u}}$, and hence interpolation will produce a physically-correct view of the object. On each image plane a line parallel to $\hat{\mathbf{u}}$ is shown.

The fundamental matrix $\mathbf{F}$ for two cameras A and B that are at different locations has the following representation (by Eq. 3.10):

$$\mathbf{F} = [\{\hat{\mathbf{e}}\}_B]_\times \mathbf{H}^\infty_{AB} \tag{7.8}$$

Here $[\cdot]_\times$ denotes the cross product matrix. When the two cameras share the same optical center, the fundamental matrix is $\mathbf{0}$ and has no meaning. However, for each moving object Ω in the scene, we can define a new kind of fundamental matrix. If, after making the fixed-camera assumption, Ω is moving in direction $\hat{\mathbf{u}}$, then the fundamental matrix *for the object* is:

$$\mathbf{F}_\Omega = [\{\hat{\mathbf{u}}\}_B]_\times \mathbf{H}^\infty_{AB} \tag{7.9}$$

The epipoles of $\mathbf{F}_\Omega$ are the vanishing points of Ω as viewed from the two reference cameras, and the epipolar lines trace out trajectories for points on Ω.

7.4.3.2 View interpolation for a single moving object

Assume the two reference cameras share the same optical center and are viewing a point ω that is part of an object Ω whose translation vector is $\hat{\mathbf{u}}$. Let $\mathbf{q}$ and $\mathbf{q} + \hat{\mathbf{u}}$ denote the position of ω at time $t = 0$ and $t = 1$, respectively (Fig. 7.8).

Assume for this subsection that the image planes of the cameras are parallel to each other and to $\hat{\mathbf{u}}$. The first half of this condition means that the third row of $\mathbf{H}^\infty_{WA}$ equals the third row of $\mathbf{H}^\infty_{WB}$

scaled by some constant λ. The second half means that $0 = \{\hat{\mathbf{u}}\}^z_A = \{\hat{\mathbf{u}}\}^z_B = (\mathbf{H}^\infty_{WA}\{\hat{\mathbf{u}}\}_W)^z = (\mathbf{H}^\infty_{WB}\{\hat{\mathbf{u}}\}_W)^z$, where $(\cdot)^z$ denotes the z-coordinate of a vector. Note that the condition can be met retroactively by using standard rectification methods [154, 107]; this is part of "prewarping" the reference views as mentioned in Section 7.4.2.

Setting $\xi = (\mathbf{H}^\infty_{WA}\{\mathbf{q}\}_W)^z = \lambda(\mathbf{H}^\infty_{WB}\{\mathbf{q} + \hat{\mathbf{u}}\}_W)^z$, the linear interpolation of the projection of ω into both cameras is

$$(1-\sigma)\frac{1}{\xi}\mathbf{H}^\infty_{WA}\{\mathbf{q}\}_W + \sigma\frac{\lambda}{\xi}\mathbf{H}^\infty_{WB}\{\mathbf{q} + \hat{\mathbf{u}}\}_W \tag{7.10}$$

Now define a virtual camera V by the matrix

$$\mathbf{H}^\infty_{WV} \stackrel{\text{def}}{=} (1-\sigma)\mathbf{H}^\infty_{WA} + \sigma\lambda\mathbf{H}^\infty_{WB} \tag{7.11}$$

Then the linear interpolation (7.10) is equal to the projection of scene point $\mathbf{q}(\sigma)$ onto the image plane of camera V (i.e., $\mathbf{H}^\infty_{WV}\{\mathbf{q}(\sigma)\}_W$ in homogeneous coordinates), where

$$\mathbf{q}(\sigma) \stackrel{\text{def}}{=} \mathbf{q} + \hat{\mathbf{u}}(\sigma) \tag{7.12}$$

$$\{\hat{\mathbf{u}}(\sigma)\}_V \stackrel{\text{def}}{=} \sigma\lambda\{\hat{\mathbf{u}}\}_B \tag{7.13}$$

Notice that $\hat{\mathbf{u}}(\sigma)$ depends only on $\hat{\mathbf{u}}$ and the camera matrices and not on the starting location $\mathbf{q}$. Thus linear interpolation of conjugate projected object points by a factor σ creates a physically-valid view (through camera V) of the entire object translated by $\hat{\mathbf{u}}(\sigma)$.

Note that in Eq. 7.13, $\hat{\mathbf{u}}(\sigma)$ is represented in basis V. Since V changes with σ it is difficult in general to characterize the trajectory $\hat{\mathbf{u}}(\sigma)$ in world coordinates. To have greater control over the interpolation process, we now prove that straight-line motion is achieved when $\{\hat{\mathbf{u}}\}_A \cong \{\hat{\mathbf{u}}\}_B$ and constant-velocity straight-line motion (i.e., linear motion) is achieved when $\{\hat{\mathbf{u}}\}_A = \lambda\{\hat{\mathbf{u}}\}_B$ (Fig. 7.9). Assume $\{\hat{\mathbf{u}}\}_A = \eta\{\hat{\mathbf{u}}\}_B$ for some scalar η. Multiplying both sides of Eq. 7.11 on the right by $\mathbf{H}^\infty_{BW}$ yields

$$\mathbf{H}^\infty_{BV} = (1-\sigma)\mathbf{H}^\infty_{BA} + \sigma\lambda\mathbf{I} \tag{7.14}$$

By multiplying both sides of Eq. 7.14 on the right by $\{\hat{\mathbf{u}}\}_B$ and on the left by $\mathbf{H}^\infty_{VB}$ the following can be derived:

$$\mathbf{H}^\infty_{VB}\{\hat{\mathbf{u}}\}_B = \frac{1}{(1-\sigma)\eta + \sigma\lambda}\{\hat{\mathbf{u}}\}_B$$

if prewarps make...	then interpolation is...
image planes parallel	physically correct
...*and* conjugate directions equal up to a scalar	...*and* depicts straight–line motion
...*and* the scalar is λ	...*and* the motion is constant–velocity

Figure 7.9 How the interpolation sequence is related to different preconditions on the reference views. Stricter preconditions lead to increased control over the output.

Multiplying both sides of Eq. 7.13 by $\mathbf{H}^{\infty}_{VB}$ now yields:

$$\{\hat{\mathbf{u}}(\sigma)\}_B = \frac{\sigma\lambda}{(1-\sigma)\eta + \sigma\lambda}\{\hat{\mathbf{u}}\}_B$$

The basis V no longer plays a role and the virtual trajectory, given by $\{\hat{\mathbf{u}}(\sigma)\}_B$, is a straight-line in basis B. If $\eta = \lambda$ then $\{\hat{\mathbf{u}}(\sigma)\}_B = \sigma\{\hat{\mathbf{u}}\}_B$ and the virtual object moves at constant velocity. The results are in basis B, but multiplying by $\mathbf{H}^{\infty}_{BW}$ or $\mathbf{H}^{\infty}_{BA}$ indicates that the results also hold in world coordinates and camera A's coordinates, thus completing the proof. Keep in mind that the world coordinate system used in this context has its origin at the shared optical center of the reference cameras.

If $\mathbf{H}^{\infty}_{BA}$ is known then the camera matrix for B can be transformed into the camera matrix for A. This allows the view from camera B at time $t = 1$ to be transformed into the view from camera A at time $t = 1$, thus producing two views of the scene from camera A at different times. For this reason, we call $\mathbf{H}^{\infty}_{BA}$ the "camera-to-camera transformation." By applying the earlier results to this special case, we derive the following corollary which forms the basis of the algorithm in Section 7.4.3.3:

If both camera matrices are equal and if $(\mathbf{H}^{\infty}_{WA}\hat{\mathbf{u}})^z = 0$, *then the camera matrix for the virtual camera* V *is just* $\mathbf{H}^{\infty}_{WA}$ *and, because* $\lambda = 1$ *and* $\{\hat{\mathbf{u}}\}_A = \{\hat{\mathbf{u}}\}_B$, *the virtual object moves at constant velocity along a straight-line path.*

7.4.3.3 Linear motion dynamic view morphing algorithm

We now present a dynamic view interpolation algorithm that will portray linear motion. The algorithm requires knowledge of $\mathbf{H}^{\infty}_{AB}$.

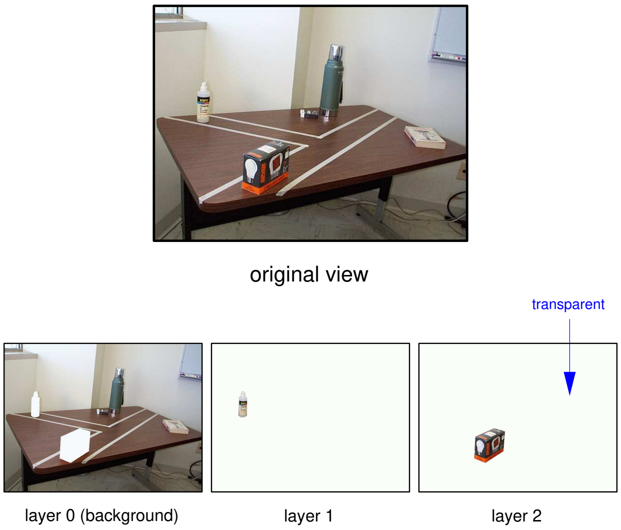

Figure 7.10 A view divided into layers. Each layer corresponds to a moving object. The single "background" object contains many different objects that all translate by the same amount.

(Step 1) *Segment both views into layers, with each layer representing a different moving object. Order the layers from nearest object to farthest object (Fig. 7.10).*

(Step 2) *Transform each layer of view B by $\mathbf{H}^{\infty}_{BA}$, thus creating a view from camera A.*

(Step 3) *Apply static view morphing to each layer separately.*

(Step 4) *Recombine the new, virtual layers in the correct depth order.*

(Step 5) *(Optional) Postwarp the new view.*

In step (3), the virtual camera will be the same for each layer by the corollary of the previous section. Furthermore, each layer will portray its corresponding object undergoing linear motion. Consequently, step (4) produces the desired linear portrayal of the entire scene.

7.4.3.4 Special case: parallel motion

In this and the following section we examine some special-case scenarios for which dynamic view interpolations can be produced without knowledge of $\mathbf{H}^{\infty}_{AB}$.

Assume a fixed-camera formulation and let $\mathbf{u}_k$ denote the displacement between the position of object k at time $t = 0$ and its position at time $t = 1$. We will say the scene consists of *parallel motion* if all the $\mathbf{u}_k$ are parallel in space.

Dynamic view morphing algorithm for parallel motion case: *Segment each view into layers corresponding to objects. Apply static view morphing to each layer and recomposite the results.*

The algorithm works because the fundamental matrix with respect to each object is the same, so the same prewarp works for each layer. The prewarp will make the direction of motion for each object be parallel to the x-axis in both views; consequently, the virtual objects will follow straight-line trajectories as measured in the camera frame. If we assume that the background object has no motion in world coordinates, then the virtual camera moves parallel to the motion of all the objects and hence the virtual object motion is straight-line in world coordinates.

7.4.3.5 Special case: planar parallel motion

We now consider the case in which all the $\mathbf{u}_k$ are parallel to some fixed plane in space. Note that this does not mean all the objects are translating in the *same* plane. Also note that this case applies whenever there are two moving objects.

Recall that in Section 7.4.3.2 the only requirement for the virtual view to be a physically-accurate portrayal of an object that translates by $\hat{\mathbf{u}}$ is that the image planes of both reference views be parallel to $\hat{\mathbf{u}}$ and to each other. In the planar parallel motion case, it is possible to prewarp the reference views so that their image planes are parallel to each other and to the displacements of all the objects simultaneously.

Dynamic view morphing algorithm for planar parallel motion case: *Segment each view into layers corresponding to objects. For each reference view, find a single prewarp that sends the z coordinate of the vanishing point of each object to 0. Using this prewarp, apply static view morphing to each layer and recomposite the results.*

The algorithm given above only guarantees physical correctness, not straight-line or linear motion. The *appearance* of straight-line motion can be created by first making the conjugate motion vectors parallel during the prewarp step [107, 110].

7.4.3.6 Dynamic scene hierarchy

This section interrelates the algorithms of the previous three sections. As always, we assume a fixed-camera formulation, meaning we choose to interpret the two reference views as having been captured by cameras that shared the same optical center.

Consider classifying each object in the scene based on the direction of its translation vector, with two objects being placed in the same class if their translation vectors are parallel. A natural hierarchy emerges based on the number of distinct parallel motion classes the scene contains.

First consider scenes that have only one motion class. If the class corresponds to the null direction vector, then the scene is static and view interpolation reduces to mosaicing. If the direction vector is non-null, view interpolations can be produced via the parallel motion algorithm (Section 7.4.3.4).

When the scene has two motion classes, the planar-parallel motion algorithm applies (Section 7.4.3.5). With four or more motion classes, $\mathbf{H}^{\infty}_{AB}$ can be determined as described in Section 7.4.4 from the four directions associated with the classes, and the linear motion algorithm applies (Section 7.4.3.3). For scenes with exactly three motion classes, either the planar-parallel algorithm applies or else $\mathbf{H}^{\infty}_{AB}$ can be approximated after making reasonable assumptions about the reference cameras [107].

7.4.3.7 Affine cameras

The mathematical development for affine cameras, which includes orthographic cameras, is similar to that for pinhole cameras. However, except in special cases, no camera-to-camera transformation exists between the reference cameras. Hence it is typically impossible to guarantee linear motion for the virtual objects. On the other hand, interpolation of conjugate points always produces a physically-valid virtual view, without needing to make the image planes parallel. Prewarps can be applied to align conjugate directions and thus achieve straight-line motion. However, in general it is only possible to align at most three conjugate directions. For a complete discussion, see [107, 110].

7.4.4 Finding relative camera calibration

The problem of determining $\mathbf{H}_{AB}^{\infty}$ is central to the linear motion algorithm of Section 7.4.3.3. $\mathbf{H}_{AB}^{\infty}$ can be determined from four conjugate directions by a well-known result used in mosaicing [176] (because conjugate directions become conjugate points if we treat the reference cameras as being co-centered).

If the fundamental matrix can be determined for two objects in the scene and if the objects are not moving parallel to each other, then $\mathbf{H}_{AB}^{\infty}$ can be determined directly from these two fundamental matrices. The previous fact is proven in [107], which also gives a method for approximating $\mathbf{H}_{AB}^{\infty}$ from two conjugate directions by making a reasonable assumption about the internal parameters of typical cameras.

7.4.5 Applications

Dynamic view morphing has many potential applications; we list a few here: filling a missing gap in a movie, creating a "hand-off" sequence to switch from one camera view to another, creating virtual views of a scene, removing obstructions or moving objects from a sequence, adding synthetic moving objects to real scenes, projecting motion into the future or past, stabilizing and compressing movie sequences, and creating movies from still images.

7.4.6 Experimental results

We have tested the concepts of this section on a variety of scenarios. Fig. 7.11 shows the results of three tests, each as a series of still frames from a view interpolation sequence. The left-most and right-most frames of each strip are the original reference views, while the center two frames are virtual views created by the algorithm.

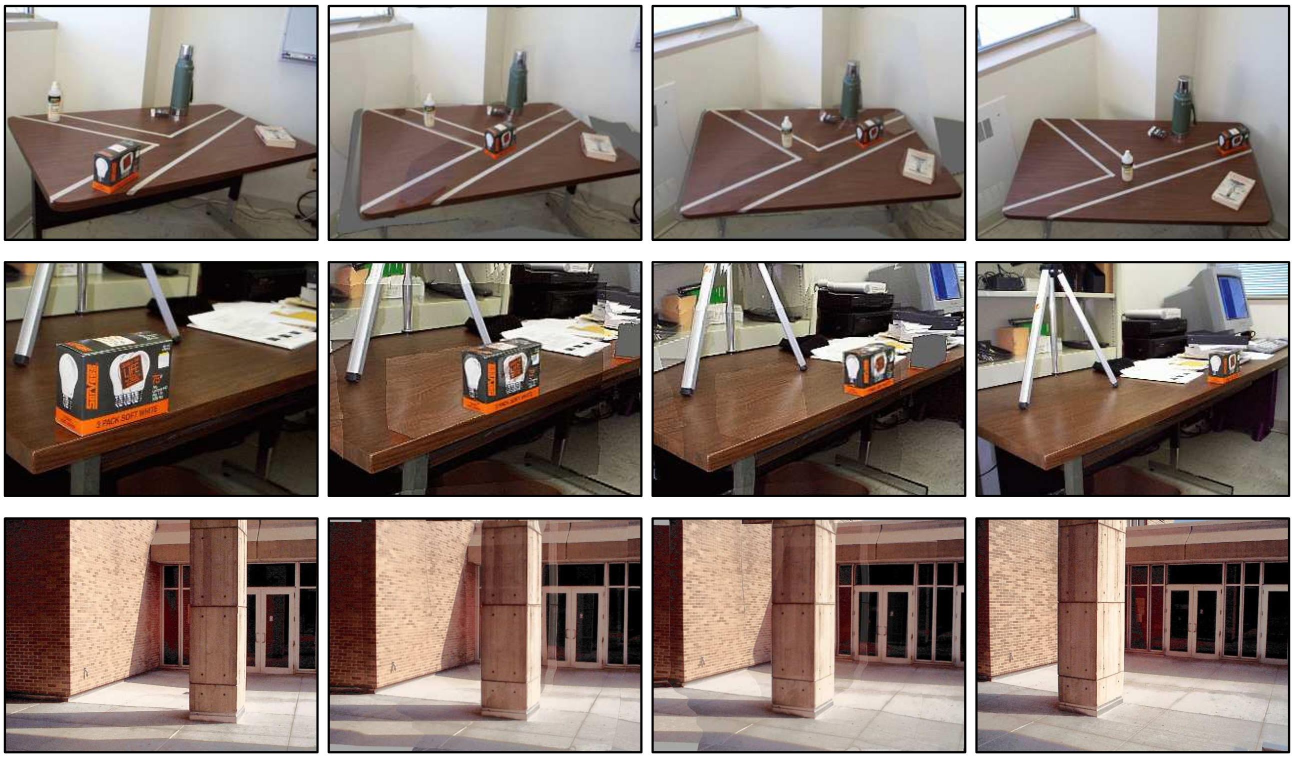

Figure 7.11 Experimental results.

To create each sequence, two preprocessing steps were performed manually. First, the two reference views were divided into layers corresponding to the moving objects. Second, for each corresponding layer a set of conjugate points between the two views was determined. Since our implementation uses the Beier-Neely algorithm [14] for the morphing step, we actually determined a series of line-segment correspondences instead of point correspondences. For each sequence, between 30 and 50 line-segment correspondences were used (counting every layer).

For all the sequences, the camera calibration was completely unknown, the focal lengths were different, and the cameras were at different locations.

The first sequence is from a test involving three moving objects (counting the background object). Since $\mathbf{H}^{\infty}_{AB}$ could only be approximated, the appearance of straight-line motion was achieved by aligning the conjugate directions of motion for each object during the prewarp step [107]. An object's direction of motion is given by the epipoles of the object's fundamental matrix. Instead of calculating the objects' fundamental matrices, we determined the epipoles directly from the vanishing points of the tape "roads."

The second sequence involves two moving objects (counting the background object) and a dramatic change in focal length. The third sequence demonstrates the parallel motion algorithm (Section 7.4.3.4). The scene is actually static, but the pillar in the foreground and the remaining background elements are treated as two separate objects that are moving parallel to each other.

7.4.7 Conclusion

This section presented a method for interpolating between two views of a dynamic scene. The method requires that, for each object in the scene, the movement that occurs between the first and second views must be equivalent to a rigid translation. The algorithm produces virtual views that portray one version of what might have occurred in the scene. It is only necessary that the image planes of the reference cameras be parallel to each other and to the motion of an object for the interpolated view of the object to be physically correct. With more conditions on the reference cameras, the object can be portrayed moving along a straight-line path and even moving at constant velocity along a straight-line path. Interpolated views of a complete dynamic scene can be synthesized by separately creating interpolated views of the scene's component objects and then combining the results.

By choosing to interpret the views as coming from the same position in space, a single theory has been created that applies to many different possible situations. In particular, the same theory applies whether or not the original reference cameras were actually co-centered. Since it is impossible to know from the reference views themselves how the original reference cameras were positioned relative to each other, the fixed-camera formulation is a natural default assumption. The virtual camera can be chosen to move along *any* trajectory; the choice simply alters the interpretation of the virtual views. The fixed-camera formulation also allows for a simple and intuitive development of the underlying mathematics of the theory.

The topics of this section, as well as many additional topics and observations, are discussed in much greater detail in [107].

7.5 View interpolation of turntable motion

In this section, view interpolation from two views of a dynamic scene containing purely rotational motion is demonstrated. The method is analogous to the dynamic view morphing method of Section 7.4.3 except (1) the virtual camera has constant internal parameters (thus avoiding undesirable distortion effects) and (2) the object motion is rotational rather than straight-line. The result given here is similar to the result given by Fitzgibbon et al. [52] except that no explicit scene reconstruction is performed; in the purest image-based-rendering sense, pixels are merely shifted around the flat images to create new, physically-correct views. It is also unclear how the result of Fitzgibbon et al. could be used to actually generate interpolation sequences, as their goal was scene reconstruction. Without extra information metric scene reconstruction is impossible from turntable reference views, meaning the scene cannot be simply reconstructed and rotated.

7.5.1 Turntable sequences

By Chasles' Theorem, mentioned in Section 4.1, any series of rotations applied in sequence will produce a transformation equivalent to a rotation around a single axis. Thus, when discussing arbitrary rotational motion, it is only necessary to consider rotation around a single axis. Consequently, any purely-rotational scene motion we might consider is equivalent to the turntable setup depicted in Fig. 7.12. In this scenario, a single camera, which is mounted on a tripod and fixed in orientation and internal parameters, views an object turning on an invisible turntable; this is the "fixed-camera formulation" for rotational motion. While both reference views for this scenario have the same internal parameters, the results we will derive under this assumption will also apply to scenarios in which the cameras are different but are internally calibrated.

The scenario in Fig. 7.13(b), equivalent to the turntable scenario, is more recognizable to vision researchers. It consists of two cameras in different locations viewing a static scene and thus there is obviously a fundamental matrix $\mathbf{F}$ between the reference views. Since the fundamental matrix is derived solely from correspondences in the reference views (without regard to how the reference

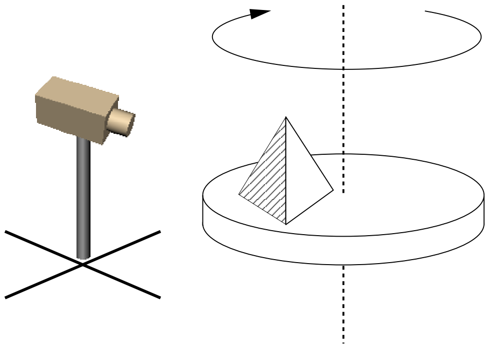

Figure 7.12 Turntable scenario.

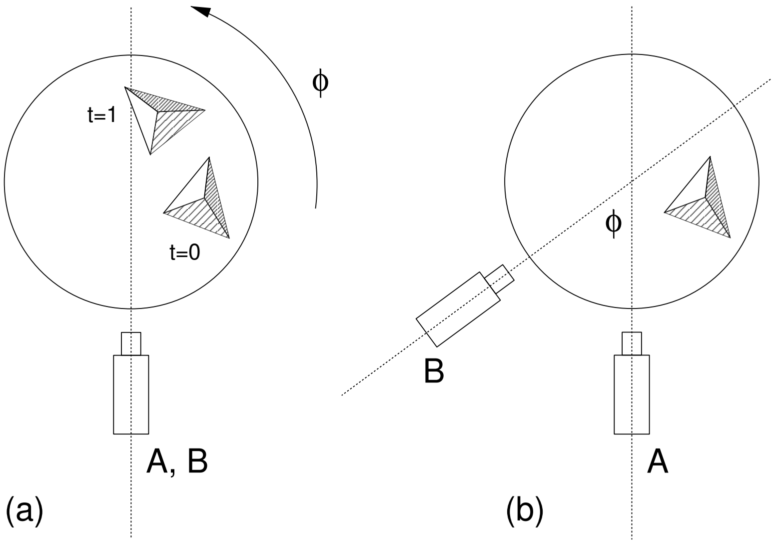

Figure 7.13 (a) The original turntable setup. (b) An alternative but equivalent interpretation of the reference views, using the principle of relativism.

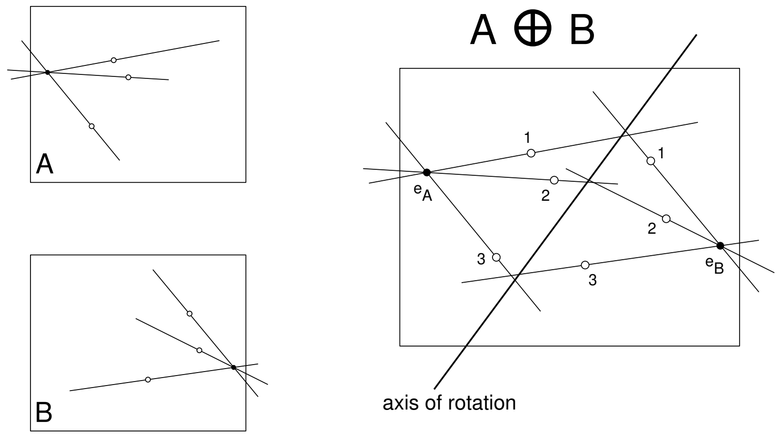

Figure 7.14 Each view in the left column shows the view's epipole and some conjugate scene points with their associated epipolar lines. On the right the two views have been superimposed; the axis of rotation is found by fitting a line to the intersection of corresponding epipolar lines.

views were captured), this fundamental matrix $\mathbf{F}$ can also be associated with the turntable scenario depicted in Fig. 7.13(a).

Now consider the view through the fixed camera in Fig. 7.13(a) as the turntable rotates, and let A and B denote the camera at time $t = 0$ and $t = 1$, respectively. Any point that lies on the axis of rotation will appear to stay in exactly the same location throughout the rotation. Thus if $\mathbf{q}$ is a point on the rotation axis then $\{\mathbf{q}\}_A \cong \{\mathbf{q}\}_B$. By Eq. 3.13

$$\{\mathbf{q}\}_B^\top \mathbf{F} \{\mathbf{q}\}_A = 0 \tag{7.15}$$

and thus

$$\{\mathbf{q}\}_A^\top \mathbf{F} \{\mathbf{q}\}_A = 0 \tag{7.16}$$

The latter equation is sufficient for finding the axis of rotation, at least when the fundamental matrix is near perfect. Unfortunately, fundamental matrices are notoriously difficult to compute accurately, and we have found that another approach works better with real data.

First, let $\mathbf{p}$ be an arbitrary position in space. Thinking about the scenario as in Fig. 7.13(b), notice that the two camera's optical centers and the position $\mathbf{p}$ define the epipolar plane for $\mathbf{p}$. This plane intersects the rotation axis at some position $\mathbf{q}$, and $\mathbf{q}$ projects into the same position in each view; i.e., $\{\mathbf{q}\}_A \cong \{\mathbf{q}\}_B$. Switching back to the fixed-camera formulation in Fig. 7.13(a), consider the epipolar lines through $\{\mathbf{p}\}_A$ and $\{\mathbf{p}\}_B$, respectively, both drawn in the one fixed view. These epipolar lines will intersect at the point $\{\mathbf{q}\}_A$, which lies on the rotation axis. Thus, since $\mathbf{p}$ was arbitrary, by finding the intersection points of conjugate epipolar lines we locate points on the rotation axis. The axis of rotation can be determined by first intersecting all conjugate epipolar lines (from any conjugate point pairs that have been provided to find $\mathbf{F}$) and then using linear regression to fit a line to these intersection points (Fig. 7.14). Notice that the conjugate point pairs are first used to find $\mathbf{F}$ and then reused in combination with $\mathbf{F}$ to find the axis of rotation. This provides numerical stability to the algorithm: the axis could have been found solely from $\mathbf{F}$, but when the data has noise the original conjugate point set contains more information than $\mathbf{F}$ alone.

Points on the rotation axis are not the only ones that satisfy Eq. 7.16. Let the term *biepipolar plane* denote the unique plane that is perpendicular to rotation axis (in world coordinates) and contains the optical center of the camera (under the fixed-camera formulation in Fig. 7.13(a)). The term refers to the fact that both epipoles are contained in the line defined by this plane intersecting the image plane. This line, termed the *biepipolar line*, can be found by plotting both epipoles in a single view and drawing a line through them. Note that for any point $\mathbf{p}$ on the biepipolar plane,

$$\{\mathbf{p}\}_B^\top \mathbf{F} \{\mathbf{p}\}_A = 0$$

because the biepipolar line is the epipolar line for both $\{\mathbf{p}\}_A$ and $\{\mathbf{p}\}_B$. Thus the set $\{\mathbf{p} : \{\mathbf{p}\}_B^\top \mathbf{F} \{\mathbf{p}\}_A = 0\}$, when projected into either camera view, consists of both the axis of rotation and the biepipolar plane as seen in that view. The biepipolar plane will be important in the interpolation process.

It is interesting to note that, even if the internal calibration of the cameras is unknown, the rotation axis for a turntable sequence can be determined from two views by using the same conjugate points that might be used more typically to find the fundamental matrix. In fact, much more can be determined from this minimal amount of input data. The next section discusses how to create a synthetic interpolation sequence from two views showing the object undergoing a complete cycle on the turntable.

7.5.1.1 Turntable view synthesis algorithm

An overview of the algorithm is given below, followed by a detailed explanation of each step. The algorithm assumes a fixed-camera formulation like in Fig. 7.13(a); for example, it makes reference to a single view.

(Step 1) *Find the epipoles (e.g., from the fundamental matrix between the reference views) and the axis of rotation.*

(Step 2) *Transform the reference view so that the epipoles are in the $z = 0$ plane and the axis of rotation (as a line in the views) is parallel to the y-axis.*

(Step 3) *Find and apply (to the 2D view) the 2×2 affine transformation that makes the orbit of each feature point a circle. This process includes finding the center of rotation for each conjugate point pair.*

(Step 4) *Rotate feature points around their respective centers of rotation to produce a new synthetic view of the scene.*

(Step 5) *Postwarp the synthetic view by applying the inverse of the transformations applied in steps 3, 2, and 1.*

Step 1 has already been discussed and step 2 is a straight-forward transformation that can be accomplished in many ways. Step 3 is the most complicated part of the algorithm. After steps 1 and 2, the camera view is looking down on the biepipolar plane. We do not know what the internal calibration of the camera was to begin with nor what changes our transformations have introduced, but we know that in the correct Euclidean frame the following must hold:

- each conjugate-point pair lies on a circle whose center is on the axis of rotation
- the arc length between each conjugate pair (measured in radians along the circle) is the same for each pair; this amount is how much the turntable rotated

What is needed is an affine transformation of the biepipolar plane that makes the above conditions true.

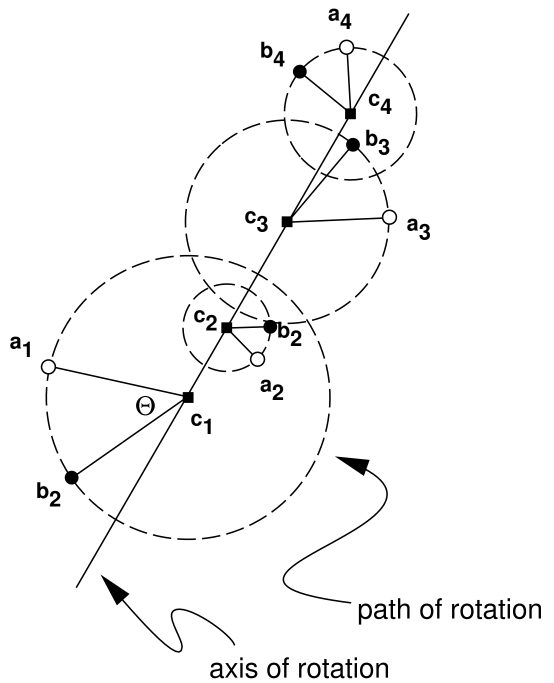

Figure 7.15 Overhead view of biepipolar plane after step 3.

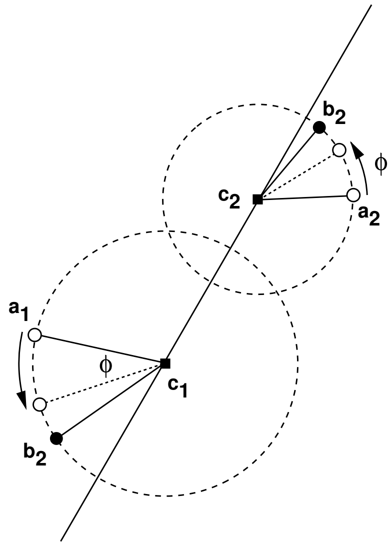

Figure 7.16 New views are created by rotating each point by ϕ along its appropriate circular path.

Fig. 7.15 shows the view looking down on the biepipolar plane after the correct affine transform has been applied; $\mathbf{a}_i$ and $\mathbf{b}_i$ are conjugate pairs (seen in view A and view B, respectively). The center of rotation for each pair is $\mathbf{c}_i$ and θ is the angle of rotation, which is the same for every pair. Note that $\mathbf{a}_i$, $\mathbf{b}_i$, and $\mathbf{c}_i$ are 2D positions in the camera view: $\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i \in \mathbb{R}^2$. Writing down the Euclidean conditions directly:

$$\mathbf{M}(\mathbf{a}_i - \mathbf{c}_i) = \mathbf{R}_\theta \mathbf{M}(\mathbf{b}_i - \mathbf{c}_i) \tag{7.17}$$

Here, $\mathbf{M}$ is a 2×2 matrix representing the sought-after affine transform and $\mathbf{R}_\theta$ is a 2×2 rotation matrix corresponding to θ.

Let $\mathbf{a}_i = [u, v]'$ and $\mathbf{b}_i = [x, y]'$. In step 2 the view was transformed so that the axis of rotation was parallel to the y-axis. Hence we can set $\mathbf{c}_i = [b, \underline{T}]'$ where $\underline{T} \in \mathbb{R}$ is unknown and b is the x-coordinate of the axis of rotation. Note we will use underlined capital letters for all unknowns.

In summary:

$$\mathbf{a}_i = \begin{bmatrix} u \\ v \end{bmatrix}, \quad \mathbf{b}_i = \begin{bmatrix} x \\ y \end{bmatrix}, \quad \mathbf{c}_i = \begin{bmatrix} b \\ \underline{T} \end{bmatrix} \tag{7.18}$$

A key insight is to realize that there are many 2×2 matrices $\mathbf{M}$ that will meet the desired conditions. For instance, if $\mathbf{M}$ works then so does $\mathbf{R}_\phi \mathbf{M}$ for any rotation matrix $\mathbf{R}_\phi$. This is because the sought-after geometric relationships are preserved by rotations. The relationships are also preserved by scaling. Since $\mathbf{M}$ is entirely defined by how it transforms the basis vectors $[1, 0]'$ and $[0, 1]'$, if $\tilde{\mathbf{M}}$ is a satisfactory solution for $\mathbf{M}$ we can always modify $\tilde{\mathbf{M}}$ by a rotation and scaling so that $\tilde{\mathbf{M}}[0, 1]' = [0, 1]'$. That is, we can assume the affine transform $\mathbf{M}$ preserves the y-basis vector:

$$\mathbf{M} = \begin{bmatrix} \underline{A} & 0 \\ \underline{C} & 1 \end{bmatrix} \tag{7.19}$$

The final insight is to simply fix θ and solve for the remaining unknowns. We can fix θ to be the true θ (if we happen to know it). Alternatively, since the unknowns can be found for any choice of θ, any θ is satisfactory (that is, there is some scene for which that choice of θ is correct).

Setting $\alpha = \cos\theta$ and $\beta = \sin\theta$ and using Eq. 7.18 and Eq. 7.19 with Eq. 7.17 gives:

$$\begin{bmatrix} \underline{A}u - \underline{A}b \\ \underline{C}u + v - \underline{C}b - \underline{T} \end{bmatrix} = \begin{bmatrix} \alpha & -\beta \\ \beta & \alpha \end{bmatrix} \begin{bmatrix} \underline{A}x - \underline{A}b \\ \underline{C}x + y - \underline{C}b - \underline{T} \end{bmatrix}$$

This can be rewritten as a linear system to allow solving for the unknowns:

$$\begin{bmatrix} u - b - \alpha x + \alpha b & \beta x - \beta b & -\beta \\ -\beta x + \beta b & u - b - \alpha x + \alpha b & \alpha - 1 \end{bmatrix} \begin{bmatrix} \underline{A} \\ \underline{C} \\ \underline{T} \end{bmatrix} = \begin{bmatrix} -\beta y \\ -v + \alpha y \end{bmatrix} \tag{7.20}$$

Of course, there is an equation like this for each conjugate pair. $\underline{A}$ and $\underline{C}$ are the same for each conjugate pair, but $\underline{T}$ differs. Thus we get a large $2n \times (2 + n)$ linear system. If n is large, a small sample of conjugate point pairs can be used to find $\underline{A}$ and $\underline{C}$ using Eq. 7.20, and then the n unknowns $\underline{T}$ (from each conjugate pair) can be determined separately using a single closed-form equation for each.

Step 4 is illustrated in Fig. 7.16. After step 3, both views are looking straight down on the biepipolar plane and the plane is viewed correctly in Euclidean coordinates up to the accuracy of θ. Furthermore, the center of rotation for each conjugate pair is known. Thus it is straight forward to rotate each point in A by a fixed amount ϕ around its corresponding center of rotation.

Step 5 is a postwarp step for making the synthetic view look like the original views. It is easy to achieve by simply undoing all the transformations that were applied to the view. The final synthetic image will show the turntable scene rotated by ϕ (in a Euclidean sense) around the turntable axis. This rotation is relative to the starting position in view A.

Of course, only the chosen conjugate points will appear in the synthetic view. If a dense correspondence between the reference views is known, which is reasonable to acquire if the views are closely spaced, then the synthetic view will also be dense and realistic. If a sparse correspondence can be determined, techniques like image morphing might be used to reasonably complete the synthetic view. If the scene is polyhedral, other techniques can be used to create the dense correspondence.

In a real turntable sequence, there is likely to be a stationary background along with the rotating turntable. In such cases, layering can be used to create realistic synthetic views: Simply place the background and turntable on separate layers, leave the background unchanged, and recomposite the synthetic turntable image onto the stable background.

It is also possible to have arbitrary numbers of turntables visible in the views; again, layering can be used to create each synthetic view separately before recompositing for the final result. This works because each synthetic view will be through the same camera, that is, the original camera A. The separate turntables can even be rotated by different amounts. When multiple turntables are visible in the scene and their rotation axes are not parallel in space, then it may be possible to extract full Euclidean calibration; this is left for future work.

Sample results from applying the algorithm to two real reference views are given in Fig. 7.17. Errors visible in the output are due to incorrect point correspondences, or in some cases, missing information due to occlusions. Nonetheless, the overall shape of the house model is evident even at the most extreme rotation angles.

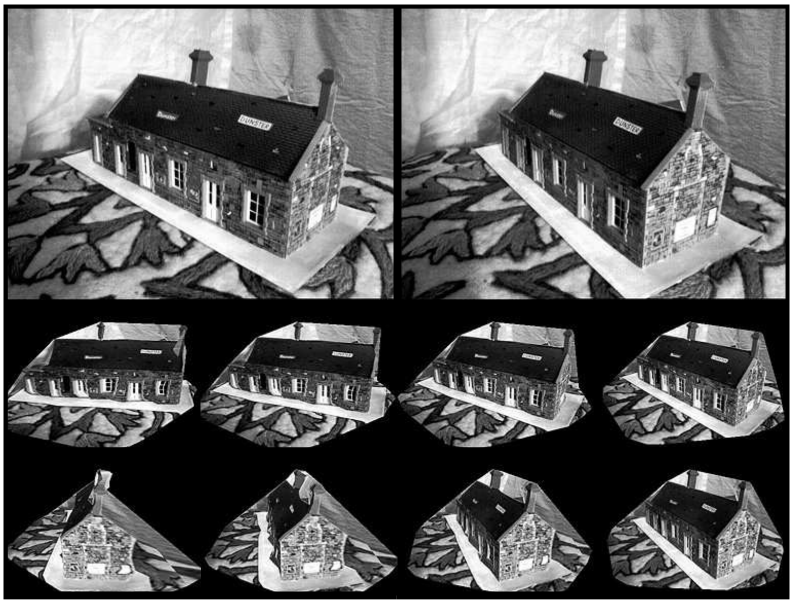

Figure 7.17 Experimental results. Top row is original two reference views.

7.5.2 Conclusion

In this section, a method for view interpolation between two views of a rotating object was presented. The interpolation sequence will show the object rotating at constant velocity and the virtual camera will have fixed internal parameters throughout the sequence. Thus the interpolation sequence will not have the distortions evident in "view morphing" techniques [154, 109] where the internal calibration of the virtual camera can change during the sequence.

The two reference views must be captured by cameras with the same internal parameters. However, it is not necessary to calculate the fundamental matrix between the reference views, which is notoriously difficult to determine correctly, as long as the epipoles or at least the biepipolar line can be determined in another way (e.g., from conjugate points). Notably, the axis of rotation can be determined directly from point correspondences.

The interpolation algorithm given here represents IBR in its purest form: the virtual views are created entirely by moving pixels around without scene reconstruction.

Chapter 8

Summary

8.1 Accomplishments

This dissertation has achieved the following:

- The concept and underlying theory of screw-transform manifolds has been presented. Although formal mathematical proofs for Conjectures 1 and 2 are lacking, the logical derivation of the screw-transform manifold algorithms (i.e., the algorithms used to define $\Phi_{\mathbf{F}}$ and $\Lambda_{\mathbf{F}}$) along with ample experimental evidence from synthetic and real data suggests the truth of the conjectures. Detailed formal proofs of the steps of the screw-transform manifold algorithms have been provided.

- Three general-purpose algorithms for finding the mutual-intersection point(s) of a series of manifolds have been presented. Such algorithms have a variety of uses; here they are used for camera self calibration since self calibration can be achieved by finding the mutual intersection point of three or more screw-transform manifolds. The voting algorithm (Section 5.2) was first presented in [104] and later used in [105]. The other two algorithms were presented in [113].

- Extensive experiments (Section 6.1 and Section 6.2) have demonstrated the efficacy and performance characteristics of the SURFIT and MCMC-based algorithms. A detailed metric scene reconstruction using only three closely-spaced views of a real scene was demonstrated with the SURFIT algorithm (Section 6.1.6 and Fig. 6.11).

- Experiments have shown that self calibration can be reliably achieved from as few as three camera views. This was established by experiments with the SURFIT algorithm that showed most trials performed with three camera views and 0 noise yielded only 1 or 2 possible calibrations and the correct calibration was usually one of these; see Section 6.1.5 for a more detailed discussion. Furthermore, a successful reconstruction from three real camera views (Section 6.1.6) was demonstrated.

- The mathematics of screw-transform manifolds for specialized motions (i.e., non-general motions) has been presented (Section 4.4). The case of turntable motion was first presented in [105]; the case of transfocal motion was presented in [113]. The case of general motion was first presented in [104].

- An important theorem partitioning all pairwise monocular camera motions into 6 categories (Theorem 1 of Section 4.4.4; also see Table 4.1 and Table 4.2) has been presented. A simple test has also been provided for each category. To the best of my knowledge, the case called "transfocal motion" has never been formally labeled or studied before Manning and Dyer [113]. The test for turntable motion was first presented in [105]; a minor flaw in the proof of the test was corrected in Manning and Dyer [113].

- Important details needed for implementing the algorithms have been presented.

- A detailed tutorial on multiview geometry has been provided that assumes only knowledge of linear algebra. This tutorial may prove friendlier than existing texts to early graduate students and advanced undergraduates interested in the topic.

- An brief history and survey of image-based rendering and camera self calibration has been presented.

- A linear algorithm [107, 112] for the affine self calibration of a stereo rig from dynamic scenes has been presented. The algorithm requires apparent linear motion in the scene. This algorithm was one of the first published uses of dynamic scenes for camera self calibration.

- A second, nonlinear algorithm for affine self calibration of stereo rigs directly from fundamental matrices has been presented here for the first time. Two views of a static scene are required; alternatively, the rig could view a rigid object undergoing general motion.

- An IBR method [108, 109] for generating physically-valid interpolation sequences between two views of a dynamic scene with apparent linear motion has been presented. This method was probably the first to perform physically-valid view synthesis of dynamic scenes from novel viewpoints.

- An IBR method for generating physically-valid interpolation sequences between two views of a dynamic scene with turntable motion has been presented here for the first time.

8.2 Conclusions

In my time as a graduate student studying image-based rendering, I have come to the following broad conclusions. First, image-based rendering is part of a larger trend in engineering in which sampling is a better, more efficient way to solve certain problems (e.g., speech synthesis). Second, most useful IBR techniques require camera calibration. In particular, IBR for dynamic scenes without camera calibration is highly restrictive in terms of what classes of motion can be portrayed and what kinds of virtual views can be created. This echoes the sentiment expressed by Werner in his doctoral thesis [190]. Third, the self calibration of a general pinhole camera is possible provided the given data (e.g., point correspondences) has sufficiently-low noise. Completely-accurate internal calibration may not be necessary to produce usable scene reconstructions and ego-motion estimates. Finally, the mutual-intersection points of a collection of manifolds can be determined quickly and reliably by direct intersection, at least for low-dimensional manifolds. This represents an ideal alternative to nonlinear optimization when performing camera self calibration.

BIBLIOGRAPHY

[1] Adelson and Bergen. *The Plenoptic Function and the Elements of Early Vision*, pages 3–20. MIT Press, Cambridge, MA, 1991.

[2] Aloimonos. *Low Level Visual Computations*. PhD thesis, University of Rochester, 1986.

[3] Appel. Some techniques for shading machine renderings of solids. In *AFIPS 1968 Spring Joint Computer Conf.*, volume 32, pages 37–45, 1968.

[4] Armstrong, Zisserman, and Beardsley. Euclidean structure from uncalibrated images. In *Proc. British Machine Vision Conference*, pages 509–518, 1994.

[5] Armstrong, Zisserman, and Hartley. Self-calibration from image triplets. In *Proc. European Conference on Computer Vision*, LNCS 1064/5, pages 3–16. Springer-Verlag, 1996.

[6] Avidan and Shashua. Novel view synthesis in tensor space. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 1034–1040, 1997.

[7] Avidan and Shashua. Non-rigid parallax for 3D linear motion. In *Proc. Image Understanding Workshop*, pages 199–201, 1998.

[8] Ayache and Lustman. Trinocular stereo vision for robotics. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 13(1):73–85, 1991.

[9] Bajura and Neumann. Dynamic registration correction in video-based augmented reality systems. *IEEE Computer Graphics and Applications*, 15(5):52–60, 1995.

[10] Baker and Kanade. Limits on super-resolution and how to break them. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 372–379, 2000.

[11] Baker and Kanade. Hallucinating faces. In *Fourth International Conference on Automatic Face and Gesture Recognition*, March 2000.

[12] Beardsley and Zisserman. Affine calibration of mobile vehicles. In Mohr and Chengke, editors, *Europe-China workshop on Geometrical Modelling and Invariants for Computer Vision*, pages 214–221. Xidan University Press, Xi'an, China, 1995.

[13] Beardsley, Torr, and Zisserman. 3D model aquisition from extended image sequence. In *Proc. European Conference on Computer Vision*, pages 683–695, 1996.

[14] Beier and Neely. Feature-based image metamorphosis. In *Proc. SIGGRAPH 92*, pages 35–42, 1992.

[15] Blanz and Vetter. A morphable model for the synthesis of 3d faces. In *Proc. SIGGRAPH 99*, pages 187–194, 1999.

[16] Blinn and Newell. Texture and reflection in computer generated images. *Communications of the ACM*, 19(10):542–547, October 1976.

[17] Bottema. *Theoretical Kinematics*. North-Holland Publishing Company, New York, 1979.

[18] Breasted. *A History of Egypt*. Charles Scribner's Sons, New York, 1905.

[19] Bregler, Covell, and Slaney. Video rewrite: Driving visual speech with audio. In *Proc. SIGGRAPH 97*, pages 353–360, 1997.

[20] Brodsky, Fermller, and Aloimonos. Shape from video. In *Proc. Computer Vision and Pattern Recognition Conf.*, volume 2, pages 146–151, 1999.

[21] Burt and Adelson. The laplacian pyramid as a compact image code. *IEEE Trans. Communications*, 31:532–540, 1983.

[22] Carlos. Switched-On Bach. Audio recording, 1968.

[23] Caspi and Irani. Alignment of Non-Overlapping sequences. In *Proc. Int. Conf. on Computer Vision*, volume 2, pages 76–83, 2001.

[24] Catmull. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, University of Utah, Salt Lake City, UT, 1974.

[25] Chai, Chan, Shum, and Tong. Plenoptic sampling. In *Proc. SIGGRAPH 00*, pages 307–318, 2000.

[26] Chen and Williams. View interpolation for image synthesis. In *Proc. SIGGRAPH 93*, pages 279–288, 1993.

[27] Cootes, Edwards, and Taylor. Active appearance models. In *Proc. European Conference on Computer Vision*, volume 2, pages 484–498, 1998.

[28] Davis. Mosaics of scenes with moving objects. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 354–360, 1998.

[29] Debevec. Rendering synthetic objects into real scenes: Bridging traditional and image-based graphics with global illumination and high dynamic range photography. In *Proc. SIGGRAPH 98*, pages 189–198, 1998.

[30] Debevec, Wenger, Tchou, Gardner, Waese, and Hawkins. A lighting reproduction approach to live-action compositing. *ACM Transactions on Graphics*, 21(3):547–556, July 2002.

[31] Debevec, Taylor, and Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proc. SIGGRAPH 96*, pages 11–20, 1996.

[32] Dellaert, Seitz, Thorpe, and Thrun. Structure from motion without correspondence. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 557–564, 2000.

[33] Demey, Zisserman, and Beardsley. Affine and projective structure from motion. In Hogg and Boyle, editors, *Proc. 3rd British Machine Vision Conference, Leeds*, pages 49–58. Springer-Verlag, September 1992.

[34] Deriche, Zhang, Luong, and Faugeras. Robust recovery of the epipolar geometry for an uncalibrated stereo rig. In *Proc. European Conference on Computer Vision*, pages 567–576, 1994.

[35] Devernay and Faugeras. From projective to euclidean reconstruction. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 264–269, 1996.

[36] Dudley. Parallel bandpass vocoder. A machine for synthesizing the human voice, 1939.

[37] E. Trucco and Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice-Hall, Upper Saddle River, N. J., 1998.

[38] Edwards, Taylor, and Cootes. Learning to identify and track faces in image sequences. In *Proc. Sixth Int. Conf. Computer Vision*, pages 317–322, 1998.

[39] Efros and Leung. Texture synthesis by non-parametric sampling. In *Proc. Int. Conf. on Computer Vision*, pages 1033–1038, 1999.

[40] Farid and Popescu. Blind removal of image non-linearities. In *Proc. Int. Conf. on Computer Vision*, pages 76–81, 2001.

[41] Faugeras, Luong, and Maybank. Camera self-calibration: Theory and experiments. In *Proc. European Conference on Computer Vision*, pages 321–334, 1992.

[42] Faugeras and Luong. *The Geometry of Multiple Images*. The MIT Press, Cambridge, Massachusetts, 2001.

[43] Faugeras. *Three-Dimensional Computer Vision, A Geometric Viewpoint*. MIT Press, Cambridge, MA, 1993.

[44] Faugeras. What can be seen in three dimensions with an uncalibrated stereo rig. In *Proc. European Conference on Computer Vision*, pages 563–578, 1992.

[45] Faugeras. Stratification of 3-dimensional vision: Projective, affine, and metric representations. *Journal of the Optical Society of America*, 12(3):465–484, 1995.

[46] Faugeras, Quan, and Sturm. Self-calibration of a 1D projective camera and its application to the self-calibration of a 2d projective camera. In *Proc. European Conference on Computer Vision*, pages I:36–52. Springer-Verlag, June 1998.

[47] Feiner, MacIntyre, Hollerer, and Webster. A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment. In *Int. Symp. on Wearable Computers*, October 1997.

[48] Feynman. *Surely You're Joking, Mr. Feynman!: Adventures of a Curious Character*. W.W. Norton, New York, 1985.

[49] Feynman. *QED: The strange theory of light and matter*. Princeton University Press, Princeton, N. J., 1985.

[50] Finsterwalder. Die geometrischen grundlagen der photogrametrie. *Jahresbericht Deutscher Mathematik*, pages 1–35, 1899.

[51] Fischler and Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[52] Fitzgibbon, Cross, and Zisserman. Automatic 3D model construction for turn-table sequences. In Koch and Van Gool, editors, *Proc. Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE '98)*, pages 155–170. Springer, 1998.

[53] Fitzgibbon and Zisserman. Multibody structure and motion: 3-D reconstruction of independently moving objects. In *Proc. European Conference on Computer Vision*, pages 891–906. Springer-Verlag, June 2000.

[54] Forsyth and Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, Upper Saddle River, N. J., 2003.

[55] Friedman. *U.S. Submarines Through 1945: An Illustrated Design History*. United States Naval Institute, 1995.

[56] Gap, Inc. Khakis swing. Television Commercial, 1998.

[57] Geyer and Daniilidis. Catadioptric camera calibration. In *Proc. Seventh Int. Conf. Computer Vision*, pages I:398–404, 1999.

[58] Gleicher. Projective registration with difference decomposition. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 331–337, 1997.

[59] Golub and Loan. *Matrix Computations. 2nd ed.* Johns Hopkins Press, Baltimore, MD, 1989.

[60] Golub and Reinsch. Singular value decomposition and least squares solutions. *Numerische Mathematik*, 28(14):403–420, 1970.

[61] Gondry and Buffin. Like a rolling stone. Music Video, 1995.

[62] Goral, Torrance, Greenburg, and Battaile. Modelling the interaction of light between diffuse surfaces. *Computer Graphics*, 18-3:213–222, July 1984.

[63] Gortler, Grzeszczuk, Szeliski, and Cohen. The lumigraph. In *Proc. SIGGRAPH 96*, pages 43–54, 1996.

[64] Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, November 1986.

[65] Harris and Stephens. A combined corner and edge detector. In *Proc. 4th Alvey Vision Conf.*, pages 189–192, 1988.

[66] Hartley. Camera calibration using line correspondences. In *Proc. Image Understanding Workshop*, pages 361–366, 1993.

[67] Hartley. Projective reconstruction from line correspondences. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 903–907, 1994.

[68] Hartley. Self-calibration from multiple views with a rotating camera. In *Proc. European Conference on Computer Vision*, pages 471–478, 1994.

[69] Hartley. A linear method for reconstruction from lines and points. In *Proc. Fifth Int. Conf. Computer Vision*, pages 882–887, 1995.

[70] Hartley and Zisserman. *Multiple View Geometry*. Cambridge University Press, New York, 2000.

[71] Hartley. Euclidean reconstruction from uncalibrated views. In Zisserman and Forsyth, editors, *Applications of Invariance in Computer Vision, LNCS 825*, pages 237–256. Springer-Verlag, 1994.

[72] Hartley. Projective reconstruction and invariants from multiple images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16(10):1036–1041, 1994.

[73] Hartley. In defence of the 8-point algorithm. In *Proc. Fifth Int. Conf. Computer Vision*, pages 1064–1070, 1995.

[74] Hauck. Neue konstruktionen der perspektive und photogrammetrie. *Crelle J. f. Math.*, pages 1–35, 1883.

[75] Heigl, Koch, Pollefeys, Denzler, and Van Gool. Plenoptic modeling and rendering from image sequences taken by hand-held camera. In *Proc. DAGM'99*, pages 94–101, 1999.

[76] Heyden. *Geometry and Algebra of Multiple Projective Transformations*. PhD thesis, Lund Institute of Technology, Lund, Sweden, 1995.

[77] Heyden and Astrom. Euclidean reconstruction from constant intrinsic parameters. In *Proc. Int. Conf. on Pattern Recognition*, pages 339–343, 1996.

[78] Horaud and Csurka. Self-calibration and Euclidean reconstruction using motions of a stereo rig. In *Proc. Sixth Int. Conf. Computer Vision*, pages 96–103, 1998.

[79] Horn. Relative orientation. *Int. J. Computer Vision*, 4:59–78, January 1990.

[80] Horn. Projective geometry considered harmful. Self published on Internet, 1999.

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

[82] Hough. Machine analysis of bubble chamber pictures. In *International Conference on High Energy Accelerators and Instrumentation*, 1959.

[83] Irani, Anandan, and Hsu. Mosaic based representations of video sequences and their applications. In *Proc. Fifth Int. Conf. Computer Vision*, pages 605–611, 1995.

[84] Irani, Hassner, and Anandan. What does the scene look like from a scene point? In *Proc. European Conference on Computer Vision*, pages 883–897, 2002.

[85] Irani and Peleg. Improving resolution by image registration. *CVGIP: Graphical Models and Image Processing*, 53:231–239, 1991.

[86] Isard and Miller. Diphone synthesis techniques. In *Proceedings of IEE International Conference on Speech Input/Output*, pages 77–82, 1986.

[87] Jain, Kasturi, and Schunck. *Machine Vision*. McGraw-Hill, St. Louis, 1995.

[88] Kanade, Rander, and Narayanan. Virtualized reality: Constructing virtual worlds from real scenes. *IEEE Multimedia*, 4(1):34–46, 1997.

[89] Koch. *Automatische Oberflaechenmodellierung starrer dreidimensionaler Objekte aus stereoskopischen Rundum-Ansichten*. PhD thesis, Hannover, 1996.

[90] Koch and Van Gool, editors. *Proc. Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE '98)*. Springer, 1998.

[91] Koenderink and van Doorn. Affine structure from motion. *J. Opt. Soc. Am. A*, 8:377–385, 1991.

[92] Kruppa. Zur ermittlung eines objektes aus zwei perspektiven mit innerer orientierung. *Sitz.-Ber. Akad. Wiss., Wien, math. naturw. Kl., Abt. IIa.*, 122:1939–1948, 1913.

[93] Kutulakos and Seitz. A theory of shape by space carving. In *Proc. Seventh Int. Conf. Computer Vision*, pages 307–314, 1999.

[94] Kutulakos and Vallino. Calibration-free augmented reality. *IEEE Trans. Visualization and Computer Graphics*, 4(1):1–20, 1998.

[95] Laveau and Faugeras. 3-D scene representation as a collection of images. In *Proc. Int. Conf. on Pattern Recognition*, pages 689–691, 1994.

[96] Levoy and Hanrahan. Light field rendering. In *Proc. SIGGRAPH 96*, 1996.

[97] Levoy, Pulli, Curless, Rusinkiewicz, Koller, Pereira, Ginzton, Anderson, Davis, Ginsberg, Shade, and Fulk. The digital michelangelo project: 3D scanning of large statues. In *Proc. SIGGRAPH 00*, pages 131–144, 2000.

[98] Lhuillier and Quan. Image interpolation by joint view triangulation. In *Proc. Computer Vision and Pattern Recognition Conf.*, volume 2, pages 139–145, 1999.

[99] Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.

[100] Lourakis and Deriche. Camera self-calibration using the Kruppa equations and the SVD of the fundamental matrix: The case of varying intrinsic parameters. Technical Report 3911, INRIA, March 2000.

[101] Lucas and Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[102] Lucasfilm Ltd. Willow. Film, 1988.

[103] MacCurdy, editor. *The Notebooks of Leonardo da Vinci*, page 343. George Braziller, New York, 1954.

[104] Manning and Dyer. Metric self calibration from screw-transform manifolds. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 590–597, 2001.

[105] Manning and Dyer. Stratified self calibration from screw-transform manifolds. In *Proc. European Conference on Computer Vision*, volume 4, pages 131–145, 2002.

[106] Manning and Dyer. Research on self calibration without minimization. Technical Report 1490, Computer Sciences Department, University of Wisconsin-Madison, 2003.

[107] Manning and Dyer. Dynamic view morphing. Technical Report 1387, Computer Sciences Department, University of Wisconsin-Madison, 1998.

[108] Manning and Dyer. Interpolating view and scene motion by dynamic view morphing. In *Proc. Image Understanding Workshop*, pages 323–330, 1998.

[109] Manning and Dyer. Interpolating view and scene motion by dynamic view morphing. In *Proc. Computer Vision and Pattern Recognition Conf.*, volume 1, pages 388–394, 1999.

[110] Manning and Dyer. Dynamic view interpolation without affine reconstruction. In Leonardis, Solina, and Bajcsy, editors, *Confluence of Computer Vision and Computer Graphics*, pages 123–142. Kluwer, Dordrecht, The Netherlands, 2000.

[111] Manning and Dyer. Environment map morphing. Technical Report 1423, Computer Sciences Department, University of Wisconsin-Madison, 2000.

[112] Manning and Dyer. Affine calibration from moving objects. In *Proc. Eighth Int. Conf. Computer Vision*, volume 1, pages 494–500, 2001.

[113] Manning and Dyer. On screw-transform manifolds. Technical Report 1482, Computer Sciences Department, University of Wisconsin-Madison, 2003.

[114] Marr. *Vision*. W. H. Freeman Co., San Francisco, CA, 1982.

[115] McMillan. *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, University of North Carolina at Chapel Hill, Chapel Hill, 1997.

[116] McMillan and Bishop. Head-tracked stereoscopic display using image warping. In *Proc. SPIE Vol. 2409A*, pages 21–30, 1995.

[117] McMillan and Bishop. Plenoptic modeling. In *Proc. SIGGRAPH 95*, pages 39–46, 1995.

[118] Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller. Equation of state calculations by fast computing machines. *J. of Chemical Physics*, 21:1087–1092, 1953.

[119] Miller and Hoffman. Illumination and reflection maps: Simulated objects in simulated and real environments. In *Course Notes for Advanced Computer Graphics Animation, SIGGRAPH 84*, 1984.

[120] Mohr, Buschmann, Falkenhagen, Gool, and Koch. CUMULI, PANORAMA, and VANGUARD project overview. In Koch and Van Gool, editors, *Proc. Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE '98)*, pages 1–13. Springer, 1998.

[121] Moons, Van Gool, Proesmans, and Pauwels. Affine reconstruction from perspective image pairs with a relative object-camera translation in between. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 18(1):77–83, January 1996.

[122] Narayanan, Rander, and Kanade. Constructing virtual worlds using dense stereo. In *Proc. Sixth Int. Conf. Computer Vision*, pages 3–10, 1998.

[123] Nayar. Catadioptric omnidirectional camera. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 482–488, 1997.

[124] Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.

[125] New York Institute of Technology. Interface. Film, 1985.

[126] Okutomi and Kanade. A multiple-baseline stereo. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 15(4):353–363, 1985.

[127] Oskarsson, Zisserman, and Astrom. Minimal projective reconstruction for combinations of points and lines in three views. In *Proc. British Machine Vision Conference*, pages 63–72, 2002.

[128] Pacific Title / Mirage Studio. The Jester. Short film, 1999.

[129] Pacific Western. Terminator 2: Judgment Day. Film, 1991.

[130] Peleg and Herman. Panoramic mosaicing with videobrush. In *Proc. Image Understanding Workshop*, pages 261–264, 1997.

[131] Pollefeys. *Self-Calibration and Metric 3D Reconstruction from Uncalibrated Image Sequences*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 1999.

[132] Pollefeys, Koch, and Van Gool. Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters. In *Proc. Sixth Int. Conf. Computer Vision*, pages 90–95, 1998.

[133] Pollefeys and Van Gool. A stratified approach to metric self-calibration. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 407–412, 1997.

[134] Pollefeys and Van Gool. A stratified approach to metric self-calibration with the modulus constraint. Technical Report 9702, K. U. Leuven – ESAT-MI2, 1997.

[135] Pollefeys and Van Gool. Stratified self-calibration with the modulus constraint. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 21(8):707–724, August 1999.

[136] Pollefeys and Van Gool, editors. *Proc. Second Workshop on 3D Structure from Multiple Images of Large-Scale Environments (SMILE '00)*. Springer, 2000.

[137] Pollefeys, Van Gool, and Oosterlinck. The modulus constraint: A new constraint for self-calibration. In *Proc. Int. Conf. on Pattern Recognition*, pages 349–353, 1996.

[138] Pryor, Furness, and Viirre. The virtual retinal display: A new display technology using scanned laser light. In *Proc. Human Factors and Ergonomics Society, 42nd Annual Meeting*, pages 1570–1574, 1998.

[139] Quan. Affine stereo calibration for relative affine shape reconstruction. In *Proc. 4th British Machine Vision Conference, Surrey, England*, pages 659–668, 1993.

[140] Quan. Invariants of six points and projective reconstruction from three uncalibrated images. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17(1):34–46, 1995.

[141] Quan. Uncalibrated 1D projective camera and 3d affine reconstruction of lines. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 60–65, 1997.

[142] Quan and Kanade. A factorization method for affine structure from line correspondences. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 803–808, 1996.

[143] Rademacher and Bishop. Multiple-center-of-projection images. In *Proc. SIGGRAPH 98*, pages 199–206, 1998.

[144] Rose, Cohen, and Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–40, 1998.

[145] Rothwell, Csurka, and Faugeras. A comparison of projective reconstruction methods for pairs of views. In *Proc. Fifth Int. Conf. Computer Vision*, 1995.

[146] Rousso, Peleg, Finci, and Rav-Acha. Universal mosaicing using pipe projection. In *Proc. Sixth Int. Conf. Computer Vision*, pages 945–952, 1998.

[147] Roy, Meunier, and Cox. Cylindrical rectification to minimize epipolar distortion. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 393–399, 1997.

[148] Sawhney and Kumar. True multi-image alignment and its application to mosaicing and lens distortion correction. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 21(3):235–243, 1999.

[149] Sawhney, Hsu, and Kumar. Robust video mosaicing through topology inference and local to global alignment. In *Proc. European Conference on Computer Vision, Vol. II*, pages 103–122, 1998.

[150] Sawnhey and Kumar. True multi-image alignment and its application to mosaicing and lens distortion correction. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 450–456, 1997.

[151] Schaffalitzky. Direct solution of modulus constraints. In *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing, Bangalore*, pages 314–321, 2000.

[152] Schmid, Mohr, and Bauckhage. Comparing and evaluating interest points. In *Proc. Int. Conf. on Computer Vision*, pages 230–235, 1998.

[153] Seitz and Anandan. Implicit scene reconstruction from probability density functions. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 28–34, 1999.

[154] Seitz and Dyer. View morphing. In *Proc. SIGGRAPH 96*, pages 21–30, 1996.

[155] Seitz and Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 1067–1073, 1997.

[156] Seitz and Dyer. View Morphing: Uniquely predicting scene appearance from basis images. In *Proc. Image Understanding Workshop*, pages 881–887, 1997.

[157] Shade, Gortler, He, and Szeliski. Layered depth images. In *Proc. SIGGRAPH 98*, pages 231–242, 1998.

[158] Shapiro. *Affine Analysis of Image Sequences*. Cambridge University Press, Cambridge, England, 1995.

[159] Shashua. Algebraic functions for recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 17(8):779–789, 1995.

[160] Shashua. Projective structure from uncalibrated images: Structure from motion and recognition. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 16(8):778–790, 1994.

[161] Shashua and Wolf. Homography tensors: On algebraic entities that represent three views of static or moving planar points. In *Proc. European Conference on Computer Vision*, pages 507–521. Springer-Verlag, June 2000.

[162] Shechtman, Caspi, and Irani. Increasing space-time resolution in video. In *Proc. European Conference on Computer Vision*, pages 753–768, 2002.

[163] Shum and He. Rendering with concentric mosaics. In *Proc. SIGGRAPH 99*, pages 299–306, 1999.

[164] Silberman. Matrix 2. *Wired Magazine*, May 2003.

[165] Sinha. *Perceiving and Recognizing Three-Dimensional Forms*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, 1995.

[166] Slama, Theurer, and Henriksen, editors. *Manual of Photogrammetry, Fourth Edition*. American Society of Photogrammetry and Remote Sensing, Falls Church, Virginia, 1980.

[167] Spetsakis and Aloimonos. Structure from motion using line correspondences. *Int. J. Computer Vision*, 4(3):171–183, 1990.

[168] Spetsakis and Aloimonos. A unified theory of structure from motion. In *Proc. Image Understanding Workshop*, pages 271–283, 1990.

[169] Stein. Lens distortion calibration using point correspondences. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 602–608, 1997.

[170] Stein. Model based brightness constraints: On direct estimation of structure and motion. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 400–406, 1997.

[171] Stein. *Geometric and Photometric Constraints and Structure from Three Views*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, June 1998.

[172] Stein. Tracking from multiple view points: Self-calibration of space and time. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages I:521–527, 1999.

[173] Sturm. Critical motion sequences for monocular self-calibration and uncalibrated euclidean reconstruction. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 1100–1105, 1997.

[174] Sturm and Quan. Affine stereo calibration. In *Proc. 6th International Conference CAIP '95, Prague, Czech Republic*, pages 838–843, September 1995.

[175] Szeliski. Image mosaicing for tele-reality applications. In *Proc. Workshop Applications of Computer Vision*, pages 44–53, 1994.

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

[177] Szeliski and Shum. Creating full view panoramic image mosaics and environment maps. In *Proc. SIGGRAPH 97*, pages 251–258, 1997.

[178] The Revolution Company. EyeVision, 2001.

[179] Tomasi and Kanade. Shape and motion from image streams under orthography: A factorization method. *Int. J. Computer Vision*, 9(2):137–154, 1992.

[180] Torr and Zisserman. Robust parameterization and computation of the trifocal tensor. *Image and Vision Computing*, 15:591–605, 1997.

[181] Triggs. Autocalibration and the absolute quadric. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 609–614, 1997.

[182] Ullman. *The Interpretation of Visual Motion*. MIT Press, Cambridge, MA, 1979.

[183] Universal Pictures. Jurassic Park. Film, 1993.

[184] Van Gool, Moons, Proesmans, and Van Diest. Affine reconstruction from perspective image pairs obtained by a translating camera. In *Proc. Int. Conf. on Pattern Recognition*, pages A:290–294, 1994.

[185] Vieville, Faugeras, and Luong. Motion of points and lines in the uncalibrated case. *Int. J. Computer Vision*, 17(1):7–41, January 1996.

[186] Village Roadshow Productions. The Matrix. Film, 1999.

[187] Walt Disney Pictures. Flight of the Navigator. Film, 1986.

[188] Walt Disney Pictures. Dinosaur. Film, 2000.

[189] Weng, Huang, and Ahuja. Motion and structure from line correspondences: Closed form solution, uniqueness and optimization. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(3):318–336, 1992.

[190] Werner. *Image-Based Visualization of Real 3D Scenes*. PhD thesis, Czech Technical University, Prague, Czech Republic, 1998.

[191] Werner, Hersch, and Hlavac. Rendering real-world objects using view interpolation. In *Proc. Fifth Int. Conf. Computer Vision*, pages 957–962, 1995.

[192] Whitted. An improved illumination model for shaded display. *Computer Graphics (Special SIGGRAPH '79 Issue)*, 13(3):1–14, August 1979.

[193] Wiley and Hahn. Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications*, 17(6):39–45, 1997.

[194] Williams. Pyramidal parametrics. In *Proc. SIGGRAPH 83*, pages 1–11, July 1983.

[195] Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, CA, 1990.

[196] Wolf and Shashua. On projection matrices $P^k \rightarrow P^2$, $k = 3, \ldots, 6$, and their applications in computer vision. In *Proc. Int. Conf. on Computer Vision*, volume 1, pages 53–67, 2002.

[197] Wolf. *Elements of Photogrammetry*. McGraw-Hill, 1974.

[198] Yang, Boyer, and Kak. Range data extraction and interpretation by structural light. In *Proc. 1st IEEE Conf. Artificial Intelligence Appl.*, pages 199–205, 1984.

[199] Yu, Debevec, Malik, and Hawkins. Inverse global illumination: Recovering reflectance models of real scenes from photographs from. In *Proc. SIGGRAPH 99*, pages 215–224, 1999.

[200] Zhang, Deriche, Faugeras, and Luong. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artificial Intelligence*, 78:87–119, 1995.

[201] Zisserman, Beardsley, and Reid. Metric calibration of a stereo rig. In *IEEE Workshop on Representation of Visual Scenes*, pages 93–100, 1995.

[202] Zongker, Werner, Curless, and Salesin. Environment matting and compositing. In *Proc. SIGGRAPH 99*, pages 205–214, 1999.

Appendix A: Mathematical Details

A.1 Derivation of the fundamental matrix

In this section we show in detail how the rising-turntable formulation of the fundamental matrix (Eq. 4.1) can be derived in a purely mechanical way, using straight-forward properties of matrix arithmetic. We also provide a list of mathematical equalities stemming from the rising-turntable formulation.

A.1.1 Useful matrix properties

We begin by stating some general matrix properties that are used in Appendix A.1.2 and Appendix A.2.

Property 1 (Matrix inverse). Let $\mathbf{M} = [\,\mathbf{m}_1\;\mathbf{m}_2\;\mathbf{m}_3\,]$, with $\mathbf{m}_i \in \mathbb{R}^3$, be an invertible 3×3 matrix. Observe that $\det(\mathbf{M}) = \mathbf{m}_1 \cdot \mathbf{m}_2 \times \mathbf{m}_3 = \mathbf{m}_1 \times \mathbf{m}_2 \cdot \mathbf{m}_3$ and

$$\begin{bmatrix} (\mathbf{m}_2 \times \mathbf{m}_3)^\top \\ (\mathbf{m}_3 \times \mathbf{m}_1)^\top \\ (\mathbf{m}_1 \times \mathbf{m}_2)^\top \end{bmatrix} [\,\mathbf{m}_1\;\mathbf{m}_2\;\mathbf{m}_3\,] = \det(\mathbf{M}) \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Since the inverse of a matrix is unique, it must be that

$$\mathbf{M}^{-1} = \frac{1}{\det(\mathbf{M})} \begin{bmatrix} (\mathbf{m}_2 \times \mathbf{m}_3)^\top \\ (\mathbf{m}_3 \times \mathbf{m}_1)^\top \\ (\mathbf{m}_1 \times \mathbf{m}_2)^\top \end{bmatrix}$$

Property 2 (Cross-product matrix and cross-product operator). For $\mathbf{a}, \mathbf{b}, \mathbf{u} \in \mathbb{R}^3$:

$$(\mathbf{a} + \mathbf{b}) \times \mathbf{u} = [\mathbf{a} + \mathbf{b}]_\times \mathbf{u} = [\mathbf{a}]_\times \mathbf{u} + [\mathbf{b}]_\times \mathbf{u} = \mathbf{a} \times \mathbf{u} + \mathbf{b} \times \mathbf{u}$$

Property 3 (Matrix multiplication). For $\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e}, \mathbf{f} \in \mathbb{R}^3$:

$$\begin{bmatrix} \mathbf{a} & \mathbf{b} & \mathbf{c} \end{bmatrix} \begin{bmatrix} \mathbf{d}^\top \\ \mathbf{e}^\top \\ \mathbf{f}^\top \end{bmatrix} = \mathbf{a}\mathbf{d}^\top + \mathbf{b}\mathbf{e}^\top + \mathbf{c}\mathbf{f}^\top$$

Property 4 (An alternative cross-product expression). Here we consider a complicated expression for the cross-product operation which is used in Appendix A.1.2. In what follows, $\mathbf{l}_{ij} = \mathbf{h}_i \times \mathbf{h}_j$ and $\mathbf{u} \in \mathbb{R}^3$. Since $\mathbf{h}_1$, $\mathbf{h}_2$, and $\mathbf{h}_3$ form a basis that spans $\mathbb{R}^3$, $\mathbf{u}$ can be uniquely expressed as $\mathbf{u} = \sigma_1\mathbf{h}_1 + \sigma_2\mathbf{h}_2 + \sigma_3\mathbf{h}_3$ for some $\sigma_1, \sigma_2, \sigma_3 \in \mathbb{R}$.

$$\begin{aligned}
(-\mathbf{l}_{13}\mathbf{l}_{23}^\top + \mathbf{l}_{23}\mathbf{l}_{13}^\top)\mathbf{u} &= (-\mathbf{l}_{13}\mathbf{l}_{23}^\top + \mathbf{l}_{23}\mathbf{l}_{13}^\top)(\sigma_1\mathbf{h}_1 + \sigma_2\mathbf{h}_2 + \sigma_3\mathbf{h}_3) \\
&= -\sigma_1\mathbf{l}_{13}\det(\mathbf{H}) - \sigma_2\mathbf{l}_{23}\det(\mathbf{H}) \\
&= \det(\mathbf{H})[\mathbf{h}_3]_\times(\sigma_1\mathbf{h}_1 + \sigma_2\mathbf{h}_2 + \sigma_3\mathbf{h}_3) \\
&= \det(\mathbf{H})[\mathbf{h}_3]_\times\mathbf{u}
\end{aligned}$$

Thus

$$[\mathbf{h}_3]_\times = \frac{1}{\det(\mathbf{H})}(-\mathbf{l}_{13}\mathbf{l}_{23}^\top + \mathbf{l}_{23}\mathbf{l}_{13}^\top)$$

Similarly, $(\mathbf{l}_{23}\mathbf{l}_{12}^\top - \mathbf{l}_{12}\mathbf{l}_{23}^\top) = \det(\mathbf{H})[\mathbf{h}_2]_\times$.

Property 5 (Intersection of two lines). Let $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3 \in \mathbb{R}^3$ be arbitrary, nonzero, linearly independent vectors. Then $(\mathbf{h}_1 \times \mathbf{h}_2) \times (\mathbf{h}_1 \times \mathbf{h}_3)$ is a vector $\mathbf{u} \in \mathbb{R}^3$ that is simultaneously perpendicular to $(\mathbf{h}_1 \times \mathbf{h}_2)$ and $(\mathbf{h}_1 \times \mathbf{h}_3)$. Thus $\mathbf{u}$ is in the intersection of the plane with normal $(\mathbf{h}_1 \times \mathbf{h}_2)$ and the plane with normal $(\mathbf{h}_1 \times \mathbf{h}_3)$. Since $(\mathbf{h}_1 \times \mathbf{h}_2)$ and $(\mathbf{h}_1 \times \mathbf{h}_3)$ are linearly independent, this intersection is a line. Note that $\mathbf{h}_1$ lies on this line. Thus

$$(\mathbf{h}_1 \times \mathbf{h}_2) \times (\mathbf{h}_1 \times \mathbf{h}_3) \cong \mathbf{h}_1$$

In projective geometry terms, this identity indicates that the line through $\mathbf{h}_1$ and $\mathbf{h}_2$ intersects the line through $\mathbf{h}_1$ and $\mathbf{h}_3$ at the point $\mathbf{h}_1$; the identity is used in Appendix A.2 but not in the derivation below.

A.1.2 Direct derivation of fundamental matrix

We can now derive the fundamental matrix formula. The fundamental matrix $\mathbf{F}$ can be written

$$\mathbf{F} = [-\mathbf{e}_B]_\times(\mathbf{H}^\infty) = [-\mathbf{e}_B]_\times\mathbf{G}\mathbf{H}^{-1}$$

where $\mathbf{H}$ and $\mathbf{G}$ are the left-most 3×3 matrices of $\mathbf{\Pi}_A$ and $\mathbf{\Pi}_B$, respectively. $\mathbf{H}$ is simply $[\mathbf{h}_1\ \mathbf{h}_2\ \mathbf{h}_3]$ and $\mathbf{\Pi}_B$ has the form:

$$\begin{aligned}
\mathbf{\Pi}_B &= \mathbf{\Pi}_A \mathbf{S}(-\gamma, -\theta) \\
&= [\,\mathbf{h}_1\ \mathbf{h}_2\ \mathbf{h}_3\ \mathbf{h}_4\,] \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & -\gamma \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= [\,\cos\theta\mathbf{h}_1 - \sin\theta\mathbf{h}_2, \sin\theta\mathbf{h}_1 + \cos\theta\mathbf{h}_2, \mathbf{h}_3, -\gamma\mathbf{h}_3 + \mathbf{h}_4\,]
\end{aligned}$$

Thus

$$\mathbf{G} = [\,\mathbf{g}_1\ \mathbf{g}_2\ \mathbf{g}_3\,] = [\,\cos\theta\mathbf{h}_1 - \sin\theta\mathbf{h}_2, \sin\theta\mathbf{h}_1 + \cos\theta\mathbf{h}_2, \mathbf{h}_3\,]$$

Evaluating $\mathbf{F}$ is straight forward but involves many terms. Working in stages, we first multiply $[-\mathbf{e}_B]_\times$ with $\mathbf{G}$ (where $\mathbf{e}_B$, the left epipole of $\mathbf{F}$, is given by Eq. A.11):

$$\begin{aligned}
[-\mathbf{e}_B]_\times \mathbf{G} &= [(1-\cos\theta)\mathbf{h}_1 + \sin\theta\mathbf{h}_2 + \gamma\mathbf{h}_3]_\times [\,\mathbf{g}_1\ \mathbf{g}_2\ \mathbf{g}_3\,] \\
&= [(1-\cos\theta)\mathbf{h}_1 \times \mathbf{g}_1 + \sin\theta\mathbf{h}_2 \times \mathbf{g}_1 + \gamma\mathbf{h}_3 \times \mathbf{g}_1, \\
&\quad (1-\cos\theta)\mathbf{h}_1 \times \mathbf{g}_2 + \sin\theta\mathbf{h}_2 \times \mathbf{g}_2 + \gamma\mathbf{h}_3 \times \mathbf{g}_2, \\
&\quad (1-\cos\theta)\mathbf{h}_1 \times \mathbf{g}_3 + \sin\theta\mathbf{h}_2 \times \mathbf{g}_3 + \gamma\mathbf{h}_3 \times \mathbf{g}_3] \\
&= [-\sin\theta(1-\cos\theta)\mathbf{h}_1 \times \mathbf{h}_2 + \sin\theta\cos\theta\mathbf{h}_2 \times \mathbf{h}_1 + \gamma\cos\theta\mathbf{h}_3 \times \mathbf{h}_1 \\
&\quad -\gamma\sin\theta\mathbf{h}_3 \times \mathbf{h}_2, \cos\theta(1-\cos\theta)\mathbf{h}_1 \times \mathbf{h}_2 + \sin\theta\sin\theta\mathbf{h}_2 \times (\mathbf{h}_1) \\
&\quad +\gamma\sin\theta\mathbf{h}_3 \times (\mathbf{h}_1) + \gamma\cos\theta\mathbf{h}_3 \times (\mathbf{h}_2), \\
&\quad (1-\cos\theta)\mathbf{h}_1 \times \mathbf{h}_3 + \sin\theta\mathbf{h}_2 \times \mathbf{h}_3] \\
&= [-\sin\theta(1-\cos\theta)\mathbf{l}_{12} - \sin\theta\cos\theta\mathbf{l}_{12} - \gamma\cos\theta\mathbf{l}_{13} + \gamma\sin\theta\mathbf{l}_{23}, \\
&\quad \cos\theta(1-\cos\theta)\mathbf{l}_{12} - \sin\theta\sin\theta\mathbf{l}_{12} - \gamma\sin\theta\mathbf{l}_{13} - \gamma\cos\theta\mathbf{l}_{23}, \\
&\quad (1-\cos\theta)\mathbf{l}_{13} + \sin\theta\mathbf{l}_{23}] \\
&= [-\sin\theta\mathbf{l}_{12} - \gamma\cos\theta\mathbf{l}_{13} + \gamma\sin\theta\mathbf{l}_{23}, \\
&\quad -(1-\cos\theta)\mathbf{l}_{12} - \gamma\sin\theta\mathbf{l}_{13} - \gamma\cos\theta\mathbf{l}_{23},
\end{aligned}$$

$$(1-\cos\theta)\mathbf{l}_{13}+\sin\theta\mathbf{l}_{23}] \quad \text{(A.1)}$$

The calculation is now finished by multiplying by $\mathbf{H}^{-1}$.

$$\begin{aligned}
\det(\mathbf{H})\{[-\mathbf{e}_B]_\times\mathbf{G}\mathbf{H}^{-1}\} &= (-\sin\theta\mathbf{l}_{12}-\gamma\cos\theta\mathbf{l}_{13}+\gamma\sin\theta\mathbf{l}_{23})(\mathbf{h}_2\times\mathbf{h}_3)^\top + \\
&\quad (-(1-\cos\theta)\mathbf{l}_{12}-\gamma\sin\theta\mathbf{l}_{13}-\gamma\cos\theta\mathbf{l}_{23})(\mathbf{h}_3\times\mathbf{h}_1)^\top + \\
&\quad ((1-\cos\theta)\mathbf{l}_{13}+\sin\theta\mathbf{l}_{23})(\mathbf{h}_1\times\mathbf{h}_2)^\top \\
&= -\sin\theta\mathbf{l}_{12}\mathbf{l}_{23}^\top-\gamma\cos\theta\mathbf{l}_{13}\mathbf{l}_{23}^\top+\gamma\sin\theta\mathbf{l}_{23}\mathbf{l}_{23}^\top + \\
&\quad (1-\cos\theta)\mathbf{l}_{12}\mathbf{l}_{13}^\top+\gamma\sin\theta\mathbf{l}_{13}\mathbf{l}_{13}^\top+\gamma\cos\theta\mathbf{l}_{23}\mathbf{l}_{13}^\top + \\
&\quad (1-\cos\theta)\mathbf{l}_{13}\mathbf{l}_{12}^\top+\sin\theta\mathbf{l}_{23}\mathbf{l}_{12}^\top \\
&= (1-\cos\theta)(\mathbf{l}_{12}\mathbf{l}_{13}^\top+\mathbf{l}_{13}\mathbf{l}_{12}^\top)+\gamma\sin\theta(\mathbf{l}_{13}\mathbf{l}_{13}^\top+\mathbf{l}_{23}\mathbf{l}_{23}^\top) + \\
&\quad \sin\theta(\mathbf{l}_{23}\mathbf{l}_{12}^\top-\mathbf{l}_{12}\mathbf{l}_{23}^\top)+\gamma\cos\theta(\mathbf{l}_{23}\mathbf{l}_{13}^\top-\mathbf{l}_{13}\mathbf{l}_{23}^\top) \\
&= (1-\cos\theta)(\mathbf{l}_{12}\mathbf{l}_{13}^\top+\mathbf{l}_{13}\mathbf{l}_{12}^\top)+\gamma\sin\theta(\mathbf{l}_{13}\mathbf{l}_{13}^\top+\mathbf{l}_{23}\mathbf{l}_{23}^\top) + \\
&\quad [\sin\theta\mathbf{h}_2+\gamma\cos\theta\mathbf{h}_3]_\times
\end{aligned} \quad \text{(A.2)}$$

A.1.3 Useful properties of the screw-transform decomposition of the fundamental matrix

The following identities can all be derived from Eqs. 4.2–4.3 by straight-forward multiplication:

$$\mathbf{F}^{\mathsf{A}}\mathbf{h}_1 = \sin\theta\mathbf{h}_2\times\mathbf{h}_1+\gamma\cos\theta\mathbf{h}_3\times\mathbf{h}_1 \quad \text{(A.3)}$$
$$\mathbf{F}^{\mathsf{A}}\mathbf{h}_2 = \gamma\cos\theta\mathbf{h}_3\times\mathbf{h}_2 \quad \text{(A.4)}$$
$$\mathbf{F}^{\mathsf{A}}\mathbf{h}_3 = \sin\theta\mathbf{h}_2\times\mathbf{h}_3 \quad \text{(A.5)}$$
$$\mathbf{F}^{\mathsf{S}}\mathbf{h}_1 = \gamma\sin\theta\mathbf{h}_2\times\mathbf{h}_3 \quad \text{(A.6)}$$
$$\mathbf{F}^{\mathsf{S}}\mathbf{h}_2 = (1-\cos\theta)\mathbf{h}_2\times\mathbf{h}_1+\gamma\sin\theta\mathbf{h}_3\times\mathbf{h}_1 \quad \text{(A.7)}$$
$$\mathbf{F}^{\mathsf{S}}\mathbf{h}_3 = (1-\cos\theta)\mathbf{h}_1\times\mathbf{h}_3 \quad \text{(A.8)}$$

Letting $\mathbf{m}=\sin\theta\mathbf{h}_2+\gamma\cos\theta\mathbf{h}_3$ so that $\mathbf{F}^{\mathsf{A}}=[\mathbf{m}]_\times$, we have

$$\mathbf{F}^{\mathsf{S}}\mathbf{m} = \sin\theta(1-\cos\theta)\mathbf{h}_2\times\mathbf{h}_1+\gamma\sin^2\theta\mathbf{h}_3\times\mathbf{h}_1+\gamma\cos\theta(1-\cos\theta)\mathbf{h}_1\times\mathbf{h}_3$$

$$
\begin{aligned}
&= \sin\theta(1-\cos\theta)\mathbf{h}_2 \times \mathbf{h}_1 + \gamma\sin^2\theta\mathbf{h}_3 \times \mathbf{h}_1 + \gamma\cos\theta\mathbf{h}_1 \times \mathbf{h}_3 - \gamma\cos^2\theta\mathbf{h}_1 \times \mathbf{h}_3 \\
&= \sin\theta(1-\cos\theta)\mathbf{h}_2 \times \mathbf{h}_1 + \gamma\cos\theta\mathbf{h}_1 \times \mathbf{h}_3 - \gamma\mathbf{h}_1 \times \mathbf{h}_3 \\
&= -(1-\cos\theta)(\sin\theta\mathbf{h}_1 \times \mathbf{h}_2 + \gamma\mathbf{h}_1 \times \mathbf{h}_3) \qquad \text{(A.9)}
\end{aligned}
$$

Finally, the two epipoles of $\mathbf{F}$ are given by:

$$
\begin{aligned}
\mathbf{e}_A &\cong \mathbf{\Pi}_A\left(\mathbf{S}(\gamma,\theta)[1,0,0,1]^\top\right) \\
&= \mathbf{\Pi}_A[\cos\theta, \sin\theta, \gamma, 1]^\top \\
&= [\,\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3\,][(\cos\theta-1), \sin\theta, \gamma]^\top \\
&= (\cos\theta-1)\mathbf{h}_1 + \sin\theta\mathbf{h}_2 + \gamma\mathbf{h}_3 \qquad \text{(A.10)} \\
\mathbf{e}_B &\cong \mathbf{\Pi}_B[1,0,0,1]^\top = (\mathbf{\Pi}_A\mathbf{S}(-\gamma,-\theta))\,[1,0,0,1]^\top \\
&= (\cos\theta-1)\mathbf{h}_1 - \sin\theta\mathbf{h}_2 - \gamma\mathbf{h}_3 \qquad \text{(A.11)}
\end{aligned}
$$

A.2 Derivation of the parameterization algorithms

This section shows in detail how algorithms A–D are derived. Note the fundamental matrix $\mathbf{F}$ between views A and B is found directly from the images themselves by identifying point correspondences or by other means. $\mathbf{F}^{\mathsf{S}}$ and $\mathbf{F}^{\mathsf{A}}$ can be determined from $\mathbf{F}$ because

$$
\mathbf{F}^{\mathsf{S}} = \frac{1}{2}(\mathbf{F} + \mathbf{F}^\top) \qquad \text{and} \qquad \mathbf{F}^{\mathsf{A}} = \frac{1}{2}(\mathbf{F} - \mathbf{F}^\top)
$$

A.2.1 Derivation of Algorithm A–1

Step A–1.1. Define $\mathbf{M}$ by

$$
\mathbf{M} = [[\mathbf{e}]_\times\mathbf{F}|\mathbf{e}]\,\mathbf{a}^\top
$$

where $\mathbf{a} \in \mathbb{R}^4$ and $\mathbf{e}$ is the left epipole of $\mathbf{F}$. It is easy to verify (by direct multiplication) that any choice of $\mathbf{a} \in \mathbb{R}^4$ produces an $\mathbf{M}$ that satisfies $\mathbf{F} \cong [\mathbf{e}]_\times\mathbf{M}$ (note that $[\mathbf{e}]_\times^3 \cong [\mathbf{e}]_\times$). It is well-known that all $\mathbf{M}$ that satisfy the equation have the form given above (e.g., [72]). Thus it is easy to find a matrix $\mathbf{M}$ for step (1).

Step A–1.2. Note that $\mathbf{h}_3 \cdot \mathbf{F}^{\mathsf{S}}\mathbf{h}_3 = 0$; this property allows the image of $\mathbf{h}_3$ to be parameterized by a single real variable κ. One way to do this is by expanding the equation $\mathbf{h}_3 \cdot \mathbf{F}^{\mathsf{S}}\mathbf{h}_3 = 0$ and then using the quadratic equation. A more elegant parameterization is given by the following derivation:

$$
\begin{aligned}
(\mathbf{H}^{\infty})\mathbf{h}_3 &\cong \mathbf{h}_3 \\
(\mathbf{M} + \mathbf{e}\mathbf{a}^{\top})\mathbf{h}_3 &\cong \mathbf{h}_3 \\
(\mathbf{M} + \mathbf{e}\mathbf{a}^{\top})\mathbf{h}_3 &= k_1\mathbf{h}_3 \\
\mathbf{e}\mathbf{a}^{\top}\mathbf{h}_3 &= (k_1\mathbf{I} - \mathbf{M})\mathbf{h}_3 \\
k_2\mathbf{e} &= (k_1\mathbf{I} - \mathbf{M})\mathbf{h}_3 \\
(k_1\mathbf{I} - \mathbf{M})^{-1}\mathbf{e} &\cong \mathbf{h}_3
\end{aligned}
\tag{A.12}
$$

The fact that $\underline{\mathbf{h}}_3$ lies on the cone $\mathbf{F}^{\mathsf{S}}$ (since $\mathbf{h}_3^{\top}\mathbf{F}^{\mathsf{S}}\mathbf{h}_3 = 0$) suggests that, as κ varies over all real numbers, Eq. A.12 will trace out the cone $\mathbf{F}^{\mathsf{S}}$; experiments also suggest this is the case.

Step A–1.3. Using Property 5 (Appendix A.1.1), Eq. A.8, and Eq. A.9

$$
\begin{aligned}
(\mathbf{F}^{\mathsf{S}}\mathbf{m}) \times (\mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_3) &= (\mathbf{F}^{\mathsf{S}}\mathbf{m}) \times (s_3\mathbf{F}^{\mathsf{S}}\mathbf{h}_3) \\
&= -s_3(1 - \cos\theta)^2(\sin\theta\mathbf{l}_{12} + \gamma\mathbf{l}_{13}) \times \mathbf{l}_{13} \\
&= -s_3(1 - \cos\theta)^2(\sin\theta\mathbf{h}_1) \\
&\cong \mathbf{h}_1
\end{aligned}
$$

Step A–1.4. Since $\mathbf{F}^{\mathsf{S}}\mathbf{h}_1 = \gamma\sin\theta(\mathbf{h}_2 \times \mathbf{h}_3)$ and $\mathbf{F}^{\mathsf{A}}\mathbf{h}_3 = \sin\theta(\mathbf{h}_2 \times \mathbf{h}_3)$ (see Eq. A.6 and Eq. A.5 in Section A.1.3), the given vector $\phi(1/s_1, \gamma/s_3)^{\top}$ is in the null space of the given matrix:

$$
\frac{\phi}{s_1}\mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_1 - \frac{\phi\gamma}{s_3}\mathbf{F}^{\mathsf{A}}\underline{\mathbf{h}}_3 = \phi\mathbf{F}^{\mathsf{S}}\mathbf{h}_1 - \phi\gamma\mathbf{F}^{\mathsf{A}}\mathbf{h}_3 = \phi\gamma\sin\theta\mathbf{h}_2 \times \mathbf{h}_3 - \phi\gamma\sin\theta\mathbf{h}_2 \times \mathbf{h}_3 = 0
$$

Since the matrix is nonzero, its rank is at least 1; since the matrix has at least one null eigenvector, its nullity is at least 1. The array has dimensionality 3×2 and the rank plus nullity add up to the minimum dimension, which is 2. Hence the nullity is 1 and the given vector generates the null space. The scalar ϕ is determined in the next step.

Step A–1.5. Use Eq. A.8. Note that θ, $\underline{\mathbf{h}}_1$, $\underline{\mathbf{h}}_3$, $\mathbf{F}^{\mathsf{S}}$, and σ_1 are known by this stage of the algorithm.

Step A–1.6. Straight-forward multiplication using the definition of $\mathbf{m}$:

$$
(\phi\mathbf{m} - \sigma_2\cos\theta\underline{\mathbf{h}}_3)/(\phi\sin\theta) = (\phi\sin\theta\mathbf{h}_2 + \phi\gamma\cos\theta\mathbf{h}_3 - \frac{\phi\gamma}{\sigma_3}\cos\theta\underline{\mathbf{h}}_3)/(\phi\sin\theta) = \mathbf{h}_2
$$

In summary, $\mathbf{h}_1$, $\mathbf{h}_2$, and $\underline{\mathbf{h}}_3$, which are the vanishing points of the x, y, and z axes, respectively, as seen in the first camera view, can be determined directly from $\mathbf{F}$ provided two real parameters κ and θ are known. Furthermore, by the method just described, $\mathbf{H}$ can be determined up to a single unknown real parameter: the scale of $\mathbf{h}_3$, which is γ. Once $\mathbf{H}$ is determined, the metric internal calibration of the camera can be found and metric scene reconstruction is possible.

Naively, since $\mathbf{K}$ is an upper triangular matrix and we are only interested in $\mathbf{K}$ up to a scale factor, we know $\mathbf{K}$ has at most 5 degrees of freedom. Our analysis shows that $\mathbf{K}$ can be parameterized by three real numbers κ, θ, and γ. The fact that $\mathbf{K}$ has only three degrees of freedom once $\mathbf{F}$ is known has been shown before (e.g., [131]). Here we have demonstrated a specific parameterization, one which has a great deal of intuitive meaning: θ is the rotation angle between the views, κ corresponds to the vanishing point of the rotation axis, and γ is the amount of translation (as a multiple of the distance between the optical center and the axis of rotation) along the screw axis during the screw transformation.

A.2.2 Derivation of Algorithm A–2

Note that $\gamma = 0$ in the case of turntable motion.

Step A–2.1. This step simply defines the quantity $\mathbf{m}$.

Step A–2.2. When $\gamma = 0$, Eq. 4.2 becomes $\mathbf{F}^{\mathsf{A}} = [\sin\theta\mathbf{h}_2]_\times$. Notice that we are fixing the scale of $\mathbf{h}_2$ to be whatever the scale of $\mathbf{F}$ happens to be. The scale of $\mathbf{h}_1$ and $\mathbf{h}_3$ must be determined in later steps so as to be consistent with the scale of $\mathbf{h}_2$ and $\mathbf{F}$.

Step A–2.3. When $\gamma = 0$, Eq. A.9 becomes $\mathbf{F}^{\mathsf{S}}\mathbf{m} = -(1-\cos\theta)(\sin\theta)\mathbf{h}_1 \times \mathbf{h}_2$. Recall that $\mathbf{l}_{ij}$ is shorthand notation for $\mathbf{h}_i \times \mathbf{h}_j$.

Step A–2.4. It must be shown that $\mathbf{F}^{\mathsf{S}}$ has a 1-dimensional null space and that $\mathbf{h}_1$ is in the null space. From Eqs. A.6–A.8 with $\gamma = 0$ we have $\mathbf{F}^{\mathsf{S}}\mathbf{h}_1 = 0$, $\mathbf{F}^{\mathsf{S}}\mathbf{h}_2 \cong \mathbf{h}_2 \times \mathbf{h}_1 \neq 0$, and $\mathbf{F}^{\mathsf{S}}\mathbf{h}_3 \cong \mathbf{h}_1 \times \mathbf{h}_3 \neq 0$. This proves both conditions since $\mathbf{h}_1$, $\mathbf{h}_2$, and $\mathbf{h}_3$ form a spanning basis for $\mathbb{R}^3$. In practice, $\underline{\mathbf{h}}_1$ is found by finding a null eigenvector of $\mathbf{F}^{\mathsf{S}}$; this eigenvector has an indeterminate scale factor and step (5) is needed to find the scale that makes $\mathbf{h}_1$ consistent with $\mathbf{F}$.

Step A–2.5. Since $\mathbf{l}_{12} = \mathbf{h}_1 \times \mathbf{h}_2$ found in step (3) is already scaled correctly to be consistent with $\mathbf{F}$, $\|\mathbf{h}_1 \times \mathbf{h}_2\| \,/\, \|\underline{\mathbf{h}}_1 \times \mathbf{h}_2\|$ gives the proper scale factor for converting $\underline{\mathbf{h}}_1$ to $\mathbf{h}_1$.

Step A–2.6. Any vector $\mathbf{u} \in \mathbb{R}^3$ can be represented as $\mathbf{u} = a\mathbf{h}_1 + b\mathbf{h}_2 + c\mathbf{h}_3$ for some scalars $a, b, c \in \mathbb{R}$. Observe:

$$\begin{aligned}
& (\mathbf{F}^{\mathsf{S}} - (1-\cos\theta)[\mathbf{h}_1]_\times)^\top \mathbf{u} \\
= \; & (\mathbf{F}^{\mathsf{S}} + (1-\cos\theta)[\mathbf{h}_1]_\times)\mathbf{u} \\
= \; & (\mathbf{F}^{\mathsf{S}} + (1-\cos\theta)[\mathbf{h}_1]_\times)(a\mathbf{h}_1 + b\mathbf{h}_2 + c\mathbf{h}_3) \\
= \; & b(1-\cos\theta)\mathbf{l}_{21} + c(1-\cos\theta)\mathbf{l}_{13} + b(1-\cos\theta)\mathbf{l}_{12} + c(1-\cos\theta)\mathbf{l}_{13} \\
= \; & 2c(1-\cos\theta)\mathbf{l}_{13} \\
\cong \; & \mathbf{l}_{13}
\end{aligned}$$

Since this relationship is true for every $\mathbf{u} \in \mathbb{R}^3$ with $c \neq 0$, it must hold for $\mathbf{u} = (1,0,0)^\top$, $\mathbf{u} = (0,1,0)^\top$, and $\mathbf{u} = (0,0,1)^\top$, thus proving $(\mathbf{F}^{\mathsf{S}} - (1-\cos\theta)[\mathbf{h}_1]_\times)^\top = [\; \mathbf{l}_{13}\; \mathbf{l}_{13}\; \mathbf{l}_{13} \;]$ up to unknown scale factors on the columns. Of course, if $(1,0,0)^\top = a\mathbf{h}_1 + b\mathbf{h}_2$ for some $a, b \in \mathbb{R}$ then the scale factor is 0 for column 1, and similarly for the other columns. The scale factor cannot be 0 for every column since $(1,0,0)^\top$, $(0,1,0)^\top$, and $(0,0,1)^\top$ form a basis for $\mathbb{R}^3$, and thus $\underline{\mathbf{l}}_{13}$ can be determined from at least one column.

Step A–2.7. $\mathbf{h}_1$ and $\mathbf{h}_3$ both lie in the plane perpendicular to $\underline{\mathbf{l}}_{13} \cong \mathbf{h}_1 \times \mathbf{h}_3$ and $\underline{\mathbf{l}}_{13}$ was determined in the previous step. The vector $\mathbf{R}(\underline{\mathbf{l}}_{13}, \kappa)\mathbf{h}_1$ also lies in this plane for every choice of κ; this vector is just $\mathbf{h}_1$ rotated within the plane $\mathbf{l}_{13}$. The procedure given in this step allows $\underline{\mathbf{h}}_3$ to point in any direction in the plane $\underline{\mathbf{l}}_{13}$. Thus for every scenario S there is some κ that produces the correct direction for $\underline{\mathbf{h}}_3$, which is sufficient for the purposes of this paper. Empirical evidence suggests that, for every choice of κ that produces an $\underline{\mathbf{h}}_3$ that is not collinear with $\mathbf{h}_1$, there is some scenario S that is consistent with this $\underline{\mathbf{h}}_3$.

Step A–2.8. γ is used as an arbitrary scale factor for converting $\underline{\mathbf{h}}_3$ to $\mathbf{h}_3$. In this case, γ does not have its normal physical interpretation as the amount of screw translation. Note that the scale of $\mathbf{h}_1$ and $\mathbf{h}_2$ is decoupled from the scale of $\mathbf{h}_3$ in the case of turntable motion.

A.2.3 Derivation of Algorithm A–3

Step A–3.1. Clear from Eq. 4.15.

Step A–3.2. Because the optical center lies on the axis of rotation, which serves as the z-axis, there is no clear choice for the direction of the x-axis as there was in earlier cases. So an arbitrary line through $\underline{\mathbf{h}}_3$ is chosen for the xz-plane, which is used to determine $\underline{\mathbf{h}}_1$ in the next step. For a given fundamental matrix $\mathbf{F}$, the same line must always be chosen.

Step A–3.3. The given parameterization allows $\underline{\mathbf{h}}_1$ to be any point on the line $\mathbf{l}_{13}$. It also restricts κ_1 to the range $(0, 1)$, which can help reduce the search space. Although not specified in order to make the algorithm description easier, κ_1 must be selected so that $\underline{\mathbf{h}}_1$ is not collinear with $\underline{\mathbf{h}}_3$.

Step A–3.4. This follows from Eq. 4.16.

Step A–3.5. As in step (3), this parameterization allows $\underline{\mathbf{h}}_2$ to be any point on the line $\mathbf{l}_{23}$.

Step A–3.6. From Eq. 4.15. Makes $\mathbf{h}_3$ and $\mathbf{F}$ have the same scale.

Step A–3.7. If $s_1\mathbf{h}_1 = \underline{\mathbf{h}}_1$ and $s_2\mathbf{h}_2 = \underline{\mathbf{h}}_2$ then we can define $\mathbf{u}_1 = \underline{\mathbf{h}}_2 \times \mathbf{h}_3 = s_2\mathbf{h}_2 \times \mathbf{h}_3$ and $\mathbf{u}_2 = \mathbf{F}^{\mathsf{S}}\underline{\mathbf{h}}_1 = s_1\mathbf{h}_2 \times \mathbf{h}_3$, where the latter formula comes from Eq. 4.16. Note that $\mathbf{u}_2$ was already computed in step (4). We can now find $\phi = s_1/s_2$ using $\mathbf{u}_1\phi = \mathbf{u}_2$:

$$\phi = \mathbf{u}_1^\top \mathbf{u}_2 / \left\|\mathbf{u}_1\right\|^2$$

Step A–3.8. γ' serves as the unknown scale factor that makes $\underline{\mathbf{h}}_1$ consistent with $\mathbf{F}$. $\gamma' = 1/s_1$.

Step A–3.9. $\gamma'\phi = (1/s_1)(s_1/s_2) = 1/s_2$.

A.2.4 Derivation of Algorithm C

Only the linear system in step (3) is of interest. We know that $\mathrm{conhin}(\mathbf{F})$ has at least one element, say $\mathbf{H}^\infty$, and that at least one $\mathbf{a} \in \mathbb{R}^3$ satisfies Eq. 4.4. The linear system arises from placing constraints on $\mathbf{a}$ using properties that we know $\mathbf{H}^\infty$ must satisfy. We will not attempt to prove that this system has a unique null eigenvector or determine under what conditions the eigenvector is unique; we can only cite our experimental results as evidence that this approach leads to a unique and correct $\mathbf{H}^\infty$.

The properties of $\mathbf{H}^\infty$ that we use are (1) the angle θ of the underlying screw rotation is encoded in $\mathbf{H}^\infty$, (2) $\mathbf{H}^\infty$ (when scaled correctly) is conjugate to a rotation matrix and fixes all points on the rotation axis (i.e., $\mathbf{H}^\infty\mathbf{h}_3 = \mathbf{h}_3$), and (3) the vanishing line of all planes that are perpendicular to the rotation axis is fixed in all views (i.e., $(\mathbf{H}^\infty)^\top\mathbf{l}_{12} \cong \mathbf{l}_{12}$).

Assume $\mathbf{H}^\infty$ is scaled so that $\det(\mathbf{H}^\infty) = 1$, making $\mathbf{H}^\infty$ conjugate to a rotation matrix, and let $\lambda \in \mathbb{R}$ be the scale factor that makes Eq. 4.4 an equality:

$$\mathbf{H}^\infty = \lambda(\mathbf{M} + \mathbf{e}\mathbf{a}^\top)$$

Then by the second property listed above,

$$\underline{\mathbf{h}}_3 = \mathbf{H}^\infty\underline{\mathbf{h}}_3 = \lambda\mathbf{M}\underline{\mathbf{h}}_3 + \lambda\mathbf{e}\mathbf{a}^\top\underline{\mathbf{h}}_3$$

leading to rows 2-4 of the linear system. Rows 5-6 come from the third property:

$$\mathbf{l}_{12} \cong (\mathbf{H}^\infty)^\top\mathbf{l}_{12} \cong \mathbf{M}^\top\mathbf{l}_{12} + \mathbf{a}\mathbf{e}^\top\mathbf{l}_{12} = \mathbf{q} + \xi\mathbf{a}$$

and so

$$(\mathbf{l}_{12})_x(\mathbf{q}_y + \xi\mathbf{a}_y) = (\mathbf{l}_{12})_y(\mathbf{q}_x + \xi\mathbf{a}_x) \tag{A.13}$$

$$(\mathbf{l}_{12})_x(\mathbf{q}_z + \xi\mathbf{a}_z) = (\mathbf{l}_{12})_z(\mathbf{q}_x + \xi\mathbf{a}_x) \tag{A.14}$$

The first row uses the angle of rotation that is encoded in $\mathbf{H}^\infty$: Because $\mathbf{H}^\infty$ is conjugate to a rotation matrix, it has eigenvalues 1, $\exp(\theta i)$, and $\exp(-\theta i)$, and since the trace of a matrix is the sum of its eigenvalues,

$$\begin{aligned} 1 + 2\cos(\theta) &= 1 + \exp(-\theta i) + \exp(-\theta i) = \mathrm{Tr}(\mathbf{H}^\infty) \\ &= \lambda\left(\mathbf{M}_{(11)} + \mathbf{M}_{(22)} + \mathbf{M}_{(33)} + \mathbf{e}_x\mathbf{a}_x + \mathbf{e}_y\mathbf{a}_y + \mathbf{e}_z\mathbf{a}_z\right). \end{aligned}$$

A.2.5 Derivation of Algorithm D

Step D.1. By assumption, $i = 1$. The goal is to find $\mathbf{a}$ such that the following holds:

$$\mathbf{H}^{\infty}_{1j} \cong \mathbf{H}_j + \mathbf{e}_j\mathbf{a}^{\top} \tag{A.15}$$

This equation comes from Eq. 4.7; we only want equality up to a scale factor because we want the coefficient of $\mathbf{H}_j$ to be 1. We will also meet this coefficient condition in steps D.3–D.5 when $i \neq 1$, ensuring that all the resulting screw-transform manifolds will be at the same overall scale (because the same set of $\mathbf{H}_j$ matrices and $\mathbf{e}_j$ vectors will be used throughout the self-calibration process).

Let $-\sigma$ be the scale factor that makes the left-hand side of Eq. A.15 equal to the right. We get

$$-\sigma\mathbf{H}^{\infty}_{1j} = \mathbf{H}_j + \mathbf{e}_j\mathbf{a}^{\top} = \mathbf{H}_j + \mathbf{a}_x\mathbf{E}_1 + \mathbf{a}_y\mathbf{E}_2 + \mathbf{a}_z\mathbf{E}_3$$

which is the linear system to be solved in this step. Since the null eigenvector will only be found up to a scale factor, it is necessary to divide by the *second* component of the eigenvector (corresponding to $\mathbf{H}_j$) to recover $\mathbf{a}$ at the correct scale.

Steps D.3–D.5. Using Eq. A.15 twice (following the pattern of Eq. 4.8), we get

$$\mathbf{H}^{\infty}_{ij} \cong (\mathbf{H}_j + \mathbf{e}_j\mathbf{a}^{\top})(\mathbf{H}_i + \mathbf{e}_i\mathbf{a}^{\top})^{-1} \tag{A.16}$$

Let $-\phi$ be the scale factor that makes the right-hand side of Eq. A.16 equal to the left. We will work in stages, first solving for ϕ then recovering $\mathbf{a}$.

Rearranging Eq. A.16 leads to:

$$\begin{aligned} \mathbf{H}^{\infty}_{ij}(\mathbf{H}_i + \mathbf{e}_i\mathbf{a}^{\top}) &= -\phi(\mathbf{H}_j + \mathbf{e}_j\mathbf{a}^{\top}) & \text{(A.17)} \\ \mathbf{H}^{\infty}_{ij}\mathbf{H}_i + \phi\mathbf{H}_j &= -(\mathbf{H}^{\infty}_{ij}\mathbf{e}_i + \phi\mathbf{e}_j)\mathbf{a}^{\top} & \text{(A.18)} \end{aligned}$$

The right-hand side of Eq. A.18 is a rank 1 matrix with columns in the same 1-dimensional space. Thus so is the left-hand side, and the cross-product of any two columns on the left-hand side must vanish. Hence, defining $[\,\mathbf{q}_1\ \mathbf{q}_2\ \mathbf{q}_3\,] = \mathbf{H}^{\infty}_{ij}\mathbf{H}_i$ and $[\,\mathbf{m}_1\ \mathbf{m}_2\ \mathbf{m}_3\,] = \mathbf{H}_j$ we get

$$\mathbf{0} = (\mathbf{q}_1 + \phi\mathbf{m}_1) \times (\mathbf{q}_2 + \phi\mathbf{m}_2) = \mathbf{q}_1 \times \mathbf{q}_2 + \phi(\mathbf{q}_1 \times \mathbf{m}_2 + \mathbf{m}_1 \times \mathbf{q}_2) + \phi^2\mathbf{m}_1 \times \mathbf{m}_2 \tag{A.19}$$

which can be solved for ϕ. This covers steps D.3–D.4.

An alternative solution for ϕ to the one given in step (4) of Fig. 4.5 arises from multiplying Eq. A.19 by $\mathbf{v}_1^\top$, leading to $\phi^2\mathbf{v}_1^\top\mathbf{v}_1 + \phi\mathbf{v}_1^\top\mathbf{v}_2 + \mathbf{v}_1^\top\mathbf{v}_3 = 0$ which can be solved with the quadratic equation. Two solutions for ϕ will arise; use either one that satisfies Eq. A.19.

Once ϕ has been determined, step (5) follows directly from Eq. A.17 using the logic of step (1). The vector $\mathbf{a}$ will have the correct scale because Eq. A.16 was derived from Eq. A.15 and the missing scale factor ϕ was determined as a separate step. In other words, the derivation ensures that the coefficient on each $\mathbf{H}_i$ is 1.

Appendix B: Implementation Details

B.1 Voting algorithm details

This section discusses some important implementation details for the voting-based manifold-intersection algorithm (Section 5.2). Part of the reason we are presenting these details is that the underlying ideas may have uses beyond the voting algorithm and screw-transform manifolds. The conditioning of Section B.2 is always needed when using screw-transform manifolds with the parameterization of $\underline{\mathbf{h}}_3$ given by Eq. A.12; this conditioning is not specific to the voting algorithm.

B.1.1 Forced-spread sampling

The voting scheme for determining manifold intersection points (Section 5.2) is simple in principle but care must be taken to sample the manifolds efficiently. If samples are generated in a completely random manner, after several "zoom-in" steps very few randomly-generated samples will lie within the current (smaller) search volume and the algorithm will converge more and more slowly. Samples must be generated so that they (1) have a high probability of lying in the current search range and (2) cover the search range uniformly so that all possible areas of convergence are accounted for. In this section, we present an approach to sampling termed the *forced-spread algorithm* that meets the goals just described; in practice, the algorithm leads to convergence quickly provided the given fundamental matrices are close to the true fundamental matrices.

Recall that each sample point on a manifold is generated from some underlying parameters, which can be thought of as the coordinates of the sample. The forced-spread algorithm generates new sample points by slightly altering the coordinates of existing (previously generated) samples. To ensure a uniform spread of samples across a manifold, only one sample in a given region is allowed to create new samples; such a sample point is termed *fecund* and the samples it generates are thought of as its *offspring*. Furthermore, to ensure the fast spread of sample points across a manifold, only the most-recently generated samples are allowed to be fecund (i.e., allowed to have offspring). This encourages exploration of the newly populated (and thus less filled out) areas of the manifold.

Also recall from the general description of the voting algorithm in Section 5.2 that the search volume V_i is subdivided into voting voxels. Since V_i is a hypercube, it is subdivided equally with *nVoteVoxelsPerSide* per side. Let η denote the dimensionality of the search space, so $\eta = 5$ in the case of direct self calibration in K-space and $\eta = 3$ in the case of stratified self calibration in a-space. Thus $(nVoteVoxelsPerSide)^\eta$ voting voxels are needed: one η-dimensional array[1] of integers is needed to tally the votes cast for each voxel and one η-dimensional array of Booleans is needed for each manifold to keep track of whether a manifold has cast a vote yet for any particular voting voxel (since each manifold is allowed to cast at most one vote for any particular voting voxel).

Keeping track of the fecund samples requires a similar mechanism: the current search region V_i is subdivided into *fecund voxels* (like voting voxels) and at most one fecund sample is allowed per fecund voxel. This can be handled with a η-dimensional array of Booleans. Let *nFecundPerSide* denote the number of fecund voxels per side of V_i; thus the necessary array stores $(nFecundPerSide)^\eta$ Boolean variables, and one such array is needed per manifold. It should be true that $nFecundPerSide = k(nVoteVoxelsPerSide)$ for some integer k to achieve a uniform spread of fecund samples over the voting voxels. Typically a value of $k = 1$ or $k = 2$ is used; larger than this defeats the purpose of using the fecund-sample mechanism to force samples to spread quickly over the manifold.

The forced-spread algorithm can now be stated; see the comments after the algorithm for further explanation about the variables:

(1) Choose an initial search volume V_0 that contains the mutual intersection point (see Section 5.2).

(2) **Initialize arrays:** There is a η-dimensional array of small integers used to keep track of votes cast for each voxel, and for each manifold i there are two η-dimensional arrays of Booleans: one for tracking whether or not manifold i has voted for a particular voxel yet, and one for marking a region of the search space as having a fecund sample associated with manifold

[1] Here an η-dimensional array is an array with η indices. This does not specify the size of the array; each dimension of the array may have its own size.

i. Only one fecund sample is allowed per fecund voxel (although other non-reproducing samples are allowed to lie within a fecund voxel).

(3) **Seed the manifolds:** For each screw-transform manifold, generate random points on the manifold (by choosing random (κ, θ, γ) triplets or (κ, θ) doublets, depending on η) until one lies in the initial search region V_0.

(4) **Clear lists and set variables:** For each manifold i, initialize two empty lists S_i and F_i. The list S_i will hold all samples so far generated for manifold i that lie within current search volume V_j. The list F_i will hold fecund samples. Add the seed sample points from step (3) to both lists. Mark all manifolds as "active."

(5) **Generate samples:** Let $|L|$ denote the size of list L. For each manifold i marked "active" with $|F_i| <$ *minimumFecundCount*:

(5.1) Allow the *nAllowedToReproduce* (about 20 is good) most recently created fecund samples on manifold i to produce offspring. Use the *range dithering* approach described in Appendix B.1.2 when generating the offspring.

(5.2) Add all offspring to the samples list S_i. If an offspring lies in a voting voxel that manifold i has not yet voted for, cast a vote for the voxel and mark the voxel so that manifold i cannot vote again for it.

(5.3) If an offspring lies in a fecund voxel that has no fecund samples yet from manifold i, add the offspring to the list F_i and mark the fecund voxel so that manifold i will have no more fecund samples within it (this will not affect how other manifolds use that fecund voxel).

(5.4) If a manifold i seems unable to generate any more fecund samples within a reasonable amount of tries, mark the manifold as "not active" (the manifold is not keeping pace with the other manifolds and is being thrown out).

(6) If some manifold i marked "active" has $|F_i| <$ *minimumFecundCount*, return to step (5).

(7) If *minimumFecundCount* $<$ *sufficientFecundCount*, increase *minimumFecundCount* by *minimumFecundCountIncrement* and return to step (5).

(8) **Determine if voting has proceeded long enough:** Determine the voxel v that has the most votes. If v has less votes than the threshold needed for "zooming in" then increase *minimumFecundCount* by *minimumFecundCountIncrement* and return to step (5). If, over time, *minimumFecundCount* continues to be incremented without a mutual intersection point being found, signal failure and exit the algorithm. Alternatively, the algorithm could backtrack to a previous, larger search region and the voxel that led to the current failure could be barred from receiving votes in the future.

(9) **Check for linear solution:** Check the "linearity" of all manifolds marked "active." If enough are sufficiently linear, find the point of intersection of the linear manifolds and return this as the answer; see Eq. 5.1 and accompanying text.

(10) **Zoom-in step:** If no linear solution exists, perform the zoom-in process by following the steps below, then reset *minimumFecundCount* to its initial value and return to step (5).

(10.1) Determine the new, smaller search region V' as follows:

(10.1.1) Determine the center of V'. Typically, the voting voxels near the max voxel v (from step (8)) will also have a large number of votes and may well contain the sought-after point of mutual intersection. Thus find the center of mass of all voting voxels in a volume centered on v and use this as the center of the new search region. For instance, find the center of mass of all voting voxels within 2 voxels of v. The "mass" of a voting voxels is the number of votes it received.

(10.1.2) The width of the new search region is taken to be, for example, half the width of the old search region.

(10.3) Reset (clear) the arrays used to keep track of votes. These are the arrays described in step (2). The array sizes remain the same, so there is no need to reallocate memory. However, the entries will correspond to voxels in the new, smaller search region V'.

(10.4) For each manifold i marked "active," create an initial samples list $S_i{}'$ for the new search region V' by keeping only those samples from S_i that lie in V'.

(10.5) Treating the members of $S_i{}'$ as newly generated samples, follow the procedure of step (5.2) to cast votes in the new search region and follow the procedure of step (5.3) to create an initial fecund samples list $F_i{}'$and mark fecund voxels.

(10.6) Replace V with V', S_i with $S_i{}'$, and F_i with $F_i{}'$.

At first, all manifolds are considered "active." Over time however, some manifolds (e.g., outliers) will no longer have a presence in the current search region. These manifolds are marked as "not active" and will thenceforth not be considered. Manifolds that are having difficulty generating samples in the current search region (e.g., due to a bad parameterization) will also get marked "not active."

The variable *minimumFecundCount* is used to keep the number of fecund samples on each manifold about equal. This mechanism is necessary because each manifold has a different shape and parameterization, and on some manifolds fecund samples will be easy to generate while on others they will arise slowly. Once a manifold reaches its quota of fecund samples (set by the variable *minimumFecundCount*), it will not generate new samples until the other manifolds catch up. We typically initialize both the variable *minimumFecundCount* and the constant *minimumFecundCountIncrement* to 50.

The constant *sufficientFecundCount* in step (7) is used to prevent voting decisions from being made until a sufficient number of votes have been cast by each manifold.

Now consider how the forced-spread algorithm meets the goals of efficient sampling. First, the fecund samples are spread out from each other helping to ensure uniform coverage of the manifold in the current search region. Next, since only the fecund samples can reproduce, new samples are generated evenly over the manifold. Finally, since only the most recently-generated fecund samples reproduce, the "unexplored" areas on the frontiers of expansion receive the new samples. If older fecund samples had been allowed to continue producing offspring, lots of new samples would be generated in areas of the manifold that had already been sampled, wasting computational effort.

B.1.2 Range dithering

When points on a screw-transform manifold are near each other, they will have similar underlying manifold coordinates; i.e., similar values of κ, θ, and γ. The closer the points are, the closer the manifold coordinates are. After several zoom-in steps, all samples on a particular manifold will be near each other and thus will have similar manifold coordinates. The key to efficiently generating new samples on the manifold is to only use coordinates that are in the range of existing samples in the current search region. Thus after step (10) but before returning to step (5) the following can be done: For each manifold i, find the maximum and minimum values of κ, θ, and γ for all samples in the list S_i; call these κ_{min}, κ_{max}, and so on for θ and γ. Let $\Delta_{i,\kappa} = (\kappa_{\text{max}} - \kappa_{\text{min}})/10$, and so on for θ and γ. Now in step (5), whenever a fecund sample is used to generate a child on manifold i, the coordinates of the new offspring sample are chosen by slightly altering (i.e., "dithering") the parent's coordinates, using $\Delta_{i,\kappa}$, $\Delta_{i,\theta}$, and $\Delta_{i,\gamma}$ as a guide for what "slightly altering" means. The specific method used by our implementation is given in the pseudocode below, which is for modulus-constraint manifolds and thus does not involve coordinate γ. In the code, **dKa** and **dTh** denote $\Delta_{i,\kappa}$ and $\Delta_{i,\theta}$.

```
// Goal:    produce a manifold sample point near a given
//          sample point, using range dithering
// Input:   coordinates (Ka, Th) of given manifold point;
//          dKa and dTh (see discussion)
// Output:  coordinates (newKa, newTh) of new sample point
// Notes:   rand01() returns a random number in range [0,1]
//          randNP() returns a random number in range [-1,1]
r=rand01();
if (r>0.80) then begin
   newKa=Ka+randNP()*dKa*0.1;
   newTh=Th+randNP()*dTh*0.1;
end else
if (r>0.30) then begin
   newKa=Ka+randNP()*dKa;
   newTh=Th+randNP()*dTh;
end else
```

```
if (r>0.05) then begin
   newKa=Ka+randNP()*dKa*10;
   newTh=Th+randNP()*dTh*10;
end else begin
   newKa=Ka+randNP();
   newTh=randNP()*3.1415926535;
end
return manifold point with coordinates (newKa, newTh)
```

B.2 Conditioning the parameterization of h_3

It is important to realize that the equation $\underline{\mathbf{h}}_3 = (\kappa\mathbf{I} - \mathrm{M})^{-1}\mathrm{e}$ used to find $\underline{\mathbf{h}}_3$ in step (2) of Fig. 4.2 can represent an ill-conditioned system. This means that, under the right conditions, small changes in κ will lead to large changes in $\underline{\mathbf{h}}_3$. We now discuss how to condition the system.

Note that $\underline{\mathbf{h}}_3$ is the image of the vanishing point of the screw axis. If one considers the viewing sphere around a camera's optical center, then the possible locations of $\underline{\mathbf{h}}_3$ form a 1-dimensional manifold on the surface of the sphere. As κ goes towards infinity, $\underline{\mathbf{h}}_3$ approaches e from one direction along this manifold (e is also on the manifold) and as κ goes towards negative infinity, $\underline{\mathbf{h}}_3$ approaches e from the other direction. Once κ gets large enough (e.g., $|\kappa| > 10$), $\underline{\mathbf{h}}_3$ is very close to e and stops changing in any meaningful way. Thus we can assume $\kappa \in [-\mu, \mu]$ for some fixed, sufficiently-large μ. This means we can treat κ as being a real number in $[0, 1]$, which then gets mapped into $[-\mu, \mu]$.

The observation that κ has a finite range provides a way to condition the equation for $\underline{\mathbf{h}}_3$: establish a map from $[0, 1]$ to $[-\mu, \mu]$ that produces $\underline{\mathbf{h}}_3$ at approximately regularly-spaced intervals along the 1-dimensional manifold on the viewing sphere. A very approximate estimate of internal calibration should be used to stretch the viewing sphere so that "regularly-spaced intervals" is meaningful; that is, the Euclidean distance between each sample $\underline{\mathbf{h}}_3$ (normalized to unit length) should be roughly equal in metric space (hence the need for approximating the internal camera calibration).

The conditioning map in our implementation sends 50 equally-spaced "guide" numbers in $[0, 1]$ to 50 numbers in $[-\mu, \mu]$ that yield evenly-spaced $\underline{\mathbf{h}}_3$ in approximate metric space. The remaining members of $[0, 1]$ are mapped by linearly interpolating the image of the nearest two guides. Fig. B.1 shows our conditioning algorithm. Despite its simplicity, this method conditions the samples very effectively using ideas similar to bisection methods; in particular, it is immune to problems that might arise from trying to calculate derivatives for this unstable system. Our implementation uses $a = -\mu$, $b = \mu$, and $N = 50$. Furthermore, $f(k) = (k\mathbf{I} - \mathbf{M})^{-1}\mathbf{e} / \|(k\mathbf{I} - \mathbf{M})^{-1}\mathbf{e}\|$. For $N = 50$ we use 10000 iterations of the loop (steps 2-5); for $N = 100$ we use 100000 iterations.

It is possible that the best way to condition the parameterization of $\underline{\mathbf{h}}_3$ is to use a completely different parameterization. Recall that $\underline{\mathbf{h}}_3^\top \mathbf{F}^\mathsf{S} \underline{\mathbf{h}}_3 = 0$ by Eq. A.8 and thus $\underline{\mathbf{h}}_3$ lies on the cone $\mathbf{F}^\mathsf{S}$. Hence κ could be used to simply parameterize all points on the intersection of cone $\mathbf{F}^\mathsf{S}$ with the unit sphere (after the camera space has been normalized using an estimate of internal calibration, as described above). We have not experimented with this parameterization so can say nothing further about how well it might work.

ALGORITHM (CONDITIONING)

Goal: Find N real numbers $k_i \in [a, b]$ with $k_1 = a$ and $k_N = b$ such that $\|f(k_i) - f(k_{i+1})\|$ is approximately the same for all i. Note the range space of f is $\mathbb{R}^n$.

Comment: Since only k_j changes during each iteration, it can be efficient to precompute the value of $f(k_i)$ for each i and store these values in an array, changing only the entry corresponding to k_j during the loop.

(1) Initialize k_i to be equally spread across $[a, b]$; i.e., $k_i = a + (b - a)(i - 1)/(N - 1)$

(2) Pick one of the samples k_j at random, avoiding the two endpoints; i.e., let j equal a random number between 2 and $N - 1$.

(3) Find the distance between the image of k_j and the images of its two neighbors; i.e., let $d_0 = \|f(k_j) - f(k_{j-1})\|$ and $d_1 = \|f(k_j) - f(k_{j+1})\|$.

(4) Change k_j so that the image of k_j is more equally-spaced between the images of its neighbors. We do this as follows: If $d_0 < d_1$ then let $k_j = k_j + (k_{j+1} - k_j)/10$; otherwise, let $k_j = k_j + (k_{j-1} - k_j)/10$.

(5) Repeat from step (2) a fixed number of times dependent on N; experiments can be performed ahead of time to determine a good, general-purpose number of iterations.

Figure B.1 Algorithm for conditioning the coordinate system of a 1D manifold.

INDEX

a-space, 80
absolute quadric, 59–61
abstract feature points, 86
affine calibration, 69, 141
affine matrix, 53
affine reconstruction, 53
affine transformation, 53
apparent linear motion, 142

biepipolar line, 173
biepipolar plane, 173

calibration, 1, 31
calibration space, 5
camera, 35
  coordinates, 34
  matrix, 32
  obscura, 35
  projection function of, 31
  triplet, 31
  view, 31
camera matrix
  derivation, 35
  Platonic form of, 32
  specific to a basis, 32
camera relativism, 68
camera-to-camera transformation, 163
conjugate points, 50
coordinate system of a manifold, 74
coordinates
  grid, 36
  homogeneous
    direction, 41
    image position, 40
    world position, 39
  image, 36
  of camera, 34
  transforming
    for directions, 47
    for planes, 45
    world to camera, 39
critical motion sequences, 77, 82
cross-product matrix, 48

direct self calibration, 73
direction vectors, 40
displacements, 81
dual Htensor, 144
duality
  of lines and planes, 43
  of points and lines, 43
dynamic scene, 138
dynamic view morphing, 158

ego motion, 4
environment mapping, 10
epipolar constraint, 5
epipolar geometry, 2, 46–52
epipolar lines, 49
epipolar planes, 49
epipole, 46
  calculating, 51
  right and left, 52
equality, scalar, 40
essential matrix, 19
Euclidean reconstruction, 55
Euclidean transformation, 55
external calibration, 32

fecund sample, 207
fecund voxel, 208
field-of-view, 45
fixed-camera formulation, 144, 160

for rotational motion, 170
focal length, 38
forced-spread algorithm, 207
fundamental matrix, 5, 19, 48–52
calculating, 50, 63–65
definition, 48
equivalence with epipolar geometry, 52
for an object, 146, 161
monocular, 69
not invertible, 49
physical interpretation, 48
transpose of, 51

general motion, 70
goodness of solution, 124–126
granularity, 117

H-infinity matrix, 43
homogeneous
direction representation, 41
image coordinates, 40
intersection of two lines, 44
line representation, 44
plane representation, 42
world coordinates, 39
homography induced by the plane at infinity, 43, 47, 68

IBR, *see* image-based rendering
IBRM, *see* image-based rendering and modeling
image coordinates, 32
image of a point, 34
image plane, 34
image sphere, 46
image-based rendering, 4, 9
image-based rendering and modeling, 15
internal calibration, 32
K, **31**
in coordinate system w, 32
upper-triangular matrix, 36
interpolation sequences, 9
iris, 34

K, *see* internal calibration
Kruppa-constraint manifold, 74

lattice position, 103
line on the plane at infinity, 45
linear motion, 159

manifold, 74
metric calibration, 69
metric reconstruction, 55
model, 15
modulus constraint, 77
modulus-constraint manifold, 80
monocular fundamental matrix, 69
mosaics, 4

object, 142, 159
background, 164
object width, 112
offspring, 207
optical center, 31

pairwise camera motion, 81
parallel motion, 165
pencil of planes, 49
photogrammes, 18
photogrammetry, 3, 18
physically correct, 10
pinhole camera, 1, 31
basic
projection algorithm of, 33, 36–37
history, 34–35
physical interpretation, 34–39
projection function of, 32
plane at infinity, 41–42
Platonic
concept, 28
direction, 40
equations, 45
label, 28
matrix, 32, 43
vector, 28
plenoptic modeling, 4

point correspondence
  dense, 62
  finding, 61–63
  individual, 50, 62
  sparse, 62
point on the plane at infinity, 41
point-on-line test, 44
prewarping, 162
principal point, 38
projection function, 31
projection of a point, 34
projective geometry, 39–46
projective reconstruction, 20, 53, 76
  from fundamental matrix, 58–59
projective transformation, 53

R, **31**
range dithering, 100, 209
reconstruction, 56
  affine, 53
  Euclidean, 55
  hierachy of, 57
  hierarchy of, 52
  metric, 55
  projective, *see* projective reconstruction
reference views, 9
reflection mapping, 10
relative calibration, 68
rising turntable, 68
rising-turntable formulation, 68

sampling, 11
scene, 31
scene reconstruction, 52–61
screw axis, 66
screw transformation, 66
search region, 98
self calibration, 2, 20
sketching a manifold, 97
static scenes, 138
static view morphing, 158
stereo rig, 25
stratified self calibration, 75
structure from motion, 138
sub-lattice position, 103
surface fitting, 5
synthetic views, 9

T, **31**
texture, 63
texture mapping, 9
tilt of camera, 67
torpedo data computer, 5
transfocal motion, 81, 84, 180
triangulation, 3, 57–58
turntable motion, 82

unifocal motion, 81

view interpolation, 9
view morphing, 157, 178
view-dependent texture, 16
virtual views, 9
voting voxels, 98

weak calibration, 69
weakly calibrated, 22
wire grid, 1, 2
world camera, 47
world coordinates, 31

zoom-in step, 98, 99, 105