Vision-assisted modeling for model-based video representations

by

Shawn C. Becker

B.S. in Design Engineering Technology, Brigham Young University 1987
M.S. in Computer Science, Brigham Young University 1990

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1997

Author .... [illegible] ....
Program in Media Arts and Sciences,
School of Architecture and Planning
October 25, 1996

Certified by .... [illegible] ....
V. Michael Bove Jr.
Associate Professor
Program in Media Arts and Sciences,
School of Architecture and Planning
Thesis Supervisor

Accepted by .... [illegible] ....
Stephen A. Benton
Chairman, Departmental Committee on Graduate Students
Program in Media Arts and Sciences,
School of Architecture and Planning

Vision-assisted modeling for model-based video representations

by

Shawn C. Becker

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
on October 25, 1996, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The objective of this thesis is to develop algorithms and software tools for acquiring a structured scene representation from real imagery which can be used for 2-D video coding as well as for interactive 3-D presentation.

A new approach will be introduced that recovers visible 3-D structure and texture from one or more partially overlapping uncalibrated views of a scene composed of straight edges. The approach will exploit CAD-like geometric properties which are invariant under perspective projection and are readily identifiable among detectable features in images of man-made scenes (i.e. scenes that contain right angles, parallel edges and planar surfaces). In certain cases, such knowledge of 3-D geometric relationships among detected 2-D features allows recovery of scene structure, intrinsic camera parameters, and camera rotation/position from even a single view.

Model-based representations of video offer the promise of ultra-low bandwidths by exploiting the redundancy of information contained in sequences which visualize what is largely a static 3-D world. An algorithm will be proposed that uses a previously recovered estimate of a scene's 3-D structure and texture for the purpose of model-based video coding. Camera focal length and pose will be determined for each frame in a video sequence taken in that known scene. Unmodeled elements (e.g. actors, lighting effects, and noise) will be detected and described in separate compensating signals.

Thesis Supervisor: V. Michael Bove Jr.
Title: Associate Professor
Program in Media Arts and Sciences,
School of Architecture and Planning

Doctoral Dissertation Committee

Thesis Advisor .... [illegible]

V. Michael Bove Jr.
Associate Professor
Program in Media Arts and Sciences

Thesis Reader ....

Alex P. Pentland
Associate Professor of Computers,
Communication and Design Technology
Program in Media Arts and Sciences

Thesis Reader .... [illegible]

Seth Teller
Assistant Professor of Computer Science and Engineering
Department of Electrical Engineering and Computer Science

Contents

Chapter 1

Introduction

A great deal of information about the original camera geometry, scene structure and texture can be gleaned from even a small number of uncalibrated views provided the scene contains a sufficient amount of structural information. Certain CAD-like geometric relationships which are invariant under perspective projection and normal lens distortion are visible in man-made scenes and can be readily identified. By exploiting these properties among manually specified critical sets of 2-D features this thesis proposes to find a coarse estimate for lens distortion, center of projection, camera pose and scene geometry. A refinement stage then uses all feature correspondences from multiple views to refine camera and scene parameters while maintaining known geometric constraints. Image textures from multiple views are registered into common surface coordinates and then merged according image resolution and visibility.

If matching and property assumptions of features are correct, if detector noise is reasonable, and if textures are invariant over all views, the final result is a fairly photorealistic textured static 3-D scene model which is true to all views. While this model has several useful applications, model-based video coding is of particular interest. Using the model as prior scene knowledge, we can attempt to recognize novel views of that scene. This task of *pose-determination* can be applied to each frame in a video sequence. Most existing approaches to pose-determination rely solely upon point or line feature-based matching. The fact that we have an *a-priori* scene model consisting of both geometry and texture, rather than geometry alone, should improve the accuracy and convergence of pose-determination even in the presence of significant levels of noise (due to modeling error) and clutter (e.g. dynamic unmodeled elements like people).

If the difference between real and predicted sequences is appreciable, it may be possible to segment unmodeled elements into coherent image regions (i.e. 2-D sprites) corresponding to actor extent as viewed from the camera. These 2-D actor sprites can be assigned arbitrary but meaningful 3-D attributes such as having planar orientation normal to the viewing camera and parallel to the scene's vertical axis. If contact between 2-D sprites and any surface of the scene can be determined (e.g. the bottom of the sprite touches the ground plane) or occlusion with known objects can be determined (e.g. the sprite is behind the table) then the sprite can also be given appropriate 3-D position and scale. We might also be able to identify and model other sources for appreciable error, such as environmental lighting changes, shadows, and noise.

1.1 Issues

1.1.1 Motion vision

Existing motion vision approaches like *structure-from-motion* and *shape-from-stereo* use signal processing techniques to detect and match points or lines among several views. For example, optical flow over many views can be used to recover qualitative measures of camera-relative motion and depth [52, 54, 55, 58]. Multi-baseline-stereo approaches [34] use optical flow or matched features over views with known motion to recover scene structure and texture. Structure-from-motion approaches[24, 25] use large numbers of automatically detected and tracked features over almost-calibrated views[1] to recover camera-relative motion and depth.

1.1.2 Modeling

At the other end of the scale CAD approaches to scene modeling use critically small sets of features carefully specified with manual intervention. For example, the scene modeling approaches used to create environments for visual simulation [120, 124] use known cameras and scene geometry found from architectural plans or physical measurements. In [98] the user manually corresponds user defined image lines to 3-D edges of manually created sets of parametric volumes which are then automatically refined to minimize observed deviation from pre-calibrated cameras. Other systems, like [127], use manually specified point and line features with calibrated cameras at roughly known camera positions.

1.1.3 Shape from wireframes and contours

Another group of approaches use *shape-from-wireframe* and *shape-from-contours* methods which attempt to use features with known properties to recover object structure or shape. Shape-from-wireframe algorithms aim to recover 3-D geometry of a polyhedral scene (e.g. "blocks world" or "LEGO") from a single line drawing. Initial work [104, 106, 105] focussed on the problem of partitioning wireframe scenes into separate trihedral polyhedral objects made up of collections of labeled vertices and edges. More recently algorithms have been presented [60, 61] which find 3-D polyhedral interpretations of wireframe drawings that match human observations amazingly well. In the special case of trihedral scenes with known vanishing points [13] show that line detection can be constrained and in some cases labeled automatically.

Object contours (i.e. the locus of visible surface points either tangent to the view or discontinuous in surface curvature) are another type of image cue which convey rich geometric information and are stable with respect to illumination changes. Algorithms have been presented [62] to recover a description of generalized cylinders by fitting an observed or hypothesized cross section to an observed set of contours. Such algorithms can be used to reconstruct a number of complex shapes, shapes for which polyhedrals are poor approximations.

[1]Where all intrinsic parameters except focal length are known.

1.2 Motivations

Digital cinematography, virtual reality and telepresence applications are increasing the demand for photorealism in synthetic 3-D models. To achieve this type of visual quality, some researchers are investigating the use photographic images of real scenes directly rather than attempting to synthesize geometry and texture [73, 98, 127]. Besides attacking the ill-posed problem of structure recovery this requires that we also face problems of proper registration and merging of texture samples from multiple views. Accuracy and photographic realism is critical especially if the model is to be used for other applications like model-based video coding where we hope to use a scene model as a stable representation of a scene over multiple views and lighting conditions.

Excellent image-based reconstruction results have been obtained using structure-from-motion techniques. However, since state of the art techniques for feature detection and tracking are far from perfect, some sort of human intervention is typically required at that stage to get quality results. In light of the fact that human intervention is often used anyway, existing techniques suffer due to their modeling simplicity since they do not take advantage of powerful geometric properties readily available in many images.

Existing image-based CAD modeling approaches have also given excellent results. However, these approaches typically require an initial topologically complete and dimensionally approximate 3-D solution to start with. A faster and simpler approach might be to specify structural constraints among features in a purely 2-D environment. Existing approaches also impose restrictive viewing conditions by requiring the use of pre-calibrated cameras so the image acquisition stage must be a planned event. It would be more convenient if we had the freedom to recover a 3-D model from an existing still image or a video sequence taken from a totally unknown camera.

Another motivating factor for this work is to contribute to the goal of realizable 3-D video capture from monocular 2-D cameras. While the Media Lab's Spatial Imaging and Television of Tomorrow groups have been successful in creating holographic video displays [117, 118, 119] the task of acquiring three-dimensional video content is still an ongoing challenge. Also, since acquisition of photorealistically textured 3-D models directly from images or film is so difficult in itself few have attempted to go beyond the acquisition problem towards exploiting that knowledge for model-based video coding. With the help of an able thesis committee I feel that I have successfully made new contributions in this area.

1.3 Thesis Committee

Besides the fact that the members of this thesis committee share my enthusiasm for this topic they were chosen because of the particular interests and expertise that each can bring to this effort.

Professor Bove has experience in the topics of multi-sensor fusion, digital representations for video, and hardware systems for video coding/decoding. His knowledge and resources will make it possible to develop methods for 3-D reconstruction from real imagery of man-made scenes which can be used for low-bandwidth 2-D video coding. The results of this research will also contribute to previous work he's done with interactive 3-D structured video.

Professor Teller has experience in designing efficient representations for storage, retrieval and rendering of large complex graphical databases. Since texture rendering is an inherent part of this approach to video coding and decoding, his knowledge may help find methods to animate photorealistically textured 3-D scenes with incremental changes in camera position. His shared interest in acquiring realistic scenes for visual simulation also make him a natural choice as a committee member.

Professor Pentland has experience in photogrammetric methods for scene recovery as well as dynamic scene understanding. His suggestions will be valuable for developing methods to recover a representation for scene structure which is optimally stable over multiple views, to use that model to recognize novel views of that scene, and to segment image regions containing human figures when the background is known. Results of this research could be used to provide photorealistic 3-D scenes for immersive virtual environments.

1.4 Goals and contributions

The main objective of this thesis was to develop algorithms and software tools for acquiring a structured scene representation from real imagery which is amenable for 2-D video coding as well as for interactive 3-D presentation. New and significant contributions include:

- A new method of camera calibration which uses the vanishing points of three mutually orthogonal sets of parallel lines to solve lens distortion, center of projection and camera pose simultaneously.
- A new structure and motion algorithm, which besides using point and line correspondences among images, additionally exploits geometric constraints like coplanarity, parallelism and perpendicularity which can be defined with a modicum of user intervention.
- A new algorithm for tracking camera pose from live video in a scene with known texture and geometry with straight edges.
- A new algorithm for actor segmentation from live video in a scene with known texture and geometry.
- An improved method for model-based video coding which is amenable to interactive 3-D manipulation and presentation, and describes a video sequence in terms of a textured static 3-D scene model, changing camera pose, dynamic 2-D actor sprites, and a residual error signal.

Chapter 2

Background

To familiarizes the reader with issues of image-based scene reconstruction the following sections review issues in camera calibration, structure-from-motion algorithms and pose-determination. We will review recovery of structure given calibrated cameras with known pose. We will also review current work in model-based video coding and virtual reality over the World Wide Web; two areas that could benefit from the availability of highly realistic and accurate models recovered from new or preexisting imagery.

2.1 Camera calibration

Images created with a compound lens system are typically modeled by a pin-hole projection equation which describes how 3-D elements in the scene are projected onto 2-D positions in a frame buffer. This non-linear projection equation can be described as a set of intrinsic and extrinsic camera parameters. Intrinsic parameters include radial and decentering (also known as tangential or prismatic) lens distortion terms, horizontal pixel scale, pixel shear, principal point (also known as image center or optical center) and effective focal length. Extrinsic parameters (also known as camera motion or pose) describe the camera's position and rotation with respect to some reference frame: either some object in the scene, one of several fixed cameras, or a moving camera's position at some instance in time. Scene structure refers to the 3-D geometry of visible points, lines and surfaces. Whereas *camera calibration* algorithms use known (or partially known) scene structure to determine a camera's intrinsic parameters *pose determination* algorithms use known (or partially known) scene structure and precalibrated intrinsic camera parameters to determine the camera's extrinsic parameters.

2.1.1 Calibration using direct measurements

Intrinsic and extrinsic camera parameters can be obtained from physical measurements of camera itself or by using three-dimensional coordinates of known world points and their corresponding 2-D projections in the image. Typical approaches [1, 2, 3, 4, 5, 6, 7] employ specially constructed calibration templates (usually a planar grid) with a large number of precisely positioned fiducial points or lines. Due to the non-linear nature of the projection equation these calibration approaches use iterative nonlinear optimization techniques and thus are dependent upon having a reasonable initial estimate. However, by using large numbers of precisely known template features very accurately calibration parameters can be found.

If distortion can safely be considered negligible, Roberts [102] shows that the remaining parameters can be combined to form a single homogeneous 3 by 4 matrix which can be solved given 6 or more known world points and their 2-D projections in two images. To determine of the position of the camera relative to known object points, Ganapathy [103] shows how this 3 by 4 matrix can be inverted into its constituent intrinsic and extrinsic parameters.

2.1.2 Affine calibration

This matrix can also be used with corresponding pairs of 2-D points to recover a more qualitative description of scene structure. As described by Faugeras [9, 10] and others [11], given 4 non-planar point correspondences point-wise scene structure can be recovered up to an arbitrary affine transformation relative to the positions of those four points in three-dimensional space. This formulation completely evades the effects of camera calibration by recovering only relative information at the expense of losing metric. The subsequent gain is that these approaches are more stable, simpler and better behaved. Such an approach is appropriate when accurate metric information is not required, such as in robotic systems using binocular stereo vision where relative positioning is all that is needed.

2.1.3 Calibration using three-point perspective

Another class of techniques lets us make assumptions about geometric shapes of elements or relationships among elements in the scene. This offers advantages over approaches using direct scene measurements since we can recover geometry to an arbitrary scale factor from a single image. For example, it is well known that the projection of a set of parallel lines in a scene form a set of lines in an image which meet at a common *vanishing point.* If a scene contains three mutually orthogonal sets of parallel lines (i.e. a rectangular solid or right parallelepiped) an image of that scene can contain one, two or three vanishing points. Interior and exterior views of man-made buildings are prime examples of scenes containing these structures. If an image contains three vanishing points, there exist several properties of projective geometry which are particularly useful for camera calibration.

- the 3 vanishing points of the 3 sets of parallel lines form a triangle whose orthocentre is the principal point or image center.
- the vector from the center of projection to a vanishing point is parallel to the set of lines in the scene.
- the vanishing points of all lines that lie in a 3-D plane in the scene lie on a common line in the image which is called the vanishing line for that plane.
- the position of the 3 vanishing points change only due to camera rotation not camera translation.

Caprile and Torre [12] and Wang and Tsai [14] exploit these properties to determine principal point and focal length from the vanishing points in a view of a cube. These algorithms assume known or negligible lens distortion. Since they use so few features these calibration approaches are quite sensitive to detection errors.

While previous examples use 3-point perspective to calibrate cameras using rectangular solids of *known dimensions* Becker and Bove [16] show how one or more perspective projections of a right parallelepiped with 3-point perspective can be used to recover rectangular object dimensions as well as intrinsic and extrinsic parameters. Camera position and object size are solved to an arbitrary scale factor and horizontal pixel scale cannot be recovered and must be assumed unitary. No numerical results of estimation errors were presented but it was found that mean estimation error of center of projection (and consequently estimations for camera pose and scene structure) increases as the object's angle of view decreases, as mean line segment length decreases, or as true focal length increases.

Robust algorithms for finding vanishing points among sets of image lines are presented in [17, 19, 18]. Error analysis of vanishing point estimation is discussed by Kanatani in [20, 21, 22].

The disadvantage of 3-point perspective approaches to camera calibration is the requirement that sufficient rectangular structure is visible in the scene. The clear advantage is that the majority of intrinsic and extrinsic parameters can be accurately recovered without requiring explicit measurements, but rather simply by taking advantage of geometric properties of objects visible in a single image.

The 3-point perspective approach is actually a special case of affine calibration, the non-metric technique for recovering the view to world transformation up to an arbitrary affine scale. This transformation can be brought to metric by simply specifying 4 distances between tetrahedral vertices, or equivalently, 3 angles and 1 true length.

2.1.4 Plumb-line techniques

Another example of an approach that takes advantage of geometric properties in scenes to solve intrinsic camera parameters is Brown's [23] *plumb-line* technique for solving lens distortion. Given a single image of a scene of parallel edges the effects of lens distortion can be corrected by iteratively finding the distortion parameters that minimize curvature among points in each line in the distorted image. This approach to solving lens distortion differs from template-based calibration techniques in that no knowledge of world points is required. Rather, linear relationships between points (i.e. straightness) is all that is known.

Becker and Bove [16] improve upon the plumb-line technique by exploiting the vanishing point property of parallel lines as well as Brown's straightness constraint. They present an iterative technique for finding the distortion parameters which minimize the vanishing point dispersion derived using Kanatani's [20] statistical foundations. Using this formulation lines which are parallel in the scene are constrained to be straight as well as share a common vanishing point in the image.

2.1.5 Projections of spheres

Assuming negligible radial lens distortion and a scene with two or more spheres Stein [8] solves principal point, effective focal length and horizontal pixel scale by analyzing each sphere's elliptical projection on the image plane. This approach is sensitive to distortion since when even small amounts of radial lens distortion are present the affects of focal length and radial distortion are impossible to differentiate.

2.1.6 Self calibration using point matching

Another approach that is gathering interest is to use large numbers of features, and/or large numbers of frames, to overconstrain the non-linear observation equations used in structure and motion determination (which we review in greater detail in section 2.2). If these equations are sufficiently overconstrained intrinsic parameters (like focal length) can be solved along with extrinsic camera parameters and scene structure simultaneously as shown by Azarbayejani and Pentland [24, 26].

2.2 Structure-from-motion algorithms

The recovery of scene structure from 2-D images alone is a convenient and powerful means of quickly acquiring knowledge about a scene. Classic examples of approaches to visual scene reconstruction consist of a variety of so called "shape-from-X" algorithms where X might be shading, texture, focus, contour, wireframe, stereo, multi-baseline, or known motion. Structure-from-motion approaches, use the visible artifacts of "motion parallax" over two or more views to recover unknown scene structure as well as unknown camera motion. These approaches typically assume that intrinsic camera parameters have already been found by calibration, though some so-called "self-calibrating" techniques attempt to recover some intrinsic parameters as well, as mentioned in section 2.1.6. Feature correspondences are used to determine the 6 extrinsic parameters that describe the pose (i.e. position and rotation) of one camera relative to another up to an unknown scale in position. Having determined relative pose (or relative orientation) the 3-D position of features relative to one camera can be found by triangulation.

2.2.1 2-D to 2-D point correspondences

Examples of structure-from-motion techniques which use critical sets of matching 2-D points among two or more images include [32, 29, 30, 31]. To combat difficulties presented with trying to solve the non-linear projection equation with noisy or erroneous feature matches, some researchers are using long image sequences to overconstrain the problem. Several approaches have been presented [40, 41, 24] which use incremental estimation frameworks based on the Extended Kalman Filter or EKF [39]. Others [33, 27] use batch estimation frameworks having higher computational overhead, but potentially better accuracy, since they avoid the linearizing assumptions inherent in the EKF.

2.2.2 2-D to 2-D planar points and orthogonal corners

If these matched 2-D points in multiple images have some known structure this structure can be exploited to generate additional equations and this results in requiring fewer points for finding a solution for 3-D point position and camera pose. Several approaches [49, 43, 42] show how 4 or more matching points known to be coplanar provide solutions for incremental camera rotation, plane orientation and translation to an unknown scale factor. Others [15] use orthogonal corners as features, (i.e. points that mark the intersection of 3 mutually orthogonal edges). Given a calibrated camera one orthogonal corner and two point or line correspondences are all that are needed to determine motion and structure uniquely.

Given a pair of uncalibrated images we would not expect a structure-from-motion algorithm to automatically recognize such high level properties of points. We include them, however, because *their use in conjunction with an interface tool* is of exceptional value.

2.2.3 2-D to 2-D line correspondences

Straight line features offer definite advantages over point features in that they are prominent in man-made scenes and they are easier to detect and track. Straight line features provide useful structural information (as shown in section 2.1) and since they have more image support they can be localized more accurately. While these properties make lines advantageous for use in structure-from-motion algorithms they require more views and more correspondences than points. As described in [29, 35, 38] at least 3 views of 6 or more line correspondences are required, as long as the lines are not arranged in certain degenerate configurations.

2.2.4 Optical flow and spatiotemporal derivatives

Optical flow can be interpreted as the field of 2-D image motion caused by 3-D motion of a rigid object with respect to an imaging sensor. Theoretically, it can be used to recover relative motion and depth as shown by Prazdny [50], but its use for anything but a qualitative measure of structure and motion in real image sequences is questioned by Barron and others [51]. Structure-from-motion algorithms using optical flow and spatiotemporal brightness gradients directly have been presented by Adiv [52], Horn [54, 55] and Heeger and Jepson [58].

2.3 Pose-determination from known structure

Suppose we use camera calibration and structure-from-motion techniques to recover an accurate photorealistic model of a scene from a few stills or key frames of video. In this section we review how to automatically determine camera pose given a new view of that scene.

Pose-determination is usually considered a subset of the problem of object recognition where given a database of objects the task is to recognize the object in a real image. A typical strategy for object recognition is to take each object and find the transformation (i.e. camera pose) which aligns it to best fit some set of features in the 2-D image. If an object's aligned fit achieves some threshold that set of features is identified as that object with an associated pose.

2.3.1 Critical sets of 2-D to 3-D matches for pose-determination

Pose-determination techniques use the same photogrammetric methods as those mentioned in section 2.1.1 except that they are optimized to take advantage of only critically small sets of matches which are typically available in uncontrolled situations.

For the case of point features Ullman [93] shows how 3 pairs of 2-D to 3-D point matches can be used to find a unique transformation describing weak perspective (i.e. orthography and scale factor). For normal perspective 3 point pairs can be used to form a fourth-degree polynomial in one of the unknowns thus resulting in as many as 4 possible solutions for pose. Haralick [64] gives a good review of these techniques and their relative numerical stability. If 4 or 5 point correspondences are available either a linear least-squares solution can be found [29] or Fischler and Bolle's RANSAC method [63] can be used to find the solution that best describes all observations. Ganapathy [103] presents a method that in general produce a unique solution given 6 or more point correspondences as long as the points are not collinear on the image plane.

Use of line correspondences may be preferred over points because of their relative ease of detection and tracking. As shown by Liu, Huang and Faugeras [65], EKF [39] can be used to first find camera rotation given 3 line correspondences, and then 3 additional line correspondences are sufficient to determine camera translation. To explain why even non-critical sets (i.e. more than the theoretic minimum) of line correspondences do not always give robust results Navab and Faugeras [35] show that for certain degenerate configurations no unique solution for pose can be found. Kumar and Hanson [66] present a RANSAC-based iterative technique which minimizes error due to rotation and translation simultaneously and claim its superiority to techniques that solve for rotation and translation separately.

If individual matches between 2-D and 3-D point features can't be made, but sets of points can, Nagao and Grimson [88, 89] show how 2nd order statistics of spatial distributions of matched clusters of point features can be used to solve camera pose under weak perspective projection up to an unknown image plane rotation. Nagao and Horn [90] present a similar technique that uses direct moments of image intensities and intensity gradients.

2.3.2 Strategies for feature-based pose-determination

Approaches for determining arbitrary pose given a hypothesized model can be categorized into 3 groups. Pose search methods [38, 82] attempt to test the space of all poses to find the one that best fits the model to the observed image. Pose clustering methods [87] use techniques like the generalized Hough transform [17] to accumulate evidence for possible coordinate transformations from all possible matches. And alignment methods [93, 63, 86] compute pose from all possible matches of critical sets and then see if any of the remaining model features are supported by image features.

If we consider that four 2-D to 3-D matches are needed to find a unique camera pose, in the general perspective case, then straight out testing of all hypothetical pairs of model and image features has 8th order cost in the number of features. Region segmentation and region matching effectively groups features into smaller sets for comparisons. This brings the testing cost down only quadratic order in the number of regions. Approaches for color histogram-based region segmentation and matching are found in [94, 85]. Texture analysis and synthesis results can also be used for texture-based region segmentation as shown in [96, 37, 97].

Another way to prune the number of hypothesis tests needed for pose determination is to consider only those 2-D and 3-D features that have similar identifying characteristics. Properties which are invariant (i.e. consistent) even under large changes in viewing or lighting conditions are particularly useful. Some examples include junction labeling of points [84], color opponency of points and lines [89], as well as color and texture metrics for region matching.

2.3.3 Strategies for iconic-based pose-determination

While feature-based approaches to pose-determination use a difference measure which is based on mis-registration of 2-D point or line features iconic-based approaches can be used instead which compare image depths or image intensities directly. Holtzman's [82] uses a straightforward pose search strategy using textures. Viola [91] iteratively finds the pose parameters which maximize the mutual information (i.e. functional *consistency*) between pixel depths of the rendered model and observed image intensities.

Pose-determination : Synopsis

Feature-based methods for pose-determination are combinatorically expensive but robust in the presence of clutter, noise and occlusion. Iconic-based methods for pose-determination require a full texture rendering at each iteration and thus are extremely processor intensive. However, they neatly side-step the inherent problems of feature detection and matching and thus may be more applicable to parallel architectures that implement both 3-D texture rendering and 2-D image processing pipelines. For these reasons, iconic approaches also lend themselves nicely to coarse-to-fine refinement of pose over different scales. The relative strengths of feature- and iconic-based strategies suggest that a coarse to fine strategy using first iconic and then feature matching might be useful.

2.4 Recovering structure from known pose

Once camera parameters have been calibrated and camera pose has been determined stereo recovery techniques aim to estimate camera-relative depths for pixels in an image. The main challenge for stereo-based recovery is correctly matching pixels among one or more views. Given a short baseline (or distance between cameras), iconic correlation is straightforward and unambiguous since disparities between matching pixels are small. Small disparities arising from short baselines, however, result in depth estimates which are less precise due to narrow triangulation. As the baseline is lengthened the disparity and thus depth accuracy increases but the chance of getting an incorrect match increases.

To attack these dual problems Barnard [44] uses a stochastic method to integrate stereo matches over multiple scales. Correspondences over large baseline disparities are found by using a coarse-to-fine strategy over multiple levels of a Gaussian pyramid in a pair of stereographic images. Kanade, Okutomi and Nakahara [45] use a multiple-baseline stereo method that minimizes the sum of squared-difference values as a function of horizontal disparity over a more continuous range of baselines. In [34] Matthies and Kanade use Kalman filtering to integrate depth estimates over multiple baselines. Koch [46] fits a 3-D mesh to detected and matched 2-D features in stereoscopic image sequences to automatically reconstruct buildings.

More recently, Mellor, Teller and Perez [47] present a technique for recovering depth and surface normal distributions for each pixel in a sequence of epipolar images. The mean and variance of this distribution at each pixel describe estimates for depth and estimation error which can be used for achieving a maximum likelihood fit of models directly to the images.

Coorg and Teller [48] use absolute camera pose information to identify correspondences among point features in hundreds or thousands of images as continuations of line segments in image space. A geometric *incidence counting* algorithm matches features by extruding them into an absolute 3D coordinate system, then searching 3D space for regions into which many features project. Matches from hundreds or thousands of images are then used to iteratively refine camera pose.

2.5 Model-based video coding

Communicating a higher-level model of the image than pixels is an active area of research, particularly for the application of low-bandwidth video transmission for teleconferencing. The basic idea is to have the transmitter and receiver agree on the basic model for the image. The transmitter then sends parameters to manipulate this model in lieu of picture elements themselves. These types of model-based decoders are essentially computer graphics rendering programs which trade generality for extreme efficiency in its restricted domain.

2.5.1 Model-based head-and-shoulders coding

One example of a highly restricted domain is the special case of low-bitrate coding of human figures. What could be the first example of this type of coding was presented in 1983 by Forchheimer and Fahlander [69] which used animation to describe facial image sequences. They suggest the notion of giving the sender and receiver a shared set of models of objects seen in common image sequences (i.e. a human face and shoulders on a static background). For the special case of head-and-shoulder scenes Li, Roivainen and Forchheimer [70] suggest that motion parameters for head-and-shoulder scenes be separated into global rigid motion and local deformations due to facial expressions. Essa's work [71] represents the state of the art in compact model-based representations for facial expression where deformations measured by optical flow on a known 3-D face model are analyzed to find an expression vector. To make facial expression appear more life-like on the decoder Provine and Bruton [92] explore psychovisual models that "trick" the eye just as psychoacoustic models are used to "trick" the ear in audio coding.

We can get more generality if the decoder can be given a model on the fly which is then manipulated by a low-bandwidth channel. In the next sections we review the spectrum of approaches used to describe video ranging from 2-D image warping to 3-D scene modeling.

2.5.2 Image-warping models describing a static scene

A video sequence can be thought of as a group of rigid objects moving in a static scene. Contributing to the goal of determining static visual content from a set of monocular views with a moving camera Adelson and Bergen [72] formalized the concept of 2-D visibility over the entire sphere of possible directions as the plenoptic function. If we are given images from a camera with purely rotational motion (i.e. motion is constrained to rotate approximately about the center of projection) the set of images can be registered and "mosaiced" onto a common sphere of possible directions.

Methods for finding transformations that register images into the plenoptic function use either point correspondences (or equivalently optical flow) or are iconic in nature and thus require no explicit point matches. Aizawa [67] presents an iterative technique for finding pure rotation, focal length and zoom given point correspondences in a pair of views. McMillan and Bishop [73] describe transformations needed to project images onto a common cylinder, claiming to be able to solve 5 intrinsic parameters as well as 3 parameters of pure rotation from 12 or more feature correspondences. Mann and Picard [74] present a feature-less iconic approach for estimating the 8-parameter perspective transformation used to reproject and register multiple images onto a common plane via "video-orbits" for image mosaicing. Feature-less techniques are also used by Szeliski [28] to recover video mosaics under conditions of pure rotation or scenes which are well-approximated as a single plane.

If the focal length is considered infinite (i.e. having viewing conditions of pure orthography) a 6-parameter affine transformation describing image flow over an entire region can be found. Teodosio and Bender's Salient Stills algorithm [75] was one of the first well-known techniques for finding affine transformation parameters. Hsu [77] and Kermode [78] have presented extended approaches for image mosaicing and resolution enhancement.

If focal length is considered infinite and camera-roll, scale and shear are held to be fixed, then the camera pan and tilt or object motion simplify to a purely 2-D translational model such as the block-wise translation model used for MPEG-I and MPEG-II digital video formats.

2.5.3 Toward detecting static 3-D representations from video

Algorithms have been developed which aim to interpret the field of visual motion as distinct objects moving relative to the camera. Attempts have been made by Wang and Adelson [76] and Hsu [77] to iteratively classify optical flow into regions which fit a common set of affine motion parameters. In this way they effectively model an image sequence as a set of planar textured "layers" being viewed under orthography. Kermode [78] developed a hybrid video coder that uses similar principles but adds capabilities for image mosaicing and resolution enhancement and works within the current MPEG digital video coding framework.

Adiv [52] attempts to fit optical flow to locally planar patches, and then does local hypothesis testing to aggregate patches that have compatible motion parameters (i.e. the 8-parameter planar perspective transformation used by Tsai and Huang [49]). Adiv's result can be considered to similar to Wang and Adelsons [76] except that known finite focal length is assumed.

Here we begin to see a blurring between efforts of video coding and total 3-D scene reconstruction. As pointed out by Barron [51] structure-from-motion algorithms are ill-suited for general video coding due to the difficulty of detecting and tracking features caused by the aperture problem[1]. It is also difficult to automatically cluster optical flow vectors into regions of similar planar flow due to its non-linear form and sensitivity to noise. In contrast, affine models effectively model optical flow over a region as flow caused by viewing a moving planar texture under orthography. Although the affine model is only an approximation to true perspective flow it is in practice more tractable because it has a linear solution which uses only 1st order spatiotemporal brightness derivatives.

Mann [74] has been able to accurately find projective flow by iteratively fitting 1st and 2nd order derivatives to a polynomial mapping and then converting that to a correct projective mapping but only for the entire image, not for individual image regions.

2.5.4 Dimensionalization

Holtzman [82] sidesteps the optical flow perspective segmentation problem for video-coding by allowing partially known scene geometry. In an initial frame 3-D planar polygons describing key surfaces in the scene are manually placed via a CAD system relative to the initial camera's origin with known focal length. In this process, which he calls "dimensionalization", the user aligns known world geometry to observed image features and serves to bootstrap the process for coding video sequences created with a moving camera in a static scene. Relative pose for the next frame is found by densely searching the 6-parameter space to find the pose that minimizes pixelwise difference between the next image and textures from previous image transformed according to plane geometry and new camera pose. In this way Holtzman showed that the concept of "dimensionalization" offered practical advantages over other methods of video coding in that it made 3-D model-based video coding practical if allowed the initial startup cost of manually specifying the 3-D scene.

[1]The aperture problem is that image flow can only be resolved in the direction of available texture detail

2.6 Strategies for video transmission and decoding of images of synthetic scenes

Suppose we have a 3-D model which can be manipulated to best represent a given video sequence. Ignoring for a moment the challenge of creating the initial model and then parameterizing it to match incoming frames of video, what can be said for efficiencies this framework allows for compression and transmission.

This can be answered by reviewing work done in compressing rendered images of known synthetic scenes. Levoy [101] postulates having a high-performance graphics engine act as a rendering server to lower-performance client machines. To handle the problem of transmitting high resolution images of synthetic scenes the graphics server first renders the scene at high resolution. It then produces a simpler compressed model of the scene which can be rendered efficiently by any client. To recreate a high-resolution view the client need only to render the simpler compressed scene model and then add in the residual difference image. This residual signal can be compressed for low-bandwidth transmission using JPEG or MPEG.

Subjective experiments show that for low to medium level of detail scenes at compression rates of more than 1:25 high resolution renderings compressed using this technique look better than standard JPEG compression at the same rate. Levoy warns, however, that for synthetic scenes whose complexity approximates that of natural scenes, the method will convey little or no advantage for lossless transmission of single images for the reasons that complex sythetic models cannot be easily compressed and are too large to transmit.

2.7 A new strategy for model-based representations of real scenes

Our approach differs in that it attempts to represent the background of a real (i.e. not synthetic) video sequence into geometry with low to medium complexity. We will make heavy use of texture to compensate for lack of geometric complexity, but that texture will come from only a few frames of video. The resulting surface texture images can be JPEG compressed. Dynamic camera pose will then be describing using 6 numbers and the composite actor channel will be MPEG compressed. The actor channel should have a fixed bandwidth, lower than that used by full screen video. Bandwidth used by frame-wise pose information is nearly negligible and the scene model takes up a one-time initialization cost. Since this fixed initialization cost gets averaged over time, we can claim that the average bit per frame rate actually decreases toward the pure cost of the actor channel as the length of the sequence increases.

If lossless coding is required residual pixel differences arising from errors in modeling, pose detection and actor segentation can be sent as another MPEG compressed channel.

We should point out, however, that the 3-D and interactive capabilities made possible by this kind of model-based description of video are more important and interesting than the possible gains in compression rates. For this reason we will focus more on issues of with physical validity (i.e. correct camera pose and coherent actor sprites) than on lossless transmission and compression rates.

2.8 Isis : a custom delivery mechanism for model-based video

Isis is a Scheme-like interpretive programming language written by Stefan Agamanolis [81] which can be used as a stand-alone programming environment as well as interface language to packages of C functions that do specialized operations from networking and file access to synchronized processing of graphics and sound.

Agamanolis uses this framework to create what we will refer to as the Interactive Model-Based Video Decoder (or IMBVD). This decoder uses a format for model-based video which has objects and a script. Objects have 2-D, 2.5-D and 3-D geometry, audio, and video properties. The script specifies how object properties change as a result of user manipulation and with the persistent passage of time. If run on a platform with sufficient processing power, the IMBVD can handle real-time audio and video rendering/composition as the user interactively manipulates camera parameters and global valuators. Individual objects have geometry, video texture and audio and as the point of view changes stereo sound and video texture for each object are rendered at the appropriate location relative to the viewer. The IMBVD also handles special video effects like filtering, warping and chroma-keying as well as audio effects like echo and reverbration.

Though Isis can work on any platform a high-performance implementation of the IMBVD exists on the Cheops [107, 108] platform, the Media Lab's modular platform that performs computationally intensive stream operations in parallel in specialized hardware. Since the architecture is optimized for real-time 2-D and 2.5-D alpha-depth-buffered composition, but not for 3-D texture mapped rendering, textured 3-D objects are pre-rendered as 2.5-D objects (i.e. color + alpha + depth) from an arbitrary set of scripted views. During playback the IMBVD on Cheops thus supports interactive viewing within that subset of views.

2.9 VRML : another delivery mechanism for model-based video

Of late, the increasing availability of graphics rendering hardware, explosive interest of the Web, and proliferation of the Virtual Reality Modeling Language (VRML) as a standard 3-D scene description language has ignited a flurry of interest in creating engaging environments. In 1994 David Raggett proposed the concept of using a new markup language to extend the World Wide Web to support platform independent virtual reality [122]. He envisioned a high level language describing VR environments as compositions of logical elements. Geometry and complex behaviors of these elements could be described at different levels of detail and retrievable as shared resources from the network via the universal reference locator (URL) scheme. In time this concept become codified into a specification that has paved the way for VRML browsers on many platforms.

In an effort to lay the groundwork for VRML the initial specification described little more than static geometry and texture in scenes [125]. From its initial conception, however, the formative goal, has been to support dynamic properties of elements such as object behavior, audio and video, as outlined by Pesce [126]. Now as academic and commercial interests for virtual reality applications on the Internet gain momentum, VRML designers are rapidly adding capabilities to encompass those goals. Eventually, VRML agents may also be able to support data streams to deliver live actor geometry and MPEG-II video textures. VRML may soon become a practical delivery mechanism for model-based video over the Internet, but until then, Isis is the clear viewer of choice.

Chapter 3

Framework for vision-assisted modeling

The ultimate goal of this thesis is to find a physically valid three-dimensional model-based representation from one or more two-dimensional images. One interesting application of this work is to use that model as the basis for representing a digital video sequence. Handling digital video in such a fashion can provide both compression advantages and the ability to support interactive or personalized display of visual information [107]. As real-time 3-D texture-rendering capabilities become more affordable and widespread, it becomes more and more reasonable to consider the use of three-dimensional models of real scenes in lieu of two-dimensional signal-based image representations.

How does one first go about obtaining a photorealistic and accurate computer graphics model of a static scene? This thesis goal is achieved in the guise of an interactive photogrammetric modeling package (see Figure 3-1). It allows basic CAD-like constraints (e.g. coplanarity, parallelism, and perpendicularity) to be defined directly on 2-D lines and curves in a single photograph or still frame of video.

The vision-assisting modeling package is implemented as three software components: `sceneBuild`, `sceneAnalyze`, and `sceneView`.

`sceneBuild` is a graphical user interface which lets the user load a small number of still photographs or frames of video into zoomable viewports and overlaying color images with points, lines and curves. Three dimensional constraints among points, lines, and curves are specified by manually grouping them into 3-D placeholders: features, edges, directions, and surfaces. Information gathered for each project is arranged into a *sceneDatabase* file which is used by `sceneAnalyze`, a batch processing program, that takes the 2-D features and user-specified 3-D constraints and updates the sceneDatabase to describe a 3-D textured scene. `sceneView` is a rendering program that is used to interactively visualize the computed 3-D scene. It allows interactive camera placement, spline interpolated camera path planning, and can be used to create a synthetic fly-thru movie sequence by doing antialiased texture rendering on each camera. Chapters 7, 8 and 9 review the implementation and use of these modules in detail.

This chapter (Chapter 3) lays out necessary mathematical framework upon which the modeling technique rests. The camera model is introduced in section 3.1. The remaining sections discuss issues regarding projective geometry representations for image primitives. Issues regarding probabilistic representations for projective geometry may be found in the Appendix.

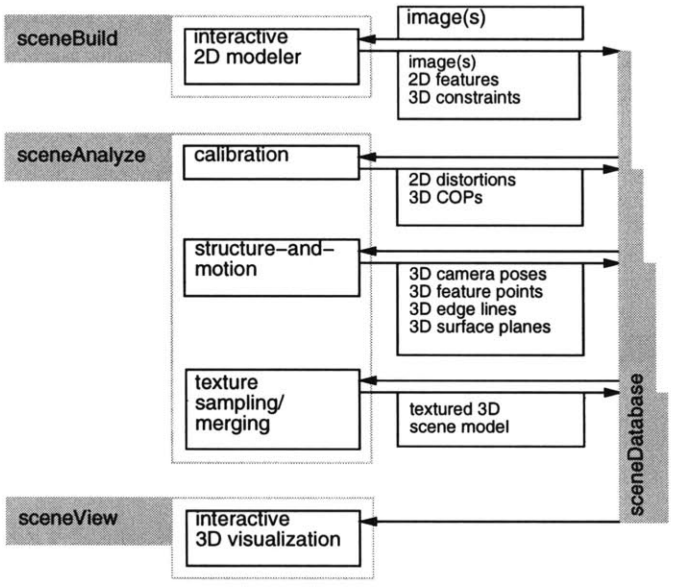


Figure 3-1: Framework for vision-assisted modeling. `sceneBuild` is a GUI application used to define 2-D features and 3-D constraints among them. Each automated processing stage in `sceneAnalyze` contributes more information to the composite `sceneDatabase` until a full textured 3-D scene model is recovered. `sceneView` has a 3-D GUI for view manipulation and supports wireframe and textured rendering.

As seen in Figure 3-1 `sceneAnalyze` is made up of three processing stages each of which build upon data gathered from previous stages. Chapters 4, 5 and 6 present the key algorithms which are used for vision-assisted modeling. Chapter 4 discusses algorithms used in the *calibration* stage. Chapter 5 discusses the algorithms used in *structure-and-motion* stage to coarsely estimate and then iteratively refine the 3-D geometry of cameras, features, edges, directions and surfaces of a polyhedral scene until it optimally matches 2-D projections in the observed image. Chapter 6 introduces a *texture sampling/merging* algorithm which samples image textures for visible planar surfaces and then merges texture contributions from multiple views. Chapter 6 also presents techniques for maximizing resolution and reducing blending artifacts.

After reviewing the actual programs which implement the algorithms for vision-assisted modeling in chapters 7, 8 and 9 chapter 10 presents experimental results analyzing structure recoverability on synthetic and real-world scenes. Chapter 10 also provides a user performance analysis showing how well the `scene` package works as a 3-D from 2-D modeling platform and compares it to other related approaches to vision-assisted scene modeling.

Besides interactive visualization another interesting application for this photorealistic 3-D model is for the purpose of *model-based video coding.* Chapter 11 details an algorithm used to automatically decompose motion video sequence of a known scene into pose parameters, 2-D actor billboard sprites (with position and planar orientation) and a 2-D residual error signal. In chapter 12 this algorithm is implemented as a program called `sceneTrack` which takes digitized frames of video, a model of the scene, and an initial camera position and outputs a 3-D video representation compatible with existing Isis viewers.

3.1 Camera model

The transformation from world coordinates ${}^w\mathbf{p} = [{}^wp_x, {}^wp_y, {}^wp_z]^T$ to distorted film coordinates ${}^d\mathbf{p} = [{}^dp_x, {}^dp_y]^T$ can be treated more simply as a composition of several transformations.

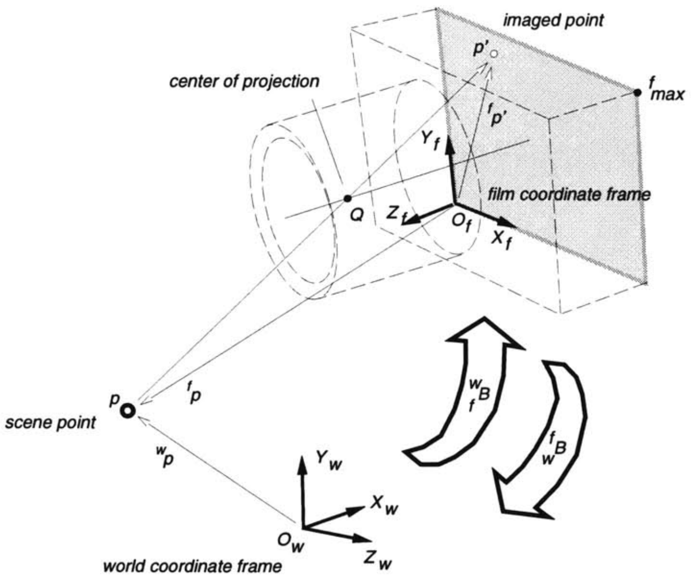


Figure 3-2: The camera model.

As seen in Figure 3-2 we choose a camera model which, besides using the simplifying pinhole projection model, attempts to describe the important geometric features of the actual physical camera. Using Paul's notation [116] the world-relative position and orientation of the film plane is defined by a 4×4 coordinate frame ${}^w_f\mathbf{B}$ whose principal unit axes and origin are defined by the 4×1 homogeneous vectors $\mathbf{x}_f$, $\mathbf{y}_f$,$\mathbf{z}_f$ and $\mathbf{o}_f$.

$$ {}^w_f\mathbf{B} = [{}^w\mathbf{x}_f | {}^w\mathbf{y}_f | {}^w\mathbf{z}_f | {}^w\mathbf{o}_f] \tag{3.1}$$

The *rigid body transformation matrix* is the 4×4 matrix used to get film-frame-relative coordinates from world-frame-relative coordinates

$$ {}^f\mathbf{p} = {}^f_w\mathbf{B}\, {}^w\mathbf{p} \tag{3.2}$$

where ${}^f_w\mathbf{B}$ is simply the inverse of ${}^w_f\mathbf{B}$.

The *perspective transformation* is the non-linear mapping ${}^u_f\mathcal{P}$which uses the film-relative center of projection, ${}^f\mathbf{c} = [{}^fc_x {}^fc_y {}^fc_z]^T$ (where ${}^fc_z$ is focal length in pixels), to project film-frame-relative scene position ${}^f\mathbf{p}$ into predistorted film-relative image coordinates ${}^u\mathbf{p}$.

$$ {}^u\mathbf{p} = {}^u_f\mathcal{P}\left({}^f\mathbf{p}\right) \tag{3.3}$$

As seen in Figure 3-2 the *center of projection* (or COP[1]) is placed between the scene and the film plane. This representation allows proper decoupling between changing focal length

[1]Equivalent terms for center of projection include optical center and nodal point. The point on the image plane nearest to the center of projection is commonly called the principal point, piercing point or

and changing camera pose. As noted by Azarbayejani[110] it also correctly maintains the effects that uncertainties in focal length and camera pose have on determining 3-D feature position via triangulation. More justification for this choice of camera representation may be found in Section A.2 in the Appendix.

This transformation is found from the relations

$$\frac{{}^u p_x}{{}^u p_w} = \frac{{}^f c_x {}^f p_z - {}^f c_z {}^f p_x}{{}^f p_z - {}^f c_z} \tag{3.4}$$

$$\frac{{}^u p_y}{{}^u p_w} = \frac{{}^f c_y {}^f p_z - {}^f c_z {}^f p_y}{{}^f p_z - {}^f c_z} \tag{3.5}$$

and can be described using the 4×3 matrix,

$$\begin{bmatrix} {}^u p_x \\ {}^u p_y \\ {}^u p_w \end{bmatrix} = \begin{bmatrix} 1 & 0 & \beta {}^f c_x & 0 \\ 0 & 1 & \beta {}^f c_y & 0 \\ 0 & 0 & \beta & 1 \end{bmatrix} \begin{bmatrix} {}^f p_x \\ {}^f p_y \\ {}^f p_z \\ 1 \end{bmatrix} \tag{3.6}$$

where negative inverse focal length is $\beta = -1/{}^f c_z$ and projected image coordinates ${}^u p_x$ ${}^u p_y$ are normalized by ${}^u p_w$. Note that this perspective model assumes unit aspect ratio and handles the orthographic case.

The *lens distortion equation* describes the non-linear relationship between ideal undistorted pixel coordinates ${}^u\mathbf{p}$ and observed pixel coordinates ${}^d\mathbf{p}$ which have been subjected to lens distortion

$$\begin{aligned} {}^u p_x &= {}^d p_x + \bar{x}\left(K_1 r^2 + K_2 r^4 + K_3 r^6\right) + \left[P_1\left(r^2 + 2\bar{x}^2\right) + 2P_2 \bar{x}\bar{y}\right]\left[1 + P_3 r^2\right] \\ {}^u p_y &= {}^d p_y + \bar{y}\left(K_1 r^2 + K_2 r^4 + K_3 r^6\right) + \left[P_2\left(r^2 + 2\bar{y}^2\right) + 2P_1 \bar{x}\bar{y}\right]\left[1 + P_3 r^2\right] \end{aligned} \tag{3.7}$$

where $[K_1 K_2 K_3]$ are radial distortion coefficients and $[P_1 P_2 P_3]$ are decentering distortion coefficients that account for the case when the center of distortion and the principal point are not coincident. Displacement terms $\bar{x}$, $\bar{y}$ and r are normalized by the maximum film radius R.

$$\bar{x} = \frac{{}^d p_x - {}^f c_x}{R} \quad \bar{y} = \frac{{}^d p_y - {}^f c_y}{R} \quad r = \sqrt{\bar{x}^2 - \bar{y}^2} \tag{3.8}$$

Since equation (3.7) outputs an undistorted point for an observed point already subjected to lens distortion, we might describe this as an *undistortion* mapping which is functionally of the form

$${}^u\mathbf{p} = {}^u_d\mathcal{U}\left({}^d\mathbf{p}\right) \tag{3.9}$$

The inverse mapping which would be used to synthesize point-wise lens distortion is functionally of the form

$${}^d\mathbf{p} = {}^u_d\mathcal{U}^{-1}({}^u\mathbf{p}) = {}^u_d\mathcal{D}({}^u\mathbf{p}) \tag{3.10}$$

where the distorted coordinate ${}^d\mathbf{p}$ for a particular undistorted coordinate ${}^u\mathbf{p}$ is given. The

photogrammetric image center. The distance from the principal point to the center of projection is called the principal distance or effective focal length entire *projection equation* can now be described as

$$ {}^{d}\mathbf{p} = {}^{d}_{u}\mathcal{D}\left\{ {}^{u}_{f}\mathcal{P}\left({}^{f}_{w}\mathbf{B}\ {}^{w}\mathbf{p} \right)\right\} \tag{3.11} $$

3.2 Projective Geometry

3.2.1 Projective duality of points and lines

The projection of a 3-D point feature in the scene passes thru the center of projection $\tilde{\mathbf{c}}$. Given only the projected 2-D image of a point, its 3-D camera-relative position is constrained to lie along the *projection vector.* Using this camera model we define this projection vector as having direction $\tilde{\mathbf{p}} = (\tilde{\mathbf{c}} - \mathbf{p}) = [c_x - p_x, c_y - p_y, c_z]^T$[2].

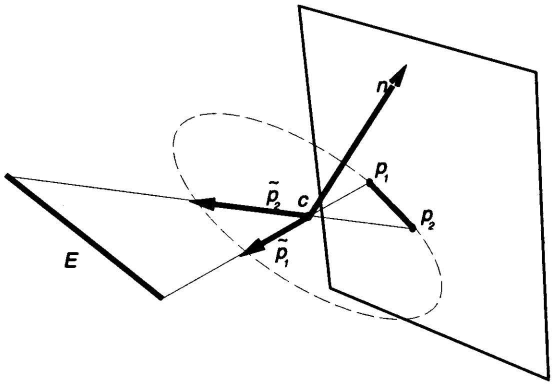


Figure 3-3: The cross product of the projection vectors of any two points defines the *projection plane normal* $\mathbf{n}$ of the line that passes through those two points

The projective representation of a straight image line is found from the projection vectors of any two points on that line. Referring to Figure 3-3, let E denote a 3-D edge in a scene with world-relative direction $\hat{\mathbf{d}}$. If $\mathbf{p}_1$, $\mathbf{p}_2$ are two imaged points believed to belong to that edge and if $\tilde{\mathbf{c}}$ is a hypothetical center of projection, then under conditions of noiseless line detection and zero lens distortion, E can be considered to lie in the *projection plane* which contains $\mathbf{p}_1$, $\mathbf{p}_2$ and $\tilde{\mathbf{c}}$.

The unit normal vector $\mathbf{n}$ of this projection plane has the following properties

$$ \mathbf{n}^T(\mathbf{p} - \tilde{\mathbf{c}}\ ) = \mathbf{n}^T(\tilde{\mathbf{c}}\ - \mathbf{p}) = \mathbf{n}^T\tilde{\mathbf{p}} = 0 \tag{3.12} $$

or equivalently

$$ n_x p_x + n_y p_y = \mathbf{n}^T\tilde{\mathbf{c}} \tag{3.13} $$

for all image points $\mathbf{p} = [p_x, p_y]^T$ which lie in that image line. Therefore, $\mathbf{n}$ may be found from the cross product of the two projection vectors[3]

$$ \mathbf{n} = \tilde{\mathbf{p}}_1 \times \tilde{\mathbf{p}}_2 \tag{3.14} $$

[2]Note that most computer vision and photogrammetry approaches typically consider the the center of projection to be the camera frame's origin, thus greatly simplifying definition of a point's projection vector. We choose this alternate representation since the COP is unknown.

[3]The careful reader will note that either sense of $\mathbf{n}$ describes the image line that contains both $\mathbf{p}_1$ and $\mathbf{p}_2$.

A projective duality exists between image points and lines. Whereas the cross product of two projective point vectors defines a projection plane normal, conversely, as seen in Figure 3-4 the cross product of any two projection plane normals $\mathbf{n}_1$ and $\mathbf{n}_2$ results in the vector $\mathbf{d}$. Vector $\mathbf{d}$ is the projective vector of the point $\mathbf{V}$ which contains both lines. Thus $\mathbf{V}$ forms the intersection of the two lines $\mathbf{e}_1$ and $\mathbf{e}_2$. In the special case that $\mathbf{e}_1$ and $\mathbf{e}_2$

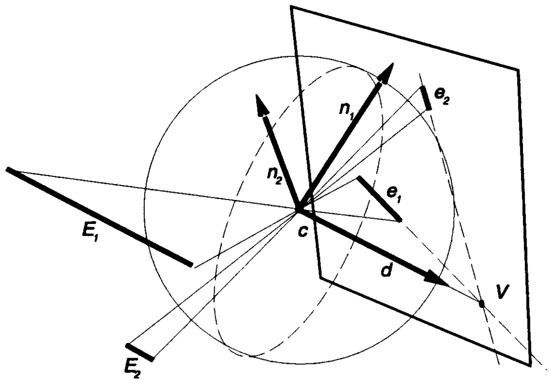


Figure 3-4: The cross product of two projection plane normals ($\mathbf{n}_1$ and $\mathbf{n}_2$) is a projection vector ($\mathbf{d}$) which points in the direction of the image point shared by both lines

are actually the imaged lines of two parallel 3-D scene edges E_1 and E_2 then by mutual orthogonality to the projection plane normals, the projective vector $\mathbf{d}$ is also parallel to the 3-D edges. In this case, we will refer to $\mathbf{d}$ as the *vanishing point direction* of a set of parallel lines and the projected point V will be called the *vanishing point.*

3.2.2 Projective representation for vanishing points

Since vanishing points are important cues for calibration it is important to choose a representation that allows accurate and robust estimation of vanishing points from points and lines. To emphasize the utility of the projective reprentation, consider an image which contains a pair of image segments which are parallel and not collinear. It would be difficult to describe the vanishing point of this line pair if we insisted on using only image points to represent line intersections because, in this case, no image point exists that describes the intersection. The vanishing point is easily described in projective terms, however, as a 3-D projection vector. The direction insensitive nature of this represenation also makes it well-suited for conditions of orthography where parallel 3-D lines always project to parallel 2-D lines in an image.

3.2.3 Estimating vanishing point direction from many lines

Now suppose there are a total of N non-collinear line segments $\mathbf{e}_j$, $j = [1, N]$ specified as belonging to a particular direction. As N increases, the subsequent set of positive and negative pairs of unit projection normals $\pm\mathbf{n}_j$ forms a symmetric great circle on the sphere of possible directions (see Figure 3-5) whose center is $\tilde{\mathbf{c}}$. The plane that minimizes the mean square error to this ring of projection normals will itself have a normal vector which defines the optimal vanishing point direction, $\mathbf{d}$.

Due to line detector noise, each line segment $\mathbf{e}_j$ can be treated probabilistically as a multivariate Gaussian distribution $p_{n_j}(\mathbf{n}) \sim N(\mathbf{n}_j, \Lambda_{n_j})$. A detailed discussion of proba-

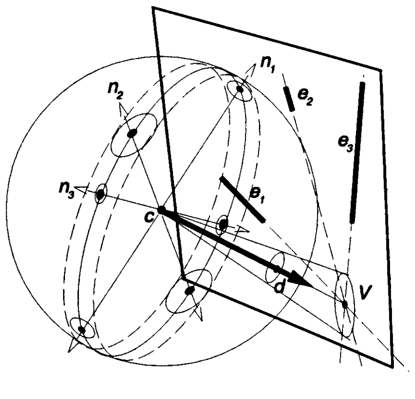


Figure 3-5: Optimal vanishing point estimation

bilistic representations of points and lines due to detector/measurement error is deferred to section A.4 in the Appendix.

The optimal mean and covariance of $\mathbf{d}$ can be found by analyzing the aggregate distribution of all unit projection normals. Since this set consists of samples which are themselves Gaussian distributions we can define the aggregate distribution as the sum of distributions, weighted according to the believability of each sample. If we treat each line as equally probable the aggregate distribution can be defined as $p_n(\mathbf{n}) = \frac{1}{N}\sum_{j=1}^{N} p_{n_j}(\mathbf{n})$.

Using basic moment generating properties the mean of this distribution is

$$\begin{aligned} \bar{\mathbf{n}} &= E[\mathbf{n}] = \int \mathbf{n} p_n(\mathbf{n}) d\mathbf{n} = \int \mathbf{n} \frac{1}{N}\sum_{j=1}^{N} p_{n_j}(\mathbf{n}) d\mathbf{n} \\ &= \frac{1}{N}\sum_{j=1}^{N}\int \mathbf{n} p_{n_j}(\mathbf{n}) d\mathbf{n} = \frac{1}{N}\sum_{j=1}^{N} E[\mathbf{n}_j] = \frac{1}{N}\sum_{j=1}^{N} \mathbf{n}_j \end{aligned} \tag{3.15}$$

similarly the covariance of the aggregate distribution of projection normals is

$$\begin{aligned} \Lambda_{\mathbf{n}} &= E[\mathbf{n}\mathbf{n}^T] - \bar{\mathbf{n}}\bar{\mathbf{n}}^T \\ &= \frac{1}{N}\left[\sum_{j=1}^{N} E[\mathbf{n}_j\mathbf{n}_j^T]\right] - \bar{\mathbf{n}}\bar{\mathbf{n}}^T = \frac{1}{N}\left[\sum_{j=1}^{N} \Lambda_{n_j} + \mathbf{n}_j\mathbf{n}_j^T\right] - \bar{\mathbf{n}}\bar{\mathbf{n}}^T \end{aligned} \tag{3.16}$$

To determine the optimal vanishing point direction, we find the 3×3 eigenvector and eigenvalue matrices Φ and Λ such that

$$\Lambda_{\mathbf{n}} = \Phi\Lambda\Phi^T \tag{3.17}$$

where Φ is an orthonormal matrix whose columns are eigenvectors

$$\Phi = [\phi_1|\phi_2|\phi_3] \tag{3.18}$$

and Λ is the diagonal matrix of eigen values associated with each eigenvalue

$$\Lambda = diag(\lambda_1, \lambda_2, \lambda_3) \tag{3.19}$$

whose elements are sorted such that

$$\lambda_1 \geq \lambda_2 \geq \lambda_3 \tag{3.20}$$

Since the aggregate distribution of projection normals $\mathbf{n}_j$ for $j = [1, N]$ is nearly planar the first two columns of Φ span the best fit plane and the eigenvector associated with the minimum eigenvalue λ_3 is the optimal plane normal. Thus, the optimal vanishing point direction, $\hat{\mathbf{d}}$, is simply

$$\hat{\mathbf{d}} = \phi_3 \tag{3.21}$$

Shortly, we shall see that the minimum eigenvalue λ_3 itself is also quite useful.

Chapter 4

Calibration

This chapter describes calibration algorithms used in this thesis for vision-assisted modeling. Section 4.1 presents a new framework for estimating radial and decentering distortion terms of a lens. This least-squares approach requires that distorted lines have been classified into a small number of common directions. The estimated distortion terms can be used to either undo the effects of lens distortion on points, lines and pixel intensities or to synthesize lens distortion effects after rendering straight-edged geometries.

In section 4.2 we then present a least-squares approach for roughly estimating center of projection. This technique depends upon having an image of a scene containing straight edges of a right parallelepiped viewed under 3-point perspective. For heavily cluttered scenes it will also require manual grouping of lines. The distinct advantage of these calibration techniques over others is that no direct measurements are required.

4.1 Correcting for lens distortion

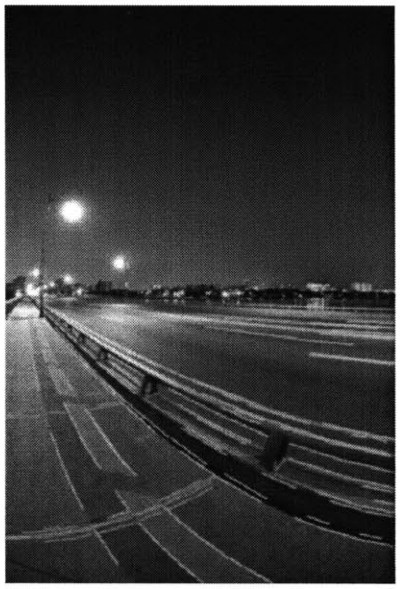


Figure 4-1: The effect of lens distortion on straight lines before correction.

As noted by D. C. Brown [23], in the absence of distortion, the central projection of a straight line is itself a straight line. Most approaches to lens distortion estimation attempt to do so by minimizing curvature along curves or among points which should be straight[1]. We instead make the observation that as shown in Figure 4-2(a) in the absence of distortion, the central projection of a set of straight parallel lines is itself a set of straight lines that intersect at a distinct vanishing point. A set of N non-overlapping segments results in $\frac{N}{2(N-2)}$ possible intersection points. Without distortion, all candidate vanishing points will be identical. Under lens distortion, however, the projection of this set of lines yields a set

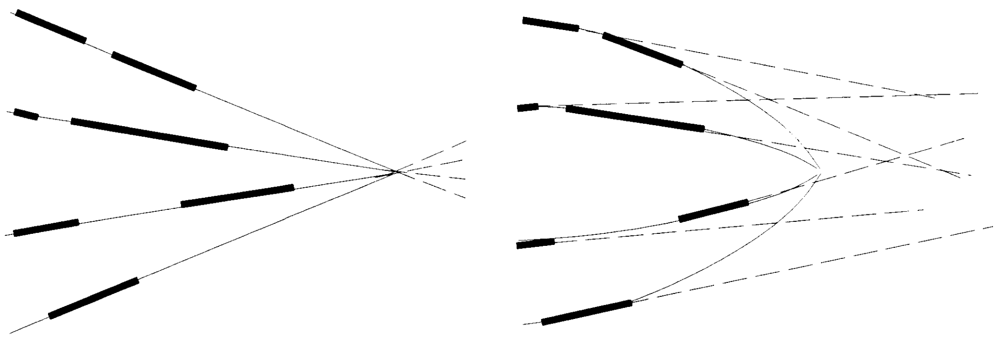


Figure 4-2: Vanishing point of a set of parallel scene lines (a) without lens distortion is compact and distinct (b) with lens distortion is dispersed and "confused".

of curved lines that if broken into piecewise linear segments (as done by some line detection techniques), the extension of these segments will not result in a common intersection point. Instead, the set of candidate vanishing points will be dispersed as shown in Figure 4-2(b).

Resuming the Section 3.2.3 discussion of estimating the vanishing direction of a set of lines, as line detection noise and lens distortion are introduced, the set of candidate vanishing point directions become dispersed and the aggregate set of projection normals becomes less constrained to a plane. Since λ_3 describes the minimum thickness of the aggregate distribution it can be interpreted as a measure of vanishing point dispersion.

This suggests that λ_3 can be used as a measure of the amount of distortion introduced by the projection equation (3.11). To correct for lens distortion, our goal then, is to find the distortion parameters that minimize the sum of all λ_3 for all d_i directions in a scene. Using equations (3.9), (A.5) and (3.21) we define the observation function

$$f(d_i, \mathbf{e}_{i,j}; K_1, K_2, K_3, P_1, P_2, P_3) = \sum_{i=1}^{M} \lambda_{3_i}(\mathbf{e}_{i,j}\ , j = [1, N_i] \tag{4.1}$$

where d_i denotes the i-th of M directions and $\mathbf{e}_{i,j}$ describes the j-th of N_i lines that belong to that direction. Starting with an initial assumption of zero distortion, i.e. $\mathbf{x} = \mathbf{0}$ we use the iterative downhill simplex method outlined in [83] to find the value for $\mathbf{x} = [K_1, K_2, K_3, P_1, P_2, P_3]$ that minimizes the observation function (4.1).

[1]The traditional approach for solving lens distortion is explained in section A.1 in the Appendix.

Figure 4-3: After correcting for lens distortion.

4.1.1 Optimal lens distortion estimation over multiple views

If lens distortion is constant over multiple views, equation (4.1) requires only a slight change

$$f(d_i, \mathbf{e}_{i,j,k}; K_1, K_2, K_3, P_1, P_2, P_3) = \sum_{k=1}^{K} \sum_{i=1}^{M} \lambda_{3_i}(\mathbf{e}_{i,j,k} \ , j = [1, N_i k] \qquad (4.2)$$

where d_i denotes the i-th of M directions and $\mathbf{e}_{i,j,k}$ describes the j-th of $N_i k$ lines that belong to that direction and are visible in the k-th of K images.

4.1.2 Advantages over plumb-line methods

Plumb-line methods (originating with Brown[23]) traditionally image templates of parallel lines and require detection of points along individual curves. The error metric for modeling effectively tries to minimize curvature along each independent curve by summing deviation from the mean square error straight line that fits the curve (see Appendix section A.1).

Our new approach, like Brown's method, also effectively mimimizes curvature given a scene with parallel edges, since curvature increases variance of the aggregate set of projection normals. In addition, however, our approach exploits the additional constraint that lines from a parallel set of edges must intersect at a common point on the Gaussian sphere. This should help get a more unique and correct result.

From a classification standpoint our method offers the advantage that it does not require detection of continuous curves. Instead, this method requires only that detected line segments be grouped into parallel sets. This is much more convenient since, in general, scenes contain many more edges than edge directions.

4.1.3 Automatically classifying lines into parallel sets

The vanishing point estimation framework can be used to automatically identify all lines in an image that may be parallel to two or more selected lines in the scene. By describing lines as projection normals and the vanishing point as the vanishing point direction we can identify all lines that have unit normals within some threshold of the plane formed by the vanishing point direction as potentially being parallel to the given direction.

If the number of directions in the scene are known the Random Sample and Consensus (RANSAC[63]) algorithm can be adapted to automatically find the pair of projection normals which form a plane that fits the plurality of unclassified normals within some activation threshold. To automatically find all directions this can then be done iteratively until all but some expected percentage of lines are accounted for (i.e. clutter).

4.1.4 Undistorting the intensity image

Once the distortion parameters have been estimated equation (3.9) can be used to correct the unwanted distortions at the endpoints for all image segments. However, if we wish to correct the entire distorted intensity image given estimated distortion parameters, we need to invert this equation to find distorted image coordinates for each pixel in the undistorted output image. This equation must also be inverted if we wish to synthesize a distorted wireframe scene representation for visualization and testing.

To do this we use the iterative downhill simplex method [83] to find the roots of non-linear multivariate equations. To get the initial estimate for the distortion coordinates ${}^{d}\mathbf{p}$ given an undistorted point ${}^{u}\mathbf{p}$ we rewrite equation (3.9) as a displacement function $u(\cdot)$

$$\begin{aligned} {}^{u}\mathbf{p} &= {}^{d}\mathbf{p} + {}^{u}_{d}\mathcal{U}\left({}^{d}\mathbf{p}\right) - {}^{d}\mathbf{p}\Big) \\ &= {}^{d}\mathbf{p} + u({}^{d}\mathbf{p}) \end{aligned} \tag{4.3}$$

Since the displacement function u is locally smooth we can say that

$$u({}^{u}\mathbf{p}) \approx u({}^{d}\mathbf{p}) \tag{4.4}$$

and, therefore, make the approximation

$${}^{d}\mathbf{p} \approx {}^{u}\mathbf{p} - u({}^{u}\mathbf{p}) \tag{4.5}$$

To invert (3.9) for a particular undistorted point ${}^{u}\mathbf{p}$ we first define the error function $f(\cdot)$

$$f({}^{d}\mathbf{p}) = {}^{u}_{d}\mathcal{U}({}^{d}\mathbf{p}) - {}^{u}\mathbf{p} = 0 \tag{4.6}$$

We then initialize the estimator ${}^{d}\hat{\mathbf{p}}$ to (4.5) and use linear regression [83] to iteratively converge to a solution for a distorted point.

4.1.5 Stochastic image unwarping/resampling

Let $\mathbf{R_u}$ denote the region covered by one pixel in the undistorted image as seen in Figure 4-4 and let $\mathbf{R_d}$ be the corresponding region in the distorted image. $\mathbf{R_d}$ may cover an area much smaller than a single pixel or it may cover an area extending over many pixels, depending upon the local characteristics of the distortion mapping. To undo the effects of distortion without incurring sampling artifacts like aliasing or jaggies, the intensities covered by $\mathbf{R_d}$ need to be filtered and averaged or interpolated before being used as the intensity at the undistorted location $\mathbf{R_u}$.

We use a stochastic sampling approach in which a random set of samples are taken in

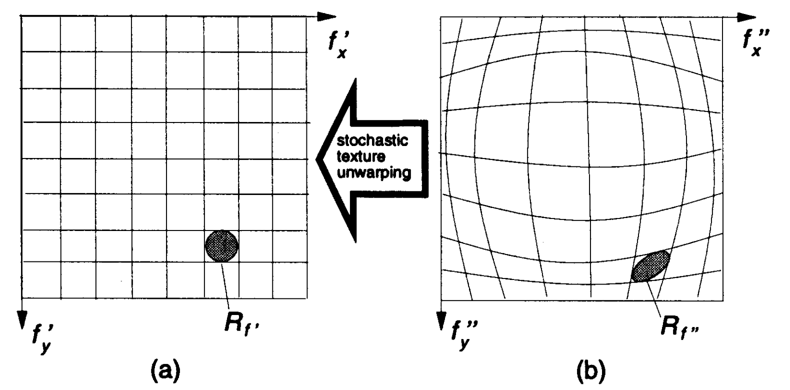


Figure 4-4: A pixel region in an undistorted image (a) is given the average of stochastically sampled intensities in a region of the distorted image (b).

the distorted image so as to match the distribution of $\mathbf{R_d}$; its covariance is a by-product of each iteration in the linear regression used to solve (4.6). Enough random subpixel samples are generated to cover $\mathbf{R_d}$, the area[115] of which is

$$\text{Volume}_N = (N-1)^{N/2}|\Lambda_{R_d}|^{1/2} \tag{4.7}$$

where N is the dimensionality and in this case is 2. Intensities at each subpixel sample location are bilinearly interpolated, and then the entire set is averaged together and used as the new intensity for the undistorted pixel $\mathbf{R_u}$.

4.2 Estimating center of projection

When a rectangular parallelepiped is imaged under conditions giving rise to 3-point perspective the 3 vanishing points on the image ($\mathbf{v}_a$, $\mathbf{v}_b$, and $\mathbf{v}_c$) and the true center of projection $\mathbf{c}$ form a right tetrahedron. Recall that the sphere used for optimal estimation of vanishing point directions and lens distortion in section 3.2 uses only some *approximate* center of projection, $\tilde{\mathbf{c}}$. Since vanishing points are fundamentally a property of lines on the image they do not change appreciably as $\tilde{\mathbf{c}}$ changes. Therefore, we can use $\tilde{\mathbf{c}}$ to project the initial vanishing point directions into the image plane and then exploit this right tetrahedral property to refine $\tilde{\mathbf{c}}$ until we get an optimal solution for the true center of projection, $\mathbf{c}$.

The film-relative vanishing point $\mathbf{v}_i$ for the i-th direction may be found by intersecting the line that contains $\tilde{\mathbf{c}}$ and has direction $\mathbf{d}_i$ with the film's XY-plane.

$$\mathbf{v}_i = \tilde{\mathbf{c}} - (\tilde{c}_z/d_{i_z})\hat{\mathbf{d}}_i \tag{4.8}$$

where $\tilde{\mathbf{c}}$ as the approximated center of projection, $\hat{\mathbf{d}}_i$ is the i-th direction vector pointing toward the film plane, and d_{i_z} denotes the z-coefficient of $\hat{\mathbf{d}}_i$.

Since the true center of projection $\mathbf{c}$ and the vanishing points form a right tetrahedron we have the following constraints

$$\left\{ \begin{array}{l} (\mathbf{v}_a - \mathbf{v}_b)^T(\mathbf{c} - \mathbf{v}_c) = 0 \\ (\mathbf{v}_b - \mathbf{v}_c)^T(\mathbf{c} - \mathbf{v}_a) = 0 \\ (\mathbf{v}_c - \mathbf{v}_a)^T(\mathbf{c} - \mathbf{v}_b) = 0 \end{array} \right. \tag{4.9}$$

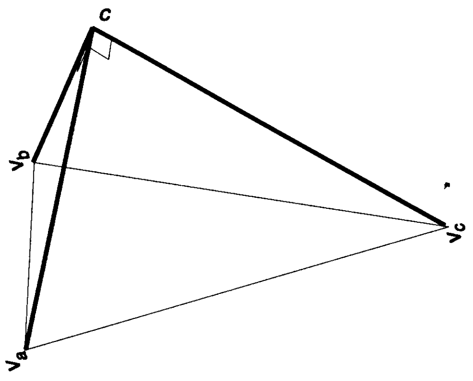


Figure 4-5: Estimating principal point and principal distance from three vanishing points of mutually orthogonal parallel lines

from which $\mathbf{c}$ is the solution to a set of 3 linear equations

$$\begin{bmatrix} (\mathbf{v}_a - \mathbf{v}_b)^T \\ (\mathbf{v}_b - \mathbf{v}_c)^T \\ (\mathbf{v}_c - \mathbf{v}_a)^T \end{bmatrix} \mathbf{c} = \begin{bmatrix} (\mathbf{v}_a - \mathbf{v}_b)^T \mathbf{v}_c) \\ (\mathbf{v}_b - \mathbf{v}_c)^T \mathbf{v}_a) \\ (\mathbf{v}_c - \mathbf{v}_a)^T \mathbf{v}_b) \end{bmatrix} \tag{4.10}$$

4.2.1 Solving c from a set of 3 non-orthogonal directions

If the user-defined directions corresponding to these directions are known, but not mutually orthogonal, then

$$\begin{cases} (\mathbf{v}_b - \mathbf{c}\ )^T(\mathbf{v}_c - \mathbf{c}\ ) = \alpha \\ (\mathbf{v}_c - \mathbf{c}\ )^T(\mathbf{v}_a - \mathbf{c}\ ) = \beta \\ (\mathbf{v}_a - \mathbf{c}\ )^T(\mathbf{v}_b - \mathbf{c}\ ) = \gamma \end{cases} \tag{4.11}$$

where α, β and γ are the cosines of the known angles between each pair of directions. Since these equations introduce $\mathbf{c}^T\mathbf{c}$ terms the problem becomes quadratic in 3-D. By folding the 3 terms of $\mathbf{c}$ into one the equation can be converted to a 4'th order polynomial for which, in general, 4 real solutions can be found.

4.2.2 Improving center of projection over multiple views by fusing multiple ML-estimates

If film-relative center of projection is known to be constant over multiple views then we wish to use that additional information to improve estimation accuracy. Letting $\mathbf{c}_k$ and Λ_{c_k} describe candidate distributions found by making measurements from each of $k = [1, K]$ images, the mean and covariance of fused ML-estimate are defined as

$$\begin{aligned} \hat{\mathbf{c}} &= \mathbf{\Lambda}_{\hat{c}} \sum_{k=1}^{K} \mathbf{c}_k \mathbf{\Lambda}_{c_k}{}^{-1} \\ \mathbf{\Lambda}_{\hat{c}} &= \left[\sum_{k=1}^{K} \mathbf{\Lambda}_{c_k}{}^{-1} \right]^{-1} \end{aligned} \tag{4.12}$$

Chapter 5

Structure and Motion

This chapter discusses the algorithms used in the *structure-and-motion estimation* stage of the modeling package. Sections 5.1 and 5.2 discuss algorithms used to estimate camera rotation and translation. Section 5.3 describes how we fix the 3-D geometry of features, edges, directions and surfaces of a polyhedral scene. In special cases, a single view is sufficient to find an approximate solution. Given multiple views, a least squares solution for scene structure is found where possible. Section 5.5 describes the *parameter refinement* stage which iteratively improves our rough parameter estimates until they define a scene model which optimally agrees with the observed set of 2-D image elements. This is done while simultaneously maintaining prespecified geometric constraints among 3-D scene elements.

5.1 Absolute rotation

Having solved for distortion and center of projection knowledge of any 3 scene directions that span $\mathbf{R}^3$ and their corresponding vanishing points in a single image allows us to find a least squares estimate of the camera's absolute rotation. To do this we form the 3×3 matrix ${}^w\mathbf{D}$ whose columns are the three user-specified world-relative line directions

$$ {}^w\mathbf{D} = \left[{}^w\hat{\mathbf{d}}_1 | {}^w\hat{\mathbf{d}}_2 | {}^w\hat{\mathbf{d}}_3\right] \tag{5.1}$$

and likewise use the corresponding undistorted film-relative vanishing point direction vectors found in (3.21) as columns of the matrix ${}^f\mathbf{D}$

$$ {}^f\mathbf{D} = \left[{}^f\hat{\mathbf{d}}_1 | {}^f\hat{\mathbf{d}}_2 | {}^f\hat{\mathbf{d}}_3\right] \tag{5.2}$$

The rotation matrix ${}^f_w\mathbf{R}$ which brings these two rotation frames into registration is found as

$$ {}^f_w\mathbf{R} = {}^f\mathbf{D}\,{}^w\mathbf{D}^{-1} \tag{5.3}$$

To guarantee orthonormality, ${}^f_w\mathbf{R}$ is converted to a quaternion 4-vector, unit normalized, and then converted back into a 3×3 rotation matrix as described by McKay[109].

5.1.1 Rotation sense ambiguity

Vanishing points alone are not enough to uniquely define the world-relative camera rotation since each set of parallel lines can, depending upon camera pose, project a vanishing point along either sense of its world-relative direction vector. Rotation ambiguity can be resolved if the octant of the rotation is known, i.e. thru a simple 3-D interface to define the rough world-relative rotation. Another image-based approach is to provide the film-relative sense of at mean two edge directions in each image, such as by allowing the user to place arrowheads on two lines to specify the film-relative projection of the positive sense of their associated direction. If the arrow head for a given line points away from that line's vanishing point then that film-relative vanishing point direction is negated before composition into (5.2). This is done for two of the directions. The sign of the third direction is adjusted so that the handedness (as indicated by the sign of the triple products) are equivalent, or so that

$$({}^f\hat{\mathbf{d}}_1 \times {}^f\hat{\mathbf{d}}_2) \cdot {}^f\hat{\mathbf{d}}_3 = ({}^w\hat{\mathbf{d}}_1 \times {}^w\hat{\mathbf{d}}_2) \cdot {}^w\hat{\mathbf{d}}_3 \tag{5.4}$$

5.2 Absolute translation

The rotational matrix ${}^f_w\mathbf{R}$ of (5.3) forms the upper left rotational component of the the rigid body transformation matrix ${}^f_w\mathbf{B}$ introduced in section 3.1. The 4×4 matrix ${}^f_w\mathbf{B}$ can be described as the composition of several component elements

$${}^f_w\mathbf{B} = \begin{bmatrix} {}^f_w\mathbf{R} & {}^f_w\mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{5.5}$$

${}^f_w\mathbf{t}$ is a 3×1 vector which is the translational component of ${}^f_w\mathbf{B}$. It describes the position of the world coordinate frame's origin with respect to the camera's film reference plane. We can recover ${}^f_w\mathbf{t}$ using the known (i.e. user-defined) 3-D endpoints ${}^w\mathbf{p}_i$ of one edge in the scene and their corresponding undistorted 2-D film projections ${}^u\mathbf{p}_i$ in a single view.

For the case of two points, we first define the film-relative projection vectors ${}^f\tilde{\mathbf{p}}_0$ and ${}^f\tilde{\mathbf{p}}_1$ for the two image points ${}^f\mathbf{p}_0$ and ${}^f\mathbf{p}_1$ as introduced in section 3.2. Recall that our camera model (see Figure 3-2) places the center of projection between the film plane and the scene so each ray points from the imaged point in the film plane, thru the center of projection ${}^f\mathbf{c}$ toward the scene.

$${}^f\tilde{\mathbf{p}}_i = {}^f\mathbf{c} - {}^u\mathbf{p}_i \tag{5.6}$$

Our goal is to find the scale factors which make the difference between the scaled rays equivalent to the difference between the two user-defined points in the world. We define

$${}^w\mathbf{d} = {}^w\mathbf{p}_1 - {}^w\mathbf{p}_0 \tag{5.7}$$

and since this difference vector is positionless we can use the rotation matrix (5.3) to convert it to film-relative coordinates

$${}^f\mathbf{d} = {}^f_w\mathbf{R}\,{}^w\mathbf{d} \tag{5.8}$$

We now have the situation illustrated in Figure 5-1 and wish to find the scale factors s_0 and s_1 that minimize the difference equation

$$(s_1{}^f\tilde{\mathbf{p}}_1 - s_0{}^f\tilde{\mathbf{p}}_0) - {}^f\mathbf{d} \tag{5.9}$$

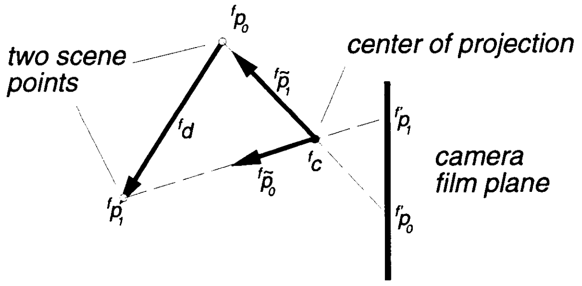


Figure 5-1: Solving absolute translation from a single view given two points and known absolute rotation

which can be found as the mean square solution of

$$\mathbf{U}\mathbf{s} = \left[-{}^f\tilde{\mathbf{p}}_0 | {}^f\tilde{\mathbf{p}}_1\right] \begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \mathbf{d} \tag{5.10}$$

where $\mathbf{U}$ is the 2×2 matrix formed by composition of the 2 rays and $\mathbf{s}$ in the 2×1 vector of scale factors. If we assume for the time being that the first imaged point ${}^u\mathbf{p}_0$ is noiseless then we now have the relation

$${}^f\mathbf{p}_0 = {}^f \mathbf{c} + s_0 {}^f\tilde{\mathbf{p}}_0 \tag{5.11}$$

from which we easily solve the translation component ${}^f_w\mathbf{t}$ of the rigid body transformation matrix since from (3.2) and (5.5)

$${}^f\mathbf{p}_0 = {}^f_w\mathbf{R}\, {}^w\mathbf{p}_0 + {}^f_w\mathbf{t} \tag{5.12}$$

5.2.1 Solving translation given multiple points and known rotation

This approach can be generalized for $n > 2$ points by adding rows to $\mathbf{U}$ $\mathbf{s}$ and $\mathbf{d}$ of (5.10) for each new known 3-D point and its 2-D projection to find the $n + 1$ mean square error solutions to the set of $3n$ equations

$$\begin{bmatrix} -{}^f\tilde{\mathbf{p}}_0 & {}^f\tilde{\mathbf{p}}_1 & 0 & \cdots & 0 \\ -{}^f\tilde{\mathbf{p}}_0 & 0 & {}^f\tilde{\mathbf{p}}_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -{}^f\tilde{\mathbf{p}}_0 & 0 & 0 & \cdots & {}^f\tilde{\mathbf{p}}_n \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \\ \vdots \\ s_n \end{bmatrix} = \begin{bmatrix} {}^f_w\mathbf{R}({}^w\mathbf{p}_1 - {}^w\mathbf{p}_0) \\ {}^f_w\mathbf{R}({}^w\mathbf{p}_2 - {}^w\mathbf{p}_0) \\ \vdots \\ {}^f_w\mathbf{R}({}^w\mathbf{p}_n - {}^w\mathbf{p}_0) \end{bmatrix} \tag{5.13}$$

5.3 Estimating polyhedral scene structure

Vanishing points of parallel sets of lines are key elements for camera calibration. Coplanar groupings of lines are likewise instrumental in building up a polyhedral description of the scene from a single view. Just as parallel lines are grouped into **directions**, coplanar lines can be grouped into **surfaces** whose 3-D plane is defined in standard 4 parameter form such that

$$m_x x + m_y y + m_z z + m_d = 0 \tag{5.14}$$

for any 3-D point $\mathbf{x} = [x, y, z]^T$ which lies in the plane. $\mathbf{m} = [m_x, m_y, m_z]^T$ denotes the plane's nonzero normal vector and the distance from the origin to the nearest point on the plane is $\frac{-m_d}{||\mathbf{m}||}$.

It follows that a plane's projection normal can be estimated from the cross product of the direction vectors of all edges that lie in that plane. A plane's position can be fixed if any feature or edge that lies in that plane is itself fixed. How can this be used to fix the orientation and position of all surfaces in a scene?

5.3.1 Requirements for recovering polyhedral scene structure from a single view

- The scene must be an "origami" scene
- At least one surface must contain a feature or edge with known 3-D position.
- Each surface must have 2 or more directed lines whose 3-D directions are either known or can be recovered: i.e. from vanishing points, from construction by projection of a single line onto a previously fixed surface, or from triangulation in 2 or more views.

Those lines which are shared by two surfaces mark the "hinge" or line intersection between two planes. Scenes that are fully connected in the sense that each planar surface is connected to some other surface via a shared visible line are sometimes called "origami" scenes [13]. If this condition is held, and if normal vectors for each surface can be found then we can then iteratively recover a complete polyhedral description of the scene from a single view.

5.3.2 Origami scene storyboard

After structural relationships among points and lines have been specified (i.e. by grouping those elements into edges, directions and surfaces) and once a sufficient number of unknowns have been set geometric information cascades throughout this web of relationships. Panels (a) through (i) in Figure 5-2 illustrate this process for fixing polyhedral structure in an origami scene as a simple storyboard.

In panel 5-2(a) we manually assign two points in the far corner of the scene arbitrary but meaningful 3-D positions. The bottom feature will be treated as the room's origin and is thus given position $[0,0,0]^T$. The feature above it has arbitrarily been given the position $[0,1,0]^T$.

In panel 5-2(b) we manually assign meaningful 3-D directions to a triad of lines such that they form a right handed triad. Since the point we arbitrarily tagged to be the origin is known to be collinear with each of these lines[1] the 3-D position of those edges can be fixed. At this point, if vanishing points for each direction can be found, they are used with the pair of known points to fix the camera (i.e. solve for camera distortion, center of projection and pose).

In panel 5-2(c) newly fixed features and edges are used to fix the geometry of highlighted surfaces which contain them. Conversely, as shown in panel 5-2(d), highlighted feature and edge elements lying in newly fixed 3-D surfaces can be fixed by projecting points and lines from the fixed camera.

In panel 5-2(e) the two highlighted edges are able to inherit their 3-D position and direction from collinear features and the dashed parallel edge with known direction.

[1]By reason that they share the same surfaces

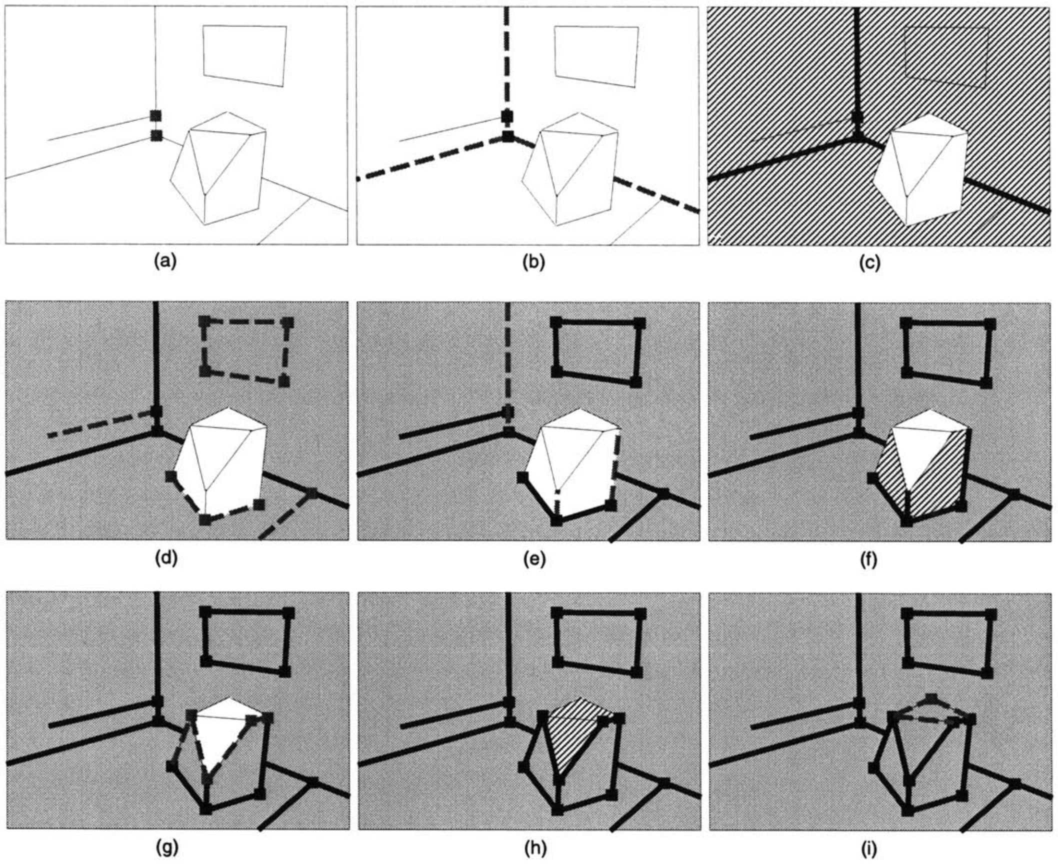


Figure 5-2: Process for fixing an "origami" scene

In panel 5-2(f) the two highlighted surfaces are able to inherit their 3-D position and surface normal given 2 or more newly fixed member edges. Just as shown in 5-2(d), features and edges are fixed by projection in panel 5-2(g).

In panel 5-2(h) two more highlighted surfaces are fixed by inheriting their 3-D position and surface normal given 3 or more newly fixed member features.

And finally, the remaining features and edges are fixed using known surfaces and projection in panel 5-2(i).

Recall that all 2-D image elements, whether specified manually or automatically, have some degree of placement error which can be described probabilistically. Sections A.11, A.12 and A.13 in the Appendix detail equations which are used to estimate probabilistic 3-D positions of cameras, features, edges, and surfaces in a polyhedral origami scene given noisy 2-D image measurements.

5.3.3 Iterative origami fixing algorithm

Having fixed the camera, features, edges and surfaces visible in a single view we can use those results to calibrate and fix the camera and visible elements for successive views. We refer to this iterative process as the "origami-fixing" algorithm.

- If an image has enough points and lines with fixed features and directions then that view's camera is calibrated as shown in Chapter 4 and pose is found as shown in Section 5.1 and 5.2.
- Fixed features and edges are used to fix unfixed surfaces (see Section A.11).
- Fixed surfaces are used to fix unfixed trihedral features (see Section A.13.1) and to fix any unfixed bi-intersecting edges (see Section A.12.1).
- Fixed poses and surfaces are used to fix unfixed features and edges by projecting from image points and lines onto fixed surfaces (see Sections A.13.2 and A.12.2).
- Fixed poses and coplanarity constraints are used to fix as yet unfixed features and edges by triangulating from two or more known views (see Sections A.14 and A.12.3).
- Unfixed directions are given the least-squares fit vector of all fixed edges that belong to that direction.
- Continue this process until no new features, direction, edges, surfaces or cameras have been fixed.

5.4 Error propagation and optimal "fix-path"

If the specified 3-D constraints among detected or drawn 2-D primitives are physically valid the iterative *origami fixing algorithm* will return **one** solution. But how do we know if this solution is the best solution?

All measurements have some noisyness or uncertainty associated with them. When we estimate the position and orientation of some object it will be based on noisy predictions of previously fixed objects which are connected to it as well as on noisy estimates of intrinsic camera geometry and extrinsic camera pose. In addition, the inherent non-linear nature of these projection and triangulation equations causes small deviations to be magnified with distance. The result of these facts is that errors accumulate rather quickly.

The fact that we keep careful track of error covariances during estimation allows us to factor in that information to make decisions along the way that continually minimize uncertainty. This is done by using maximum likelyhood estimators which essentially weight the contributions of different measurements by their inverse covariance.

As illustrated by the *origami fixing algorithm* we also use a depth-first strategy for scene recovery. We try to make the most of constraining geometric information as soon as possible by fixing all elements which can possibly be fixed as soon as enough constraining information is available. This can be likened to a "greedy" minimization process the danger of which is that globally optimal paths will be ignored in the rush to reach a local minimum.

Another way to look at the problem of the depth-first strategy is that the chosen *fix path* (the actual order in which all elements are fixed) will have an effect on the final outcome. Though fixing an element as soon as possible serves to generally minimize the graph radius (i.e. the maximum of all minimum paths to reach the graph boundary) it does not guarantee that the sum of all path lengths is globally minimized.

5.4.1 Finding optimal "fix-path" by combinatoric search

Unfortunately, no global estimation strategy is guaranteed to achieve a global minimum except a full combinatoric search where we test all possible fix paths. This cannot be likened to the classic NP-complete "traveling salesman" problem (for which simulated annealing appears successful[83]) because, in our case, new errors (or costs) are not additive, or even linear, but are strongly dependent upon the mean and covariance of the dependent elements.

One possible framework to efficiently search the entire combinatoric space [2] is to use a binary stack implementation of a full-search tree. All fix paths will not be searched to completion since many will quickly accumulate errors that exceed a previously found minimum path.

Another possible solution is to stochastically sample the set of all possible fix paths. Given our database-driven scene model, this can easily be achieved by running the *origami fixing algorithm* multiple times from scratch, each time randomizing the database's order of access.

Structure recovery errors increase as the distance from the two absolutely known origin features increase. Also, feature and edge positional errors are related to the inverse cosine of their plane angle. Given some initial scene solution we might be able to decrease the global reconstruction error by automatically finding the scene elements which have the greatest effect on the rest of the scene. These elements can momentarily be considered as absolutely known, a global solution is calculated, and the scene is either rescaled according to the initial user input, or the user is prompted to specify positions for those key elements.

[2] As suggested by Dan Gruhl at MIT.

5.5 Parameter refinement Algorithm

Rather than attempting to combinatorically test all possible fix paths we have chosen, instead, to find the local minimum using traditional linear regression techniques. Scene surfaces, camera pose and camera COPs are adjusted in a global refinement stage while keeping scene directions and camera distortions fixed.

- A scene vector is formed which contains all refinable (i.e. non-user specified) camera and surface parameters using up to 9 terms for each camera (3 for COP, 3 for rotation and 3 for translation)[3] and 1 term for each surface[4].
- Define a framework which takes the current parameterization of all cameras and surfaces and uses the methods outlined in chapter 5 and detailed in the Appendix to temporarily get an optimal least-squares reconstruction for all visible 3-D edges.
- Define a cost function of the scene parameter which is based on the weighted sum of misregistrations between predicted 3-D edges and their observed image lines as described in Section A.15 in the Appendix.
- Repeat this process iteratively using the downhill simplex method[83] to find the scene parameter that locally minimizes this cost function.
- Use final scene parameterization to update cameras and surfaces.
- Use final cameras and surfaces to project observed lines into scene to get infinite 3-D edges as described in Section A.12 in the Appendix. Then clip infinite edges to the maximum extent of observed image lines. Note that due to occlusions and edge detector noise, these endpoints will not correspond to the physical endpoints, thus we make no claims about accuracy of edge length, only edge direction and position.

Performance note

As discussed in Section 10.3 the parameter refinement stage typically reduces the residual 2-D observation errors found in the initial rough estimation stage by a factor of 1.5 to 2.0.

[3]Any of these terms can be considered fixed for the entire set of cameras. For example, if we are processing a set of stills taken from a continuous video sequence, the two principal point parameters are considered to be constant but refinable global parameters which are shared by all cameras.

[4]Since scene direction vectors are considered fixed.

Chapter 6

Multi-view texture recovery

Having refined camera and surface geometries image pixels can now be projected onto visible regions of surfaces. Each surface has one or more image polygons which define its visible extent in one or more views. Every image polygon consists of an ordered list of observed image lines. The refined 3-D edges associated with each line are used to create a closed 3-D boundary.

Using known camera and surface geometry these image regions are rectified (or "un-perspectivized") and registered into common rectangular surface coordinates. Where multiple images contribute intensity values to the same surface region pixel intensities may not match precisely due to several different factors (which we will explore in section A.17).

One way to sidestep all the problems associated with view-dependent texture differences is to blend textures from different views in the rendering stage as suggested in [98]. In an effort to keep the model simple and compact (and to keep it compatible with existing VRML viewers) we choose instead to find a static description for texture.

Our task will be to sample image texture contributions for each surface in all views in which it is visible. These texture contributions will then be merged according to differing resolutions due to distance and foreshortening and according to differing extents due to occlusions and framing.

6.1 Texture rectification

To preserve the highest resolution of textural information available in all views we rectify planar texture, thus effectively undoing perspective, while retaining a measure of confidence for each sample so that textures from multiple perspectives can be merged by weighted averaging. As described by Mann and Becker [42] the non-linear 2-D from-warping[1] that embodies the 8 *pure parameters* of a planar patch under perspective projection [49] is of the form

$$ {}^{u}\mathbf{p} = \frac{\mathbf{A}\,{}^{t}\mathbf{p} + \mathbf{b}}{\mathbf{c}^{T}\,{}^{t}\mathbf{p} + 1} \tag{6.1}$$

where ${}^{u}\mathbf{p}$ and ${}^{t}\mathbf{p}$ are 2-D pixel coordinates in the perspective warped film image and perspective corrected texture image. The 8 pure parameters are described in the 2×2 matrix

[1]A *from-warping* is where the texture value for a region in the destination image is pulled from some region in the source image. The opposite *to-warping* takes a texture value in the source image and sends it to some region in the destination image. The prior approach is the one used for image warping since it guarantees that every region in the destination image receives texture values.

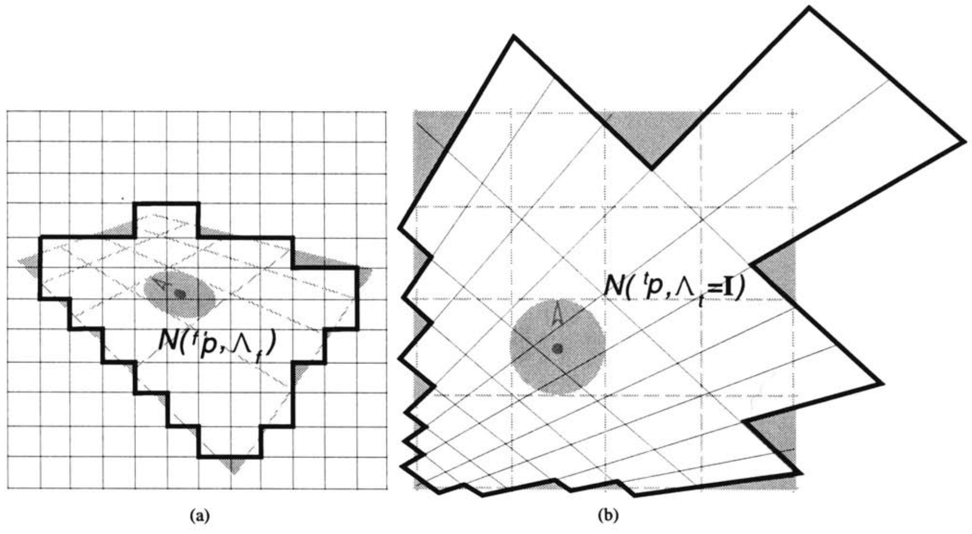


Figure 6-1: Projection of black-edged square image pixels (a) onto texture plane (b) and back-projection of grey unit texture element with distribution $N({}^t\mathbf{p}, \mathbf{\Lambda_t} = \mathbf{I})$ onto foreshortened region in image plane with distribution $N({}^u\mathbf{p}, \mathbf{\Lambda_u})$

$\mathbf{A}$, and the 2×1 vectors $\mathbf{b}$ and $\mathbf{c}$. This 2-D to 2-D mapping can be used to warp any planar surface under one perspective view into any other perspective view. This solution comes from solving 4 pairs of linear equations of the form

$$ {}^t p_x = \begin{bmatrix} {}^u p_x & {}^u p_y & 0 & 0 & 1 & 0 & -{}^u p_x {}^t p_x & -{}^u p_y {}^t p_x \end{bmatrix} \cdot \mathbf{P} \tag{6.2} $$

$$ {}^t p_y = \begin{bmatrix} 0 & 0 & {}^u p_x & {}^u p_y & 0 & 1 & -{}^u p_x {}^t p_y & -{}^u p_y {}^t p_y \end{bmatrix} \cdot \mathbf{P} \tag{6.3} $$

where the 8 pure parameters are contained in $\mathbf{P} = [Axx, Axy, Ayx, Ayy, bx, by, cx, cy]^T$.

To rectify the texture which contributes to a given surface from a particular camera we first find a set of four vertices in the texture image ${}^t\mathbf{p} = [{}^t p_x, {}^t p_y]$ which bound the texture space of the surface. Texture vertices are then converted to world-relative coordinates ${}^w\mathbf{p}$ and projected into image coordinates ${}^u\mathbf{p}$ using the projection equation (3.11). These 4 pairs of coordinates are then used in equations (6.2) and (6.3) to solve for the 8 warping parameters.

We can now use stochastic sampling (as discussed in section 4.1.5 for lens distortion correction) to average intensities within a region in an observed image which map onto any element in the texture region. Given a certain texture element, ${}^t\mathbf{p}$, the corresponding film region that maps intensities onto it has mean position ${}^u\mathbf{p}$ from (6.1), and assuming unit covariance for the texture region $\mathbf{\Lambda_t} = \mathbf{I}$ (see figure 6-1) the covariance of the corresponding film region is defined as

$$ \mathbf{\Lambda_u} = \partial \mathbf{u}/\partial \mathbf{t} \partial \mathbf{u}/\partial \mathbf{t}^T = \mathbf{A} - \mathbf{c}\mathbf{b}^T(\mathbf{c}^{T t}\mathbf{p} + 1)^2 \mathbf{A} - \mathbf{c}\mathbf{b}^T(\mathbf{c}^{T t}\mathbf{p} + 1)^{2^T} \tag{6.4} $$

6.2 Resolution dependent texture merging

The covariance matrix $\mathbf{\Lambda_u}$ also plays an important part in the task of merging data from multiple sources because its inverse covariance $\mathbf{\Lambda_u}^{-1}$ defines the foreshortened projection a unit image pixel makes at that location in the texture image. Since the determinant of a 2-D covariance matrix is linearly related to the area of a confidence interval, from equation (4.7), we can use the inverse of the determinant, $R_i = \frac{1}{\det(\mathbf{\Lambda_u})}$ as a convenient way to measure the relative degree of detail or resolution that the i-th image is able to contribute to a certain texture element. A low R-value means that image detail has been smeared out in the texture image. Conversely, a high R-value (i.e. much greater than 1) signifies that the image had more detail than we bothered to use in the texture image.

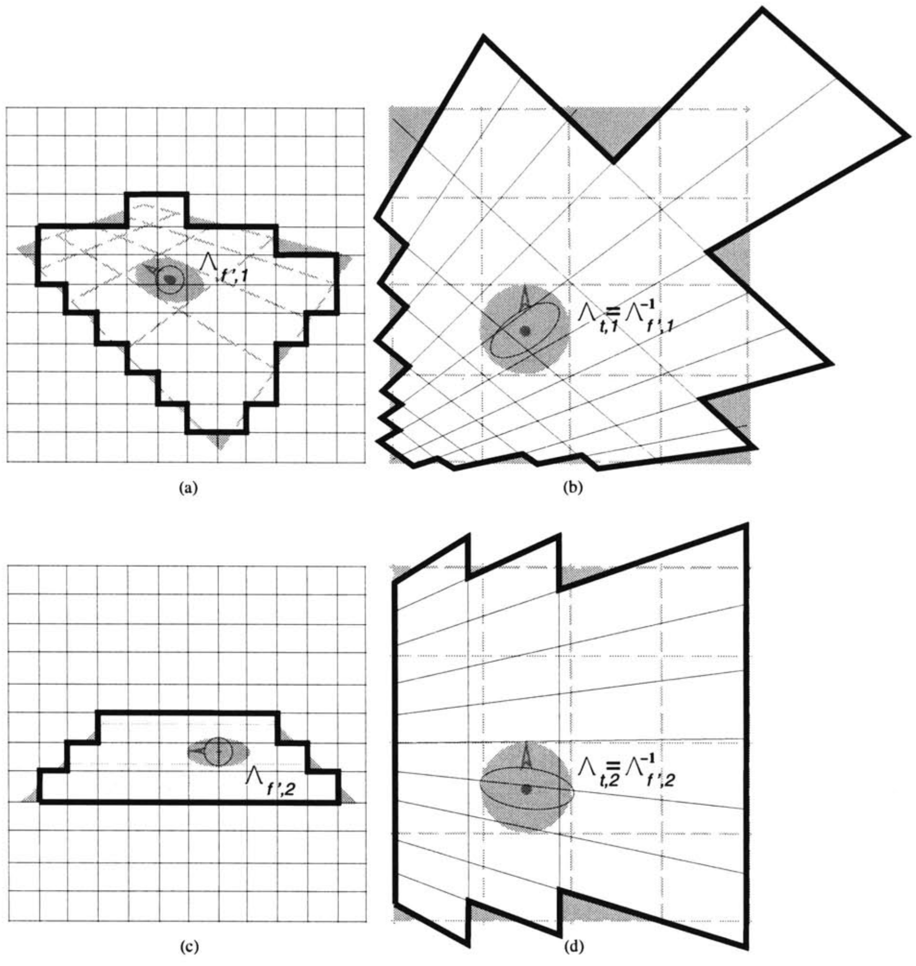

$\Lambda_{f',1}$
$\Lambda_{t,1} = \Lambda^{-1}_{f',1}$
(a)
(b)
$\Lambda_{f',2}$
$\Lambda_{t,2} = \Lambda^{-1}_{f',2}$
(c)
(d)

Figure 6-2: Projection of black-edged square image pixels from two images (a) and (c) onto texture planes (b) and (d). Notice that the foreshortened projection of a unit image (a) pixel to texture (b) is slightly smaller than the projection of (b) onto (d). This indicates that the values from (a) contribute higher resolution and thus should be prefered to contributions coming from (c).

To guarantee maximum detail when recovering texture from multiple views we can use this as the basis for resolution dependent merging. If $\mathbf{X}_i$ describes the average values for red, green, blue and alpha transparency from the i-th of N views, and R_i is the relative resolution, the simplest blending function is to maximize resolution.

$$\mathbf{X}_{\text{out}} = \mathbf{X}_i \text{ where } i = \text{argmax}(R_i) \tag{6.5}$$

This approach may cause visible discontinuities at equi-resolution contours. We may instead use a weighted average to smoothly blend contributions according to relative resolution.

$$\mathbf{X}_{\text{out}} = \frac{\sum_{i=1}^{N} \mathbf{X}_i R_i}{\sum_{i=1}^{N} R_i} \tag{6.6}$$

6.3 Alpha-blending to reduce view-dependent intensity changes

Vignetting is a common artifact in real imaging situations where apparent image intensity falls off due to the combined effects of aperture foreshortening and changing aperture distance as seen from a point on the film. One artifact caused by vignetting is that when attempting to blend texture information from multiple views the original image boundaries are clearly seen as intensity discontinuities.

Specularity adds to this problem. Were the reflectance characteristics of real surfaces perfectly diffuse then they would appear the same intensity regardless of camera rotation or position. In actuality, however, all real surfaces, even carpeting and sidewalk, seem to have some specular component. These differences among views are again particularly apparent at the image boundaries.

To reduce the visibility of view-dependent intensity changes we give image pixels alpha transparency values that increase with radial distance from the principal point. This has the effect of smoothing together contributions from different images and hiding differences at the image boundaries.

The alpha-blending function mixes contributing RGB values according to their associated alpha-transparency value. Assuming we have all contributing $\mathbf{X}_i$ values simultaneously[2] we can use the blending function

$$\mathbf{X}_{\text{out}} = \frac{\sum_{i=1}^{N} \mathbf{X}_i A_i}{\sum_{i=1}^{N} A_i} \tag{6.7}$$

6.4 Texture recovery algorithm

We can now present an algorithm for projecting image texture data onto planar surfaces which merges contributions according to visibility and resolution.

- In each view, use optimal world polys and camera parameters to do a depth-buffer rendering of the entire polyhedral scene. At each bin in the depth-buffer save a unique I.D. which specifies which surface texture is visible. To reduce artifacts we may choose to render at twice the normal resolution.

- Also, in each view initialize an alpha buffer that fades to full transparency as a function of distance to the image boundary or distance from the origin to soften artifacts at image boundaries caused by vignetting, specularity and lighting changes.

[2]this condition removes the disadvantage of normal alpha transparency blending in which rendering order has a great effect on the final output color and transparency

- For each poly of a surface find the linear transformation, ${}^{t}_{w}T$, which transforms 3-D world coordinates ${}^{w}\mathbf{p}$ into 2-D texture image coordinates ${}^{t}\mathbf{p}$ and use it to transform world poly vertices into texture poly coordinates.

- Use the refined camera parameters of each view ${}^{t}_{w}T$ and equations (6.2) and (6.3) to find the 2-D perspective mapping ${}^{u}_{t}\mathcal{P}_2$ from (6.1) which transforms 2-D texture positions ${}^{t}\mathbf{p}$ directly to undistorted 2-D film coordinates ${}^{u}\mathbf{p}$.

- Use scan conversion to visit all texture elements (with position ${}^{t}\mathbf{p}$) which have any portion in common with the texture poly.

- For a given texture element use ${}^{u}_{t}\mathcal{P}_2$ to find its projected position ${}^{u}\mathbf{p}$ in each view.

- Look at the bin in the depth buffer that includes position ${}^{u}\mathbf{p}$. If the I.D. at that location matches the I.D. of this surface/poly that location in the view contributes to the texture of this surface at ${}^{t}\mathbf{p}$.

- Evaluate the Jacobian of ${}^{u}_{t}\mathcal{P}_2$ at ${}^{t}\mathbf{p}$ to determine the extent of the film region that this texture element casts onto. Stochastically filter RGBA (red, green, blue, alpha-transparency) values in that region of the image to find the average contribution of RGBA that the view makes to this texture element.

- Weight RGB contributions of each view according to film resolution as defined in the blending functions (6.5) or (6.6) and alpha-transparency as defined in (6.7) and save it as the best RGB value for this texture element.

- When all texture elements have been processed there will be elements inside the texture poly regions which still have undefined RGB values. This is due primarily to obscuring surfaces and partially due to jaggies at the polygon edges caused by pixel quantization in the depth buffer rendering stage. Median filtering can be used to fill in these holes.

- Visualize the polyhedral scene by rendering each 3-D surface as a 2-D texture image clipped by its bounding polygon.

Chapter 7

sceneBuild

`sceneBuild` is the graphical front end to a novel sort of 3-D modeler that lets the user remain in the 2-D image domain. It is tightly partnered with a 3-D recovery engine called `sceneAnalyze` which implements the algorithms presented in chapters 4 through 6. `sceneBuild` lets a user load a small number of still photographs or frames of video into zoomable viewports and then supplies drawing tools for overlaying color images with points, lines and polygons. `sceneBuild` also lets the user define three dimensional constraints among 2-D *image elements* (points, lines, and polys) by allowing manual grouping of these into color-coded sets of 3-D *scene elements* (features, edges, directions, or surfaces).

This chapter reviews `sceneBuild`'s basic command structure and then gives step-by-step instructions on how to model a scene from one or more photographs.

More on-line documentation for `sceneBuild` may be found at `http://www.media.mit.edu/s̃beck/research.html`.

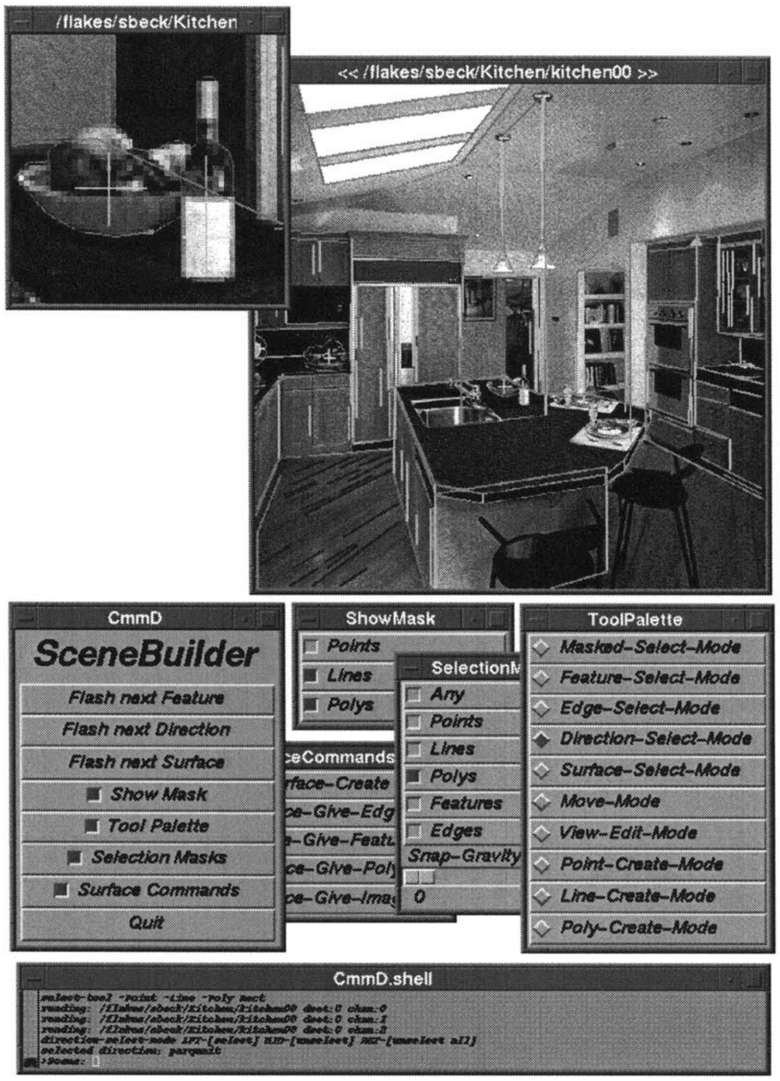
7.0.1 Using the *select-modify* command paradigm

There are four different ways to issue instructions in `sceneBuild` : a Tcl-based command-line interpreter shell, a Tk-based GUI (menus, buttons, sliders, etc.), viewport mouse action modes (draw, select, move, etc.) and viewport quick-keys.

All of these command interfaces use the *select-modify* paradigm. Easy-to-use selection tools (pick, intersect, box or lasso) allow the user to select or deselect one or more 2-D elements visible in any viewport. Visibility and selectability of different 2-D types are controlled by the *show-mask* and the *selection-mask* and selected elements are displayed as bright red. The selected group of image elements is then modified when some action or command is selected either from the ASCII command line interpreter shell, from pull-down menu-items, from always visible command button icons, or from viewport quick-key command aliases.

Note that we will use the word "select" to denote 2-D image elements (points, lines, polys) which have been chosen or highlighted. The word "active" denotes 3-D scene elements (features, edges, directions, surfaces) which are active for subsequent commands.

The *select-modify* paradigm is fairly standard and is easy to use. For example, to group some lines into the active direction the user simply lassos several lines turning them bright red and then presses the "Line-give-direction" command icon. The color of the lines will then change to the color of the direction that they now reference. The active direction can be changed by using the quick-key alias 'd' in any viewport to cycle thru all color-coded directions.



```
select-tool -Point -Line -Poly Rect
reading: /flakes/sbeck/Kitchen/kitchen00 dest:0 chan:0
reading: /flakes/sbeck/Kitchen/kitchen00 dest:0 chan:1
reading: /flakes/sbeck/Kitchen/kitchen00 dest:0 chan:2
direction-select-mode LFT-[select] MID-[unselect] RGT-[unselect all]
selected direction: parquet
>Scene: 
```

Figure 7-1: A screen shot of `sceneBuild`. The top-most windows are zoomable viewports used to view the image data with overlying elements. The center panels provide GUI controls and action icons. An ASCII terminal interface is shown in the bottom window.

Since one 2-D image element can reference many 3-D scene elements (i.e. a line can belong to an edge, a direction and one or more surfaces) scene element membership visualization is done primarily by flashing when color is already used as another grouping cue (i.e. flash all lines that belong to the active surface).

7.0.2 States and action-modes

`sceneBuild` maintains several states or global variables which are sometimes used as built-in arguments for different operations.

```
selected-image-elements
active-feature
active-edge
active-direction
active-surface
show-mask
select-mask
action-mode
```

The action-mode controls what happens when mouse clicks and drags are done in the viewports. As with normal drawing programs, mode changes are made by selecting action icons in a command window. The mode can also be changed from script or command line in the interpreter shell. The current mode is easily identifiable by the mouse cursor when over a viewport. The different action-modes and their cursors include

This mode is used to slide and zoom the viewport's view into the image.

`view-edit-mode`	`resizable box cursor`

These modes are used to create new image elements

`point-create-mode`	`circle and cross-hairs cursor`
`line-create-mode`	`cross-hairs cursor`
`poly-create-mode`	`X-windows cursor`

These modes are used to select and modify existing image elements

`select-elements-mode`	`pointer-cursor`
`move-selected-elements-mode`	`cross-arrows cursor`

These modes are used as a convenient way to activate scene elements by selecting image elements.

`feature-select-mode`	`X-windows cursor`
`edge-select-mode`	`X-windows cursor`
`direction-select-mode`	`X-windows cursor`
`surface-select-mode`	`X-windows cursor`

7.1 Using `sceneBuild` to model a scene

We'll now review the different information gathering stages needed to model a scene : selecting an appropriate photograph, image capture, line detection/specification, specifying parallel constraints, specifying an arbitrary world frame and scale, specifying coplanar constraints and optional specification of additional line correspondences.

7.1.1 Selecting an appropriate photograph

First of all, the photo you choose to model from must consist primarily of straight lines. If significant amounts of lens distortion are present then long uninterrupted lines help improve distortion correction. If focal length or principal point are unknown then conditions of 3-pt perspective among long lines should exist as well. Texture extent improves as original camera focal length decreases, but geometric accuracy is maximized in the middle ranges of focal length (see vision-assisted modeling results in chapter 10). To get the best accuracy from a totally uncalibrated view you will get best results if mostly rectangular objects, such as building exteriors or room interiors, are viewed from low or elevated corner views.

`sceneBuild` was designed and written to work with images from a magazine or even hand sketched drawings. But if you wish to model an existing room or building with your own camera you may opt to take certain liberties to insure maximum accuracy and quality. Section A.16 in the Appendix gives some guidelines when planning your own on-site photo shoot.

7.1.2 Image capture

One or more *images* of any size, orientation, focal length, distortion, or exposure can be used. Still images can be digitally scanned to arbitrary resolution using a flatbed image scanner or individual frames of video can be digitized using standard video capture hardware.

Digital cameras like the Kodak DC50 or Apple's QuickTake 100 can be used as long as the pixel aspect ratio is 1:1. Photo scanners with 1:1 aspect ratio can also be used but accuracy will be affected, even if focal length is held fixed over the entire photo-shoot, since photo development and flat-bed scanning introduce unavoidable misregistrations, so the principal point will be different among views.

Required image format : datfiles

`sceneBuild` takes digital images which are in MIT's *datfile* format. A datfile is a directory which contains a binary `data` file with 8 bits of raw non-interleaved per pixel per channel (i.e. RRRR...GGGG...BBBB... for 24-bit color or YYYY... for 8-bit grey) and an ASCII `descriptor` file with the format:

```
(_data_sets 1)
(_channels 3)
(_dimensions 480 640)
(_data_type unsigned_1)
```

where in this case height is 480 and the width is 640.

7.1.3 Line detection

To detect lines automatically `sceneBuild` uses a line detection algorithm written by Stefan Agamanolis[80] which finds and traverses continuous chains of edge pixels detected by the Canny[79] edge detection algorithm.

Each *line* in an image is initially given its own 3-D *edge* placeholder. If two lines in the same image are made to refer to the same 3-D edge using "line-give-edge" then we say those two lines are collinear. `sceneAnalyze` uses line collinearity constraints to help solve

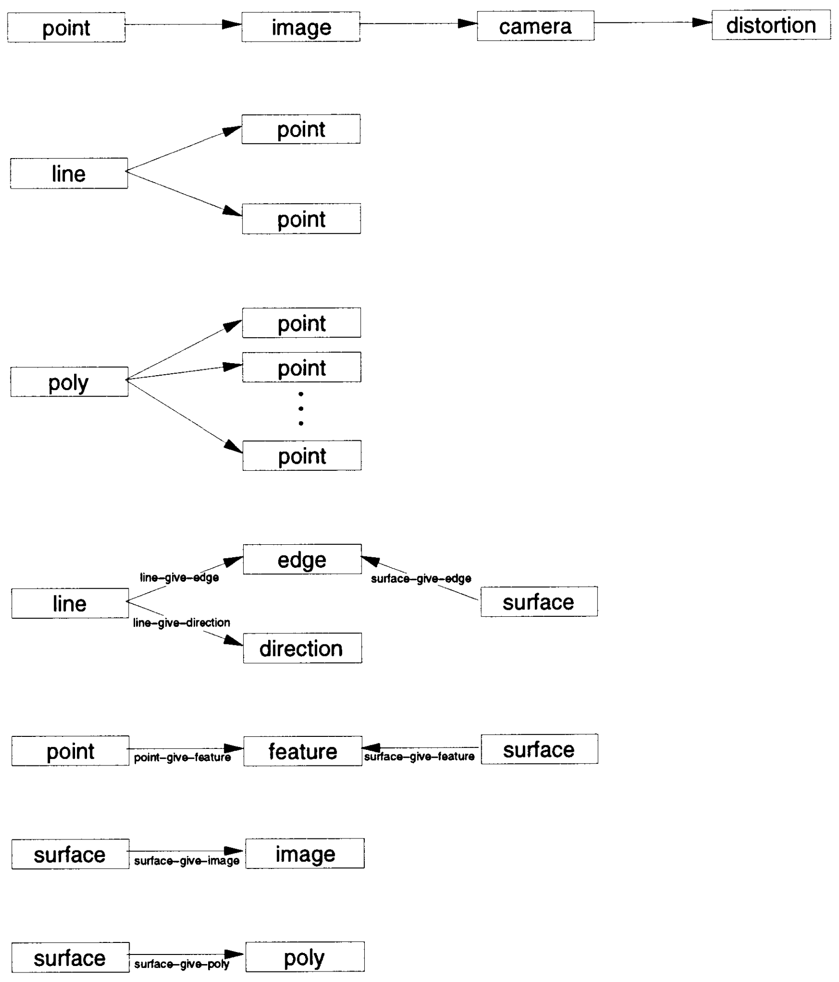


Figure 7-2: Membership hierarchy of image elements and scene elements

lens distortion. This relationship between lines and edges can be seen in figure 7-2 which shows the membership hierarchy of image elements and scene elements.

If two lines in different images refer to the same 3-D edge then we say that those two lines are matched or corresponded. `sceneAnalyze` uses line-correspondences to help constrain the solution of the camera pose as well as the position of the 3-D edge. Explicit line matching among views is encouraged but not required since some matching is done implicitly for "hinge-lines" which are shared by multiple surfaces. Under noisy conditions, however, explicit line matching helps improve camera pose and structure accuracy.

7.1.4 Specifying parallel constraints

All 2-D lines which come from a set of parallel 3-D edges in the scene, regardless of the image they are in, need to be assigned the same *direction* scene element. A direction is a placeholder for which `sceneAnalyze` estimates a world-relative direction vector. Every 3-D edge element inherits its direction vector from a 3-D direction and `sceneAnalyze` estimates the world-relative position for that edge's midpoint.

`sceneBuild` offers the choice to view lines according to the display characteristics of their parent edges, directions or surfaces. Therefore, as for edges and surfaces, direction elements can be assigned graphical attributes (i.e. unique line color, line thickness and dash style) to help visualize common member lines.

Automatic line classification

Lines can be explicitly assigned to a direction using the "line-give-direction" command. To accelerate this process the "direction-select-common-lines" command can be used. To understand how it works, note that under perspective projection, 3-D edges which are parallel in the scene give rise to 2-D image lines which share a vanishing point. Whenever a direction is assigned two or more lines in one image a least squares vanishing point is calculated and assigned to it. Once it has a vanishing point the "direction-select-common-lines" command finds and selects all lines that cross within some user specified distance of that vanishing point. "line-give-direction" can then be used to make the assignment[1].

7.1.5 Specifying coplanarity constraints

Coplanar lines, regardless of the image they are in, must be manually assigned to have the same *surface* element. Similar in purpose to edge and direction elements a surface element is a placeholder for which `sceneAnalyze` computes an infinite 3-D plane. Lines are marked as belonging to a surface with the "surface-give-edges" command.

Assigning lines a common surface serves to implicitly match sets of lines among multiple views. For example, recall the notion of "hinge-lines" (from Section 5.3). If a given line in one image is assigned to two surfaces then that line can be considered the intersecting hinge-line between those two planes. If both surfaces are visible in another image then the hinge line common to those surfaces in that image is implicitly matched to the first hinge line. We can make the claim of simplifying the problem of specifying line matches since matching groups of lines among several images is easier than matching individual lines [2].

7.1.6 Specifying surface visibility

Surfaces serve as a mechanism for specifying coplanarity constraints among features and edges. Surfaces can also receive pixel intensities from different images to form a texture map. If a surface wants to receive a texture map that surface must be marked as being visible in an image using the "surface-give-image" command.

By default, if a surface is marked as visible from a particular image it will receive pixel intensities from the convex-hull bounded region which contains all points and lines of features and edges which belong to that surface.

Visual extent can be overridden by manually creating one or more disjoint or overlapping polygons and using the "surface-give-poly" command. This is useful when we want to impose the constraint of geometric coplanarity on elements in two surfaces but need to have two disjoint texture surfaces. For example, our scene might have two flat tables at opposite sides of a room which have the same height. We would group the edges of both table tops into a single surface to impose the coplanarity constraint. We would then group the edges

[1]At or near the horizon line for a plane, edges in that plane which have different scene directions produce collinear lines in the image. For these lines direction classification must be done manually.

[2]We should stress again, however, that accuracy of the final reconstructed scene may improve if non-hinge lines (i.e. lines belonging to only one surface) are explicitly matched among images

of each table into separate polygons so the floor and chairs would be visible in-between the tables.

Additional visibility control is provided by the use of the "surface-give-shadow-poly" which allows us to define polygons that occlude visibility from large visible extents.

7.1.7 The role of features

Just as matched lines in different images reference the same edge corresponding points are made to reference the same *feature*. Features and referencing points are automatically created by `sceneAnalyze` wherever a set of 3 lines are shared among 3 intersecting trihedral surfaces. Features and referencing points can also be automatically created wherever 2 non-parallel and coplanar matched edges intersect. `sceneAnalyze` then solves the optimal 3-D world-relative position for each feature.

Like edges, features can also be made subject to a coplanarity constraint of some surface by using the "surface-give-feature" command.

To create a mesh surface in a calibratable origami scene point correspondences among multiple images can be specified explicitly using the "point-create-mode" in `sceneBuild`. This action-mode allows creation of a new feature placeholder and then creates one referencing point at user specified positions in each image. Features can be created explicitly using the "feature-create" command and pre-existing points (i.e. those already belonging to lines or polys) can be matched to a feature using the "point-give-feature" command.

7.1.8 Specifying an arbitrary world frame and scale

In order for `sceneAnalyze` to be able to establish a coordinate frame for the scene the `sceneDatabase` must contain at least three non-coplanar directions with orthogonal world vectors. By default, `sceneBuild` creates the orthonormal triplet of X, Y and Z directions if it is started without loading an existing `sceneDatabase` file. Directions can be renamed and reoriented using the "direction-rename" and "direction-set-direction" commands. These directions are considered to be *known* or *fixed* and will form a basis that establishes a rigid coordinate frame for all 3-D scene elements and cameras. Each image needs to have at least two lines for each of these three coordinate directions in order to perform camera calibration and to estimate the camera's absolute rotation.

To establish arbitrary camera position and scale two 3-D features must be created and assigned known 3-D positions. The vector between the points should agree with the orthonormal triplet of directions. The coordinate scale used for the feature position is in pixels but the distance between the two features is arbitrary since 2-D approaches for 3-D recovery cannot recover scale.[3] These registration features can be created with the "feature-create" command and given a 3-D position with "feature-set-position". Each feature also needs to be activated and attached to appropriate surfaces that it lies in using the "surface-give-feature" command. At least one image needs to have one point for each of the two registration features in order to estimate the camera's absolute translation. All other images must share at least two coplanar features or one coplanar edge with another image in order to satisfy the origami connectivity requirement (described in Section 5.3.1).

[3]Which explains why photos of realistic looking scale models can easily trick us into believing that they're actual size. To make sure the final computed 3-D scene is out in front of the camera make the scene-units distance between the two feature a few times greater than the pixel distance between them in the image.

To help eliminate rotational ambiguity (see Section 5.1.1) one line each of at least two known directions must be given an arrow in each image. Line arrows are placed to indicate the heading of a line's direction vector as viewed in a particular image[4]. This can be done manually by simply clicking one end of any line with the middle mouse button.

7.1.9 Review of `sceneAnalyze` requirements

Each of these requirements are checked by `sceneAnalyze` before starting the reconstruction algorithm. If any of the preconditions are not satisfied `sceneAnalyze` provides a moderately instructive warning message along with the name and index of the offending element when appropriate.

- The scene must contain three directions with known orthonormal world-relative vectors.
- The scene must contain two features with known world-relative position.
- Each image needs to have at least two lines for each of three coordinate directions in order to perform camera calibration and to estimate the camera's absolute rotation.
- At least one image needs to have points for each of the two registration features in order to estimate the camera's absolute translation.
- All other images must share at least two coplanar features or one coplanar edge with another image in order to satisfy the origami connectivity requirement.
- One line each of at least two known directions must be given an arrow in each image.
- All origami scene restrictions must be satisfied (see Section 5.3.1).

7.2 `sceneDatabase` file format

`sceneBuild` maintains an internal database that describes 2-D point, line and poly image elements and their memberships to 3-D scene elements (these interrelationships are diagrammed in Figure 7-2).

Elements in this database can be output serially to an ASCII `sceneDatabase` file that describes record index, element type and field values. This database file format is used by all `scene` programs. To illustrate the simple ASCII format we show the element entries that describe a single image line.

```
                ⋮
elem[ 12]={
        type="point";
        img=[6];
        pos=[251;107]
```

[4]Alternate methods are to provide the octant number explicity or to provide a very rough estimate of camera pose, but putting arrows on lines is both simpler and provides a useful graphical context which helps the user remember orientation when working with many images with very different poses.

```
};
elem[ 13]={
        type="point";
        img=[6];
        pos=[241.2345;163.0813]
};
elem[ 14]={
        type="line";
        img=[6];
        edge=[9];
        dir=[3];
        arrow=[1];
        pnts=[12,13]
};
                ⋮
```

Note that element 14 is a *line* that references *points* 12 and 13 as its endpoints. All three elements are visible in *image* 6. Line 14 also belongs to *direction* 3 and has been given an arrowhead pointing from point 12 towards point 13 that describes the heading of direction 3 in this image. The careful observer will note that point 12 has a truncated pixel position while point 13 has a floating point pixel position. This is because point 13 may have been adjusted while at a zoomed in scale factor thus allowing subpixel placement while point 12 was detected, created or last adjusted at normal scale.

Initially, the scene file created by `sceneBuild` contains only positional information about 2-D points and lines. CAD-like constraints among lines and points are defined by way of membership in *image/cameras*, *features*, *edges*, *directions* and *surfaces* whose 3-D positions are unknown (except for those used to define the coordinate frame). `sceneAnalyze` uses these 3-D relationship among 2-D elements to solve the three-dimensional positions of *cameras*, *features*, *directions*, *edges* and *surfaces*.

7.3 `sceneBuild` multi-process diagram

The `sceneBuild` application is implemented as several client-server component processes. `sceneBuildView` is the server process which receives commands from one or more client processes. To allow immediate graphical response the server manages the memory resident `sceneDatabase` and raw image data. The server is written in C and calls Xlib routines to handle rapid visualization and graphical manipulation of scene and image elements.

Client-server intercommunication is done by protected modification of atoms in the root X-Window's property list. This message passing mechanism is compatible with Tcl/Tk's TkSend method thus making it possible for any Tcl/Tk script or X11 process to open a connection with `sceneBuildView` to do message passing and remote procedure calls. The complete separation of server functionality from client graphical user interface allows us to refine each part independently, so we can for example, easily build and test new GUI ideas. Two such stand-alone clients have been written: `sceneBuildCmmd` and `sceneBuildCmmd.tk`.

`sceneBuildCmmd` is a lightweight ASCII interpreter shell which uses GNU's readline library to support emacs-style text editing, command and filename completion, history, and script file processing. This stand-alone process runs in its own window and is used

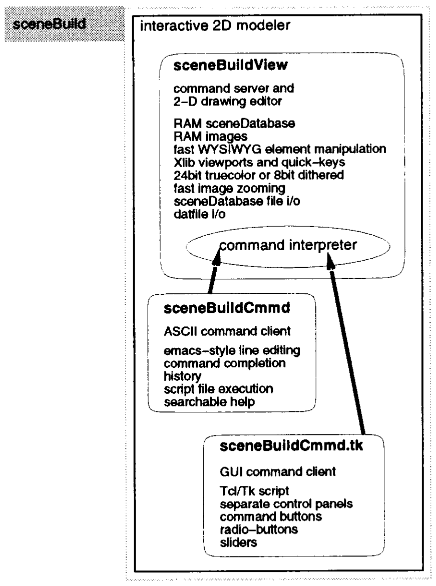


Figure 7-3: `sceneBuild` process diagram

to send ASCII commands to the `sceneBuildView` server. The server receives and interprets these commands from all connected clients in its own event loop [5]. In this way the `sceneBuildCmmd` client acts as the server's terminal, filtering both ASCII input and output[6].

`sceneBuildCmmd.tk` is a Tcl/Tk script that also sends commands to the `sceneBuildView` server. It's use of command buttons, radio-buttons and sliders presents a functional and attractive front-end to the `sceneBuild` application. These GUI command interfaces are simply aliases for ASCII-style commands which are sent to the server using Tk's "send" routine and are identical to those sent by the ASCII command client.

[5]This is also the way "quick-key" commands are handled when keys are pressed while any of the server's viewports are active.

[6]Many `sceneBuildView` commands output text to `stdout` stream of its terminal window. If `sceneBuildView` and `sceneBuildCmmd` are started from the same terminal window (as done by `sceneBuild` and `sceneBuild.tk` scripts) then both server and client will output to the same `stdout` stream. The `stdin` stream, however, is not shared; the `sceneBuildCmmd` ASCII shell has sole access to that stream since `sceneBuildView` never attempts to read from it.

Chapter 8

sceneAnalyze

sceneAnalyze is a stand-alone batch program that inputs a sceneDatabase file of 2-D image elements, processes those elements and then outputs a new sceneDatabase file with fixed 3-D geometry and texture added. It has processing stages which implement the algorithms described in chapters 4 through 6. Command line options are provided to control which processing stages are applied to the data so that if any stage runs into problems you can go back to sceneBuild to supply the necessary information and then resume processing where you left off.[1]

8.1 sceneAnalyze processing stages and command-line options

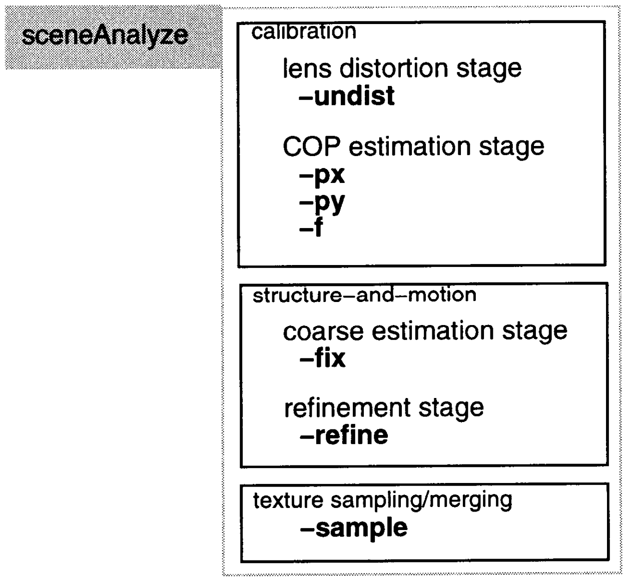


Figure 8-1: sceneAnalyze processing stages diagram

[1]Command line options allow you to do all stages at once or to do them in separate calls saving intermediate sceneDatabase files along the way.

8.1.1 -undist options used to solve and correct for lens distortion

The lens distortion stage is an optional processing stage that, with the user's discretion, can be applied if images used in the **sceneDatabase** have appreciable lens distortion[2]

It has two sub-tasks: distortion estimation and stochastic unwarping. Once lens parameters have been estimated the mapping can be applied to all geometric image elements (points, lines, polygons) as well as to the textures of each image.

Since image dewarping is computationally expensive it is typically applied only once and then undistorted versions of all input images are saved as datfiles. The new output **sceneDatabase** element file is made to reference these new undistorted datfiles.

The optional **-undist** parameters are of the form

```
-undist <estimate> <resample> <clipping> <scaling> <scaleFactor>
```

The boolean **<estimate>** flag expects a value of 0 or 1. It is used to indicate whether or not you wish to reestimate lens distortion parameters. Parameters should be reestimated whenever the lines in a scene are altered.

The boolean **<resample>** flag is used to indicate whether or not you wish to resample images and create undistorted images. Resampling should be done whenever the distortion parameters have changed. The **<estimate>** and **<resample>** options have been separated so distortion estimation and line adjustments can be done iteratively if desired.

The **<clipping>** option, which expects either the word **interior** or **all**, indicates what portions of the image are to be included in the output undistorted image (see Figure 8-2).

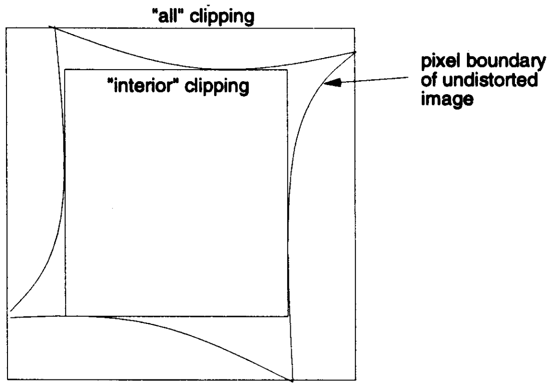


Figure 8-2: Clipping styles for undistortion warping.

The **<scaling>** rule, which can have one of the values **resolution width** or **geom**, gives additional control over the final size of the output undistorted image. When images have barrel (or fisheye lens) distortion right-angled corners in the distorted image unwarp to extended "wings" in the undistorted image. This usually results in making the undistorted image have a much greater extent than the original. The scaling rule controls the overall size of the undistorted image. **resolution** indicates that the size of the undistorted image should be chosen so that no high frequency detail is lost in the warping process. **width** indicates that the sampling rate should be scaled so that the output image has the same

[2]Use this processing stage only when the effects of lens distortion are greater than the effects of noisy line detection or placement.

width as the input distorted image. `geom` indicates that the output pixels fit within the exact same geometry as the original image.

The `<scaleFactor>` option expects a floating point value. This factor is applied after all other clipping and scaling rules have been applied to provide additional control on the overall scale of the output image.

8.1.2 `-px, -py` and `-f` supply COP estimation priors

`sceneBuild` provides commands to explicitly set the principal point and focal length in pixels for an individual camera. In general, however, this information is not known. The COP priors `-px, -py and -f` let the user supply more general a-priori information for any unknown cameras.

Command line formats for the optional COP estimation rules

```
-px (variable | constant | <pixels> | center )
-py (variable | constant | <pixels> | center )
-f  (variable | constant | <pixels> )
```

The `-px` and `-py` flags refer to the image buffer pixel coordinates (also referred to previously as the film-relative coordinates) of the principal point. The `-f` flag refers to the effective focal length in pixels. Together these 3 parameters define the film-relative coordinates of the COP.

Each COP flag expects one of several different values. The keyword `variable` indicates that the given COP parameter changes from camera to camera. The keyword `constant` indicates that the given COP parameter is the same for all cameras in the file. The keyword `center` indicates that the given principal point parameter is believed to be the image center. The COP flags can also be given an explicit floating point `<pixel>` value.

The COP estimation priors are used extensively in `sceneAnalyze`'s `-fix` and `-refine` stages.

8.1.3 `-fix` runs the coarse structure-and-motion estimation stage

This processing option requires no special arguments. It informs `sceneAnalyze` to start the iterative origami fixing algorithm described in Section 5.3.3.

8.1.4 `-refine` runs the structure-and-motion refinement stage

This processing stage can only be applied after the scene has been "fixed" using the `-fix` option. This stage implements the iterative parameter refinement algorithm described in Section 5.5. It has two parameters:

```
-refine <tolerance> <numSearches>
```

`<tolerance>` is a floating point stopping tolerance for the downhill simplex algorithm: the iterative search algorithm stops when the errors of all simplex vertices are within the given `<tolerance>` of each other.

`<numSearches>` is an integer that controls how many coarse to fine searches should be used. The downhill simplex algorithm is run `<numSearches>` times, each time with a stopping tolerance of `<tolerance>` raised to decreasing powers of 10.

Most experiments and modeling examples have used `<tolerance>` and `<numSearches>` values of 0.01 and 3.

8.1.5 `-sample` runs the texture sampling stage

The texture sampling stage implements the algorithms described in Chapter 6. This stage creates 2-D texture maps for each surface in the scene marked as visible as described in Section 7.1.6.

```
-sample <path> <scaleRule> <scaleFactor>
```

`<path>` is the absolute path name[3] of the directory which will be created to hold the scene's 2-D texture maps.

`<scaleRule>`, which can have the value `uniform` or `independent`, specifies what kind of sampling rate should be used for each surface. `uniform` indicates that sampling rate (in texture elements per world units) should be the same for all surfaces. `independent` lets each surface independently choose a sampling resolution so that the resolution available from all contributing images is optimally retained. If this option is used, then surfaces which were nearer to some camera will get more texture elements and surfaces far away from any camera will be given only a very few.

`<scaleFactor>` is the floating point pixels per texture element factor that scales the overall sampling resolution of all surfaces. This value must be chosen wisely since it controls the dimensions of the texture images used for each surface. If a typical reconstructed surface has dimensions of 1000.0 units then the 24 bit image representing the texture map for that surface will have dimensions of `<scaleFactor>` times 1000.0. [4]

8.1.6 Miscellaneous `sceneAnalyze` command-line options

`-elem <inputElemFile>` is used to specify the source `sceneDatabase` element file.

`-out <outputElemFile>` is used to specify the destination `sceneDatabase` element file.

`-classify <imageNumDirs>` performs automated classification of image lines into `<imageNumDirs>` directions as described in Section 4.1.3. This processing stage is experimental and works only with relatively uncluttered scenes.

`-view <featureSize> <downSampleRate>` lets the user view the 3-D scene interactively. `<featureSize>` specifies the floating point cube radius used to visualize 3-D features. `<downSampleRate>` is a positive integer that initializes the viewer with downsampled texture maps.

`-usrcop` enables the user to bypass a search-optimization stage used in COP estimation thus forcing it to use the provided COP priors as absolute knowledge rather than as helpful suggestions.

`-cop` can be used as the option of last resort if insufficient information exists in the image to determine the COP (i.e. the view has only one-point or two-point perspective) and resulting scene recovery errors are unacceptably high. It begins either a fullsearch

[3]We reemphasize that the path must be absolute and not relative.

[4]If you are not sure what the reconstructed scene dimensions are we suggest first testing the `-sample` stage using a low `<scaleFactor>` (like 0.1 or even 0.01) to avoid long texture sampling times and excessively large file sizes. After viewing the results with `sceneView` invoke `sceneBuild` with `-sample` again using a higher `<scaleFactor>`.

or a recursive downhill search thru COP parameter space attempting to reconstruct the scene from scratch using each COP hypothesis. Cost is based on the same cost function as that used for the refinement stage. Its effect differs from that of the refinement stage because the choice of COP before scene fixing has a much greater affect on the final structure-and-motion solution.

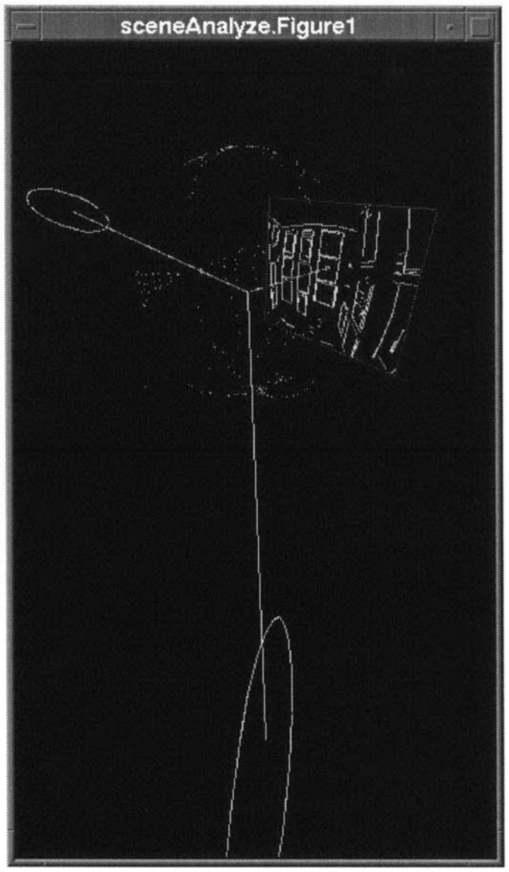


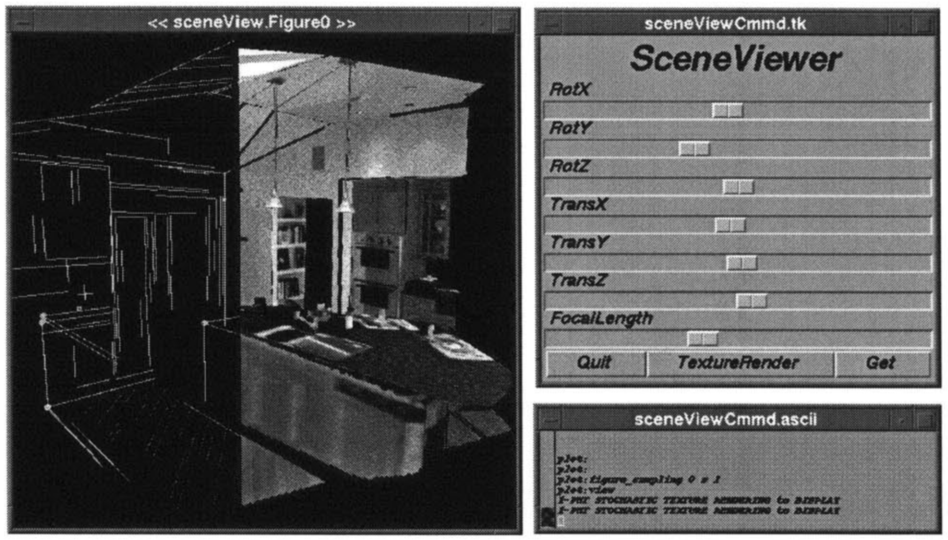
Figure 8-3: `sceneAnalyze` provides interactive 3-D viewing of intermediate results for visual debugging.

-debug <level> is used to set the verbosity level of `sceneAnalyze` (the default quiet level being 0). When the debug mode is activated iterative 3-D visualization aids are provided to aid in understanding intermediate results of the structure-and-motion recovery algorithm. Figure 8-3, for example, shows visualization of vanishing point covariance minimization during lens distortion estimation.

-room, -cube and -exp are used to run batches of synthetic experiments. Each option: creates synthetic cameras and scene geometry; renders a wireframe or textured view of the synthetic scene, adds Gaussian white noise to image elements, runs the processing stages to recover scene and camera parameters, determines the recovery errors, and writes results to some log file.

Chapter 9

sceneView



```
plot:
plot:
plot:figure_sampling 0 s 1
plot:view
1-PNT STOCHASTIC TEXTURE RENDERING to DISPLAY
1-PNT STOCHASTIC TEXTURE RENDERING to DISPLAY
```

Figure 9-1: A screen shot of **sceneView**. The left-window is a 3-D viewport that allows mouse-dragging to interactively rotate, translate and zoom the scene. The viewport is split showing the real-time wireframe rendering mode and the slower anti-aliased texture rendering mode. The top-most panel on the right provides GUI controls for modifying the camera's pose and action icons for rendering. **sceneView**'s ASCII terminal window is shown at the bottom right.

sceneView is a stand-alone 3-D viewer that loads a **sceneDatabase** file and its texture datfile images so the 3-D scene can be viewed interactively or texture rendered as frames to a datfile movie. It has an ASCII terminal interface to an imbedded Tcl interpreter. The interpreter allows Tcl-scripted control of the renderer's display options and camera parameters for the purpose of creating simulated fly-by or fly-thru movies of the scene model.

sceneView is bi-modal in the sense that the user can switch between two operating modes: interactive viewing mode and ASCII terminal mode. Each of these modes has its own event loop[1] and each has commands to switch to the other operating mode.

9.1 sceneView's interactive viewing mode

Rendering is done using custom graphics routines written in C and Xlib by the author that enable real-time wireframe visualization and slower high-quality antialiased stochastic texture sampling for either 24 bit truecolor displays or dithered for 8 bit displays.

Mouse-drag and other Window events are bound to Tcl-scripted handlers which can modify intrinsic and extrinsic parameters of the viewing camera. Standard object rotate, object translate and camera zoom handlers are bound by default, but new interpreted handlers can be easily rewritten or downloaded while in terminal mode letting us experiment with new view manipulation methods on the fly.

Viewport key-press events are also handled by the Tcl-interpreter. For example, this Tcl script which displays a wire-frame movie loop of all defined cameras is executed whenever the `m` key is pressed in a viewport.

```
proc Key-m { } {
  % scan the two parameters returned by camera-select as integers
  % and put them into variables selected and numframes
  scan [ camera-select ] "%d %d" selected numframes;
  for { set i 0 } { $i < $numframes } { incr i } {
    % give the i'th camera to the viewport
    camera-select i;
    % tell the viewport to render its camera
    Key-x;
  }
}
```

Pressing the question mark key while in a viewport (`Key-?`) gives on-line help pages. The exclamation point (`Key-!`) switches `sceneView` into its ASCII terminal mode, thus temporarily suspending any handling of viewport window events.

9.2 sceneView's ASCII terminal mode

This operating mode gives us direct access to `sceneView`'s Tcl interpreter. It supports emacs-style line editing, command and file name completion, history, script file execution and searchable on-line help. It is in the ASCII terminal that new Tcl procedures, such as `Key-m`, are written and named according to the event they should handle. Use the `help` command in `sceneView`'s ASCII terminal to see a listing of more named handlers.

9.2.1 sceneView's camera array commands

`sceneView` provides a set of routines to create and manage an array of cameras. As shown in figure 9-2 (a) and (b) we can create a set of keyframes and then use `camera-interp` to calculate a smooth closed loop of interpolated cameras among those keyframes. Use `camera-movie` to see a wireframe version of your newly created movie.

[1]Which as of this writing do not run concurrently

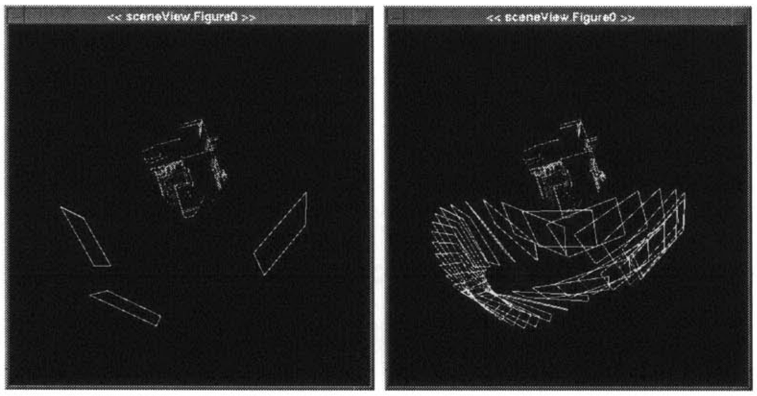


Figure 9-2: Using cubic interpolation among 3 keyframes (a) to produce a smooth closed loop camera path (b).

Here are some of the most relevant camera array commands.

`view` exits the ASCII interpreter shell and returns to interactive viewing.

`camera-select` select a camera to display in the active figure.

`camera-keep` copies the camera of the active figure into the selected camera.

`camera-new` inserts a new camera after the selected camera.

`camera-interp <numNewCameras>` uses the camera array as key frames and does linear or cubic interpolation between them to create a new longer camera array.

`camera-movie <numLoops>` does multiple wireframe render loops of all views.

9.2.2 Rendering a synthetic movie to a datfile

Once the array of cameras has been created it can be used by the renderer to produce a set of frames which can then be output as a continuous datfile movie file viewable by Isis or xdat.

`camera-render <datfile> [ filled | wire ]` Texture (or wireframe) render the camera array to a 24-bit datfile movie which can be viewed by xdat or converted to an MPEG file.

`figure-sampling <figure> [ nearest | stochastic <maxSamples> ]` This command sets the specified figure's texture rendering style. `nearest` sets the rendering style to fast "nearest neighbor" interpolation or zero-order sample and hold approach. Use the `stochastic` option to use antialiased stochastic sampling and use `<maxSamples>` to set the limit on how many rays can be cast to find intensity samples for a single texture element.

9.3 sceneView's remote client support

`sceneView` uses a client-server process framework similar to that used by `sceneBuild` to support Tcl/Tk's TkSend-style interprocess communication (see Figure 9-3). `sceneViewCmmd.tk`

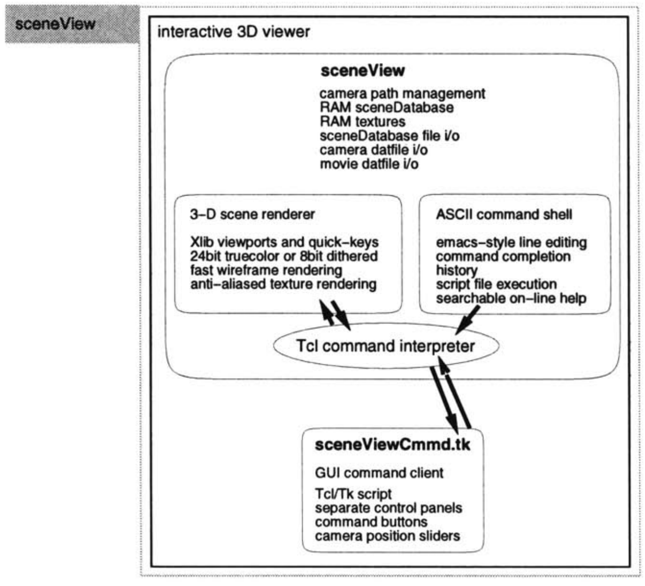


Figure 9-3: `sceneView` process diagram

(shown on the right above and in figure 9-1) is one such external Tcl/Tk command client that uses this mechanism to provide convenient GUI controls for manipulation of the camera's COP, rotation and translation using sliders and command buttons. The handlers for these widgets send Tcl commands to `sceneView`'s interpreter,which then executes them as if they arrived from its own terminal window.

9.4 VRML converter

`sceneDatabase` files can be converted to OpenInventor VRML1.0 files using the `sbeckScene2iv` program[2] available on Media Lab SGI workstations. The resulting file can then be viewed over the web using any one of the latest VRML viewers or browser plug-ins on most operating systems.

[2]`sbeckScene2iv` was written by Eugene Lin.

Chapter 10

Vision-assisted scene modeling results

In this chapter we present experimental results for structure recoverability on synthetic and real-world scenes. Section 10.1 first compares our new distortion correction technique to the traditional plumb-line approach. Section 10.2 does a statistical error analysis for rough pose and structure recovery from a single view using tens of thousands of synthetic experiments. Section 10.3 shows typical results for single view recovery and refinement from a simple parallelepiped object. Section 10.4 presents results found by modeling scenes from multiple views while pointing out limitations of the approach and areas of future work. Section 10.5 provides a user performance analysis showing the through-put rate using scene package as a 3-D from 2-D modeling platform. Finally, in Section 10.7 we compare our approach to related work in this area.

10.1 Distortion correction

10.1.1 Comparison of plumb-line and vanishing point approaches

The standard "plumb-line" approach for solving geometric lens distortion, first introduced by Brown [23], iteratively finds the parameters that minimize a cost function defined as the sum of displacement areas between curved set of points and the line that fits those points in a least squares manner (see section A.1 in the Appendix).

The plumb-line approach for straightening long curves should perform well for solving geometric lens distortion in controlled calibration settings. It may be inappropriate, however, for solving distortion directly from images in non-controlled settings because it requires detection of long continuous curves. In real imaging conditions we think ourselves fortunate to be able to detect short linear segments, much less long continuous piecewise linear curves.

Recall that our new approach for estimating lens distortion defines its cost function as the sum of vanishing point variances for each direction in the scene. Where the traditional plumb-line approach requires grouping of line segments into continuous curves our new approach requires grouping of line segments only into common directions.

To compare the effectiveness of the two approaches, we can vary the allowed spanning length of piecewise linear curves and measure the residual error between curves in the correct image and those in the undistorted control image. Whenever we allow full length curves (a length factor of 100%) we have situations optimal for the plumb-line curve-straightening approach and the RMS of pixel displacements should be small. As we reduce the length factor allowing curves to span only some fraction of the image we gradually approach conditions more akin to real imaging conditions which are probably less favorable for the plumb-line approach. Therefore, as the length factor decreases the RMS residuals for the plumb-line approach should be higher than those incurred when using the vanishing point approach which is not as sensitive to curve length.

To test this hypothesis a series of experiments were conducted which use distorted views of a simple planar scene consisting of infinitely long straight lines (see Figure 10-1).

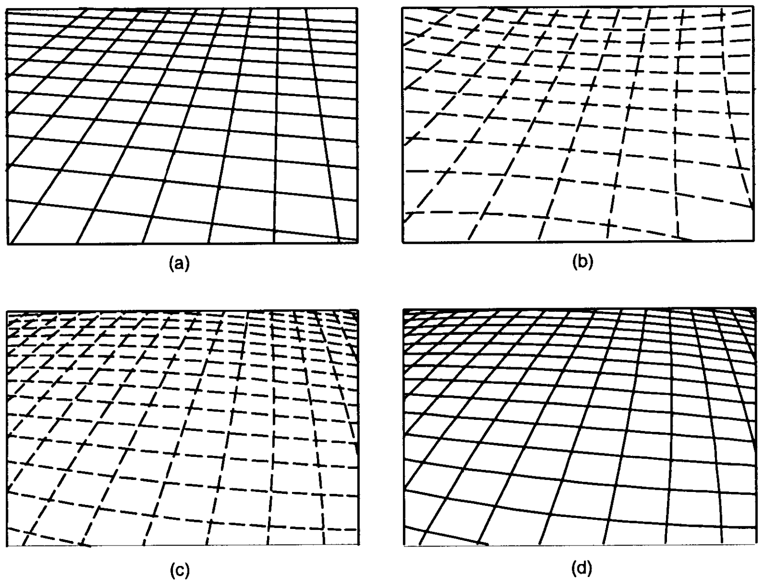


Figure 10-1: (a) Undistorted view of a set of planar edges with continuous curves; (b) highly fractionalized view (no curves with length > 20 % of image width) with negative radial distortion; (c) highly fractionalized view with positive radial distortion; (d) non-fractionalized view (curves can have length up to full image width) with positive radial distortion.

Figure 10-1(a) shows the undistorted control image. Figures 10-1(b) and (c) show negative and positive radial distortions which are highly fractionalized. Figure 10-1(d) shows a non-fractionalized view with high positive radial distortion. The residual RMS pixel errors[1] are shown for different percentages of radial distortion[2] in Figure 10-2[3].

Interpretation of results

First of all, note that, as predicted, the plumb-line technique has high residual errors when

[1]RMS pixel error is the square root of the mean of squared distances between points.

[2]An image with 5.0% radial distortion has been warped so that a pixel on the furthest horizontal edge has been displaced by 5% of the image radius.

[3]The data on these graphs represent 80 renderings (4 different distortions each with line fragmentation at 20 different scales) with each view submitted to 2 different iterative distortion estimation techniques each requiring 10 to 500 iterations in 1 to 50 seconds on a 200 MHz 64bit DEC station Alpha running Digital Unix 3.0 with 64Mb RAM.

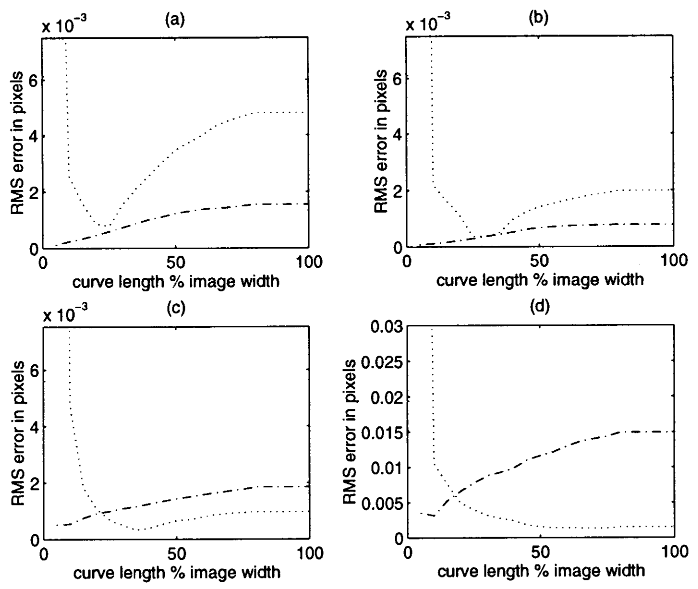


Figure 10-2: Residual RMS pixel errors from the original undistorted model using the plumb-line (dotted) vs. vanishing point (dot-dashed) approach for length factors from 0.05 to 1.0. These results are shown at radial distortion levels of (a) 5.0%; (b) 2.5%; (c) -2.5% and (d) -5.0%

the allowed curve length is only a small fraction of the entire image width[4]. This applies for all levels of radial distortion. The reason for this failure is because, when piecewise linear curves contain no more than one line segment, there is no curve to straighten and the residual curvature error is correctly measured as zero, although the image is still distorted.

Also note that for positive (fish-eye or barrel) distortions, Figure 10-2(a) and (b), the vanishing point approach offers better recovery than the plumb-line approach for all curve lengths. However, the plumb-line approach appears to do better for the less common negative (pin-cushion) radial distortions when line segments have been grouped into long continuous curves.

From these experiments we can conclude that plumb-line style curve straightening can get comparable and sometimes better results than the vanishing point approach, but that accuracy comes with the cost of requiring that line segments be grouped into common curves. The new vanishing point approach in contrast has good recoverability with or without curve grouping. In addition, since the typical man-made scene has fewer directions than edges we can claim that the vanishing point approach can more easily solve distortion with less or even no user interface.

Another shortcoming of the plumb-line approach is that it does not guarantee a geometric distortion correcting warp which is projectively valid. The plumb-line technique can find a warping that straightens long curves. However, there is no constraint that those straightened lines intersect at a common vanishing point. Our approach simply makes the observation that given the additional knowledge of common direction we can exploit that knowledge to maintain both straightness and convergence constraints in an elegant fashion.

[4]In these cases the residual is actually that of the observed distorted image compared to the undistorted control image since the initial zero distortion estimate is accepted as the minimizing location.

10.1.2 Comparison of different orders of complexity

We would like to determine how many distortion terms are needed to successfully correct for lens distortion. Equation 3.7 which describes radial and decentering distortion is actually an infinite Taylor series using only odd powers of radial distance. To estimate distortion parameters we truncate the series thus ignoring higher order terms. Which terms do we ignore?

Most existing calibration approaches seek only to recover the coefficients for the first and sometimes for the third powers of radial distance [1]. Some point out that higher orders terms of radial and decentering distortion cannot be recovered under noisy imaging conditions [2]. We might reason that attempting to estimate those additional terms only adds more unknowns to a somewhat ill conditioned problem, thus serving only to undermine the recoverability of the more important lower order terms.

To test the usefulness of higher order terms in the radial and decentering distortion equation we will use the experimental scene used in section 10.1.1 and vary over percentage of lens distortion but compare combinations of terms used in Equation 3.7. Figure 10-3 shows residual RMS pixel errors from the original undistorted model after attempting to solve for 6 different combinations of higher order radial and decentering terms[5].

Interpretation of results

We see that when the actual applied distortion is a pure first order radial distortion we may be able to get a better correction if we attempt to solve higher order terms as well. Whether our cost function is based on plumb-line deviation or vanishing point variance, if we allow higher order terms to vary it actually brings us closer to the correct solution even though those cost functions are unrelated to it (i.e. these cost functions are not in any way a distance metric from the original undistorted image but rather a cost function of local properties of the image). Therefore, it seems reasonable to say that in the presence of unmodeled effects like noise or mapping error, estimating for higher order terms can improve our estimate.

[5]The data on these graphs represent 42 renderings (21 different distortions each with line fragmentation at 2 scales) with each view submitted to 12 different iterative distortion estimation techniques (2 different constraint rules and 6 different sets of coefficients to solve for) each requiring 100 to 500 iterations in 4 to 20 seconds on a 200 MHz 64bit DEC station Alpha running Digital Unix 3.0 with 64Mb RAM.

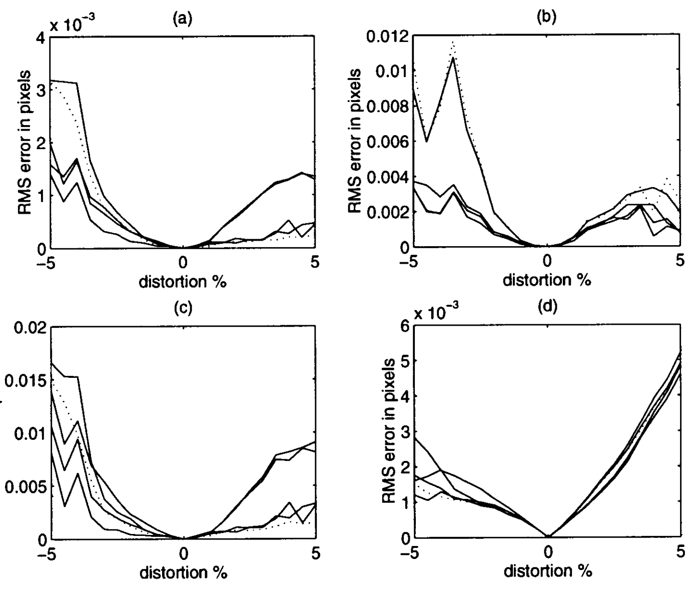


Figure 10-3: The dotted line shows the residual RMS pixel errors from the original undistorted model at different degrees of pure radial distortion when attempting to solve only for the 1st order term of radial distortion. The other 5 solid lines show the results when attempting to solve for different combinations of higher order terms. Figure (a) shows RMS errors after using the vanishing point constraint on view with with max curve length of 10% of image width and (b) is the same but using the plumb-line technique. Figure (c) and (d) are identical to (a) and (b) but allowing max curve length to match the entire width of the image.

K_1	1.105763E-01
K_2	1.886214E-02
K_3	1.473832E-02
P_1	-8.448460E-03
P_2	-7.356744E-03

Table 10.1: Results of radial and decentering distortion estimation : Museum scene

10.1.3 Real distortion correction example

Figure 10-4 shows a typical example of positive radial distorted view of a man-made scene[6]. Line detection[7] is done by first detecting edge pixels using Canny's approach [79] and then fitting line segments to edge pixels subject to a breaking angle and minimum edge length. Line segments are then manually grouped into parallel sets and the vanishing-point constraint method is used to solve the parameters for three terms of radial distortion (K_1, K_2, K_3) and two terms of decentering distortion (P_1, P_2).

Figures 10-5 and 10-6 show edges and edge normals before and after distortion correction.

[6]This is from an exhibit of student art displayed at MIT's List Gallery of Contemporary Art, photographed January 1994 by Araz "H.I.C." Inguilizian.

[7]As implemented by Stefan Agamanolis

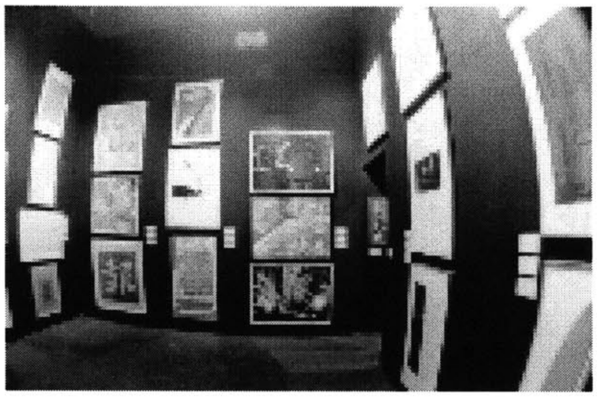


Figure 10-4: Original distorted museum view

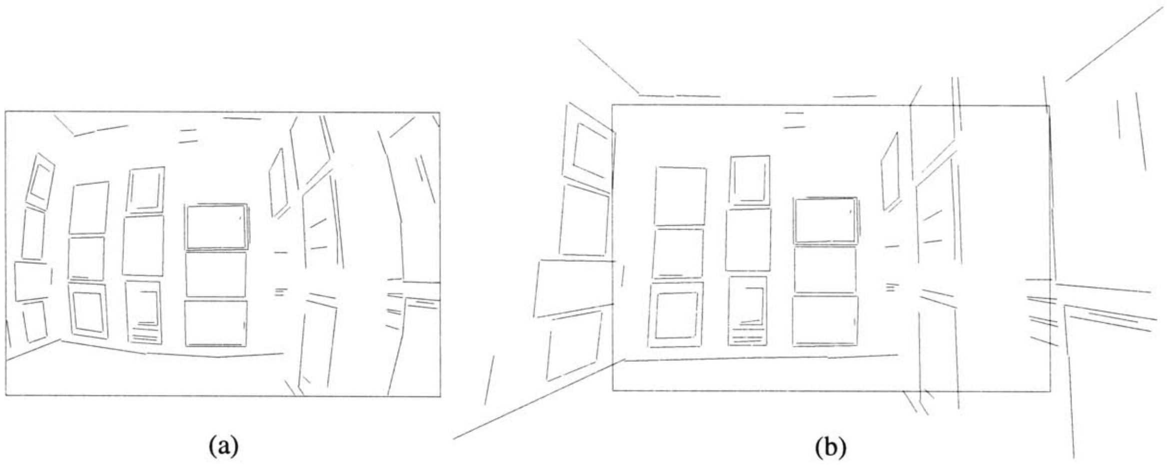


Figure 10-5: Camera calibration: (a) detected edges in original view with lens distortion (b) edges after distortion correction

Figure 10-7 is then created using stochastic resampling to apply the estimated distortion mapping to all pixels in the scene as described in Section 4.1.5.

Interpretation of results

First of all note that the scene contains very few long continuous curves. Notice that the tops and sides of the picture frames are not all set at equal levels. While the image has many X, Y and Z edges, few of them could be extended to belong in the same curve. These facts make the image ill-suited for correction using the traditional plumb-line technique.

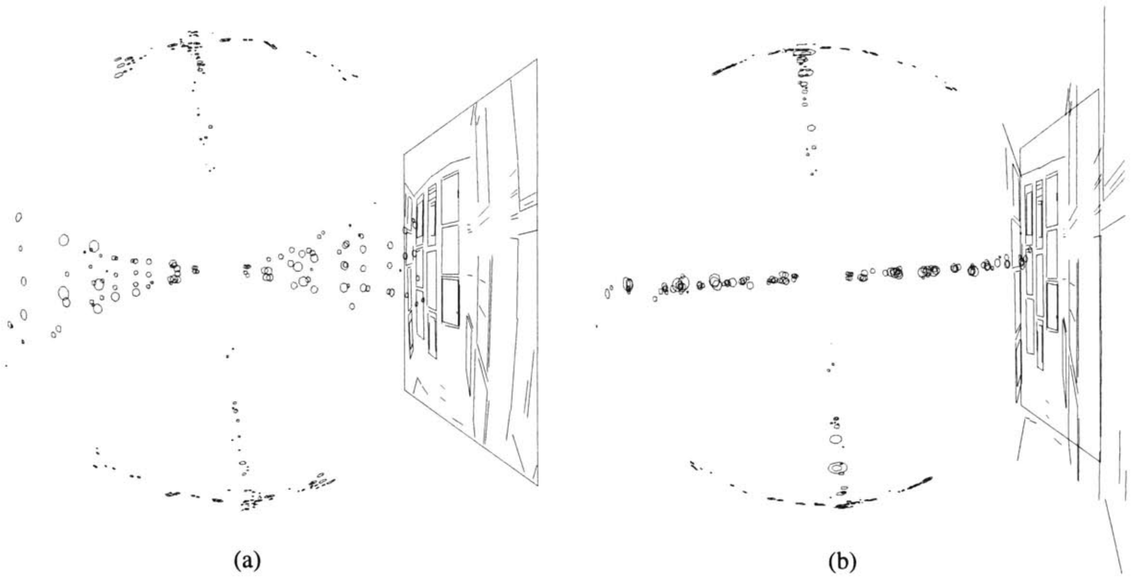


Figure 10-6: Results of camera calibration on distributions of projection normals (a) detected edges and normals from the original distorted view (b) edges and normals after distortion correction. Notice the flattening effect that vanishing point minimization has on the plane of edge normals whose normal points towards a vanishing point in the image.

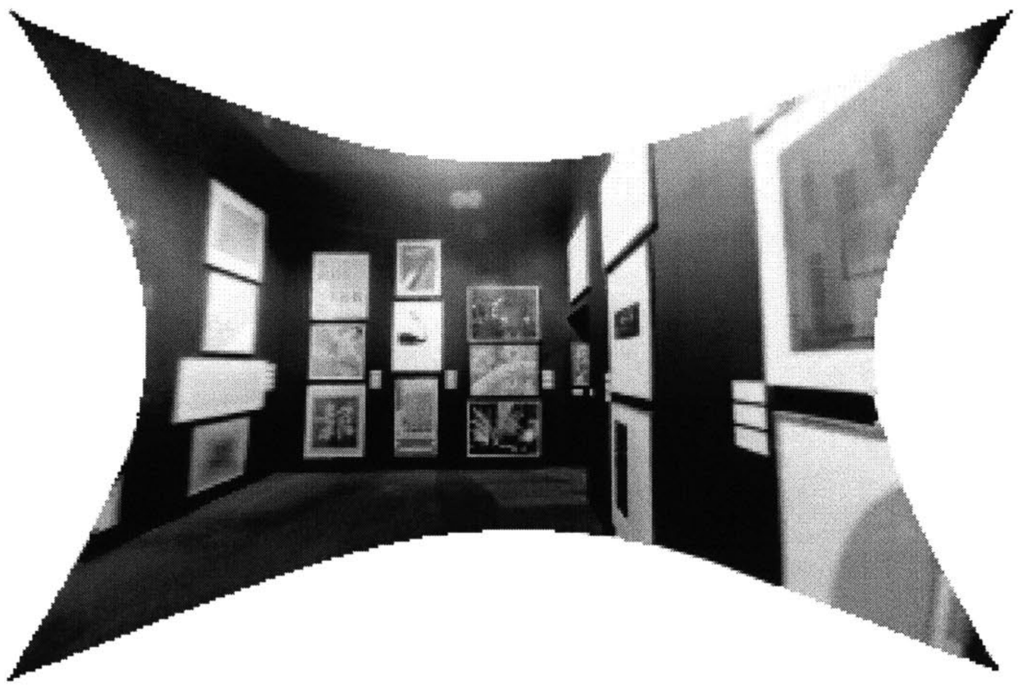


Figure 10-7: New undistorted museum view

10.2 Synthetic single-view experiments

In this section we will use synthetic experiments to determine the accuracy of our origami fixing algorithm for recovering camera parameters and structure from a single view in the presence of noise. This is the initial rough estimation stage which precedes the iterative refinement stage.

The following four sections show the results from experiments in which we vary the radius of uniform pixel noise, true focal length, principal point bias, and spin angle. For each experiment we show a series of 4 graphs detailing *RMS* errors of film-relative center of projection, world-relative camera rotation, world-relative translation and world-relative structure as a function of the freely varying parameter. Each graph has 100 samples in the freely varying parameter. Statistics at each sample are taken over 100 trials, thus these graphs show the results of 40,000 separate calibration and scene recovery experiments.

Each experiment starts by generating a synthetic trihedral scene shown in Figure 10-8. The 3-sided cube is 80mm on each side and has simulated "detected" features positioned along each principal axis $L = 40$mm from the origin and has parallel sets of edges placed every 13.3mm. The scene is viewed with a simulated $F = 50$mm focal length camera with 36×24mm film from a distance of $Z = 250$mm. The direction from the cube's origin to the camera's principal point is described by x-tilt and y-spin angles with default values of 45.7^o and 30^o respectively about the cube's x and y axes.[8]

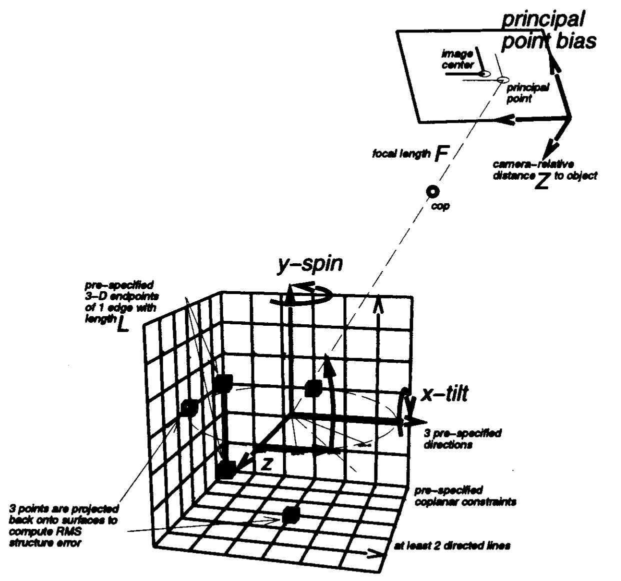

principal point bias
image center
principal point
focal length F
camera-relative distance Z to object
cop
y-spin
pre-specified 3-D endpoints of 1 edge with length L
x-tilt
3 pre-specified directions
Z
pre-specified coplanar constraints
3 points are projected back onto surfaces to compute RMS structure error
at least 2 directed lines

Figure 10-8: Synthetic scene used for single-view experiments

Edge endpoints are projected and clipped to lines in the camera's film plane. Lines are then broken up into short segments to simulate the discontinuous nature of line detectors in the presence of noise. Segment endpoints are then given uniform random noise and then truncated to an integer resolution of 480×320 pixels. Some lines are given arrows as needed to eliminate rotation ambiguity. Edge direction and coplanarity information is retained in the rendering stage. All synthetic views are considered uncalibrated, i.e. no

[8]Tilt/spin angles of 45.7356 and 45 degrees align the view point along the cube's $[1, 1, 1]^T$ axis, an orientation which maximizes conditions for 3-point perspective.

a-priori information about optical center is assumed. Also, we do not add or attempt to correct for radial or decentering distortion.

10.2.1 Sensitivity to pixel noise

For the first series of experiments we use the standard setup but vary the additive uniform pixel noise from a radius of 0 to 3 pixels. Figures 10-9(a)-(d) graph the behavior of reconstruction errors for various camera and scene parameters.

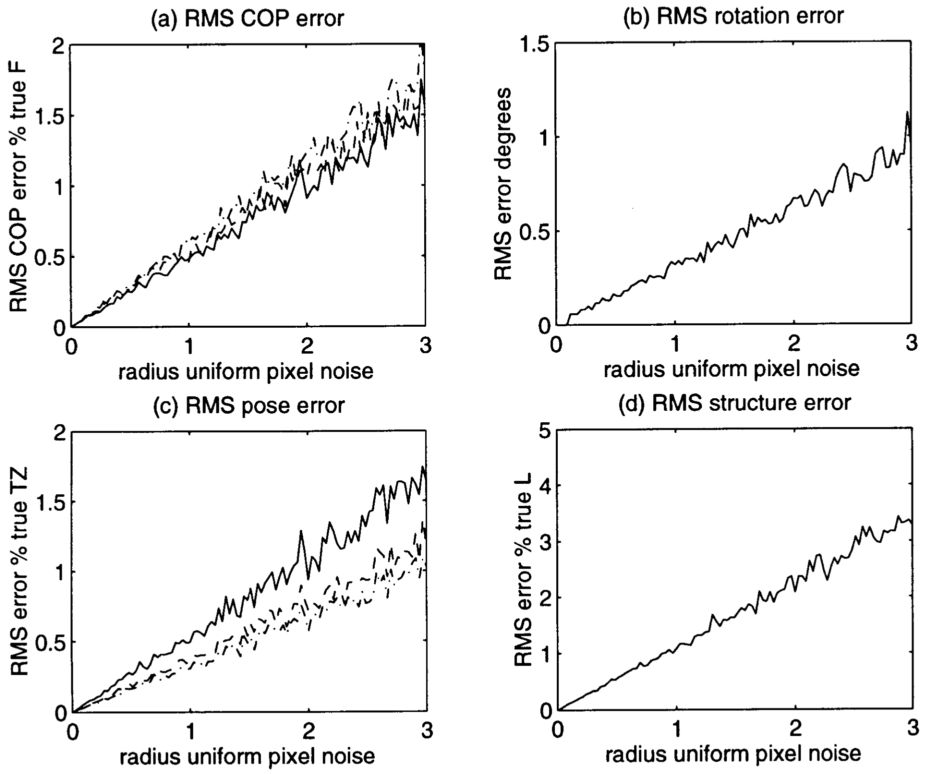


Figure 10-9: Variation in RMS errors of estimated (a)COP, (b)rotation, (c)position and (d)structure as a function of increasing pixel noise.

Note that the RMS errors for COP, rotation, position and structure are all linear with respect to pixel noise. The dotted lines in (a) and (c) show the X and Y parameters for COP and position and the solid lines show the Z parameters. Also note that in Figure 10-9(c) that when noise is 1 pixel, the recovery error for camera Z distance is around 0.6% of the actual distance. This error value is near to the theoretical limit of recoverability as detailed in Section A.7 in the Appendix.

10.2.2 Reconstruction sensitivity to focal length

In this section we use the standard setup treating all camera parameters as unknown but let the actual focal length vary over a typical range of focal length values. Figures 10-10(a)-(d) show the behavior of the reconstruction errors.

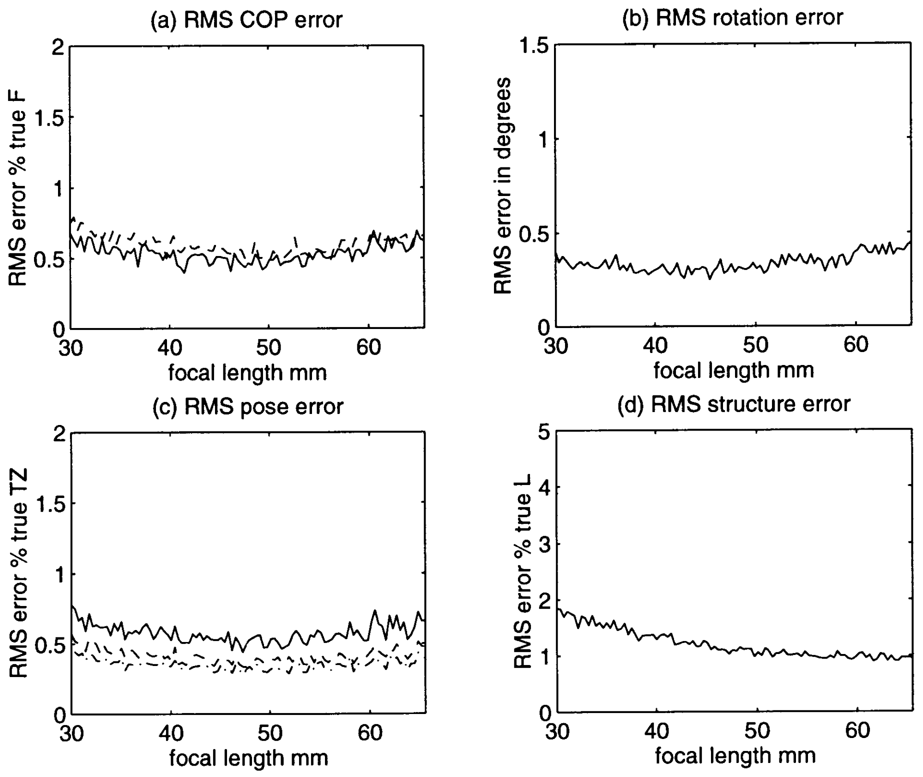


Figure 10-10: Variation in RMS errors of estimated (a)COP, (b)rotation, (c)position and (d)structure as a function of actual focal length in the presence of up to 1 pixel uniform placement noise.

Note that since the default focal length is 50mm and the default noise is 1 pixel, comparison of figures 10-9 and 10-10 at these locations show similar values as expected. The flatness in figures 10-10(a)-(c) indicates that internal camera parameters are not strongly affected by true focal length over this range of lens lengths. Figure 10-10(d), however, shows that structure error decreases slightly as focal length increases. This is correctly predicted in the theoretical treatment of structure recoverability as a function of varying focal length detailed in Section A.8 of the Appendix.

10.2.3 Sensitivity to unknown bias in principal point

Previous situations made no assumptions about principal point position. Here we displace the true principal point from the geometric center of the film by some number of pixels and study the ability of the algorithm to recover all camera parameters and scene geometry. Figures 10-11(a)-(d) show the behaviors of reconstruction errors over the entire range of unknown principal point bias from 0 to 20 pixels in an image with dimensions 480×320 in the presence of uniform positional noise of 1 pixel.

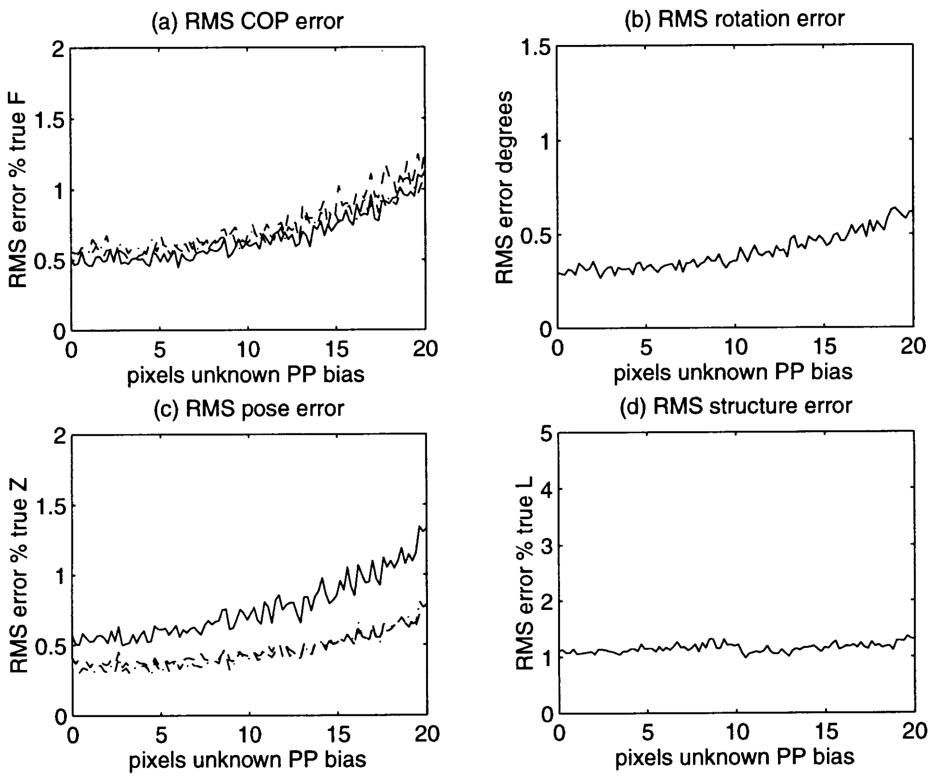


Figure 10-11: Variation in RMS errors of estimated (a)COP, (b)rotation, (c)position and (d)structure as a function of unknown principal point bias in the presence of up to 1 pixel uniform placement noise.

From these experiments we can see that increasing principal point bias has an increasing effect on camera COP, rotation and position accuracy. It seems to have relatively no effect, however, structure recovery error. This may be due to the fact that any errors in principal point can largely be explained as a camera-relative shift in camera position. The principal point can be treated as the image center and camera position will compensate for it. Structure can then still be correctly recovered but the principal point and camera position will be biased resulting in increasing *RMS* errors for those camera parameters.

10.2.4 Sensitivity to relative rotation

Our approach for estimating center of projection requires conditions of 3-point perspective when no a-priori camera knowledge is given. If the camera's position and rotation are defined by tilt and spin angles of rotation about the world's X and Y axes respectively we can explore a range of situations. Specifically, we keep the tilt angle constant and vary the spin angle about the scene's vertical Y axis. A starting spin angle of 0 places the camera in the scene's Y-Z-plane thus making a 2-point perspective situation since the scene's X axis is parallel to the film plane. Increasing the spin angle brings us to 3-point perspective until spin angle of 45 gives us maximum conditions of 3-point perspective.

Figures 10-12(a)-(d) show the behavior of reconstruction errors over the entire range of unknown spin angles, from 0 to 45 degrees.

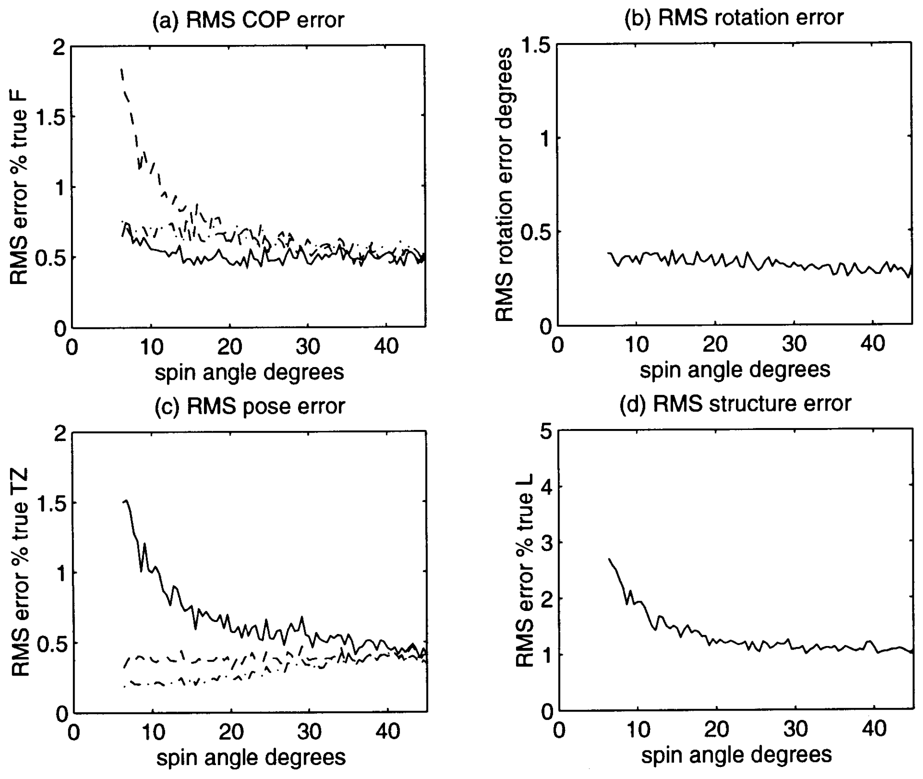


Figure 10-12: Variation in RMS errors of estimated (a)COP, (b)rotation, (c)position and (d)structure as a function of y-spin angle in the presence of up to 1 pixel uniform placement noise.

These results show that this approach fails when conditions approach 2-point perspective (i.e. having spin angle of less than 9 degrees). Large error variances (and low recoverability) for the estimated center of projection cause high variances in turn for camera position and scene structure. Qualitative and theoretical treatments for this situation are provided in Sections A.9 and A.10 of the Appendix. Rotation errors, in Figure 10-12(b), are unaffected until 1- or 2-point perspective conditions are fully attained because large variations in vanishing point positions are foreshortened when vanishing points are distant from the image center.

To address the sensitivity issues of using the 3-point perspective model `sceneAnalyze` provides a rule base that let's the user decide what assumptions can be made. The default case is that if the estimated COP variance exceeds some threshold the rule base takes over and tests different combinations of normal assumptions. The simplest of these assumptions is to constrain one or both of the principal point terms to the image center.

10.3 Real single view experiments

In this section we recover camera and scene parameters from a single view of a real scene. The `scene` package has been used by the author and others to reconstruct at least twenty scenes. This particularly simple scene was chosen to illustrate the utility of prior knowledge of the principal point and to show the improvements of accuracy that are possible when using global iterative refinement.

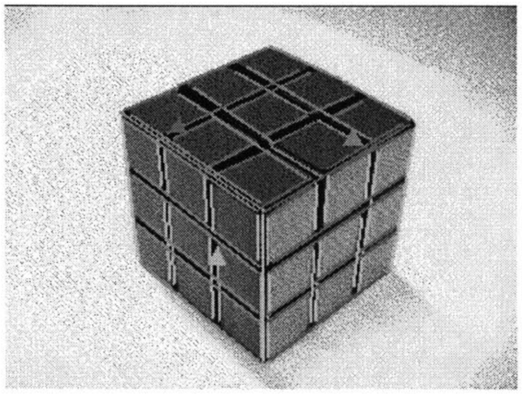


Figure 10-13: Original rubiks cube image with edges

Figure 10-13 is an image of a Rubiks cube which has been measured as being 563.6mm long on each side. Our goal is to see how well we can recover the cube's other 2 dimensions given only one from this uncalibrated 480×320 pixel image.

Following the standard approach, we do the following

- use a Canny edge detector to automatically detect lines[9]
- remove unwanted lines and draw missing lines by hand
- semi-automatically group lines into one of 3 mutually orthogonal directions
- specify an origin feature at the bottom corner
- specify a feature above the origin to be a distance of 563.6mm
- manually group lines and feature points into coplanar sets

Rubiks cube solution without an *a-priori* principal point

Tables 10.2 and 10.3 show the results of coarse and refined estimation when no prior knowledge about the camera is assumed. This are the results when using the `-fix` and `-refine` options of `sceneAnalyze`.

Note that after the two coarse-to-fine refinement stages the sum of observation residuals is 221, a 1.3 improvement over the initial coarse estimation stage of 287.

[9]Our line detection software was implemented by Stefan Agamanolis.

Stage	Sum of square errors	Iterations	tolerance
coarse estimation	294 $pixel^2$		
refinement 1.	287 $pixel^2$	76	0.1
refinement 2.	221 $pixel^2$	430	0.01

Table 10.2: Results of coarse and refined estimation using free principal point.

Feature	Estimated	Truth	Error
center of projection	186.356708;124.692576;479.272149		
cube x-size	567.2	563.6	0.745%
cube z-size	569.9	563.6	1.118%

Table 10.3: Structure errors after refinement using free principal point.

Rubiks cube solution with PP at image center

Tables 10.4 and 10.5 show the results when the `-px center` and `-py center` command-line options are used in addition to `-fix` and `-refine` in `sceneAnalyze`. These options constrain the horizontal and vertical terms of the principal point to lie at the image center during both the coarse fixing stage and during the iterative refinement stage.

Note that prior knowledge of the principal point helps improve the structure estimate: the unconstrained solution had errors of 0.745% and 1.118% for the x- and z-sizes, the image center constrained approach had errors of only 0.266% and 0.550%. It also improved on the initial coarse estimate of 675 to 356 an improvement of 1.9.

Figure 10-14 shows the final texture sampled model and estimated camera from a novel view.

Interpretation of results

This simple example shows not only that the refinement stage significantly reduces the residual observation errors left by the coarse refinement stage, but that it seems to find a solution that exceeds the theoretical limits for structure recoverability set out in section 10.2.

This seemly impossible result is in fact possible because the refinement stage embodies the sum of knowledge about the scene in terms of 1. the current estimates of camera, fea-

Stage	Sum of square errors	Iterations	tolerance
coarse estimation	675		
refinement 1.	656 $pixel^2$	35	0.1
refinement 2.	356 $pixel^2$	542	0.01

Table 10.4: Results of coarse and refined estimation while constraining principal point to image center.

Feature	Estimated	Truth	Error
center of projection	160;120;498.303894		
cube x-size	565.1	563.6	0.266%
cube z-size	566.7	563.6	0.550%

Table 10.5: Structure errors after refinement using constrained principal point.

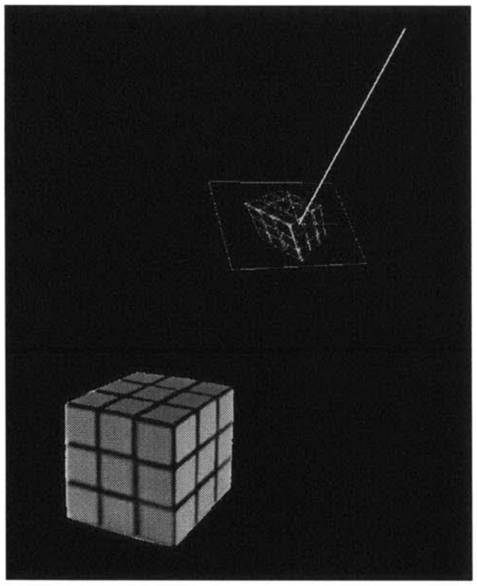


Figure 10-14: Reconstructed rubiks cube scene and camera

ture, edge and surface parameters and 2. the geometric relationships among those elements. The error limits set out in section 10.2 assume only 2-D point to 3-D feature correspondences. Determining the theoretical limits in recovery accuracy using higher level geometric relationships is difficult because it is fully dependent upon the actual configuration of the scene and cameras in question. This could be done experimentally as for lens distortion and rough scene estimation in sections 10.1 and 10.2. However, due to its inherent complexity, it is unlikely that we would be able to learn anything from it.

Further reducing reconstruction errors

Just as prior knowledge of the principal point improved our estimates for camera and scene parameters, the same can be said for any additional information we put into the system of equations.

Other strategies for reducing reconstruction errors include

- Increasing image resolution
- Taking more 3-D measurements
- Taking more 2-D observations

The first is obvious since increasing image resolution decreases the effective detector error.

We can improve the accuracy of unknown parameters by using 3-D positions of more than two features since that reduces the degrees of freedom. It is in this way that template-based camera calibration techniques are able to achieve extremely accurate results.

As pointed out by Azarbayejani and Pentland [24], as long as a multi-view estimation framework is set up so that additional 2-D observations don't add too many additional unknowns, recursive or batch refinement frameworks can be used to merge information over multiple views. The accuracy of fixed camera parameters (like distortion and center of projection) and parameters describing static scene geometry will improve as the number of independent views increases.

10.4 Multi-view experiments

In this section we briefly show some of the results of modeling full 3-D scenes from multiple uncalibrated views.

10.4.1 Attic room

This attic room is modeled from a series of 7 uncalibrated views. All lines are defined, matched and grouped manually into 6 directions and 12 surfaces. Lens distortion is considered negligible, principal point is constrained to image center (even though the images were scanned by hand from slides) and focal lengths are considered constant.

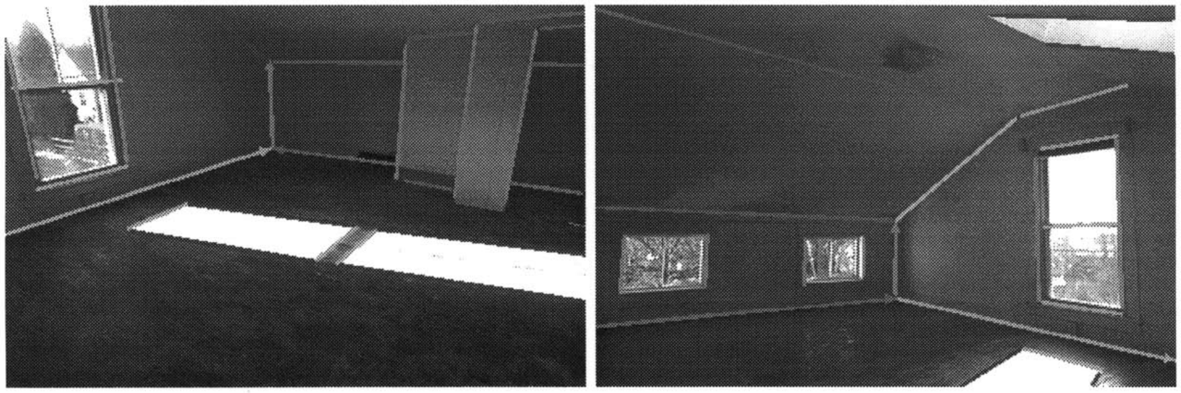


Figure 10-15: Two of the 7 original uncalibrated views with overlaid lines. Notice that lines are color-coded by direction.

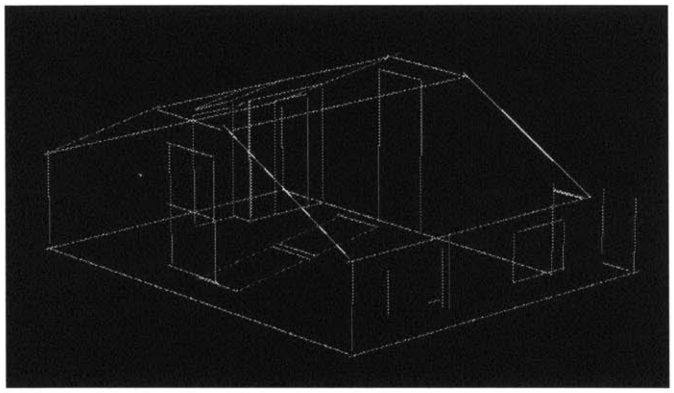


Figure 10-16: A wireframe drawing of the resulting 3-D scene geometry.

Careful review of still frames and pre-rendered or real-time animations of the reconstructed scene reveals slight problems.

- gaps at concave corners
- overhang at convex corners
- misalignment of floor texture and multiple boundaries of sunlight on floor

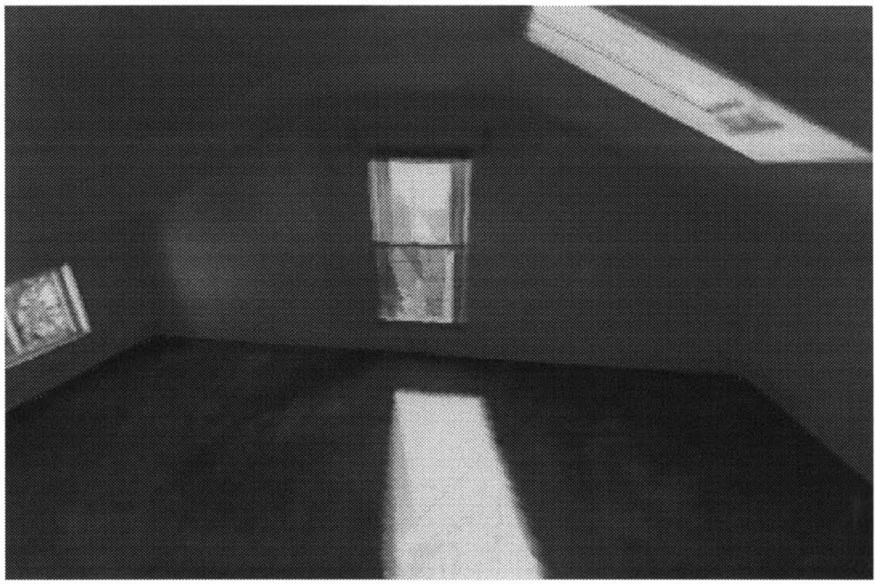


Figure 10-17: A novel view of the texture sampled geometry.

- multiple boundaries on sides of windows
- multiple boundaries on sides of skylight
- resolution discontinuities on ceiling
- color discontinuities on ceiling

Slight gaps, overhangs and texture misalignments are all due to errors in the estimated scene geometry. A major emphasis of this work is basically to minimize these sorts of errors, but we have shown that there are theoretical limitations on how well this can be achieved with a limited number of views and finite image resolution.

Many of these problems can be reduced by using higher resolution images, using more images, using pre-calibrated cameras, fully controlling the environmental lighting and taking other liberties that you can take when planning your own photo shoot as outlined in Section A.16 of the Appendix.

The multiple boundaries on the sides of the window are due to the fact that the windows are transparent and we are seeing outside objects so the view looks different from different angles. Since we model the wall and window as a single plane those different views get superimposed and we see multiple ghosts of the outside scene. This problem can be avoided by either attempting to model exterior objects as well as indoor objects, or by simply disallowing translucent objects in the scene.

Similarly, we see multiple boundaries on the sides of the skylight because we chose to model the skylight as part of the ceiling rather than model its recessed geometry explicitly. The physical world is infinitely detailed[10]. For the time being, whenever we attempt to reconstruct the physical world, we will have to choose a finite level of detail. Since we incorrectly modeled the recessed skylight as a flat part of the ceiling, that portion of the

[10]Fractal in nature.

reconstructed surface has the alpha blended composite of that region as visible from different camera positions.

Texture blending issues are an inherent property of any approach that attempts to recover a static texture description of a scene. Section 6.3 attempts to address the intensity and resolution discontinuities using resolution- and image extent-based alpha-blending to reduce view-dependent intensity changes. Debevec, Taylor and Maliks [98] elegantly avoid these problems by never attempting to recover a static texture description. Instead they use a view-dependent rendering approach which always resample texture values from those original images taken from cameras that most closely match the desired view. The two approaches offer different advantages. For our needs recovery of a static textured scene seems most appropriate.

10.4.2 Boston Public Library

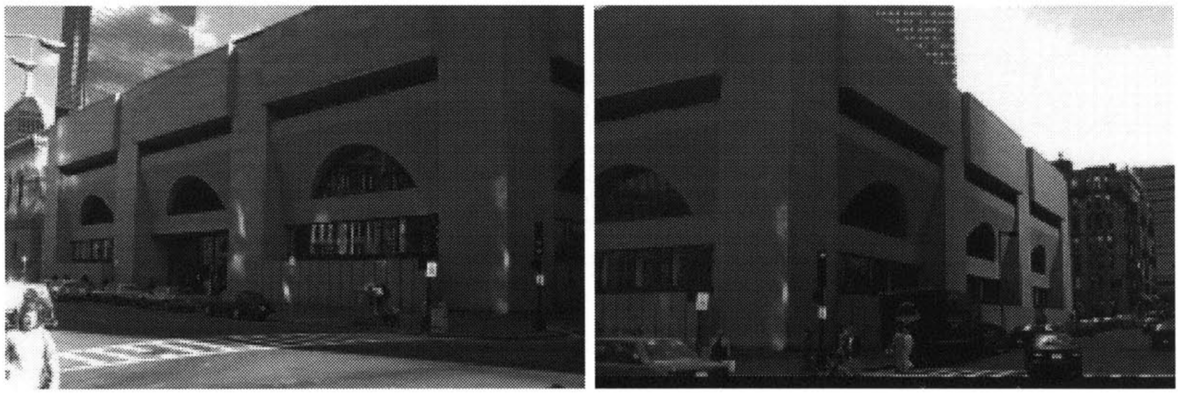


Figure 10-18: Two uncalibrated views of the Boston Public Library.

In this multi-view example we model the exterior of the new wing of the Boston Public Library from a pair of uncalibrated views shown in Figure 10-18. Notice the grossly extending edges in the wireframe view of the reconstructed model in Figure 10-19. This reveals a limitation in what is inherently our wireframe approach to scene reconstruction. When attempting to model a scene from a highly foreshortened point of view one pixel of error can lead to a huge error in reconstructed edge length. The cost function used in the refinement stage of our reconstruction algorithm refines only 3-D edge position and direction based on 2-D line position and direction. Errors in 3-D edge length and position along the infinite 3-D line are not reflected in the observation error.

Our approach fits convex texture polygons around reconstructed edges. Left alone, these sorts of edge length errors result in large errors in texture extent even though, in a wire-frame sense, the geometry is optimally aligned to the observations in all available images. One way to circumvent this problem would be to use a constructive solid geometry framework to describe texture extents rather than treat textures as individual polygons wrapped around a wireframe framework.

Another solution would be to allow constrained geometry adjustment and texture extent modeling in a 3-D setting. The current approach requires that texture polygons be defined in `sceneBuild`'s restrictive 2-D setting. Integrating the 2-D editing capabilities of `sceneBuild` with the 3-D viewing capabilities of `sceneView` into an integrated modeling environment makes sense so a user could, for example, have the ability to interactively

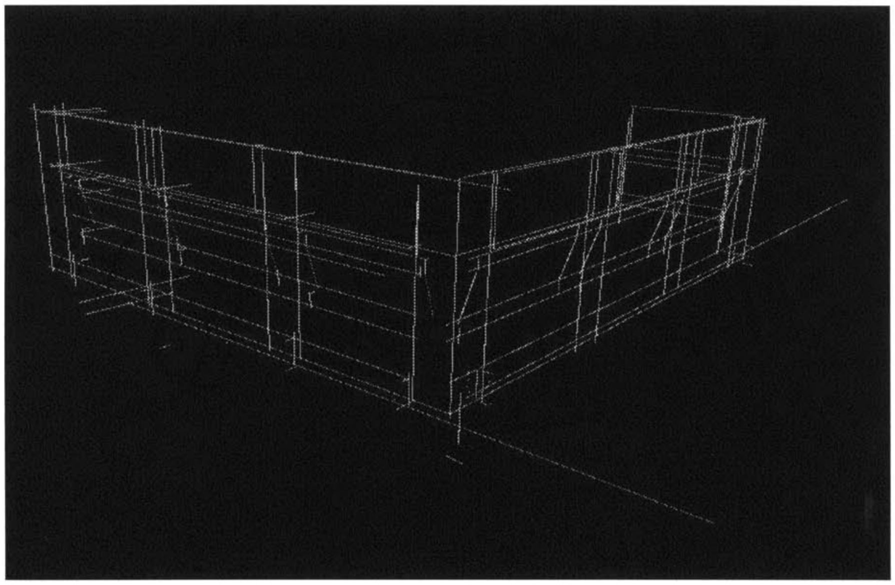


Figure 10-19: A novel wireframe view of the Boston Public Library model. Note erroneous edge lengths.

adjust 3-D edge geometry while maintaining (or at least make it more difficult to deviate from) 2-D observation constraints. It would also be useful to allow one to stretch surfaces around wireframe geometry and see the contributing image textures alpha-composited in real time. These GUI-intensive capabilities would be possible given graphics hardware with sufficient rendering speed.

These automatic approaches would be convenient, but in the meantime, we can address the edge clipping problem by manually shortening lines of edges in their respective 2-D views. The resulting texture model[11] is shown from novel views in Figures 10-20 and 10-21.

[11]Notice that the top corner is cut off since that texture is missing from the original views.

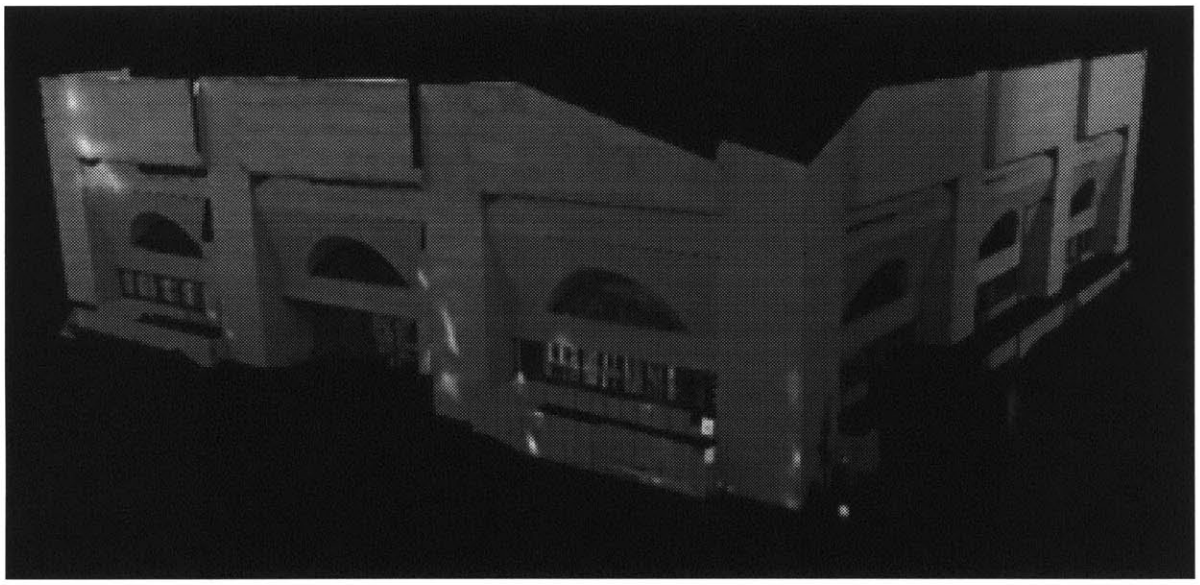


Figure 10-20: A novel textured view of the Boston Public Library model.

Figure 10-21: A close-up view of the textured Boston Public Library model.

10.5 User performance testing

To get a quantitative measure of what kind of modeling throughput can be expected when using the `sceneBuild` package this author kept a careful event log over the course of working on several modeling projects. A total of 35.5 hours was spent on a total of 8 projects, each project aimed at recovering a 3-D scene from a single high resolution scanned photograph with unknown COP and negligible lens distortion.

We should preface this review of user performance results by pointing out that modeling times vary due to many factors, most influential of which are edge clarity, scene complexity as well as the operator's spatial reasoning ability and experience with the system. The images used for these projects were typical indoor kitchen scenes like the one being processed in Figure7-1.

no. hi-res projects	avg img size in pixels	avg no. elements	avg no. surfaces	avg time hrs:min	avg throughput	
					ele/hr	srf/hr
8	1308x1364	767	46	4:30	173	10

Table 10.6: Averages over all 8 kitchen projects.

As seen in Table 10.6 the average scene had 767 elements of which 46 were surfaces. With an average modeling time of 4.5 hours the average throughput was about 173 elements per hour, 10 elements of which were surfaces. The times spent[12] are longer than normally spent processing video resolution images since this project used large images and required subpixel reconstruction accuracy.

project	image size in pixels	total no. elements	total no. surfaces	total time hrs:min	snag type
ImageG1	1207x1480	626	42	4:30	snag1
ImageG2	1486x1192	812	52	5:30	
ImageI1	1209x1469	897	52	7:30	snag2, snag3
ImageI2	1486x1197	821	42	3:30	
ImageL1	1210x1480	750	51	3:00	
ImageL2	1177x1445	955	59	5:30	snag2
ImageU1	1200x1439	754	48	4:00	snag4
ImageU2	1492x1213	526	24	2:00	

Table 10.7: Modeling project totals.

[12]The only activities counted toward length of time are: modeling time (i.e. time spent drawing, altering, and grouping points, lines and polygons), machine processing time, and model debugging time (i.e. time spent tracking down modeling failures either studying the intermediate 3-D visualization results for `sceneAnalyze` or reviewing its text output in debug mode). Time spent toward program coding, debugging or documentation are not counted toward elapsed time.

project	points	lines	polys	ftrs	dirs	edges	surfs	textures
ImageG1	236	115	37	50	4	110	42	32
ImageG2	302	160	37	67	6	153	52	35
ImageI1	325	183	39	84	5	177	52	32
ImageI2	343	160	40	53	10	148	42	25
ImageL1	258	156	43	48	6	152	51	36
ImageL2	382	165	57	88	4	160	59	40
ImageU1	291	138	41	66	8	129	48	33
ImageU2	223	80	24	75	4	76	24	20

Table 10.8: Tallies of element types created for each hi-res modeling project.

snag type	description
snag1	made incorrect feature-surface attachments
snag2	made incorrect edge-surface attachments
snag3	wasted time due to over-estimating the program's ability to solve small edge details
snag4	wasted time trying to find a way to overcome positional inaccuracies when projecting features on a glancing plane

Table 10.9: Different types of conceptual snags.

10.6 Shortcomings of the approach

The demands of correctly specifying 3-D constraints among 2-D elements are often conceptually difficult. For example, when modeling a scene from a single image one might incorrectly assume that some image elements are or are not coplanar. While a valid 3-D origami model might be recoverable, and have near zero reprojection error from the estimated camera, it will have inconsistencies that are apparent when viewing the solved model from another point of view. When such conceptual problems or "snags" arise, we are forced to suspend the normal flow of modeling to determine the cause of the error.

As shown in Tables 10.7 and 10.9 the frequency and type of conceptual snags were tracked. A total of 5 conceptually difficult snags were encountered throughout the 8 modeling projects. Of the different snag types the first two were errors in point or line grouping. The other two snag types are probably due to this author's own possibly unreasonable expectations for 3-D accuracy given the provided views. The limits of recoverability are bounded by the projective arrangement of the scene.

10.7 Comparison to related work

To get an idea of the caliber of this work we compare it to other recently publicized efforts in vision-assisted modeling.

Of all related bodies of work, Debevec, Taylor and Malik's hybrid modeling approach[98] at Berkeley is the most impressive. Using OpenGL they have made an interactive modeler that allows manual construction of a scene as parameterized 3-D primitives (i.e. rectangular solids, cylinders and surfaces of revolution). Positioning constraints among objects can be defined by hand (i.e. two objects be constrained to have abutting faces and collinear axes of revolution). Given precalibrated cameras and approximately known camera position their environment also lets a user match 2-D lines drawn on an image to 3-D lines in the preconstructed 3-D model. They then use a powerful algebreic refinement strategy to adjust camera and object parameters until edge observation errors are minimized. Surface textures are computed on the fly by a view-dependent warping approach that averages intensities between original views. They also show how surface displacement maps can be found using a stereo matching approach.

Professional photogrammetric packages (such as PhotoModeler[127]) use point correspondences in multiple views taken with precalibrated cameras with approximately known positions. Surface textures are found by mapping from the most straight-on view.

Kumar and Hanson [66] use 3 or more calibrated camera's with known relative position and hand selected and matched line segments to produce a 3-D wire-frame model. Their work seems to focus soley on wireframe recovery techniques rather than delving into the difficulties associated with modeling textures.

Azarbayejani's [110] recursive estimation approach tracks point features through sequential frames of video. The approach is fully self calibrating (except for distortion and principal point). Surface geometry and textures are found by fitting and wrapping manually selected points. He has adapted his approach to focus on applications of high precision stereo camera calibration and fast real-time blob tracking but has not continued toward recovery of textured 3-D models from images.

In contrast to these approaches, our modeling approach is entirely 2-D in nature and is based only on assigning group memberships to lines and points rather than using a preconstructed 3-D model. Like Azarbayejani, we focus on the combined problem of camera calibration, pose and structure recovery, but in addition address the problems of lens distortion and principal point recovery.

Like Kumar and Hanson we use lines which are detected or manually drawn. We can then use explicit line matchings among views but our approach of manually grouping lines into parallel and coplanar sets implicitly performs most of the line matching, therefore explicit line matching is not strictly required.

Like most other texture recovery techniques we use an iterative scheme to refine line observation errors and then undo the perspective mapping on images to create a static texture map for each surface. It is not known if other approaches address issues of resolution and intensity variation. We perform this using an alpha-blending composition rule.

Our approach is fairly unique in that it imposes the requirement of having an origami scene containing at least 3 parallel sets of mutually orthogonal lines. While this is admittedly restrictive and prone to be sensitive to detection errors this gives us the unusual ability to get a rough estimate for camera pose and surface geometries even from a single view.

Chapter 11

Model-based video coding framework and algorithms

Having provided a means for manually recovering a 3-D scene model from one or more frames of video of a scene, one application goal is to use this model as the basis for representing a real video sequence of that scene.

To do this, we present the framework for model-based video coding and decoding diagrammed in Figure 11-1.

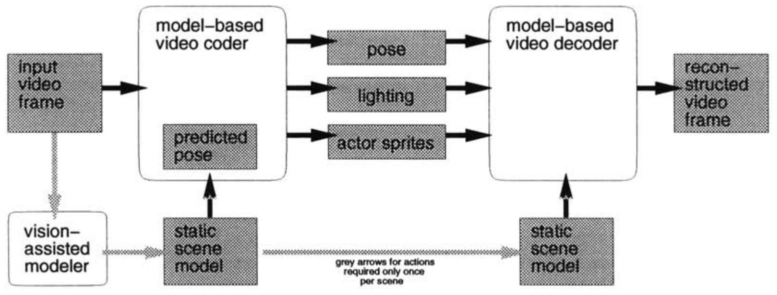


Figure 11-1: Framework for a model-based video codec using a textured 3-D scene model

11.1 Model-based video coder framework

The coder consists of analysis-synthesis stages. A model-based camera tracker (detailed in section 11.2) analyzes an incoming frame of video to determine the camera pose that brings that frame into registration with a static scene model. Texture rendering routines (described in chapter 9) are used to synthesize the background image as predicted by the optimal camera pose. Differences between the predicted background and the actual frame are then analyzed and segmented into coherent actor sprite regions and a global lighting parameter (see section 11.3).

11.2 Robust camera pose estimation using textured patches

The camera tracker (or pose estimator) is the key component of the model-based video coder. Given a static scene model and a predicted camera pose the 2-D image projections of three

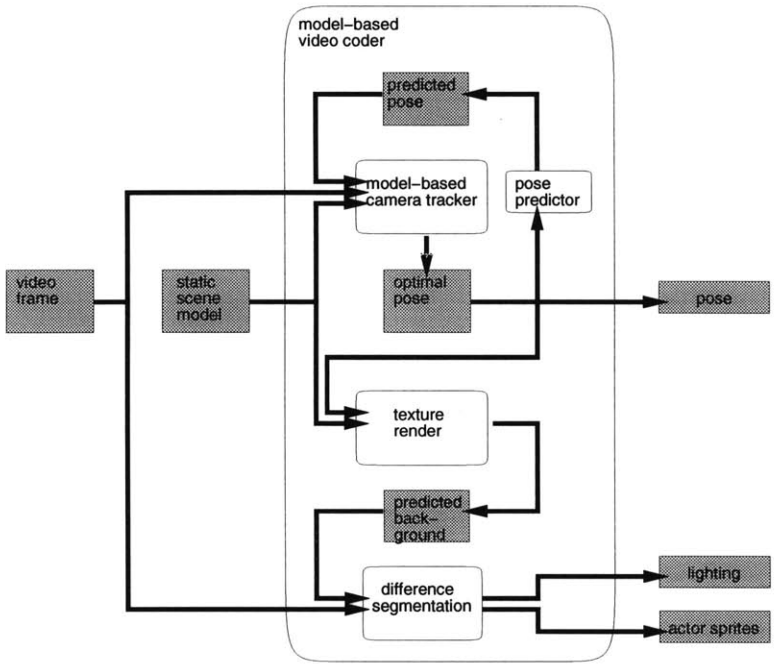


Figure 11-2: Framework for a model-based video codec using a textured 3-D scene model

or more 3-D model features can be automatically tracked in frames of live or prerecorded video. This set of 2-D to 3-D point correspondences can then be used to determine the camera's pose in each frame of video.

Since our scene model consists primarily of textured surfaces, we try to exploit that knowledge to improve the robustness of feature matching in being able to successfully reject outlier matches. This will help us find the solution for camera pose that brings the model into maximum alignment with the observed data.

The following texture patch tracking algorithm is intended for use with prerecorded video sequences of a static scene where a 3-D model is built using the first frame of the sequence. Subsection 11.2.3 discusses how this algorithm has been generalized for live video applications.

Texture patch tracking algorithm

- A trackable feature is a known 3-D point which belongs to a single surface that has a texture image. It has been tagged as having strong bi-directional spatiointensity gradients thus making it viable for tracking.

- Rather than storing all surface texture images in the pose tracker, we store only the texture patches surrounding each feature. Crop out a 32x32 rectangular portion of a texture image and store it as a texture patch for a trackable feature.

- Given a predicted camera use anti-aliased *planar texture rendering* to predict the way the planar patch should appear from that camera. Treat positions outside the patch extent as transparent treating RGB color 0,0,0 as transparent and put the *predicted patch* rendering into a 16x16 matching template.

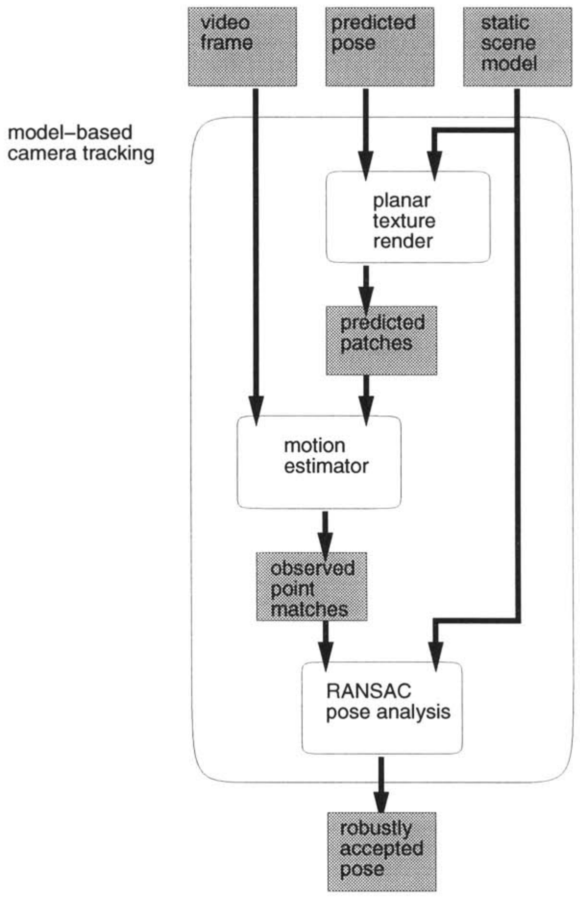


Figure 11-3: Robust camera pose estimation using textured patches

- Take a new frame of video and the matching template which has been loaded with the predicted planar patch and do *motion estimation.* Perform an 8x8 full search to find the horizontal and vertical pixel shift (the motion vector) that results in the minimum *matching cost.*

- Define *matching cost* of a particular pixel shift as the minimum sum of absolute intensity differences over pixels that are opaque in both the video frame and the matching template, normalized by the number of opaque pixels. If the number of opaque pixels falls below a prespecified threshold (40% of the template area) impose a high disabling penalty.

- Given the pixel shift of minimum cost for all opaque pixels find the distribution of the intensity differences between the video frame and the best shifted kernel. Define *quality of match* as the sample variance of this distribution.

- Consider an *observed 3D-to-2D point match* to be the following 3 items of data: 3-D model coordinates of a trackable feature, the new image position of that feature as tracked by the motion estimator and the quality of that match.

- Given all *observed 3D-to-2D point match pairs* use a random sample and consensus (RANSAC [63]) approach to robustly find the *accepted camera pose* which best explains the projected 2-D motion of all 3-D features.

- Given the 2-D projected coordinates of three known 3-D features in an image with known COP use Grunert's approach as outlined by Haralick[64] to find up to 4 possible real solutions of camera pose.
- Note that when a correct real pose solution is found, then the set of all 3-D features as projected by that pose should agree well with most observed points.
- Index all 3-way combinations of point match pairs and sort them according to decreasing summed *quality of match.*
- For each 3-combination find up to 4 hypothetical solutions for camera pose. Take each hypothetical camera pose and project all 3-D features into hypothetical 2-D image coordinates. Tally votes of evidence for that 3-combination's hypothetical pose by counting the number of hypothetical 2-D points that lie within some radius of observed 2-D points (as returned by motion estimation).
- Once the votes for all hypothetical poses of the preferred 3-combinations are taken poses are clustered according to pose-radius, i.e. all poses within some *pose-distance* are clustered into the same "voting district". 3-combinations which have a hypothetical pose belonging to the district with the most votes are considered to be robustly accepted.
- Robustly accepted pose votes are combined by weighted average where the contribution of each pose is according to its own *quality of match.* The mean pose is output by the camera tracking module as the *robustly accepted pose.* The covariance of this district is used in the difference segmentation stage to describe the precision of the pose estimate.

Calculating *pose-distance*

The texture patch tracking algorithm is responsible for grouping hypothetical camera poses which are similar. Recall that camera pose refers to the parameters which describe the world-relative rotation and translation of a camera. This is the world-relative film frame ${}^{w}_{f}\mathbf{B}$ from equation (3.1). In cases where we need to compare cameras we need to define a metric for similarity which we call *pose-distance.*

We first represent each pose as a set of 4 points, three positioned at unit distances along each principal axis and one at the camera's origin. We then define the scalar *pose-distance* $d_{i,j}$ between cameras i and j as the sum of squared differences between each camera's corresponding representative point.

$$d_{i,j} = \sum_{n=1}^{4} (\mathbf{p}_{i,n} - \mathbf{p}_{i,n})^T(\mathbf{p}_{j,n} - \mathbf{p}_{j,n}) \tag{11.1}$$

This method for comparing poses of two cameras is sensitive to differences caused by both rotation and translation and a measure of zero defines pose equality.

Note that the columns ${}^{w}_{f}\mathbf{B}$ consist of the world-relative principal vectors and the origin of a camera's film plane. Therefore, 11.1 can be rewritten somewhat more conveniently as

$$d_{i,j} = \mathrm{trace}[\mathrm{diag}({}^{w}_{f}\mathbf{B}_{i} - {}^{w}_{f}\mathbf{B}_{j})({}^{w}_{f}\mathbf{B}_{i} - {}^{w}_{f}\mathbf{B}_{j})^T] \tag{11.2}$$

11.2.1 Calculating average pose and pose covariance

Given several similar camera poses, each with its own preferential weighting, we need to find the mean pose that minimizes the weighted sum of pose-distances. To do this we cannot simply solve the weighted mean coordinate frame ${}^{w}_{f}\mathbf{B}$ because with its 12 degrees of freedom the problem would be underconstrained and it would be difficult to renormalize the matrix to describe a valid pose. We instead recast the problem describing each pose as a 7-parameter vector ${}^{w}_{f}C = [\mathbf{r}^T\mathbf{t}^T]^T$ where $\mathbf{r}$ is the 4×1 quaternion describing the orthonormal rotation component and $\mathbf{t}$ is the 3×1 translation vector. We constrain the rotation quaternion to unit magnitude, thus allowing the representation only 6 degrees of freedom.

As pointed out by Paul[116] for small differential transformations, the order of differential rotations does not matter. By the same argument, each rotation is independent, as is translation. Coupled with the fact that the weighting for each pose is scalar we can treat each parameter as an independent random variable and find the weighted sample mean and sample variance of each parameter separately. To guarantee that the final rotation is orthonormal we simply normalize the quaternion before converting into matrix form.

This 7-parameter approach seems to be an effective approximation useful for path smoothing, interpolation and extrapolation.

11.2.2 Advantages of using rendered patches for camera tracking

Transparency

We can confidently claim that the given approach for model-based pose estimation has several strong points. First of all, these texture patches are modeled as being transparent wherever they fall off the defined surface extent. The use of transparency lets us avoid many problems of occlusion. For example, given a feature on the edge of a building we can ignore the moving sky and trees in the background and concentrate soley on matching brick texture.

Quality of match

Due to occlusion, specularity, shadow, or shot noise, the motion estimation error surface might have several widely separated good matches within the search space. If the error surface is not unimodal we may want to treat all minima as potential matches, or we may instead choose to low-pass filter the template and source and try again. A simpler approach, and the one we use, is to simply assign each match a confidence factor and use that to weight its contribution to the final solution. It makes sense to define the quality of match confidence factor to be related to the variance of the pixel difference distribution because minimum variance is a measure of uniqueness (e.g. consider the delta function's relevance to autocorrelation). It allows us to assign greater trust to those matches which we are not confused about.

Link between model geometry and model texture

An earlier real-time version of the tracker was implemented which loaded the matching template with pixels from the previous frame rather than using a re-rendered patch. Unsurprisingly, this turns out to be problematic because small inevitable frame-to-frame pose estimation errors integrate over time to become noticeably large drift. After 30 or so frames the known 3-D features no longer map into the regions covered by their matching templates and we start tracking something else entirely. By using known model texture at each frame we are better able to guarantee to keep the data and model in alignment over the long haul.

Another solution for the drifting registration problem of the earlier real-time version of the tracker would have been to always use the original known pose frame as source data for the matching templates. Of course, this would result in matching problems when the view of the patch became rotated, scaled or foreshortened. We could counter that we could use the predicted camera and apply an affine or projective prewarp to the template. In effect, this is exactly what we do with patch-based feature tracking, with the addition of properly handling transparency according to the known model.

The importance of anti-aliasing vs. pose prediction

One suprising discovery was the overriding importance of preserving quality of the predicted image versus preserving quality of the predicted camera pose. Camera tracking with continuous video works rather well with zero-order sampling in pose, i.e. using the previous camera pose as the prediction for the next frame. However, the use of zero-order sampling to render each feature's predicted planar patch seriously affects the ability to track small interframe camera motion.

In retrospect, this should not be so suprising since aliasing introduces structured high-frequency noise and it is well known that even Gaussian white noise degrades motion estimation results. To prevent aliasing in the patch prediction we do stochastic texture sampling as described in 4.1.5 by casting a pencil of rays which are normally distributed about each pixel center onto the feature's surface. Intensities at each subpel locations in the surface texture are bilinearly interpolated and the entire set of intensities is averaged. The number of rays is related to the amount of surface foreshortening as viewed from that pixel's center. Without hardware texture rendering capability this limits implementation of the algorithm to purely off-line batch processing.

We can treat the camera path as if it were an object with mass moving with continuous position, velocity and acceleration. However, the order of smoothness used to predict the next camera pose seems to unimportant so long as the region of the predicted template is at least partially engaged to the actual feature to be tracked. Smooth pose prediction is done by: fitting a cubic to the previous 4 time estimates in each of the 7 pose parameters (3 for translation and 4 for the rotation quaternion); using those spline parameters to separably predict those parameters at the next time step; and finally renormalizing the quaternion. For the 4 initial frames the previous pose is used as the prediction for the next.

11.2.3 Doing pose estimation with live video in a known scene

To handle pose estimation in a live video situation we initialize the pose to be at some prespecified known origin and suspend the pose prediction stage and "search" until the matching quality of some percentage of features passes a predetermined threshold. Tracking for the moving camera will not "lock" until a recognized pose has been found. Once "locked" tracking will resume until the matching quality of enough features drops below some critical threshold, at which point we fall back into the "search" mode again.

11.3 Difference segmentation

The goal of the difference segmentation stage is to try to separate different contributions of residual differences between the predicted model rendering and a real frame of video.

As reviewed in section A.17 of the Appendix there are many sources for these errors. We will attempt to differentiate between errors caused by pose registration error and those differences caused by occlusion from unmodeled elements like actors.

11.3.1 Describing expected background YUV

To do this we first gather a statistical description of a pixel's expected background YUV color. Pixel intensities from the synthetic background are processed to find the expected mean YUV value $\mathbf{m}$ of at a given (x, y) background pixel. The expected YUV covariance $\mathbf{K}$ at that position defines the degree of deviation we expect to find for an observed pixel to be considered a background pixel. The color mean and covariance at each pixel in the background are used to measure the likelihood that a new observed pixel belongs to the background. If the likelihood falls below some threshold then that position is marked as an actor pixel.

To relate an observed (x, y) pixel $\hat{\mathbf{y}}$ from the incoming frame to our expected background YUV we say

$$\hat{\mathbf{y}} = \mathbf{m} + \mathbf{I}_g + \mathbf{v} \tag{11.3}$$

where $\mathbf{v} \sim \mathrm{N}(0, \mathbf{K})$ describes uncertainty in our prediction for mean YUV and $\mathbf{I}_g$ describes an unknown global YUV illumination bias.

The likelihood that a newly observed pixel $\hat{\mathbf{y}}$ belongs to the background can be found by appropriately thresholding its log likelihood

$$d = -(\hat{\mathbf{y}} - \mathbf{m})^T \mathbf{K}^{-1} (\hat{\mathbf{y}} - \mathbf{m}) - \ln \det \mathbf{K} \tag{11.4}$$

11.3.2 Calculating background YUV mean and variance

To estimate the expected YUV color mean $\mathbf{m}$ and variance $\mathbf{K}$ at pixel (x, y) we reason that the camera tracking stage is able to achieve only limited accuracy in estimating camera pose. As described in section 11.2.1 this inaccuracy is described by the pose error covariance. One result of this uncertainty in pose is that the projected location of each feature or surface coordinate in the model is correct only within a *circle of confusion* which is related to it's position relative to the camera. Therefore, taking erroneous camera pose into account, the expected mean color $\mathbf{m}$ of a given (x, y) pixel in the background, is the sample mean of all YUV pixels within that pixel's *circle of confusion.* The full YUV covariance $\mathbf{K}$ for that pixel is the sample covariance of all YUV in that circle.

Since expected YUV mean and covariance need to be computed at each frame, we simplify the problem by first choosing a constant circle of confusion for the entire frame. The sample mean $\mathbf{m}$ at each pixel is then the result of convolving the image with a low-pass Gaussian filter. To approximate color variance $\mathbf{K}$ we can reason that variance should be lowest in regions that have low spatial texture frequency. Conversely, differences should be highest near high-frequency intensity edges. It follows then, that the sum of squares of x-y separable derivatives of Guassian filters which both smooth and enhance edge magnitudes can be applied to the synthetic background image and used as an approximation to each YUV channel's expected intensity variance.

Chapter 12

Model-based video coding implementation and results

12.1 `sceneTrack`

Our model-based video coder is implemented as an application called `sceneTrack`. Its command line parameters include `sceneDatabase` file that describes a previously modeled scene, a video sequence of the static scene in raw RGB datfile format, an input camera file describing the predicted camera of the first frame and the name of the output camera file to store the tracking results.

During scene modeling preselected surface point features are built into the scene model and marked as trackable (using `sceneBuild`'s "features-set-trackable" command).

`sceneTrack` saves texture patches for each "trackable" feature which belongs to a single textured surface. As each new frame is encountered each patch is stochastically rendered according to the predicted camera pose and used as the template for local block motion estimation. A camera pose is then found which optimally aligns known 3-D features to the newly tracked 2-D observations. The following sections show the results of using `sceneTrack` on synthetic and real video input.

12.2 Synthetic experiments

12.2.1 UROP office

This synthetic experiment illustrates our tracking algorithm's ability to automatically determine camera pose using real textures under controlled situations. We first recover a static scene model using `sceneBuild` and `sceneAnalyze` to specify and group lines on a single uncalibrated frame of video as shown in Figure 12-1. Novel wireframe and texture mapped views of the constructed model are shown in Figure 12-2.

We then use `sceneView` to create a synthetic video stream by rendering the recovered model from a set of absolutely known camera views along a bicubic interpolated path.

By using a synthetically generated video source we are able to conduct an experiment where we avoid view-dependent intensity variations like specularity that are common in images of real scenes. More importantly, it lets us guarantee that a camera pose exists which can perfectly describe a given view. Resolution limitations on modeling accuracy make this impossible under real conditions.

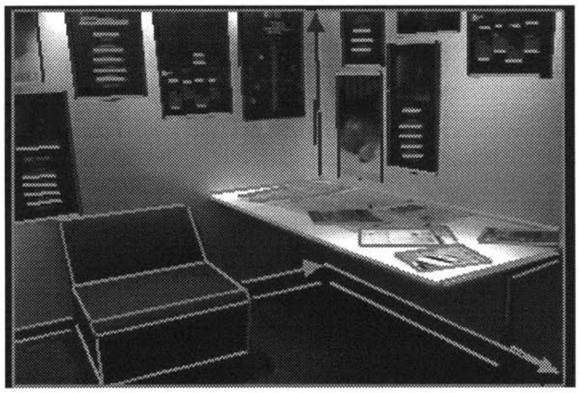


Figure 12-1: One frame of video is used to construct a model.

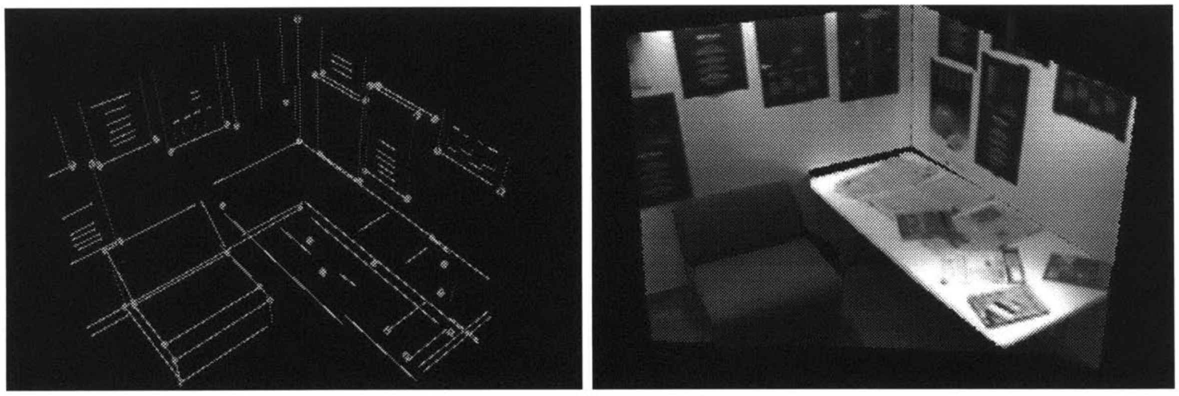


Figure 12-2: Two novel views of the textured 3-D model.

We then attempt to use the tracking algorithm to automatically track texture patches and determine camera pose. Figures 12-3(a)-(c) show the source, model and error for one frame while attempted to track camera pose from a synthetic video sequence. Since the camera path is known absolutely we can compare the real and estimated paths directly as shown in Figure 12-4.

Figure 12-5 shows that this approach is somewhat viable as a video coding technique under these special conditions. The dotted line in figure 12-5 describes RMS pixel errors when using the previous frame to predict the current frame. The solid line shows residual RMS errors when using our model-based video coding technique. Note that our approach gives lower aggregate RMS values then using DPCM (delta pulse code modulation) for lossless transmission.

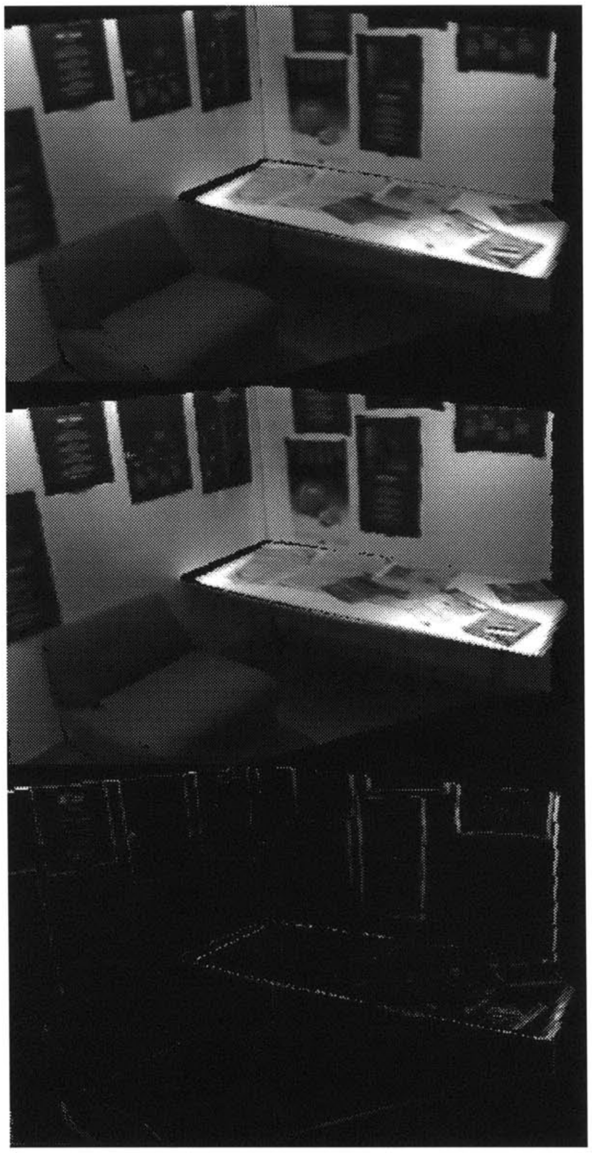


Figure 12-3: Camera tracking of a real-textured model over a synthetic camera path. (a) A synthetic input frame. (b) The model rendered at the detected pose. (c) The coder error for one frame.

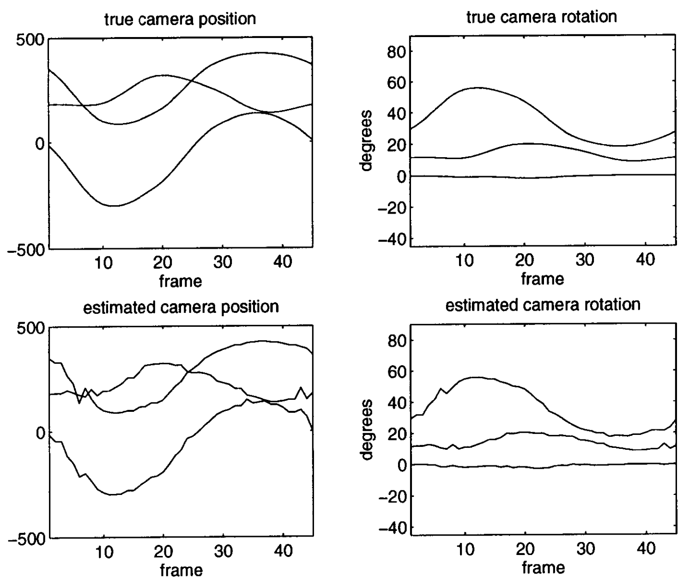


Figure 12-4: Real camera position (a) and rotation (b). Recovered camera position (c) and rotation (d).

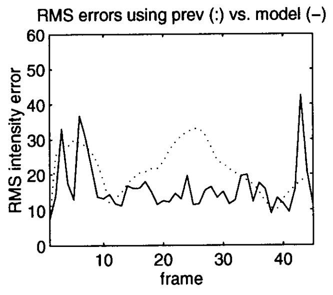


Figure 12-5: Comparing RMS pixel intensity errors when using the previous true frame to predict the current frame versus using our model-based prediction.

12.3 Real data results

12.3.1 UROP office sequence

We take a 50 frame sequence taken with a hand-held uncalibrated video camera in an office setting. The first frame of this sequence is the one that was used to create the model used in section 12.2.1. Rather than using a synthetically generated video sequence, however, here we attempt to find camera poses that bring frames from the real video into best alignment with the rendered model.

Figure 12-6 shows `sceneTrack` in action. The left image shows the original image overlaid with bordered templates that contain texture patches which have been perspective warped according to the predicted camera. These templates are used for block-wise motion estimation. Green boundaries denote high levels of matching confidence while red boundaries indicate low levels of confidence. The right image shows the wireframe model viewed from the estimated pose. Features are shown as 3-D cubes again showing confidence levels as colors ranging from green to red.

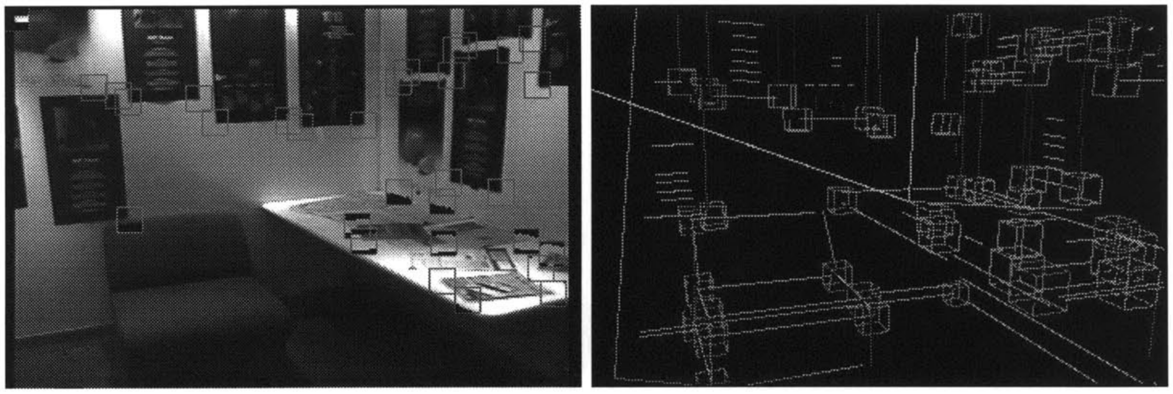


Figure 12-6: Here we see the feature-patch tracking algorithm in action. The left image shows the original image with matching warped patches. The right image shows the results of computing a new camera pose.

Figures 12-7(a)-(c) show the source, model and error for one frame while attempted to track camera pose from a synthetic video sequence. The estimated camera positions and rotations are shown in Figure 12-8 and residual RMS pixel errors are shown in Figure 12-9.

Note that the tracking algorithm does well until about frame 32 where pose estimation fails and is then unable to recover. The reason for this is due to the fact that the actual camera, at this point, is far from the starting pose which was used to model the static textures. As the camera moves towards the right texture detail on the table appears to elongate towards the position of the first frame. The predicted view then differs so much from the observed frame that patch matching is spurious.

As shown in the next example, this problem can be averted by modeling the scene from a range of views that span the expected camera path thereby guaranteeing a better description of expected texture throughout.

12.3.2 Alley sequence

Figure 12-10 shows two views from a hand-held uncalibrated video sequence of an outdoor street scene. They were used to create the model viewed in Figure 12-11.

Figures 12-12 show one frame of the input video sequence and the resulting residual error. Note that residuals are high around the edges of the frame where model surfaces are not visible in the Figures of 12-10.

Figures 12-13 and 12-14 show the resulting camera path and residual RMS pixel errors.

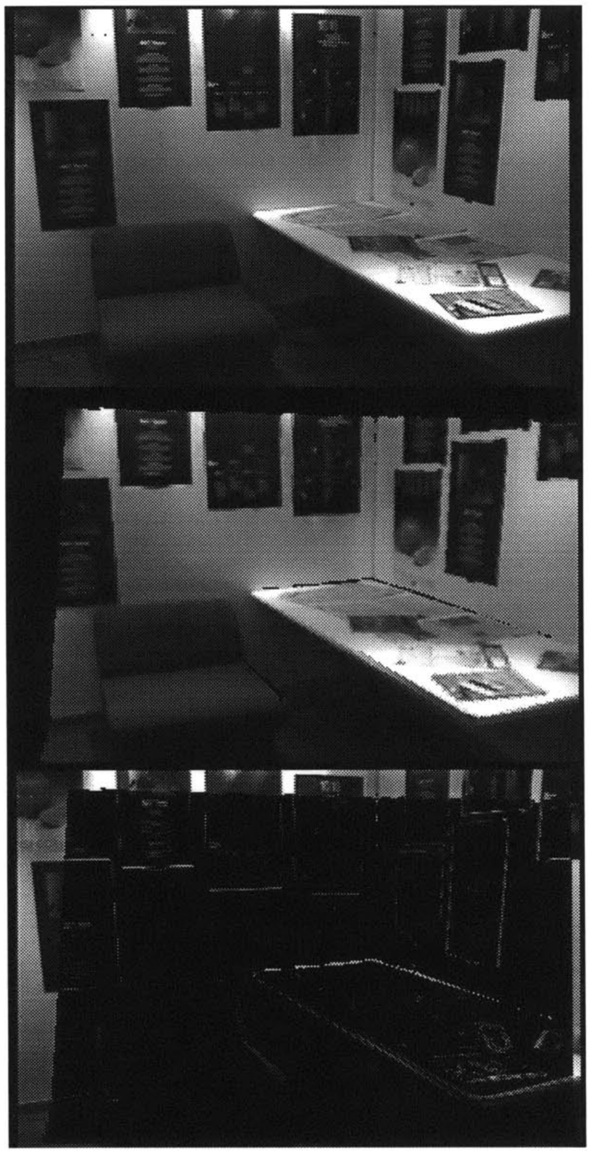


Figure 12-7: Camera tracking of a real-textured model over a real video sequence

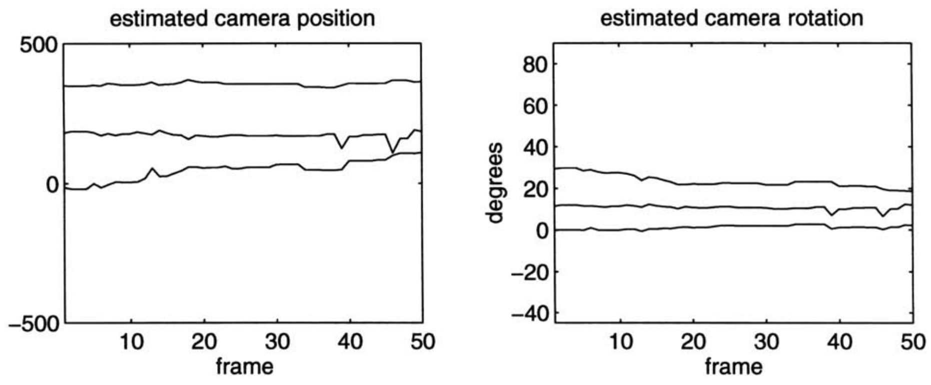


Figure 12-8: The estimated camera path.

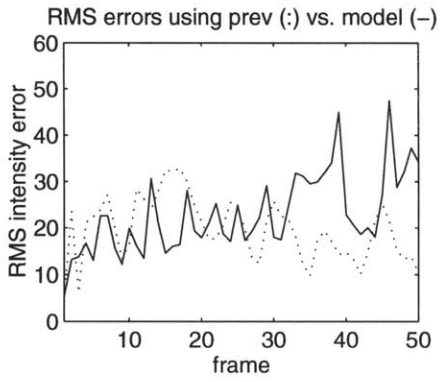


Figure 12-9: Residual RMS pixel errors using the previous frame(:) vs. using the model(-).

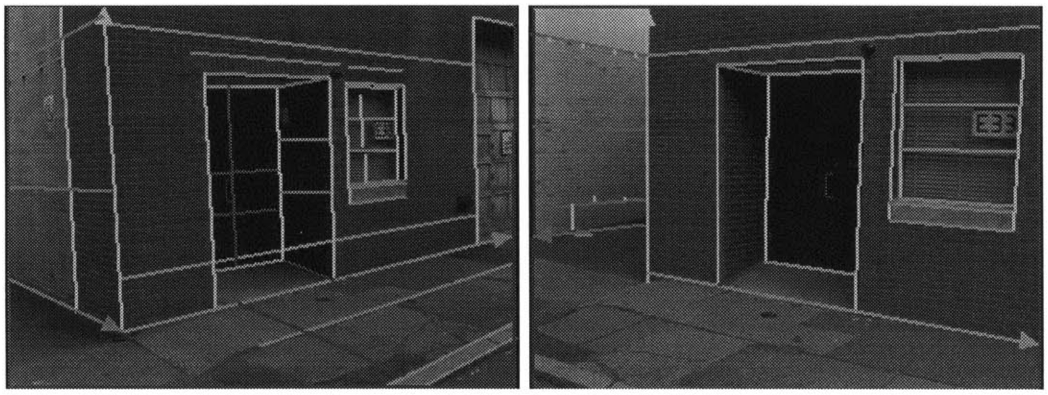


Figure 12-10: Two frames from the original video sequence used to construct the model.

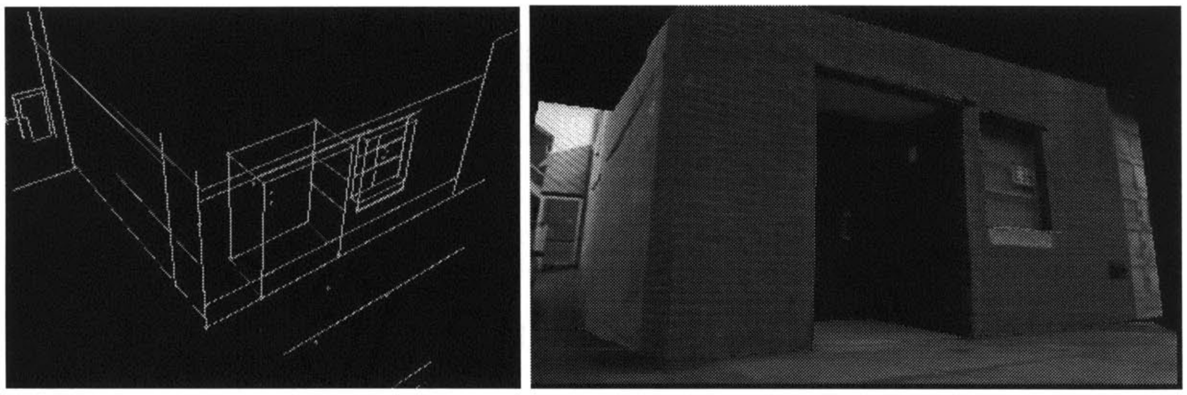


Figure 12-11: Two novel views of the textured 3-D model which was made using the two original frames.

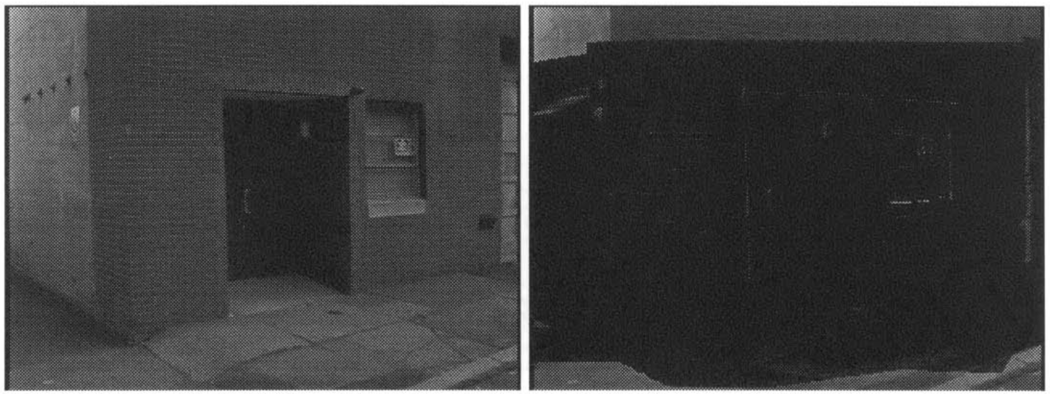


Figure 12-12: Example of one observed frame and prediction error.

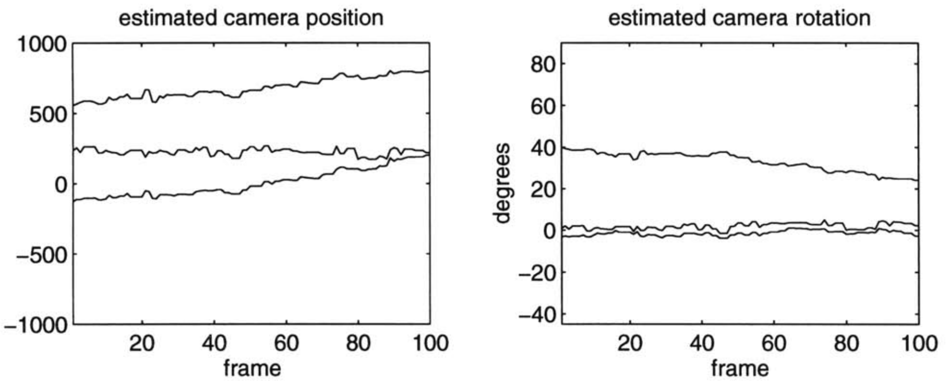


Figure 12-13: The estimated camera path.

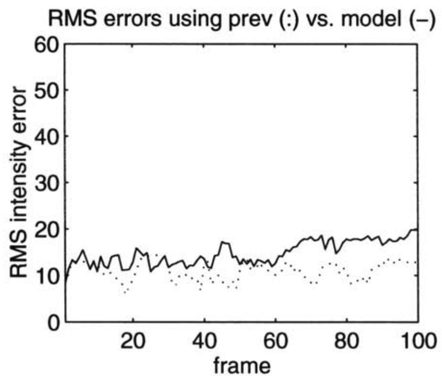


Figure 12-14: RMS pixel errors using the previous frame(:) vs. using the model(-).

Chapter 13

Conclusion

We have presented a modeling approach which exploits the proven effectiveness of vision techniques where possible and relies on human guidance where necessary to create detailed scene descriptions from random sets of uncalibrated views. Special care has been taken to rely upon only a modicum of user assistance in the form of grouping relevant features. We impose the requirement of having an origami scene containing at least 3 parallel sets of mutually orthogonal lines. While this is admittedly restrictive, and prone to be sensitive to detection errors when only a few lines are available, this approach gives us the unusual ability to find a rough estimate for camera pose and surface geometries even from a single view. Linear regression techniques are then used to refine all free camera and scene parameters to minimize discrepancies between the model and observations.

Only the most primitive of scenes are well approximated as polyhedral surfaces. Useful extensions to this work include supporting surfaces of revolution from contours and using stereo approaches to find surface height variations to eliminate texture disparities. Given approximating scene geometry, we should have enough information to determine and correct for the environmental lighting effects by eliminating shadows and shading effects from the scene model. The resulting lighting invariant representation would be suitable for realistic viewing under any lighting conditions.

The use of these models for the application of model-based video coding shows definite promise but in order to progress work needs to continue on several fronts.

For the problem of model-based video with actors we are still faced with a "chicken-and-the-egg" problem: we are not able to do actor segmentation until the background is accurately known, finding the background requires having an accurate camera pose, but pose detection is hampered because we can't segment out the actor.

The goal of full-model based video coding will be best served by continued research for segmentation of foreground material given only approximately known background. The complexity and completeness of the scene model also needs to improve so we can more closely approximate incoming frames of video. In addition, we need to find improved methods for recognizing camera poses which optimally align detail-limited models with fully detailed observations.

The ability to extract 3-D geometry from one or more uncalibrated views will find useful application in many visually related industries. To get the maximum benefit of this work, it would be particularly useful to introduce these modeling techniques into existing high quality computer aided design (CAD) products. It might also be useful to develop new specialized commercial software products that focus on specific modeling applications.

Chapter 14

Acknowledgments

My sincere thanks goes to numerous friends and colleagues. Eugene Lin for catching my vision and contributing to it. Matt Lennon, Katy Brown Evanco and Prasath Nanayakkara for being great UROPS. Stefan Agamanolis, Brett Granger and Irene Shen for their help in creating TVOT's software infrastructure. John Watlington for hardware system design excellence. Matt Antone for shared enthusiasm for parallelepiped approaches. Henry Holtzman for ground-breaking work in model-based video. Nuno Vasconcelos and Dan Gruhl for their thoughtful discussions on probability and databases. Kathi Blocher and Loni Butera for their timely assistance with model building, set preparation and video shooting. Giri Iyengar and Karrie Karahalios for digitizing my video sequences. David Tames, for being our distinguished movie producer for *The Museum* and *The Yellow Wallpaper*. Tinsley Galyean for insights on modeling, texture sampling and design of interesting and self-aware content. Steve Mann for his new perspectives on photography. Roger Kermode, Kris Popat, Lee Campbell, Ali Azarbayejani, Sumit Basu and Nassir Navab for helpful advice on vision strategies. And thanks to Barry Arons, Lisa Stifelman, Judith Donath, Bruce Blumberg, Ken Kung and Irfan Essa for their never-ending encouragement.

Special thanks goes to Bill Butera for often taking on the burden of production details, the never-ending process of testing, for timely advise and encouragement and for helping me endure to the end.

Thanks go to my generals and thesis committee members. Aaron Bobick for his thorough grasp of computer vision issues and for his desire to teach as well as encourage. Lee McKnight for bringing foresight to bear on regulation and policy decisions for technology. Seth Teller for keeping me honest enough to show the shortcomings as well as the strengths of this work. Sandy Pentland for breadth and depth of experience in vision and modeling.

I credit my early mentors Bill Barrett and Dan Olsen at BYU for infecting me with the desire to delve into the murky world of computer vision, graphics and user interface. I thank Glorianna Davenport for merging cinematography and story telling with computer graphics.

I thank Walter and Wanda Bender for their encouragement and support for me and my family. I especially thank Eileen Dorschner for kindly sustaining us during these last months at MIT.

I thank my advisor, Mike Bove, for his commitment to advancing useful research before most realize its importance. Also, I thank you for your efforts to make it possible for us in TVOT to do top level research.

And most of all thanks go to my children, Alissa and Jefferson and to my wife Linda who brings light, joy and meaning into my life and whose love and support has made it possible for me to do this work.

Appendix A

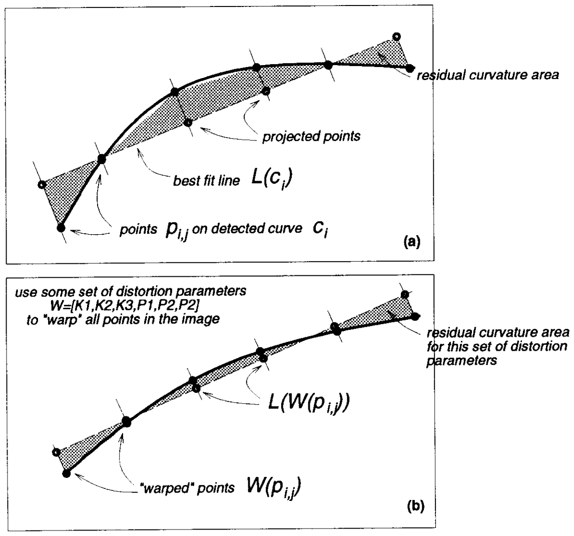
Appendix

A.1 The traditional "plumb-line" lens distortion estimation technique

In section 4.1 we introduced an observation function that measures the amount of distortion from the vanishing point variances for qualified directions in an image. This observation function is derived from Brown's [23] "plumb-line" observation function which is of the form

$$f(\mathbf{c}_i, \mathbf{p}_{i,j}; K_1, K_2, K_3, P_1, P_2, P_3) = \sum_{i=1}^{M}\sum_{j=1}^{N_i} ||W(\mathbf{p}_{i,j}) - L(W(\mathbf{p}_{i,j}))|| \tag{A.1}$$

where W denotes the parametric warping operation, L is line fitting, $\mathbf{c}_i$ is the i-th curve and $\mathbf{p}_{i,j}$ describes the j-th of N_i points in that curve as shown in Figure A-1.



residual curvature area
projected points
best fit line $L(c_i)$
points $p_{i,j}$ on detected curve C_i
(a)
use some set of distortion parameters W=[K1,K2,K3,P1,P2,P2] to "warp" all points in the image
residual curvature area for this set of distortion parameters
$L(W(p_{i,j}))$
"warped" points $W(p_{i,j})$
(b)

Figure A-1: The "Plumb-line" approach for solving lens distortion involves finding the distortion parameters that take curve points in the original image (a) and warp them into (b) so that the sum of residual curvature areas (shown in grey) for all curves is minimized.

A.2 Center of projection representation and estimation

For most computer vision setups, the COP is considered to be the origin of the camera's coordinate frame and a virtual image plane is positioned one focal length in front of the COP toward the scene. In an actual physical camera, however, the COP is an imaginary point lying somewhere along the optical axis inside a series of lenses, the physical projection image of a scene object lies inverted on the film plane, and the center of projection is positioned between the scene and the film plane. Mathematically, the two arrangements seem identical and the first simpler virtual (right-side-up) camera model is usually preferred.

In this thesis, we use the physical (upside-down) model since it more correctly models the physical imaging situation (e.g. we can take advantage of knowledge that the film plane is held rigid while the focal length is changed). Ali Azarbayejani has recently pointed out that another possible benefit of using the physical model is that it does not under-represent backprojection error as the virtual model seems to do.

Now addressing the problem of estimation, why bother trying to estimate more than just focal length? This is an apt question since the camera's center of projection is almost always located along the optical center and thus the principal point should lie at the geometric center of the projected image. Although this might be true for the film plane of the physical camera, several factors can make this untrue for the final digitized image: possible horizontal delay in video acquisition hardware, physical misregistration of a photograph or negative on the scanner bed, or uncentered cropping.

Another justification for solving principal point as well as focal length using information contained in the digitized image itself is that the geometric image center is nowhere near the true principal point when either the film or lens planes are allowed to swivel, shift or tilt. This is the normal case for shear and bellows cameras which are often used for portrait photography to control the plane of focus and in architectural photography to remove undesirable perspective cues without changing camera position[1].

A.3 Measuring precision and accuracy

To correctly represent quality of a particular estimation we should first clarify and differentiate the notions of precision versus accuracy.

Let x^* denote a known process mean and x describe a random set of experimental samples of that process.

Sample mean $\bar{x} = E_x[x]$ is the mean of the set of samples. Working with limited data the sample mean is the best estimate we can make of x^*. Sample variance $S_{\bar{x}} = E_x[(x - \bar{x})^2]$ measures compactness of the sample set. If it so happens that several trials of estimations made using independent noisy measurements agree on the same result then we can say that

[1]Architectural photographers typically try to avoid vertical keystoning when viewing a building from street level. To do this, the camera's optical axis must be parallel with the ground thus pointing at a spot on the building less than 6 feet above the ground. If a tall building is the subject of the photograph most of the bottom half of the view is wasted on the street, and the top part of the building will be cropped out unless the camera is moved far away from the building resulting in lost resolution due to distance. To obtain a full view of the building while maintaining the direction of the optical axis, the virtual image plane can be slid up (or equivalently the physical film plane can be slide down) - thus effectively reframing or recropping the visual extent until the entire building is centered in the film plate. In this case we can maintain that the principal point is centered horizontally and the unknown vertical translation can be recovered using the 2-point perspective constraint.

the estimating process has a high degree of precision. Or put in another way, we can use sample variance as a measure of recoverability of information.

Bias $b = (\bar{x} - x^*)$ is a measure of estimator accuracy or correctness. Mean square error, $MSE = E_x[(x - x^*)^2]$, is a measure of experimental deviation from the true process mean[2]. The root mean square, $RMS = \sqrt{N}\sqrt{MSE}$, is often used to bring *mse* back into magnitudes that have meaning in the original metric space.

Note, that b, MSE and RMS are measurements which are applicable only when the true process mean x^* is known. When dealing with real-world situations it is typical that the true mean is unknown. In fact, the very reason for taking multiple measurements in real-world situations is often to estimate the process mean (for example by as its sample mean $\bar{x}$).

Therefore, in real-world situations where we wish to estimate quantities which cannot be measured directly (such as camera rotation and position) we will use sample variance to measure estimator precision or recoverability. To bring results back into meaningful magnitudes we will take the square root of the sample variance $RSV = S_{\bar{x}}^{\frac{1}{2}}$.

In synthetic experiments, or in those real-world situations where the process mean can be found absolutely by direct measurement (such as positions of 3-D features), we will use RMS and b as measures for estimator accuracy.

A.4 Probabilistic representations for uncertain geometry

We describe geometric features in the scene and inside the camera as parameter vectors with Gaussian distributions. This representation allows us to describe uncertain points, lines and surfaces in such a way that their means and covariances can be manipulated and transformed between coordinate frames in an efficient and consistent manner. This representation also allows us to represent total lack of information where appropriate, i.e. when assigning arbitrary depths to scene features. In the event that we are given new information, whether contained in other images, or by way of prior knowledge of the scene, such as from a CAD description, representation of uncertain geometry also allows us to correctly merge all available information.

A.4.1 Image line representation

An image is produced by applying the projection equation (3.11) to a scene containing three-dimensional edges to get an image of line segments. Besides using the usual endpoint representation for imaged lines we also use a three-parameter form

$$\tan(e_\theta)(x - e_x) - (y - e_y) = 0 \qquad \text{(A.2)}$$

where e_θ, e_x and e_y conveniently describes the segment in such a way that direction angle and midpoint are decoupled. Since line detection is noisy, we treat each detected line as a 3×1 random Gaussian vector with mean $\mathbf{e} = [e_\theta, e_x, e_y]^T$. The three parameters are independent so the covariance matrix is diagonal $\Lambda_e = diag(\sigma_\theta^2, \sigma_x^2, \sigma_y^2)$. Since perspective mapping stretches out features at the periphery, midpoint variances σ_x^2 and σ_y^2 are directly

[2]Note that MSE confuses notions of precision and accuracy since $MSE = S_{\bar{x}} + b^2$.

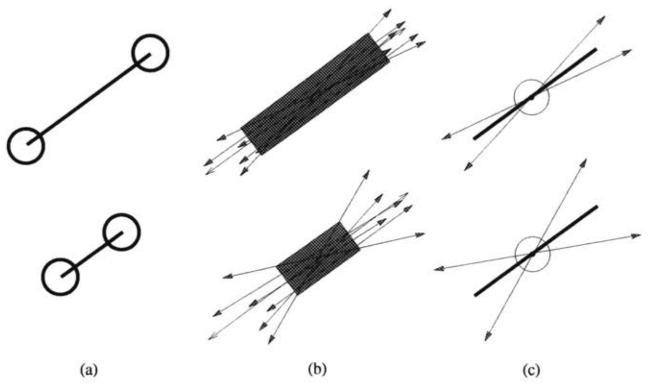


Figure A-2: Inverse relationship between segment length and angle variance: (a) results of segment detection showing confidence intervals at the endpoints, (b) a set of true lines, any one of which could have given rise the segment observed under noisy conditions, (c) confidence intervals of a probabilistic representation that decouples uncertainty in center and angle.

related to midpoint's distance R from the principal point. Using focal length $F = {}^f c_z$

$$\sigma_x = \sigma_y = (1 + (R/F)^2)\sigma_p \tag{A.3}$$

where σ_p^2 is the user supplied minimum detector variance at the principal point. If we consider pixel error to be uniform over a disk with radius ϵ, the covariance is $\epsilon^2/4$.

As illustrated in Figure A-2 angle variance, σ_θ^2 is inversely proportional to segment length L. Using the small angle approximation we find

$$\sigma_\theta = 2\sigma_x / L \tag{A.4}$$

A.4.2 Projective representations for lines

As reviewed in section 3.2.1 the 2-D image projection $\mathbf{p}$ of a 3-D point constrains the possible position of that 3-D point to be somewhere along its "projection vector" which is defined as having direction $\tilde{\mathbf{p}} = (\mathbf{c} - \mathbf{p}) = [c_x - p_x, c_y - p_y, c_z]^T$ and passes through the center of projection. Similarly the 2-D image projection of a 3-D edge constrains its position to lie somewhere in its "projection plane" which has a "projection plane normal" $\mathbf{n}$ and contains the center of projection $\mathbf{c}$ as well as the two endpoints of the 3-D edge's image projection.

If we represent the observed line segment probabilistically with mean $\mathbf{e} = [e_\theta, e_x, e_y]$ and covariance $\Lambda_\mathbf{e}$ as described in section A.4.1 the mean of the projection plane normal $\mathbf{n}$ is

$$\mathbf{n} = \tilde{\mathbf{n}} / \|\tilde{\mathbf{n}}\| \tag{A.5}$$

where

$$\tilde{\mathbf{n}} = \begin{bmatrix} e_x - \tilde{c}_x \\ e_y - \tilde{c}_y \\ -\tilde{c}_z \end{bmatrix} \times \begin{bmatrix} \cos e_\theta \\ \sin e_\theta \\ 0 \end{bmatrix} \tag{A.6}$$

which is equivalent to taking the cross product of the projection vectors of the image line's two endpoints.

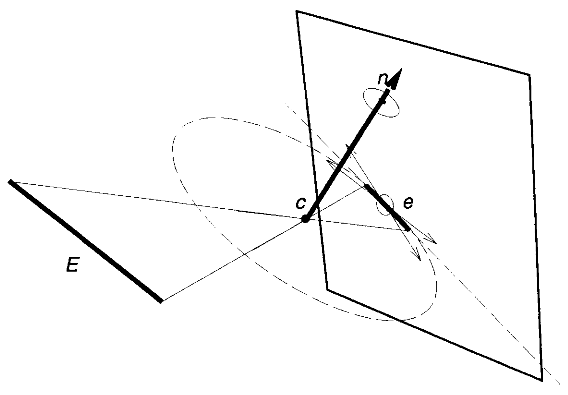


Figure A-3: Representing the probabilistic line as a probabilistic projection plane normal

A.4.3 Using the Jacobian to determine process covariance

To represent uncertainty (or confidence) of line position $\mathbf{e}$ in terms of this new projection normal representation $\mathbf{n}$, we need to likewise relate $\Lambda_{\mathbf{e}}$ to the covariance matrix $\Lambda_{\mathbf{n}}$. To do this we can use the observation[111] that for any sufficiently smooth function $g(\cdot)$ where $\mathbf{n} = g(\mathbf{e})$, the covariance Λ_n of random vector $\mathbf{n}$ can be approximated as

$$\Lambda_{\mathbf{n}} = \frac{\partial \mathbf{n}}{\partial \mathbf{e}} \Lambda_{\mathbf{e}} \frac{\partial \mathbf{n}}{\partial \mathbf{e}}^T \tag{A.7}$$

where $\frac{\partial \mathbf{n}}{\partial \mathbf{e}}$ is the 3×3 Jacobian matrix of the form $j_{a,b} = \partial n_a / \partial e_b$, letting $a = \{x, y, z\}$ and $b = \{\theta, x, y\}$ and evaluated at $\mathbf{e} = g^{-1}(\mathbf{n})$.

A.4.4 Describing Λ_n in terms of Λ_e

The covariance of the 3×1 random vector $\mathbf{n}$ is found by temporarily describing $\mathbf{n}$ by a tilt angle τ from the z-axis and a spin angle σ from the x-axis so that

$$\mathbf{n} = \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \begin{bmatrix} \sin\tau\cos\sigma \\ \sin\tau\sin\sigma \\ \cos\tau \end{bmatrix} \tag{A.8}$$

To find the tilt angle τ we need to find the minimum distance ρ from the image line e to the principal point which is found from the line equation

$$\rho = (e_x - \tilde{c}_x)\cos\sigma + (e_y - \tilde{c}_y)\sin\sigma \tag{A.9}$$

where the spin angle σ is the angle of the vector pointing from the principal point to the nearest point point on the line, and thus is orthogonal to the line angle

$$\sigma = e_\theta + \frac{\pi}{2} \tag{A.10}$$

The tilt angle, or the angle of the normal vector from the z-axis, can now be found as

$$\tau = \frac{\pi}{2} - \arctan\frac{\rho}{F} \tag{A.11}$$

where focal length F is $\tilde{c}_z$. If we form the 2×1 random vector, $\mathbf{t} = [\tau\sigma]^T$, (A.8) is now used to form the 3×2 Jacobian matrix

$$\frac{\partial \mathbf{n}}{\partial \mathbf{t}} = \begin{bmatrix} \cos\tau\cos\sigma & -\sin\tau\sin\sigma \\ \cos\tau\sin\sigma & \sin\tau\cos\sigma \\ -\sin\tau & 0 \end{bmatrix} \tag{A.12}$$

(A.11) and (A.10) are used to form the 2×3 Jacobian matrix

$$\frac{\partial \mathbf{t}}{\partial \mathbf{e}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -\frac{F}{F^2+\rho^2} & -\frac{F}{F^2+\rho^2} \end{bmatrix} \tag{A.13}$$

and the 3×3 covariance of the projection normal $\mathbf{n}$ described in terms of edge uncertainty is now

$$\Lambda_{\mathbf{n}} = \frac{\partial \mathbf{n}}{\partial \mathbf{t}}\frac{\partial \mathbf{t}}{\partial \mathbf{e}}\Lambda_{\mathbf{e}}\frac{\partial \mathbf{t}}{\partial \mathbf{e}}^T\frac{\partial \mathbf{n}}{\partial \mathbf{t}}^T \tag{A.14}$$

A.5 Computational methods for approximating covariance

The reader has no doubt noticed that we take some effort to determine the effect that non-linear transformations have on the covariance, and not just the mean of a measurement. This sort of probabilistic representation is retained to provide a measure of confidence in our estimates which is used later to do weighted merging of information from multiple sources.

While symbolic derivation of partial derivatives is often rather tedious, it is sometimes even impossible, such as when a process can only be implemented procedurally in a subroutine rather than as an equation. It is useful in these cases to consider using computational methods to approximate the effect that a mapping has on a distribution.

A.5.1 Central differences

One computational approach is to approximate the terms of the Jacobian matrix using central differences where the a, b'th term is

$$j_{a,b} \approx \frac{n_a(\mathbf{e} + W\sigma_{e_b}) - n_a(\mathbf{e} - W\sigma_{e_b})}{2W\sigma_{e_b}} \tag{A.15}$$

where W is used to scale σ_{e_b} down to some distance ϵ over which the function is locally smooth.

In general for a function given an M-dimensional input variable and an N-dimensional output, the central differences approach requires $2MN$ function evaluations (use of forward differences would require only $MN + 1$) and the final composition of Jacobian and input covariance matrices require $3M^2N^2$ floating point operations.

A.5.2 Population sampling

An alternative method for computationally approximating the covariance of a process is to represent an input distribution by a set of atomic samples oriented around a known mean. The procedural mapping is then applied to each sample point and the sample covariance is used to approximate the statistics of the output distribution. The sampling can be done with fewer function evaluations (than required for the other method) but at the cost of requiring principal component analysis to calculate eigenvectors and eigenvalues of the input covariance matrix.

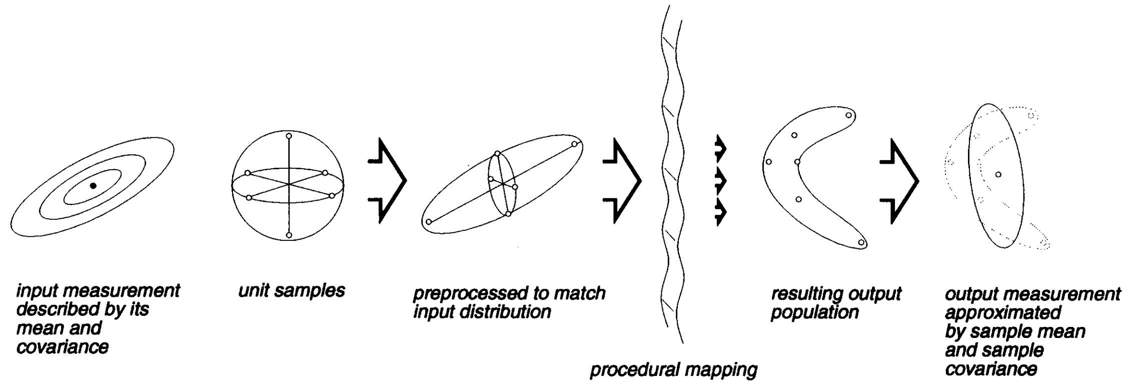

input measurement described by its mean and covariance
unit samples
preprocessed to match input distribution
procedural mapping
resulting output population
output measurement approximated by sample mean and sample covariance

Figure A-4: The effect that any mapping has on an input distribution can be approximated by using the sample covariance of a mapped population

As illustrated in Figure A-4 atomic samples are taken uniformly and symmetrically over the M-dimensional unit sphere. If there are L such M-dimensional samples, then the sample covariance of this set is $M(L-1)/L$. A critical set of samples can be represented as an $M \times 2M$ matrix $\mathbf{U}$ whose rows are the variables and whose columns are unit samples, i.e. $\mathbf{U} = [\mathbf{I}_N| - \mathbf{I}_N]$ where $\mathbf{I}_N$ is an $N \times N$ identity matrix.

Each unit sample point (or each column in $\mathbf{U}$) is scaled, sheared and translated so that the sample mean and covariance of the population match the statistics of the input distribution. The scale factor prenormalizes unit samples to make their unbiased sample covariance come out to identity. Since $\mathbf{U}$ contains $L = 2M$ sample points its value is $s = \sqrt{M - \frac{1}{2}}$. The shearing matrix is the M-dimensional equivalent of standard-deviation

$$\mathbf{S} = \Phi\Lambda^{\frac{1}{2}} \tag{A.16}$$

where Φ and Λ are the eigenvector and eigenvalue matrices. $\mathbf{S}$ can be replaced with the Jacobian matrix $\mathbf{J}$ since it too can be used to describe the covariance.

The translation term is the mean of the input distribution.

The mapping is then applied to all preprocessed input samples and the sample mean and covariance of the output population is found.

The sample covariance of the set of all candidate $\mathbf{n}_i$ is then

$$\Lambda_{\mathbf{n}_{a,b}} = \frac{1}{2M}\sum_{i=1}^{2M}(\mathbf{n}_{i_a} - \bar{\mathbf{n}}_a)(\mathbf{n}_{i_b} - \bar{\mathbf{n}}_b)^T \tag{A.17}$$

where $\bar{\mathbf{n}}$ is the sample mean

$$\bar{\mathbf{n}}_a = \frac{1}{2M}\sum_{i=1}^{2M}\mathbf{n}_{i_a} \tag{A.18}$$

and where $\mathbf{n}_i$ is the i-th sample in the set of all $2M$ candidate values for $\mathbf{n}$.

If the mapping is severely non-linear the sample mean of the output will not correspond to the mapping of the mean point. It may be thus desirable to represent the mean of the output distribution as the mapped mean.

A total of $2M^2$ floating point operations are used to apply the preprocessing step to all $2M$ $M-dimensional$ input points in $\mathbf{U}$ and a total of $6M^2N$ floating point operations are required to calculate the sample covariance. There is also the heavy cost of calculating the eigenvectors and eigenvalues of the input covariance matrix.

A.6 Non-gaussian characteristics of vanishing points and center of projection

The center of projection plays a pivotal role in the projection equation and subsequently in 3-D reconstruction from 2-D views. In order to provide a measure of confidence in the results of 3-D reconstruction, it is important to estimate error distributions of all terms leading up to that goal.

As illustrated in Figure A-5, confidence intervals of a vanishing point direction $\hat{\mathbf{d}}$ form concentric cones eminating from the origin of the sphere of directions. Note that confidence intervals of the corresponding vanishing point $\mathbf{v}_i$ which lie at the intersection of these cones with the image plane are not usually concentric. As the angle between $\hat{\mathbf{d}}$ and the film plane approaches zero the vanishing point distribution becomes increasingly coma shaped with a tail that trails away from the principal point. As a result, if we insist on making normalizing assumptions at each step the sample mean of the vanishing point (and thus subsequent estimates for center of projection) will be biased by having an exaggerated distance from the principal point.

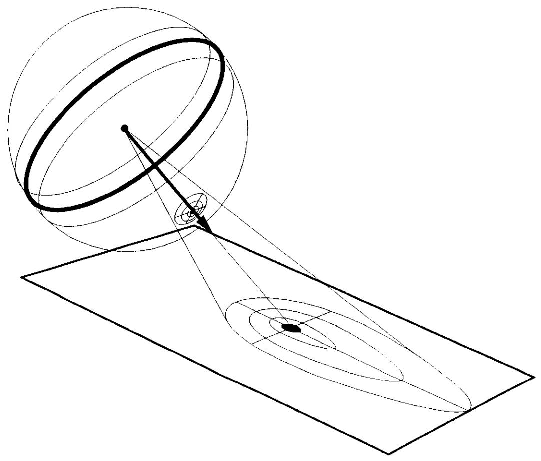


Figure A-5: Typical non-gaussian distribution of a vanishing point

A.6.1 Representing non-gaussian distributions as populations of atomic samples

To eliminate this biasing tendency when estimating $\mathbf{c}$, we might choose to temporarily avoid the normalizing assumptions and approximate each vanishing point distribution as a set of N_v point samples along a confidence interval as described in section A.5. Since any three vanishing points can be used to find any one candidate $\mathbf{c}$, all 3 sets of N_v point samples result in a total of $N_c = {N_v}^3$ candidate values for $\mathbf{c}$.

Because the vanishing points can be strongly non-Gaussian and since the center of projection equations are strongly non-linear, the actual distribution of the center of projection will usually be even more non-linear (see Figure A-6). For accuracy in subsequent camera pose and scene structure estimation it may be useful to retain this entire set of point samples to accurately describe the non-linear distribution of **c** .

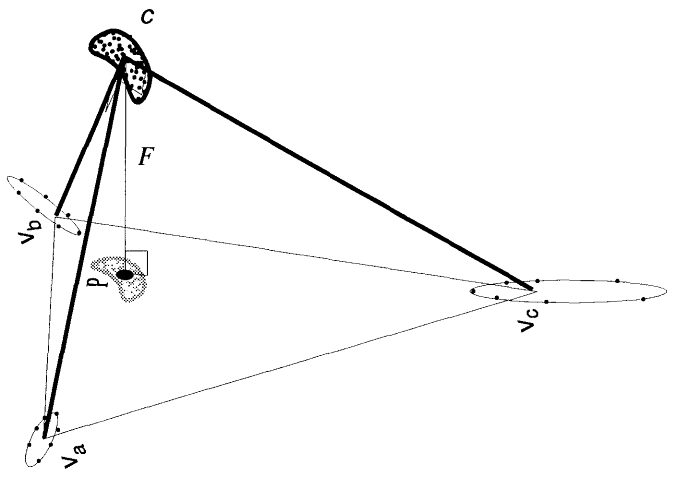


Figure A-6: Typical non-Gaussian distribution of center of projection

A.6.2 Avoiding bias in the sample mean

If for the sake of convenience we still wish to make normalizing assumptions we can use either of the computational approaches described in section A.5 to approximate the distribution of **c** . However, since the distributions of the vanishing points are generally not Gaussian, the sample mean of the output **c** population will be biased away from the mode (i.e. the value that has maximum likelyhood or ML). To avoid this biasing problem we always use the ML estimates $\mathbf{v}_i$ to compute **c** directly and use this as the mean of the distribution.

A.7 Theoretical camera distance estimation error as a function of varying detector noise

To get an idea of how well we can estimate camera distance in the presence of point placement error, suppose we view an edge with known height L and orientation from a distance Z with known focal length F so that its inverted image on the film plane has height h as shown in figure A-7. If detector noise at each endpoint of the imaged line is modeled as having a uniform distribution with radius Δn then since

$$\begin{aligned} \tfrac{h}{F} &= \tfrac{L}{Z-F} \\ Z &= F(1+\tfrac{L}{h}) \end{aligned} \tag{A.19}$$

the range of errors for the estimated camera distance ΔZ will be

$$\Delta Z = 2\frac{FL}{h^2}\Delta n \tag{A.20}$$

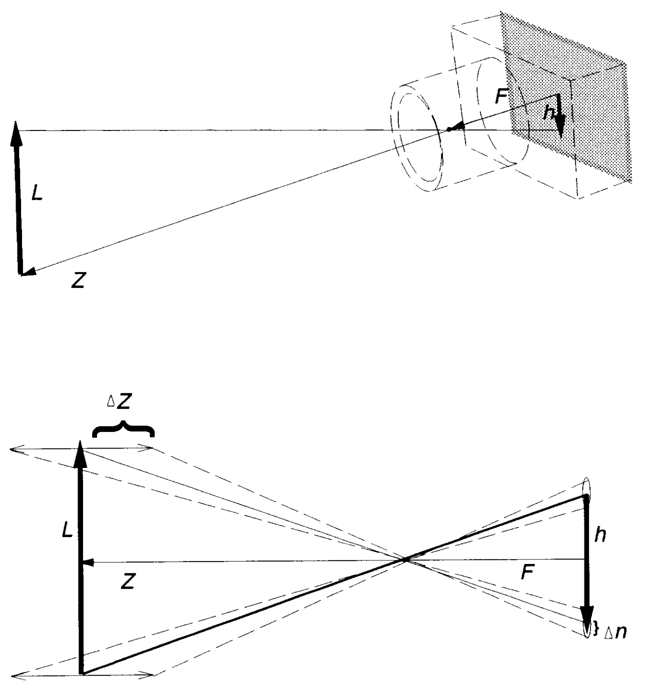


Figure A-7: Camera distance error as a function of detector noise.

which is a linear with respect to detector noise Δn.

Using the simulated setup described in Section 10.2, the object is L = 40mm tall and viewed from a distance of Z = 250mm with a F = 50mm focal length camera onto film with dimensions 36×24mm. If we digitize the image at a resolution of 480×320 pixels and then manually position the endpoints within 1 pixel of accuracy, the error in the estimated distance from the film plane to the camera will fall within ΔZ = 3mm or 1.4% of the actual distance Z. This, in fact, is similar to the kind of experimental results shown in Section 10.2.

A.8 Theoretical camera distance estimation error as a function of varying focal length in the presence of detector noise

From equation A.20 we see that camera distance error is non-linearly related to known focal length F. Figure A-8 shows that for this setup the expected error ΔZ in the estimated camera distance ranges from 1.5% to 0.5% of the true Z when known F varies over the range of typical focal lengths from 30 to 70mm.

This result assumes that actual focal length is known. When the focal length is unknown, as assumed in the synthetic experiments of Section 10.2, there are additional sources of estimation error. Of these, camera-relative orientation of the object plays the most significant part.

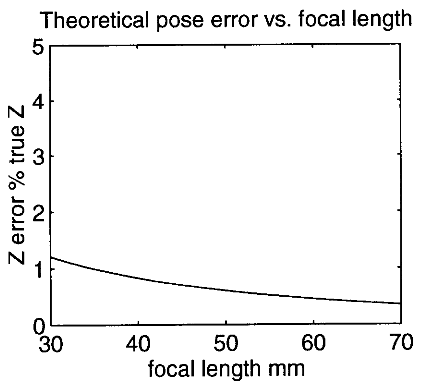


Figure A-8: Camera distance error as a function of focal length in the presence of detector noise of ± 1 pixel.

A.9 Center of projection qualitative error analysis

We find that the covariance of $\mathbf{c}$ is minimized when the triangle area of the projected vanishing points is minimized and when the interior angles are equal. An aspect of the vanishing point's comma effect is that the uncertainty of the principal point approaches infinity as any one vanishing point approaches infinity (or as any $\hat{\mathbf{d}}_{i_z}$ approaches zero). Given a scene with three mutually orthogonal edge directions this situation occurs either 1. when one or more of the camera's coordinate axes become aligned to directions in the scene (as seen in 10-12(a) for spin angles less than 9 degrees) or 2. when projection approaches orthography (i.e. the effective focal length approaches infinity) or equivalently when the calibrating parallelepiped occupies only a small portion of the view.

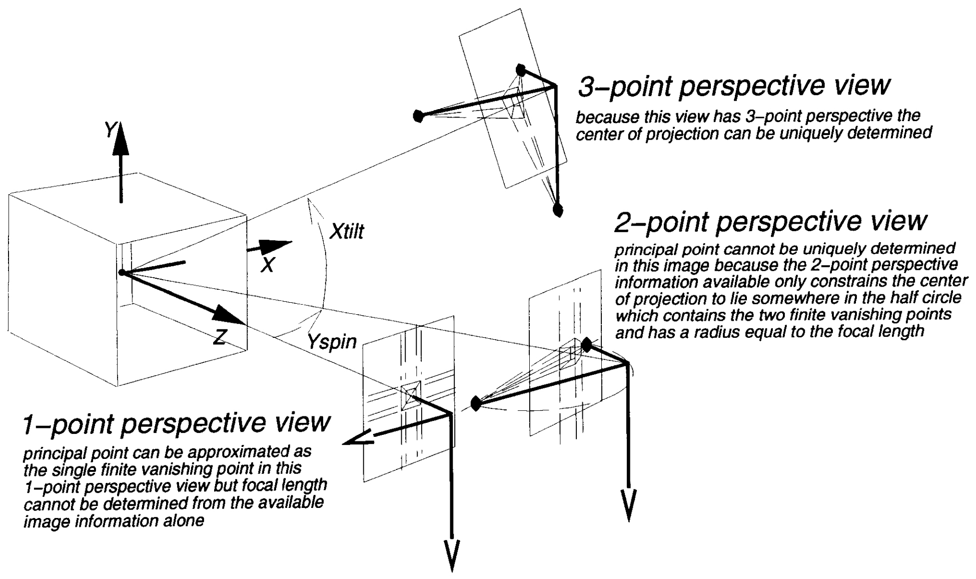

3-point perspective view
because this view has 3-point perspective the center of projection can be uniquely determined
2-point perspective view
principal point cannot be uniquely determined in this image because the 2-point perspective information available only constrains the center of projection to lie somewhere in the half circle which contains the two finite vanishing points and has a radius equal to the focal length
1-point perspective view
principal point can be approximated as the single finite vanishing point in this 1-point perspective view but focal length cannot be determined from the available image information alone
Y
X
Z
Xtilt
Yspin

Figure A-9: Three possible situations for center of projection estimation

In the event that d_{i_z} is near zero for two or more directions *a-priori* information is needed to determine center of projection. For example, given a 2-point perspective view (see Figure A-9) knowledge any one of the 3 cop parameters allows us to solve the other two. We may, in such a case, attempt to approximate the principal point as the geometric image center, a valid presumption unless the image was photographed using a bellows camera or unknowingly cropped. Given 1-point perspective the principal point can be considered to be the finite vanishing point, but focal length remains unknown.[3]

A.10 Theoretical focal length and camera distance estimation errors as functions of varying object orientation in the presence of detector noise

Our approach for estimating focal length relies entirely upon the availability of vanishing points from 3 mutually orthogonal sets of edges such as the visible edges of two walls of a rectangular solid. Since line detectors are noisy the vanishing points themselves will also be noisy. If the object consists of only a few edges the resulting vanishing points will be biased[4]. The magnitude of this bias depends upon the orientation of the object relative to the film plane. We can therefore use this information to predict the effect that object orientation has on the recoverability of focal length.

Consider a pair of square surfaces joined at a vertical edge with length L, spin angle θ, focal length F and camera distance Z as shown in Figure A-10(a). The projection of this simple object as shown in A-10(b) consists of noisy lines which result in noisy vanishing points. When θ approaches any integer multiple of 90^o one of object's two directions will become parallel with the image plane and the lines from that direction will intersect at a vanishing point whose distance from the principal point approaches infinity. From the top view we can see that the two vanishing points and the center of projection form a right triangle.

By using the geometries of Section 10.2 with 1 pixel of uniform point positioning error Figure A-11(a) shows the theoretical prediction for error limits in the estimated focal length as the relative angle between the object and the film plane varies from 0^o to 45^o. Since camera pose is strongly dependent upon having an accurate estimate for focal length, Figure A-11(b) likewise shows the theoretical limits of the recoverability of camera pose.

Note that as θ approaches 0^o the error in focal length and camera distance approaches infinity, as expected. These theoretical results explain the experimental results of Section 10.2.

A.11 Surface reconstruction

If one surface contains lines which belong to two or more directions, then the normal vector $\mathbf{m}$ of the 3-D plane of surface is simply the cross product of the member directions. Once the normal is known, if it contains any edge whose 3-D position is already known, the 4th

[3]Note that as we approach orthography the importance of the principal point should vanish since it is scaled by inverse focal length in the perspective transformation (3.6). Thus as the focal length increases, the adverse results of arbitrarily setting the principal point at the image center should decrease.

[4]A corollary of the central limit theorem[111] is that the sample mean of a Gaussian distribution approaches the true process mean as the number of sample approaches infinity. Conversely, as the number of samples is reduced, the sample mean will be biased.

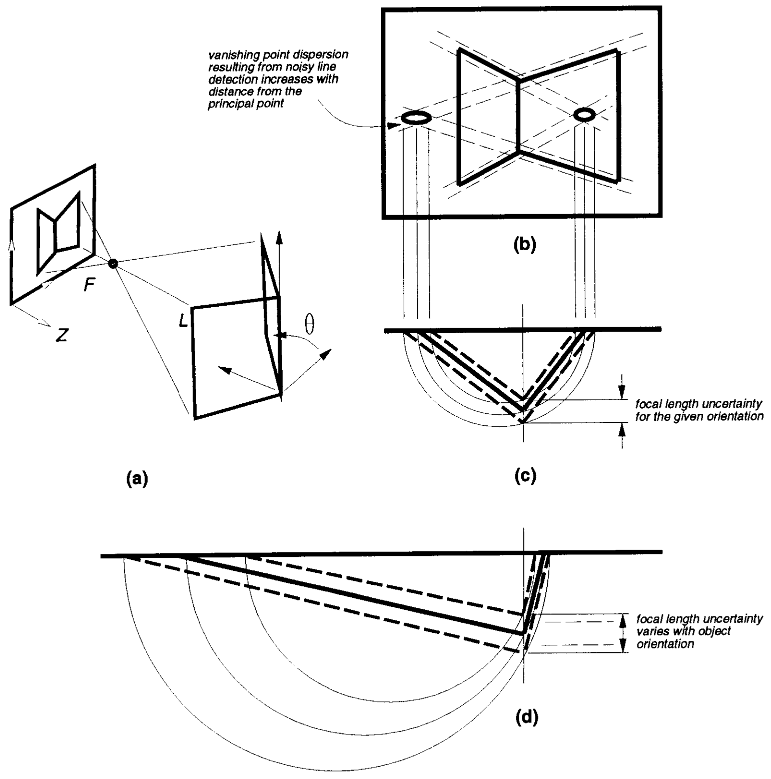

vanishing point dispersion resulting from noisy line detection increases with distance from the principal point
F
Z
L
θ
(a)
(b)
(c)
focal length uncertainty for the given orientation
focal length uncertainty varies with object orientation
(d)

Figure A-10: Focal length estimation error is sensitive to relative orientation of parallel lines. (a)A camera with a simple scene. (b)Front view of the image plane showing a simplified line error model (i.e. noiseless line angle but noisy line position) and how it affects vanishing point placement. (c)The top view of the image plane shows the range of focal length solutions given the range of vanishing points in (b). (d)Shows that as one of the object's directions are more parallel to the image plane the spread of possible focal lengths is larger. This spread approaches infinity when the object's surface is parallel to the camera's image plane.

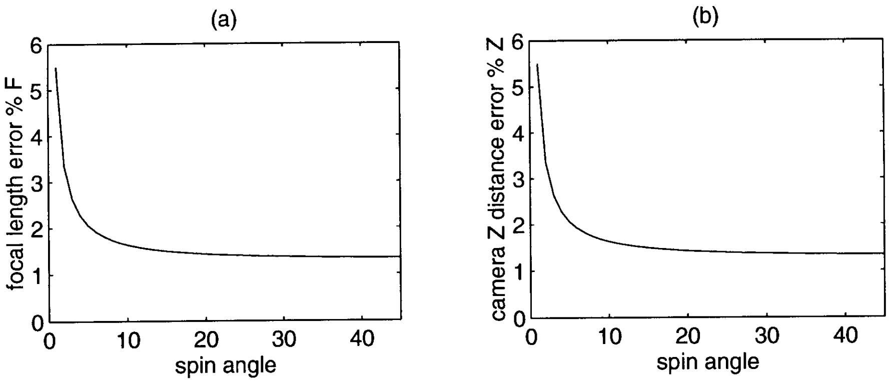


Figure A-11: Camera estimation error as functions of spin angle in the presence of 1 pixel detector noise. (a) shows focal length error as a percentage of true focal length. (b) shows camera distance error as a percentage of true camera distance.

term of the plane equation, m_d, can be found using 5.14

$$n_d = -\mathbf{m}^T\mathbf{x} \tag{A.21}$$

where $\mathbf{x}$ is any point on the fixed edge. The solution is identical if three 3-D points are known deterministically. Given three or more probabilistically known 3-D points, the optimal mean position and normal vector is found using the ML-estimator described in section 3.2.3 to find the weighted mean and covariance of the aggregate collection of distributions arranged approximately on a plane.

A.12 Edge reconstruction

A.12.1 Intersection of two 3-D planes

Any non-parallel pair of 3-D planes, $\mathbf{m}_1$ and $\mathbf{m}_2$ intersect along an infinite 3-D hinge line. Let $\mathbf{u}$ denote the direction vector of the line and let $\mathbf{x}$ be the point on the line which is closest to the origin. Since the line is shared by both planes, the line's direction vector is simply the cross product of the two surface normals

$$\mathbf{u} = \mathbf{m}_1 \times \mathbf{m}_2 \tag{A.22}$$

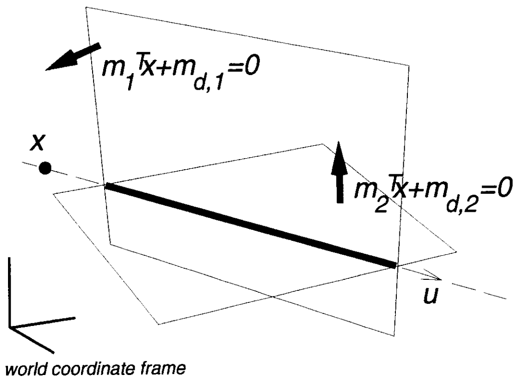


Figure A-12: Intersection of two planes

The three unknowns of line point $\mathbf{x} = [x, y, z]^T$ can be found by simultaneously solving three linear constraints

$$\begin{cases} \mathbf{u}^T\mathbf{x} = 0 & \text{(x is the point on the line which is closest to the origin)} \\ \mathbf{m}_1{}^T\mathbf{x} + m_{d1} = 0 & \text{(x is in the first plane)} \\ \mathbf{m}_2{}^T\mathbf{x} + m_{d2} = 0 & \text{(x is in the second plane)} \end{cases} \tag{A.23}$$

A.12.2 Projection of a 2-D image line onto a 3-D plane

In an imaging situation an edge can be reconstructed by projecting a single line in a known camera onto a previously fixed surface $[m_x, m_y, m_z, m_d]^T$. Given the center of projection $\mathbf{c}$ and a line's projection plane normal $\mathbf{n}$ the direction vector is subject to the following two constraints

$$\begin{cases} \mathbf{m}^T\mathbf{u} = 0 & (\mathbf{u} \text{ lies in the plane of the fixed surface}) \\ \mathbf{n}^T\mathbf{u} = 0 & (\mathbf{u} \text{ lies in the projection plane of the observed line}) \end{cases} \tag{A.24}$$

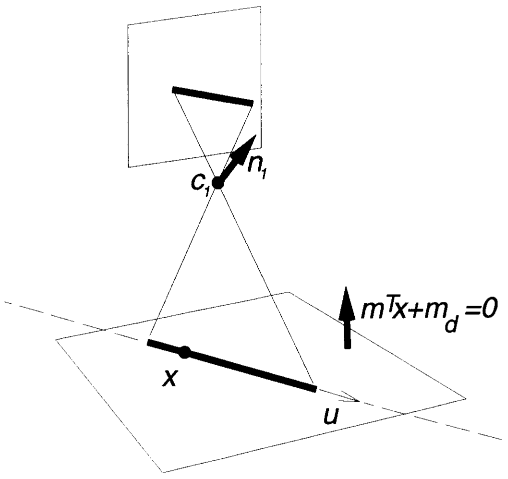


Figure A-13: Projection of a 2-D line onto a plane

and the line point $\mathbf{x}$ has three constraints

$$\begin{cases} \mathbf{u}^T(\mathbf{x} - \mathbf{c}) = 0 & (\mathbf{x} \text{ is the point on the line which is closest to } \mathbf{c}) \\ \mathbf{m}^T\mathbf{x} + m_d = 0 & (\mathbf{x} \text{ lies in the plane of the fixed surface}) \\ \mathbf{n}^T(\mathbf{x} - \mathbf{c}) = 0 & (\mathbf{x} \text{ is in the projection plane of the observed line}) \end{cases} \tag{A.25}$$

from which solutions for $\mathbf{u}$ and $\mathbf{x}$ can be solved as seen in (A.22) and (A.23).

A.12.3 Triangulation from 2-D lines in two views

An edge can also be reconstructed by triangulation from lines in multiple views. For the two view case suppose $\mathbf{c}_1$ and $\mathbf{c}_2$ are the optical centers of the cameras and $\mathbf{n}_1$ and $\mathbf{n}_2$ are the projection normals of the edge as seen in each view transformed into world-relative coordinates. The vector and point of the 3-D edge can be solved in straightforward fashion from the following constraints

$$\begin{cases} \mathbf{n}_1{}^T\mathbf{u} = 0 & (\mathbf{u} \text{ lies in the first projection plane}) \\ \mathbf{n}_2{}^T\mathbf{u} = 0 & (\mathbf{u} \text{ lies in the second projection plane}) \end{cases} \tag{A.26}$$

$$\begin{cases} \mathbf{u}^T(\mathbf{x} - \mathbf{c}_1) = 0 & (\mathbf{x} \text{ is the point on the line which is closest to } \mathbf{c}_1) \\ \mathbf{n}_1{}^T(\mathbf{x} - \mathbf{c}_1) = 0 & (\mathbf{x} \text{ lies in the first projection plane}) \\ \mathbf{n}_2{}^T(\mathbf{x} - \mathbf{c}_2) = 0 & (\mathbf{x} \text{ lies in the second projection plane}) \end{cases} \tag{A.27}$$

A.12.4 Triangulation from 2-D lines in multiple views

For the case of $N >= 2$ views of a given 3-D line, the edge direction $\mathbf{u}$ and its covariance matrix $\Lambda_\mathbf{u}$ can be calculated using the approach discussed in section 3.2.3. This involves

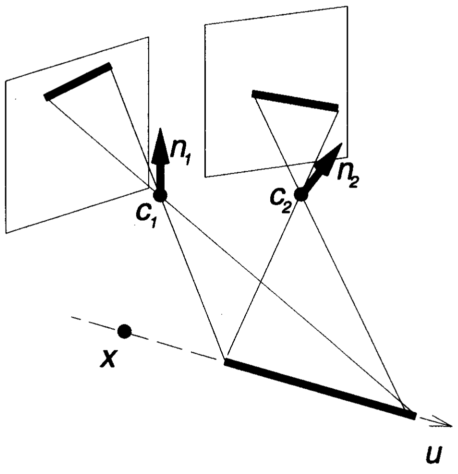


Figure A-14: Triangulation from 2-D lines in a pair of calibrated cameras

creating an aggregate set of all positive and negative pairs of unit projection plane normals $\mathbf{n}_i$, $i = [1, N]$ converted to world-relative terms. The optimal estimate of the edge direction mean $\mathbf{u}$ is the minimum eigenvector of that distribution and its covariance $\mathbf{\Lambda_u}$ is the minimum eigenvalue.

The optimal position of the line point which is closest to the first center of projection can be found by generalizing the constraints given in (A.27) to form the equation

$$\mathbf{Nx} = \begin{bmatrix} \mathbf{u}^T \\ \mathbf{n}_1{}^T \\ \mathbf{n}_2{}^T \\ \vdots \\ \mathbf{n}_n{}^T \end{bmatrix} \mathbf{x} = \begin{bmatrix} \mathbf{u}^T\mathbf{c}_1 \\ \mathbf{n}_1{}^T\mathbf{c}_1 \\ \mathbf{n}_2{}^T\mathbf{c}_2 \\ \vdots \\ \mathbf{n}_N{}^T\mathbf{c}_n \end{bmatrix} = \mathbf{b} \qquad (A.28)$$

for which $\mathbf{x}$ is the solution to the overdetermined set of linear equations. If the confidence of all measurements are considered identical then the optimal solution is the standard mean square estimator

$$\mathbf{x}_{\mathrm{MSE}} = (\mathbf{N}^T\mathbf{N})^{-1}\mathbf{N}^T\mathbf{b} \qquad (A.29)$$

Computing the ML-estimate weighting matrix for N-way line triangulation

Recall that the standard model for mean square estimation considers an output measurement $\mathbf{b}$ of a fixed but unknown phenomenon $\mathbf{x}$ subjected to a linear measurement transformation $\mathbf{N}$ and additive zero-mean Gaussian noise $\mathbf{g}$ with covariance $\mathbf{\Lambda_g}$.

$$\mathbf{b} = \mathbf{Nx} + \mathbf{g} \qquad (A.30)$$

If $\mathbf{x}$ is considered as being deterministic but unknown and $\mathbf{N}$ (which consists of plane normals) is treated as known then the covariance of $\mathbf{b}$ is the same as that of $\mathbf{g}$. Furthermore, if the covariance of $\mathbf{b}$ is not simply a scaled identity (i.e. $\Lambda_{\mathbf{b}} \neq s\mathbf{I}$ where $s > 0$) then we can weight each row measurement according to its respective degree of confidence. This is, in fact, what the maximum likelihood (ML) or weighted mean square errors estimator achieves

$$\mathbf{x}_{\mathrm{ML}} = (\mathbf{N}^T\Lambda_{\mathbf{b}}^{-1}\mathbf{N})^{-1}\mathbf{N}^T\Lambda_{\mathbf{b}}^{-1}\mathbf{b} \qquad (A.31)$$

where, in this case, $\Lambda_{\mathbf{b}}$ is the $3(n+1) \times 3(n+1)$ matrix whose diagonal is composed of the 3×3 covariance matrix of each row element of $\mathbf{b}$. From (A.28) we see that each scalar row element of $\mathbf{b}$ is the dot product of two probabilistic 3×1 vectors, the covariance of which is of the form

$$\Lambda_b = \mathbf{u}^T \Lambda_{\mathbf{v}} \mathbf{u} + \mathbf{v}^T \Lambda_{\mathbf{u}} \mathbf{v} \tag{A.32}$$

if $\mathbf{u}$ and $\mathbf{v}$ are independent and b is of the form $b = \mathbf{u}^T \mathbf{v}$.

The weighting matrix of (A.31) is, therefore,

$$\Lambda_{\mathbf{b}} = \begin{bmatrix} \Lambda_{\mathbf{u}} & 0 & \cdots & 0 \\ 0 & \frac{1}{4}(\mathbf{c}^T \Lambda_{\mathbf{n}1} \mathbf{c}_1 + \mathbf{n}_1^T \Lambda_{\mathbf{c}1} \mathbf{n}_1) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{4}(\mathbf{c}^T \Lambda_{\mathbf{n}N} \mathbf{c}_N + \mathbf{n}_N^T \Lambda_{\mathbf{c}N} \mathbf{n}_N) \end{bmatrix} \tag{A.33}$$

where the covariance $\Lambda_{\mathbf{u}}$ of the optimal line direction $\mathbf{u}$ was found previously.

Generalizing ML-estimation for all types of edge reconstruction

Although we have only shown the details of the ML estimator for the case of reconstructing edges by triangulation, this approach can easily be adapted for the other two cases of edge reconstruction (i.e. intersection of two 3-D planes, and projection from a 2-D line onto a 3-D plane).

A.13 Feature reconstruction

A.13.1 Intersection of three 3-D planes

Any set of three non-parallel 3-D planes give rise to a trihedral junction $\mathbf{x}$ which is constrained by the three linear equations

$$\mathbf{N}\mathbf{x} = \begin{bmatrix} \mathbf{n}_1{}^T \\ \mathbf{n}_2{}^T \\ \mathbf{n}_3{}^T \end{bmatrix} \mathbf{x} = - \begin{bmatrix} n_{d_1} \\ n_{d_2} \\ n_{d_3} \end{bmatrix} = \mathbf{b} \tag{A.34}$$

If we consider plane normals as unitary and deterministic and if minimum distance to the origin has been estimated probabilistically with variance $\sigma^2_{n_d}$ then the approach used in (A.33) can be used to find the ML-estimate where Λ_b is the diagonal matrix of the distance variances.

A.13.2 Projection from a 2-D point onto a 3-D plane

This is similar to the problem of projecting a noisy vanishing point direction onto the image plane except that we model more terms of uncertainty. A feature $\mathbf{x}$ in the scene is imaged as the point $\mathbf{p}$ in a view with $\mathbf{c}$ as the center of projection. We also know that feature $\mathbf{x}$ lies in a certain plane. This gives us two constraints

$$\begin{cases} \mathbf{x} = \mathbf{c} + s(\mathbf{c} - \mathbf{p}) & (\mathbf{x}\ \mathbf{c} \text{ and } \mathbf{p} \text{ are collinear}) \\ \mathbf{n}^T \mathbf{x} + n_d = 0 & (\mathbf{x} \text{ lies in the plane}) \end{cases} \tag{A.35}$$

for which the solution is

$$\mathbf{x} = (1+s)\mathbf{c} - s\mathbf{p} \tag{A.36}$$

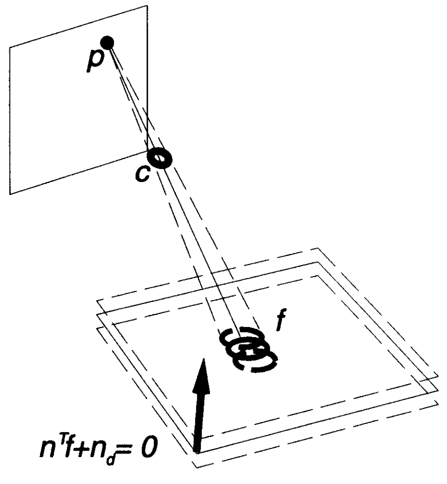


Figure A-15: Projection of a 2-D film point onto a 3-D plane

where s is the scaled distance between $\mathbf{c}$ and $\mathbf{x}$

$$s = \frac{\mathbf{n}^T\mathbf{x} + n_d}{\mathbf{n}^T(\mathbf{p} - \mathbf{c}\ )} \tag{A.37}$$

Since $\mathbf{c}$ has accumulated most of the detection errors of all image lines and points we consider it sufficient to describe the camera uncertainty in terms of $\Lambda_\mathbf{c}$ and consider the film point $\mathbf{p}$ (and the estimated rigid-body transformation) as deterministic. We will similarly consider the plane distance term n_d as probabilistic while keeping the unit plane normal deterministic.

We can now form a 4×1 vector of all probabilistic terms $\mathbf{v} = [c_x c_y c_z n_d]^T$. We use s to find the jacobians

$$\frac{\partial \mathbf{x}}{\partial s} = \begin{bmatrix} \frac{\partial f_x}{\partial s} \\ \frac{\partial f_y}{\partial s} \\ \frac{\partial f_z}{\partial s} \end{bmatrix} = \mathbf{c}\ - \mathbf{p} \tag{A.38}$$

and

$$\frac{\partial s}{\partial \mathbf{v}} = \begin{bmatrix} \frac{\partial s}{\partial c_x} & \frac{\partial s}{\partial c_y} & \frac{\partial s}{\partial c_z} & \frac{\partial s}{\partial n_d} \end{bmatrix} = \begin{bmatrix} W(n_x) & W(n_y) & W(n_z) & \frac{1}{D} \end{bmatrix} \tag{A.39}$$

where $D = \mathbf{n}^T(\mathbf{p} - \mathbf{c}\ )$ and where $W = \frac{1}{D^2}(D + \mathbf{n}^T\mathbf{c}\ + n_d)$. Equations (A.38) and (A.39) can now be used to describe the covariance of the feature point as a function of the combined positional variances of center of projection and the plane.

$$\Lambda_\mathbf{f} = \frac{\partial \mathbf{x}}{\partial s}\frac{\partial s}{\partial \mathbf{v}} \begin{bmatrix} \Lambda_\mathbf{c} & 0 \\ 0 & \sigma_{n_d} \end{bmatrix} \frac{\partial s}{\partial \mathbf{v}}^T \frac{\partial \mathbf{x}}{\partial s}^T \tag{A.40}$$

A.14 Triangulation from 2-D points in two or more views

Given a world-relative point $\mathbf{p}_i$ and COP $\mathbf{c}_i$ which forms a ray $\mathbf{u}_i = \mathbf{c}_i - \mathbf{p}_i$ in image i we wish to find the coordinates of feature $\mathbf{x}$ that best describes the triangulation of all of the rays coming from $N >= 2$ images.

For $N = 2$ images we consider $\mathbf{y}_1$ and $\mathbf{y}_2$ to be points on a pair of non-parallel yet

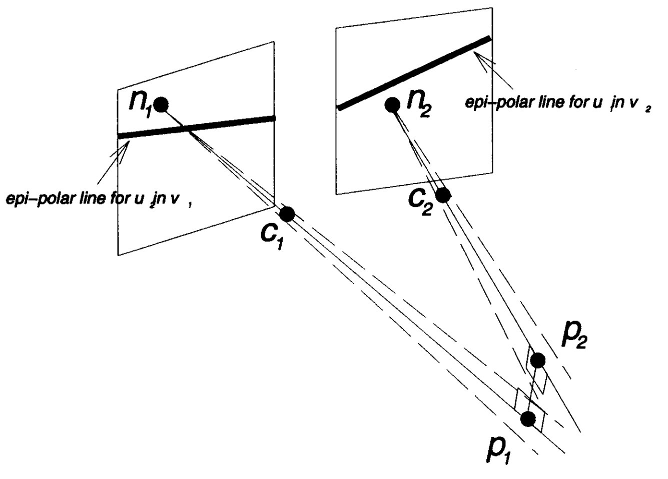


Figure A-16: Triangulation from 2-D film points in two views

possibly skew (i.e. non-intersecting) rays R_1 and R_2 with origins $\mathbf{c}_1$ and $\mathbf{c}_2$ and headings $\mathbf{u}_1$ and $\mathbf{u}_2$. To solve the triangulation problem we find points $\mathbf{y}_1$ and $\mathbf{y}_2$ which are closest to each other yet are free to slide along their respective ray. We can describe this using the following linear relation

$$\left\{\begin{array}{ll} \mathbf{y}_1 = \mathbf{c}_1 + t_1\mathbf{u}_1 & (\mathbf{y}_1 \text{ is a } t_1 \text{ parameterized point on line } R_1) \\ \mathbf{y}_2 = \mathbf{c}_2 + t_2\mathbf{u}_2 & (\mathbf{y}_2 \text{ is a } t_2 \text{ parameterized point on line } R_2) \\ (\mathbf{y}_1 - \mathbf{y}_2)^T\mathbf{u}_1 = 0 & (\text{The point } \mathbf{y}_1 \text{ on } R_1 \text{ which is closest to } R_2 \text{ is the projection of } \mathbf{y}_2 \text{ on } R_1) \\ (\mathbf{y}_1 - \mathbf{y}_2)^T\mathbf{u}_2 = 0 & (\text{Similarly, } \mathbf{y}_2 \text{ is the projection of } \mathbf{y}_1 \text{ on } R_2) \end{array}\right. \tag{A.41}$$

which can be generalized for N lines, to form the equality

$$\begin{bmatrix} bfu_1{}^T bfu_1 & -bfu_1{}^T bfu_2 & 0 & \cdots & 0 \\ bfu_2{}^T bfu_1 & -bfu_2{}^T bfu_2 & 0 & \cdots & 0 \\ bfu_3{}^T bfu_1 & 0 & -bfu_3{}^T bfu_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ bfu_N{}^T bfu_1 & 0 & 0 & \cdots & -bfu_N{}^T bfu_N \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \\ t_3 \\ \vdots \\ t_N \end{bmatrix} = \begin{bmatrix} \mathbf{u}_1{}^T(\mathbf{c}_2 - \mathbf{c}_1) \\ \mathbf{u}_2{}^T(\mathbf{c}_2 - \mathbf{c}_1) \\ \mathbf{u}_3{}^T(\mathbf{c}_3 - \mathbf{c}_1) \\ \vdots \\ \mathbf{u}_N{}^T(\mathbf{c}_N - \mathbf{c}_1) \end{bmatrix} \tag{A.42}$$

To solve this probabilistically, we can consider all $\mathbf{u}_i$ as deterministic, use uncertainties in $\mathbf{c}_i$ to define a weighting matrix and find the weighted least squares solution.

A.15 Edge projection error

In section A.12 we reviewed different methods by which known cameras and surfaces and observed image lines can be used to reconstruct 3-D edges in the scene. In this section we now define a metric for how well this 3-D scene reconstruction agrees with our 2-D observations in the images.

A.15.1 Sum of squared displacements

One way to do this is to define a cost function based on the sum of misregistrations between observed 2-D lines and the projection of their reconstructed 3-D edges in the scene. Consider an observed line segment with endpoints $\mathbf{p}_1$ and $\mathbf{p}_2$ and the projection of its predicted infinite 3-D edge in normal form

$$m_x p_x + m_y p_y + m_z = 0 \tag{A.43}$$

where

$$\mathbf{m} = \begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \begin{bmatrix} n_x \\ n_y \\ -\mathbf{n}^T\mathbf{c} \end{bmatrix} \tag{A.44}$$

since $\mathbf{n}^T(\mathbf{p} - \mathbf{c}\ ) = 0$ in (3.12). As seen in Figure A-17 the projection error at each endpoint

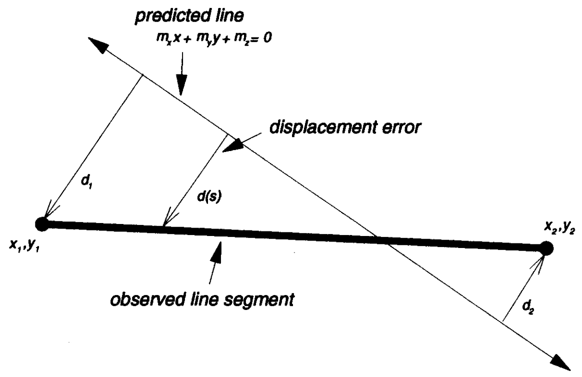


Figure A-17: Projection error of a predicted line

can be defined as the perpendicular distances to the predicted line equation, which are

$$d_1 = \frac{m_x p_{x_1} + m_y p_{y_1} + m_z}{\sqrt{m_x^2 + m_y^2}} \quad d_2 = \frac{m_x p_{x_2} + m_y p_{y_2} + m_z}{\sqrt{m_x^2 + m_y^2}} \tag{A.45}$$

Since the displacement between lines itself is linear, we can represent the displacement error at any point along the observed segment as a function $d(\cdot)$ which varies according to a scalar parameter $s = [0, 1]$

$$d(s) = d_1 + s(d_2 - d_1) \tag{A.46}$$

The sum of squared errors along the length of the observed line is then

$$\begin{aligned} SSE &= \int_0^1 d^2(s) ds \\ &= \frac{1}{3}({d_1}^2 + d_1 d_2 + {d_2}^2) \\ &= \mathbf{m}^T(\mathbf{A}^T\mathbf{B}\mathbf{A})\mathbf{m} \end{aligned} \tag{A.47}$$

where

$$m = [m_x m_y m_z]^T \quad A = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix} \quad B = \frac{1}{3\sqrt{m_x^2 + m_y^2}} \begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & 1 \end{bmatrix} \tag{A.48}$$

As Taylor and Kriegman [38] note, this error metric has the desirable quality that square errors are weighted by line length so any minimizing solution will tend to prefer longer lines whose positional accuracy is greater than shorter lines.

A.15.2 Weighted square displacement error

Another metric uses the covariance $\mathbf{\Lambda_n}$ of the projection normal $\mathbf{n}$ of an observed line segment (A.7). Given a predicted scene edge E_{pred} with direction vector $\mathbf{u}_{pred}$ and edge point $\mathbf{x}_{pred}$ we form the unit projection plane normal $\mathbf{n}_{pred}$

$$\mathbf{n}_{pred} = \mathbf{u}_{pred}{}^T(\mathbf{c} - \mathbf{x}_{pred})/\|\cdot\| \qquad (A.49)$$

the error metric is now the weighted square error between the observed and predicted projection normals

$$SSE = (\mathbf{n}_{pred} - \mathbf{n})^T \mathbf{\Lambda_n}^{-1}(\mathbf{n}_{pred} - \mathbf{n}) \qquad (A.50)$$

Since line length and endpoint pixel noise are embedded within $\mathbf{\Lambda_n}$ it can be argued that this error metric has the same desirable properties as (A.47) with the added benefit that little extra processing is required.

A.16 Planning your own on-site photo shoot

`sceneBuild` was written to work with existing photographs, images from magazines, architectural visualizations, or even hand sketched drawings. But if you wish to model an existing room or building with your own camera you may opt to take certain liberties to insure maximum accuracy and quality.

When modeling an interior space, take 4 photos with a distortionless short focal length lens (i.e. less than 30 mm) from each corner of the room facing the opposite corner from as high a vantage point as possible to maximize 3-pt perspective. Take care that one or more wall and floor features (i.e. decorations, marks, wall sockets) are in common between each neighboring pair of images. For best geometric accuracy keep the focal length constant for all views and to keep the entire scene in focus use a small aperture (i.e. approaching the pinhole situation). Small apertures require longer exposure times so in these situations mount the camera on a tripod and use a remote exposure switch to reduce vibration. If using a video camera, turn the autofocus off. To achieve an optimal blend in the reconstructed textures, surfaces must be static, opaque (non-transparent) and matte (non-glossy and non-reflecting). In addition, the lighting must be kept constant throughout the photo shoot. Don't let your shadow be visible in any view. Keep the aperture fixed so all exposures are identical. If using a video camera, turn the autogain off.

Corner shots are used to determine the camera's focal length and other internal parameters as well as overall room geometry. If the focal length is fixed or the camera parameters have already been found, then straight-on views can also be taken on each wall or surface to get high resolution texture information for planar surfaces.

Non-planar amorphous objects can be modeled by sceneBuild but only if two or more self-calibratable views are given and if surface points are manually specified in those two views.

A.17 Interpretation of errors

One important problem that needs to be addressed is how do we judge the success of the work. In photogrammetry this judgement is made by comparing known euclidean distances and angles to estimated values. In digital video coding we try to use our rough knowledge of the frequency characteristics of the human visual response to manipulate the coder's frequency response so errors resulting from compression are hidden from the observer.

Crude analytical error measures like root mean square (RMS) error are often used to quantify coder "lossy"-ness; however, all analytical image quality measures are only approximations to and are not successful against real subjective evaluations[100].

Indeed, researchers have explored the limits of the human visual system with regard to detectability of changes in images over time. Given a set of images of an altered scene even substantial structural changes are undetectable by the human visual system if the eye's difference mechanism can be suppressed. These psychovisual experiments have shown that significant structure, and even content changes, can be made to static images without the observer being able to detect these changes[5]. However, when the image difference mechanism is in place the rapid spatiotemporal discontinuities are easily noticed and the attention is drawn there.

This suggests that when attempting to model realistic scenes from photographic or video sources topological correctness may be more important than real-world geometric truth and fluidity may be more important than topological correctness. If an animated sequence made from a topologically correct and realistically textured synthetic model does not violate physics of the real world or offend human visualization expectations (e.g. like seeing empty cracks between surfaces), then the viewer would most likely be unable to point out the difference between the synthetic and real video sequences of a perfectly static scene.

A taxonomy of errors for vision assisted modeling

Just so we can see what we're up against here we list the source of errors that arise when we attempt to recover surface texture and structure from uncalibrated images:

I. Shape failure. The assumptions we make about geometric relationships among features and textures are in fact false.

- points or lines are erroneously matched
- points, lines or textures are not strictly planar causing self occlusion and disparity
- lines are not truly parallel
- angles between lines not perpendicular
- distances between points are incorrect
- lens model does not adequately model true distortion regardless of the number of parameters

[5]Experiments show that gross structural changes can be made to a scene such as changing the height of a fence, changing the position of a desk, or replacing grapes with oranges without the observer noticing the change if the difference mechanism is suppressed

II. Sensor limits. There is a limit to accuracy in parameter estimation due to sensor limitations.

- finite limit on image resolution
- partial pixel effect
- noisy feature detection/specification

III. Unmodeled elements and effects. This describes errors which remain if structural assumptions and measurements of modeled elements are correct.

- lens effects (glare, lens flare, autogain, vignetting)
- environmental lighting
- view varying texture (non-diffuse effects like specularity and transparency)
- things (chair, person)
- limits in detail: i.e. every single surface can't be modeled
- limits in shape: i.e. every shape can't be modeled

IV. Pixel noise. Even if the synthetic scene is exact, we will see differences between the synthetic and real image.

- imaging effects (glare, autogain, vignetting)
- noise in pixel color

Bibliography

[1] R.Y. Tsai, An Efficient and Accurate Camera Calibration Technique for 3-D Machine Vision, *Proceedings IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Miami Beach, Florida, pp. 364-374, June 22-26, 1986.

[2] R.K. Lenz and R.Y. Tsai, Techniques for Calibration of the Scale Factor and Image Center for High Accuracy 3-D Machine Vision Metrology, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 19, No. 5, September 1988.

[3] D. Zhang, Y. Nomura, and S. Fujii, A simple and accurate camera calibration method, *Proceedings of the SPIE*, Vol. 1822, pp. 139-148, 1992.

[4] Z.C. Lai, On the sensitivity of camera calibration, *Proceedings of the SPIE*, Vol. 1822, 1992.

[5] Y. Nomura, M. Sagara, Hiroshi Naruse, and Atsushi Ide, Simple Calibration Algorithm for High-Distortion-Lens Camera, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 14, No. 11, pp. 1095-1099, November 1992.

[6] S. Shah and J.K. Aggarwal, A Simple Calibration Procedure for Fish-Eye (High Distortion) Lens Camera, *Proceedings 1994 IEEE International Conference on Robotics and Automation* San Diego, California, pp. 3422-3427, May 8-13, 1994.

[7] S.W. Shih, Y.P. Hung, and W.S. Lin, When should we consider lens distortion in camera calibration, *Pattern Recognition*, Vol. 28. No. 3, pp. 447-461, 1995.

[8] G.P. Stein, Internal Camera Calibration using Rotation and Geometric Shapes, *Thesis for Master of Science in EECS at MIT*, February 1993.

[9] O.D. Faugeras, What can be seen in three dimensions with an uncalibrated stereo rig? *Proceedings Computer Vision EECV'92 Second European Conference on Computer Vision*, p. xv+909, 563-78, Santa Margherita Ligure, Italy, May 18-23, 1992.

[10] O.D. Faugeras, Stratification of three-dimensional vision: projective, affine, and metric representations, *Journal Optical Society of America*, Vol 12, No. 3, pp. 465-484, March 1995.

[11] A. Shashua and N. Navab, Relative affine structure: theory and application to 3D reconstruction from perspective views, In *Proceedings IEEE Conference on CVPR*, Seattle, Washington, (1994).

[12] B. Caprile and V. Torre, Using vanishing points for camera calibration, *IJCV*, Vol. 4, pp. 127-140, 1990.

[13] P. Parodi and G. Piccioli, "3D Shape Reconstruction by Using Vanishing Points", *IEEE Trans. on Pattern Anal. and Mach. Intel.*, Vol. 18, No. 2, pp. 211-217, February 1996.

[14] L.L. Wang and W.H. Tsai, Computing Camera Parameters using Vanishing-Line Information from a Rectangular Parallelipiped, *Machine Vision and Applications*, **3**:129-141, 1990.

[15] Y. Lui and T.S. Huang, Motion estimation from corner correspondences, *Proceedings International Conference on Image Processing*, Singapore, September 5-8, pp. 785-790, 1989.NEED

[16] S.C. Becker and V.M. Bove Jr., Semiautomatic 3-D model extraction from uncalibrated 2-D views, *Proceedings SPIE Visual Data Exploration and Analysis II*, Vol. 2410, pp. 447-461, San Jose, California, February 8-10, 1995.

[17] D.H. Ballard, Generalizing the Hough Transform to Detect Arbitrary Shapes, 1980.CHECK

[18] R.T. Collins and R.S. Weiss, Vanishing Point Calculation as a Statistical Inference on the Unit Sphere, *Machine Vision and Applications*, **3**:400-403, 1990.

[19] E. Lutton, H. Maitre, and J. Lopez-Krahe, Contribution to the Determination of Vanishing Points Using Hough Transform, *IEEE PAMI*, Vol. 16, No. 4. April, 1994, pp. 430-438.

[20] K. Kanatani, Statistical foundation for hypothesis testing of image data, *CVGIP: Image Understanding*, Vol. 60, No. 3, November, pp. 382-391, 1994.

[21] K. Kanatani, Statistical analysis of focal-length calibration using vanishing points, *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 6, December, 1992, pp. 767-775.

[22] K. Kanatani, Computational Cross Ratio for Computer Vision, *CVGIP: Image Understanding*, Vol. 60, No. 3, November, pp. 371-381, 1994.

[23] D. C. Brown, Close-range camera calibration, Presented at the *Symposium on close-range photogrammetry*, Urbana, Illinois, January, 1971.

[24] A. Azarbayejani and A. Pentland, "Recursive estimation of motion, structure, and focal length", *IEEE Pattern Analysis and Machine Intelligence*, April, 1994.

[25] B. Horowitz, A. Azarbayejani, T. Galyean, and A. Pentland, "Models from Video", *MIT Media Laboratory Vision and Modeling Technical Report #207*

[26] A. Azarbayejani and A. Pentland, Camera self-calibration from one point correspondence, *MIT Media Laboratory, Perceptual Computing Technical Report #341 Submitted to the IEEE Symposium on Computer Vision*, April, 1995.

[27] R. Szeliski and S.B. Kang, "Recovering 3d shape and motion from image streams using non-linear least squares", *CRL 93/3, Digital Equipment Corporation*, Cambridge Research Labs, Cambridge, MA, March 1993.

[28] R. Szeliski, "Video mosaics for virtual environments", *IEEE Computer Graphics and Applications*, pp. 22-30, March 1996.

[29] T.S. Huang and A.N. Netravali, Motion and Structure from Feature Correspondences: A Review, *Proceedings of the IEEE*, Vol. 82, No. 2, February, pp. 252-268, 1994.

[30] H.C. Longuet-Higgens, A computer algorithm for reconstructing a scene from two projections, *Nature*, **293**(10), pp. 133-135, 1981.

[31] A. Kara, K.M. Wilkes and K. Kawamura, 3D Structure Reconstruction from Point Correspondences between Two Perspective Projections, *CVGIP: Image Understanding*, Vol. 60, No. 3, November, pp. 392-397, 1994.

[32] S. Ullman, *The Interpretation of Visual Motion*, The MIT Press, Cambridge, Ma, 1979.

[33] Carlo Tomasi and Takeo Kanade, Shape and motion from image streams: a factorization method; full report on the orthographic case, *International Journal of Computer Vision*, 9(2):127-154, November, 1992.

[34] Larry Matthies and Takeo Kanade, "Kalman Filter-based Algorithms for Estimating Depth from Image Sequences", *International Journal of Computer Vision*, Vol. 3, pp. 209-236, 1989.

[35] N. Navab, O.D. Faugeras and T. Vieville, The Critical Sets of Lines for Camera Displacement Estimation: A Mixed Euclidean-Projective and Constructive Approach, *ICCV*, 1993.

[36] A. Pentland, A. and S. Sclaroff, Closed-Form Solutions For Physically Based Shape Modeling and Recognition, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol. 13, No. 7, pp. 715-730, 1991.

[37] D.J. Heeger and J.R. Bergen, Pyramid-Based Texture Analysis/Synthesis, *Computer Graphics Proceedings Siggraph*, Los Angeles, April, pp. 229-238, 1995.

[38] C.J. Taylor and D.J. Kriegman, Structure and Motion from Line Segments in Multiple Images, *Yale University Technical Report No. 0402b*, January 1994.

[39] A. Gelb, *Applied Optimal Estimation*, MIT Press, Cambridge, 1989.

[40] T.J. Broida, S. Chandrashekhar, and R. Chellappa, Recursive 3-D motion estimation from a monocular image sequence, *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 16, No. 4, July, 1990.

[41] N.Cui, J.J. Weng, and P. Cohen, Recursive-Batch Estimation of Motion and Structure from Monocular Image Sequences, *CVGIP: Image Understanding*, Vol. 59, No. 2, March, pp. 154-170, 1994.

[42] S. Mann and S. C. Becker, "Computation of some projective-chirplet-transform and metaplectic-chirplet-transform subspaces, with applications in image processing", *Proceedings DSP World Symposium*, Boston, Massachusetts, November, 1992.

[43] T. Melen, "Extracting physical camera parameters from the 3 by 3 direction linear transformation matrix", *Optical 3-D Measurement Techniques II: Applications in in-*

spection, quality control and robotics, Wichman Pub., edited by A. Gruen and H. Kahmen, pp. 355-365, 1993.

[44] Stephen T. Barnard, "Stochastic Stereo Matching over Scale", *IJCV*, 3, 17-32, 1989.

[45] Takeo Kanade, Masatoshi Okutomi, and Tomoharu Nakahara, "A Multiple-baseline Stereo Method", *Image Understanding Workshop*, DARPA92, 409-426, 1992.

[46] Reinhard Koch, "Automatic Reconstruction of Buildings from Stereoscopic Image Sequences", *Eurographics '93*, Vol. 12, no. 3, 339-350, 1993.

[47] J.P. Mellor, Seth Teller. and Tomas Lozano-Perez, "Dense Depth Maps from Epipolar Images", *MIT AI Laboratory*, AI memo 1593, November 1996.

[48] Satyan Coorg and Seth Teller, "Matching and Pose Refinement with Camera Pose Estimates", *MIT Laboratory for Computer Science*, MIT-LCS-TM-561, 1996.

[49] R. Y. Tsai and T. S. Huang, Estimating Three-Dimensional Motion Parameters of a Rigid Planar Patch, *IEEE Transactions on Acoustics, Speech and Signal Processing*, Vol. ASSP-29, No. 6, December, pp. 1147-1152, 1981.

[50] K. Prazdny, On the Information in Optical Flows, *Computer Vision, Graphics and Image Processing*, **22**, pp. 239-259, 1983.

[51] Barron, Fleet and Beauchemin, Systems and Experiment: Performance of Optical Flow Techniques, *International Journal of Computer Vision*, 12:1, pp. 43-77, 1994.

[52] G. Adiv, Determining Three-Dimensional Motion and Structure from Optical Flow Generated by Several Moving Objects, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No. 4, pp. 384-401, July 1985.

[53] B. K. P. Horn, *Robot Vision*, MIT Press, 1986.

[54] S. Negahdaripour, and B. K. P. Horn, Direct Passive Navigation, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9:1, January 1987.

[55] Berthold K.P. Horn and E.J. Weldon Jr., Direct methods for recovering motion, *International Journal of Computer Vision*, 2, 51-76 (1988).

[56] Berthold K.P. Horn, Relative orientation, *International Journal of Computer Vision*, 4, 59-78 (1990).

[57] Berthold K.P. Horn, Closed-form of absolute orientation using unit quaternions, *International Journal of Computer Vision*, 4, 629-641 (1987).

[58] David Heeger and Allan Jepson, Simple method for computing 3D motion and depth, *Proceedings 3rd IEEE International Conference on Computer Vision*, pp. 96-100, 1990.

[59] A.F. Bobick, Using Stability of Interpretation as Verification for Low Level Processing: An example from ego-motion and optic flow, *IEEE Conference on Computer Vision and Pattern Recognition*, New York, June 1993.

[60] T. Marill, Emulating the Human Interpretation of Line-Drawings as Three-Dimensional Objects, IJCV, 6:2, pp. 147-161, 1991.

[61] Yvan G. Leclerc and Martin A. Fischler, An Optimization-Based Approach to the Interpretation of Single Line Drawings as 3D Wire Frames, *IJCV*, 9:2, 113-136 (1992)

[62] R. Nevatia, M. Zerroug, and F. Ulupinar, Recovery of Three-Dimensional Shape of Curved Objects from a Single Image, /em Handbook of Pattern Recognition and Image Processing: Computer Vision, Academy Press, pp. 101-129, 1994.

[63] M.A. Fischler and R.C. Bolles, Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography, *Communications of the ACM*, Vol. 24, No. 6, pp. 381-395, June 1981.

[64] R.M. Haralick, C.N. Lee, K. Ottenberg, M. Nolle, Review and Analysis of Solutions of the Three Point Perspective Pose Estimation Problem, *IJCV*, Vol. 13, No. 3, pp. 33-356, 1994.

[65] Y. Liu, T.S. Huang, and O.D. Faugeras, Determination of camera location from 2D to 3D line and point correspondences, *Proceedings CVPR*, Ann Arbor, MI, June, 1988.NEED

[66] R. Kumar and A. R. Hanson, Robust Methods for Estimating Pose and a Sensitivity Analysis, *CVGIP: Image Understanding*, Vòl. 60, No. 3, November, pp 313-342, 1994.

[67] J.I. Par, N. Yagi, K. Enami, K. Aizawa, and M. Hatori, Estimation of Camera Parameters from Image Sequence for Model-Based Video Coding, *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 4, No. 3. June 1994.

[68] J.D. Foley, A. Van Dam, S.K. Feiner and J.F. Hughes, *Computer Graphics Principles and Practice*, 2nd Ed., Addison-Wesley, New York, 1990.

[69] R. Forchheimer and O. Fahlander, Low bit-rate coding through animation, in *Proceedings Picture Coding Symposium (PCS-83)*, Davis, pp. 113-114, March 1983.NEED

[70] H. Li, P. Roivainen and R. Forchheimer, 3-D Motion Estimation in Model-Based Facial Image Coding, *IEEE Transactions on PAMI*, Vol. 15, No. 6, pp. 545-555, June 1993.

[71] I.A. Essa, Analysis, Interpretation and Synthesis of Facial Expression, *MIT Media Laboratory Ph.D. Thesis*, February 1995.

[72] E.H. Adelson, and J.R. Bergen, "The Plenoptic Function and the Elements of Early Vision", *Computational Models of Visual Processing*, Chapter 1, Edited by M. Landy and J. A. Mavson, The MIT Press, Cambridge, Mass, 1991.

[73] L. McMillan and G. Bishop, Plenoptic Modeling: An Image-Based Rendering System, Proceedings *ACM Siggraph '95*, pp. 39-46, 1995.

[74] S. Mann and R. Picard, "Video Orbits of the Projective Group: A New Perspective on Image Mosaicing", *MIT Media Laboratory Perceptual Computing Section Technical Report No. 338.*

[75] L. Teodosio, "Salient Stills", *MIT Media Laboratory Masters Thesis*, 1992.

[76] J. Wang, T. Adelson and U. Desai, Applying Mid-level Vision Techniques for Video and Data Compression and Manipulation, *Proceedings of the SPIE: Digital Video and*

Data Compression on Personal Computers: Algorithms and Technologies, Vol. 2187, San Jose, February 1994.

[77] M. Irani, S. Hsu, P. Anandan, Mosaic-based video compression, *Proceedings SPIE: Digital Video Compression*, Vol. 2419, pp. 242-253, San Jose, California, February 8-10, 1995.

[78] R.G. Kermode, Building the BIG Picture: Enhanced Resolution from Coding, *MIT Media Laboratory Masters Thesis*, June 1994.

[79] J. Canny, A Computational Approach to Edge Detection, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-8, No. 6. Nov. (1986), pp. 679-698.

[80] Stefan Agamanolis, "Line-finder man pages", *MIT Media Lab TVOT man pages*, December, 1995.

[81] Stefan Panayiotis Agamanolis, "High-level scripting environments for interactive multimedia systems", *MIT Media Lab*, Masters Thesis, February 1996.

[82] H. Holtzman, Three-Dimensional Representations of Video using Knowledge-Based Estimation, *MIT Media Laboratory Masters Thesis*, 1991.

[83] W.H. Press, B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical recipes in C*, Cambridge University Press, Cambridge, UK, 1988.

[84] W. T. Freeman, "Steerable Filters and Local Analysis of Image Structure", Ph.D. Thesis, Massachusetts Institute of Technology, 1992. Also available as Vision and Modeling Technical Report 190, MIT Media Laboratory.

[85] C. Wren, A. Azarbayejani, T. Darrell, A. Pentland, Pfinder: Real-Time Tracking of the Human Body, *MIT Media Laboratory Perceptual Computing Section*, Technical Report No. 353, 1995.

[86] W. E. L. Grimson, D. P. Huttenlocher and T. D. Alter, "Recognizing 3D Objects from 2D Images: An Error Analysis", *MIT AI Laboratory*, A.I. Memo No. 1362, July 10, 1992.

[87] W. Eric L. Grimson and David Huttenlocher, On the Sensitivity of the Hough Transform for Object Recognition, *MIT A.I. Memo No. 1044*, May 1988.

[88] K. Nagao and W. E. Grimson, Object Recognition by Alignment using Invariant Projections of Planar Surfaces, *MIT AI Laboratory*, A.I. Memo No. 1463, February, 1994.

[89] K. Nagao and W. E. Grimson, "Recognizing 3D Objects Using Photometric Invarient", *MIT AI Laboratory*, A.I. Memo No. 1523, February, 1995.

[90] K. Nagao and B. K. P. Horn, "Direct object recognition using no higher than second or third order statistics of the image", *MIT AI Laboratory*, A.I. Memo No. 1526, February, 1995.

[91] P. A. Viola, Alignment by Maximization of Mutual Information, *MIT AI Laboratory*, A.I. Technical Report No. 1548, June, 1995.

[92] J. A. Provine and L. T. Bruton, "3-D Model Based Coding - A Very Low Bit Rate Coding Scheme for Video-conferencing", to appear in *Proc. IEEE Intl. Symp. on Circuits and Systems*, 1996.

[93] S. Ullman, An Approach to Object Recognition: Aligning Pictorial Descriptions, *AI Memo, No. 931*, December 1986.

[94] Tanveer Fathima Syeda-Mahmood, "Data and Model-driven Selection using Color Regions", *CVVV'92*, pp. 115-123, 1992. May 1993.

[95] S. Tanveer F. Mahmood, "Data and Model-driven Selection using Parallel-Line Groups", *MIT A.I. Memo, No. 1399*, May 1993.

[96] W.K. Pratt, O.D. Faugeras and A. Gagalowicz, "Applications of Stochastic Texture Field Models to Image Processing", *Proc. of the IEEE*, Vol. 69, No. 5, May 1981.

[97] K. Popat and R.W. Picard, "Novel cluster-based probability model for texture synthesis, classification, and compression", *MIT Media Lab Perceptual Computing Group Technical Report #234*, November 1993.

[98] P.E. Debevec, C. J. Taylor and J. Malik, "Modeling and Rendering Architecture from Photographs", *Technical Report UCB/CSD-96-893*, January 1996.

[99] S. Ayer, P. Schroeter and J. Bigun, "Segmentation of moving objects by robust motion parameter estimation over multiple frames", *CVVP-94*, Vol. 801, pp. 316-327.

[100] R. E. Grant, A. B. Mahmoodi, O. L. Nelson, "Chapter 11: Image Compression and Transmission", *Imaging Processes and Materials*, pp. 323-349.

[101] M. Levoy, "Polygon-Assisted JPEG and MPEG Compression of Synthetic Images", *SIGGRAPH*, pp. 21-28, 1995.

[102] Roberts, Blocks World.

[103] Ganapathy.

[104] Guzman and Falk.

[105] Clowes.

[106] Huffman.

[107] V. M. Bove, Jr., B. D. Granger, and J. A. Watlington, Real-Time Decoding and Display of Structured Video, *Proceedings IEEE ICMCS '94*, Boston MA, (1994), pp. 456-462.

[108] http://cheops.www.media.mit.edu/projects/cheops/

[109] Neil D. McKay, Using Quaternions for least-squares fitting of rotation transformations, *General Motors Research Laboratory Research Report*, CS-518, November 3, (1986).

[110] A. Azarbayejani and A. Pentland, Recursive estimation of motion, structure, and focal length, (in press) *IEEE Pattern Analysis and Machine Intelligence*, April (1994).

[111] Athanasios Papoulis, *Probability, Random Variables, and Stochastic Processes*, Third ed., McGraw Hill, New York, page 156, (1991).

[112] H. F. Durrant-Whyte, Uncertain Geometry, chapter of *Geometric Reasoning*, MIT Press, ed. D. Kapur and J. L. Munday, pp. 447-481, (1989).

[113] Alan V. Oppenheim and Ronald W. Schafer, *Discrete-time signal processing*, Prentice Hall, Englewood Cliffs, New Jersey, (1989).

[114] J. T. Tou and R. C. Gonzalez, *Pattern Recognition Principles*, Addison-Wesley, London, 1974.

[115] Richard A. Johnson and Dean W. Wichern, *Applied Multivariate Statistical Analysis*, Ed.2, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[116] Richard P. Paul, *Robot Manipulators: Mathematics, Programming and Control*, MIT Press, 1981.

[117] Stephen A. Benton, Pierre St.-Hilaire, Mark Lucente, John D. Sutter, and Wendy J. Plesniak, "Real-time computer-generated 3-D holograms," in: SPIE Proc. Vol. 1983: 16th Congress of the International Commission for Optics-Optics as a Key to High Technology (Budapest, Hungary, 9-13 August 1993) (SPIE, Bellingham, WA, 1993), pp. 536-543.

[118] Stephen A. Benton, "The second generation of the MIT holographic video system," in: J. Tsujiuchi, J. Hamasaki, and M. Wada, eds., Proc. of the TAO First International Symposium on Three Dimensional Image Communication Technologies (Tokyo, Japan, 6-7 December 1993) (Telecommunications Advancement Organization of Japan, Tokyo, 1993), pp. S-3-1-1 to -6.

[119] J. A. Watlington, Mark Lucente, C. J. Sparrell, V. M. Bove, I. Tamitani."A hardware architecture for rapid generation of electro-holographic fringe patterns" SPIE Proc. Vol. #2406: Practical Holography IX, 2406-23, (SPIE, Bellingham, WA, 1995).

[120] http://www.cs.Berkeley.EDU/s̃equin/soda/soda.html

[121] http://www.portola.com:80/TR/background.html

[122] http://vrml.wired.com/concepts/raggett.html

[123] http://quasar.poly.edu/l̃lin/GISS/sessions/SecII-A.html

[124] http://vision.ucsd.edu/

[125] http://sdsc.edu/SDSC/Partners/vrml/doc.html

[126] http://vrml.wired.com/future/scale.html

[127] http://www.photomodeler.com

[128] http://garden-docs.www.media.mit.edu/garden_docs/gnu/readline/rlman_toc.html

[129] http://cuiwww.unige.ch/eao/www/TclTk.html

[130] http://www.osf.org/motif/Motif20/index.html

[131] http://www.sgi.com/Technology/Inventor/

[132] file:///mas/garden/grad/bill/images/modeling/HowTo.html

[133]
file:///mas/garden/urop/elin/Projects/ModelMaker/Documentation/ModelMaker_docs.html

[134] http://www.microsoft.com/intdev/avr/avwhite.htm

[135] http://webspace.sgi.com/moving-worlds