

March 2003
IEEE Virtual Reality 2003 tutorial 1:

Recent Methods for Image-based Modeling and Rendering

Organizer:
Martin Jagersand
jag@cs.ualberta.ca

Authors:

Darius Burschka
Johns Hopkins University

Dana Cobzas
University of Alberta

Zach Dodds
Harvey Mudd College

Greg Hager
Johns Hopkins University

Martin Jagersand
University of Alberta

Keith Yerex
Virtual Universe Corporation

IEEE Virtual Reality 2003 tutorial 1:

Recent Methods for Image-based Modeling and Rendering

Biltmore hotel, Athenian Room
Saturday, March 22nd 2003

Schedule

7:30	Continental breakfast
8:00	Introduction and Survey Martin Jagersand
9:00	The Geometry of Image Formation Zach Dodds
10:00	Coffee break
10:30	Geometric Modeling from Images Dana Cobzas
12:00	Lunch
13:30	Texturing from Images Martin Jagersand
14:15	Real-time Visual Tracking Darius Burschka
15:00	Coffee break
15:30	Hardware Accelerated Rendering Keith Yerex
16:00	Laboratory and Individual Questions/Answers with lecturers
17:00	Tutorial ends

Abstract

A long standing goal in image-based modeling and rendering is to capture a scene from camera images and construct a sufficient model to allow photo-realistic rendering of new views. With the confluence of computer graphics and vision, the combination of research on recovering geometric structure from un-calibrated cameras with modeling and rendering has yielded numerous new methods. Yet, many challenging issues remain to be addressed before a sufficiently general and robust system could be built to e.g. allow an average user to model their home and garden from cam-corder video.

This tutorial aims to give researchers and students in computer graphics a working knowledge of relevant theory and techniques covering the steps from real-time vision for tracking and the capture of scene geometry and appearance, to the efficient representation and real-time rendering of image-based models. It also includes hands-on demos of real-time visual tracking, modeling and rendering systems.

Web Sites:

<http://www.cs.ualberta.ca/~vis/VR2003tut>

<http://www.cs.jhu.edu/CIRL/XVision2/>

Contents

1	A Survey of Image-based Modeling and Rendering		1
	Dana Cobzas and Martin Jagersand		
1.1	Modeling	1	
1.2	Image and View Morphing	2	
1.3	Interpolation from Dense Samples	3	
1.4	Pixel Reprojection using Scene Geometry	4	
1.4.1	Image mosaics	4	
1.4.2	Depth based reprojection	6	
1.5	Summary	7	
2	The Geometry of Image Formation		8
	Zach Dodds		
2.1	Camera Model	9	
2.2	2d Projective Geometry	11	
2.3	3d Projective Geometry	15	
3	Multiple View Geometry and Structure-From-Motion		19
	Dana Cobzas and Martin Jagersand		
3.1	Camera models	19	
3.2	Computation with Camera Models	24	
3.3	Two view geometry	26	
3.4	Multi view geometry	31	
3.5	Recovering metric structure	33	
3.6	A complete system for geometric modeling from images	35	
4	Texturing of Image-based Models		36
	Martin Jagersand		
4.1	Texture basis	37	
4.2	Connection between intensity change and motion	37	
4.3	Geometric texture variation	39	
4.4	Photometric variation	43	
4.5	Estimating composite variability	43	
4.6	Interpretation of the variability basis	44	
4.7	Examples of Dynamic Texture Renderings	45	
4.8	Discussion	48	
5	Real-Time Visual Tracking		50
	Darius Burschka and Greg Hager		
5.1	Motivation	50	
5.2	Visual Tracking in Monocular Images	50	
5.3	Tracking Moving Objects	52	
5.3.1	Recovering Structured Motion	52	

5.3.2	An Efficient Tracking Algorithm	55	
5.4	Illumination-Insensitive Tracking	58	
5.5	Making Tracking Resistant to Occlusion	60	
5.5.1	Planar Tracking	62	
5.6	Direct Plane Tracking in Stereo Images	63	
5.6.1	Planar Disparities	63	
5.6.2	Plane Tracking	65	
6	Implementation of Real-Time Texture Blending and IBMR systems		
	Keith Yerex and Martin Jagersand		67
6.0.3	Software choices: OpenGL or DirectX	67	
6.0.4	NVIDIA Register Combiners	68	
6.0.5	Summary	70	
6.1	IBMR System	70	
6.1.1	User interface	72	
6.1.2	Video capture and tracking	73	
6.1.3	Structure and texture editor	74	
6.1.4	Real-time renderer	76	
7	Discussion and Outlook		77
A	Biographies		86

1 A Survey of Image-based Modeling and Rendering

Dana Cobzas and Martin Jagersand

The confluence of computer graphics and vision has spurred much research in modeling scene structure and appearance from images, and building image-based models that allow re-rendering of novel views. Image-based modeling and rendering (IBMR) encompass a wide range of theory and techniques. The field is still young, and giving it a precise definition at this point would be premature. However, a common characteristic of IBMR methods is that images play a more central role. While traditionally computer graphics has focussed on transforming 3D data into 2D image projections, image based rendering (IBR) techniques shifts this emphasis to 2D - 2D image transforms. This opens up a span of possibilities, and in the recent literature image-based methods both with and without the use of an explicit 3D scene geometry have been published.

Another reason for the wide range of methods are that IBMR have been developed with different goals in mind. For example, three promising reasons are:

1. Consumer cam-corders and web cams are inexpensive, and ubiquitous. The ability to capture 3D models from cameras would open up the possibility for consumer and small business image editing to transition from 2D to 3D.
2. Despite significant work on both principles and methods of conventional computer graphics, it has been difficult to generate photo-realistic renderings. IBMR holds promise in that starting with photos, it short-cuts many of the difficult steps such as the precise a-priori modeling of surface reflection properties, and exact ray tracing of lighting.
3. Just as for other computational hardware, the performance of graphics cards have increased dramatically over the last decade. Still, the desire for realism and great detail often limit the frame rate of rendering to below real time. Applying IBMR techniques, images of certain canonical views can be pre-computed, and then in real-time warped and interpolated into the desired final video rate view sequence.

The two first of the above share the common feature that images not only are used in the internal representation, but is also the input data from which the graphics model is built. We will focus on this type of techniques in the rest of the paper.

1.1 Modeling

The traditional approach for generating detailed 3D geometric models of the surrounding environment is a challenging problem for computer graphics community. The 3D model can be generated using a CAD modeler or directly from real data (for example range data obtained from range finders or stereo) [13]. The 3D model is usually represented using polygonal, bicubic parametric curves, constructive solid geometry or space subdivision (such as octrees). New views are then rendered directly from this 3D model. The problem with these traditional *geometry based* modeling and rendering systems is that the modeling is slow and hard, and is very difficult to create realism, because the geometry of the objects found in the real world is complex, and because it is very difficult to model lighting and reflection. An alternative approach to these

systems is the *image-based* modeling and rendering system. They combine computer vision and computer graphics algorithms to create a model directly from images and use this representation to generate photorealistic novel views from viewpoints different than the original ones. Using this approach the model is very easy to acquire, and the cost of rendering is independent on scene complexity, because the sample images contain the complexity of the scene. The realism of the rendered images depends on the input images, so they produce photorealistic renderings. One of the biggest problems with image-based rendering systems is that they can not or can hardly accommodate changes in the scene (new objects, changes in lightning). Another problem is that they require a lot of memory because of data redundancy, but considering the evolution of computer systems this is not so challenging in our days.

The idea behind the image-based rendering (IBR) systems is that they try to sample the *plenoptic function* [35] for the desired scene. The plenoptic function represents *the intensity of the light rays passing through the camera center at every location, at every possible viewing angle* (see figure 1). So, in general it is a 5D function, but depending on the scene constraints it can have few degrees of freedom. There are three distinct categories of IBR techniques, depending on the modality of encoding the plenoptic function and pixel transfer. They are: *image and view morphing*, *interpolation from dense samples* and *pixel reprojection using scene geometry* that includes the *image mosaics*. In the sections below I will briefly describe each of these categories.

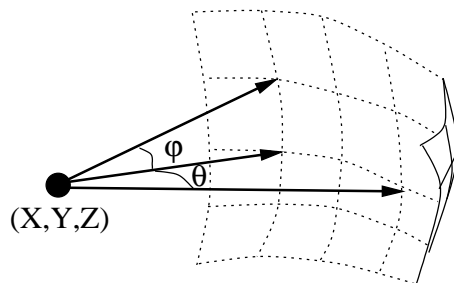


Figure 1: Plenoptic function

1.2 Image and View Morphing

Image morphing techniques generate intermediate views by image interpolation without considering the geometry of the scene. The new views are therefore geometrically incorrect (see figure 2). One of the best known techniques is Beier and Neely's feature based image metamorphosis [1]. They used corresponding line segments to smoothly transform a source image into a destination image using simple pixel interpolation. A more advanced technique, called view morphing, was developed by Seitz and Dyer [45]. They generate new views of a scene that represent a physically-correct transition between two reference views. The motion of the camera is restricted to a line that connects the two centers of projection. The intermediate views are created by first prewarping the images in order to align the projection planes, then morphing using interpolation, and then postwarping the interpolated images. Manning and Dyer extend this idea by creating dynamic view morphing [32]. A similar approach, called view interpolation was created by Chen and Williams [5]. Arbitrary viewpoints with constraints on the viewing angle are generated by

interpolating images stored at nearby viewpoints. Their technique requires calibration and full correspondence map of sample images. This is very difficult to obtain from real images, so their algorithm was tested with synthetic images.

Image and view morphing techniques are very fast and simple, but they produce nonrealistic and geometrically incorrect renderings approximated from the sample images. We cannot use these techniques for robot navigation where the modeled environment must be geometrically correct.

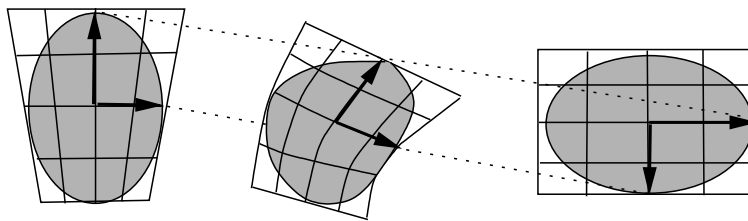


Figure 2: Interpolation can bring incorrect views

1.3 Interpolation from Dense Samples

This class of methods first build up a lookup table by taking many image samples of an object or a scene and then reconstruct images from arbitrary viewpoints by interpolating the stored table. This lookup table is an approximation of the plenoptic function. One of the biggest advantages of these methods is that pixel correspondence is not necessary and they are fast, but they require extensive data acquisition, high memory requirements and knowledge about the camera viewpoint during data acquisition. That is why they are more suitable for synthetic scenes.

Two examples of this approach have been presented in SIGGRAPH'96: *light field rendering* [31] and the *lumigraph* [15]. They both use a 4D parameterization of the plenoptic function if the scene is restricted to a bounding box (see figure 3a). The interpolation scheme used by Levoy and Hanrahan [31] approximates the resampling process by interpolating the 4D function from nearest samples. Lumigraph [15] is reconstructed as a linear sum of the product between a basis function and the value at each grid point.

More recently Shum and He [47] presented a 3D plenoptic function called *concentric mosaics*. In this case the camera motion is restricted to planar concentric circles and concentric mosaics are created by composing slit images taken at different locations. Novel views are rendered by combining the appropriate captured rays (see figure 3b). A similar approach is described in [40] where a *stereo panorama* is created using a camera with one left and one right slit that is rotating along a circle.

For this class of techniques the workspace is limited within the space defined by the samples, a cube for the lightfield and lumigraph, and a disc for concentric mosaics, so they are not suitable for our application of robot navigation.

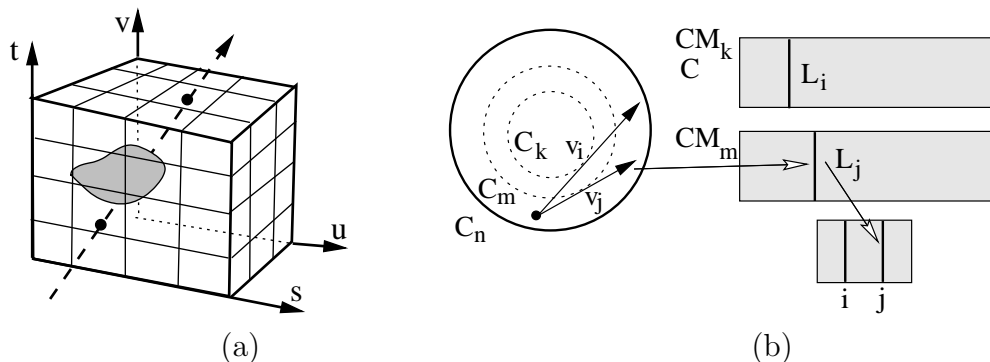


Figure 3: (a) Lumigraph and lightfield parametrisation (b) Concentric mosaics

1.4 Pixel Reprojection using Scene Geometry

This class of techniques use relatively small number of images with the application of geometric constraints to reproject image pixels at a given camera viewpoint. Some of the most used constraints are: depth or disparity values, epipolar constraint, and trilinear tensor. The rendered images are geometrically correct, but most of the methods require sparse or full disparity or depth map which is very difficult to recover from real images. I will first talk about the image mosaics, which are one of the most primitive image-based models, and then about the other techniques that use depth or disparity in order to produce the image-based model.

1.4.1 Image mosaics

One of the simplest image-based modeling techniques is image mosaicing. The term “mosaic” refers to the combination of at least two images to yield a higher resolution or larger image. This is a very old technique and was developed long before the age of digital computers. It appeared shortly after the photography was invented in 19th century, when images acquired from balloons or tops of the mountains were manually pieced together to create maps. Today mosaics are used in many applications like whiteboard and document scanning as an aid to video conferencing, reconstructing 3D scenes from multiple nodes [51, 52, 48], video compression [25], architectural walkthroughs, virtual museums, cartoons [61], telepresence, tele-medicine, and astronomy.

In order to register the images onto a bigger one they must be related by a linear projective transformation (homography). This is possible when the images sample a planar surface, or are taken from the same point of view. The general form of a projective transformation is:

$$s \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{H}_{11} & \mathbf{H}_{12} & \mathbf{H}_{13} \\ \mathbf{H}_{21} & \mathbf{H}_{22} & \mathbf{H}_{23} \\ \mathbf{H}_{31} & \mathbf{H}_{32} & \mathbf{H}_{33} \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

where (u, v) and (u', v') are corresponding pixels in two images, s is a scale factor and \mathbf{H} is a non-singular matrix (defined up to a scale factor).

Creating an image mosaic involves three problems: **projecting** the images on the desired surface (planar, cylindrical, spherical), and correcting geometric deformations caused by different

types of lenses, **registering** the images into a common coordinate system, and **correcting errors** resulting from the registration process.

There are different types of image projection depending on the desired application and acquisition technique. The simplest set of images to mosaic are pieces of a *planar* scene such as a document or a whiteboard. As mentioned before, these pieces are related by a linear projective transformation, so they can be registered together into a larger planar image [51]. While planar mosaics are convenient representation for relatively small field of view (less 180°), they become problematic for wider scenes. In those circumstances either *cylindrical* or *spherical* representations are more suitable (panoramic or panspheric mosaics). Cylindrical panoramic mosaics are created by projecting images taken from the same point of view onto a cylindrical surface [48, 52]. They can be used for fixed location visualization (Quick Time VR) [4] or variable location visualization (plenoptic modeling) [35] and for recovering 3D structure of the environment [28]. Spherical panoramic mosaics can be created either by projected planar images taken from a fixed center of projection onto a sphere [51], or using special lenses and mirrors [38].

Images with parallax, which do not satisfy any of the two conditions mentioned above, can also be composed into a mosaic. Irani *et al* [25] used a polynomial transformation with more than eight degrees to compensate nonlinearities due to parallax. An interesting approach is presented in [9] where an intermediate image is used for registering two images under arbitrary camera motion. Another solution is to use a one dimensional camera to scan scenes. This can be realized using conventional cameras by combining strips taken from a sequence of neighboring images. In this way Peleg [41] and Tsuji [62] created a panoramic mosaic from images along an arbitrary path.

In order to create the image mosaics, images have to be registered or matched. Carefully calibrated cameras prior to the acquisition process can eliminate this step, but this is very inconvenient for the user. The image registration techniques used in the literature include:

- **Manual registration** methods where the operator has to manually align the images.
- **Feature based methods** that manually or automatically detect specific features in the images, compute correspondences, and then estimate the camera motion.
- Finding the motion that will best align the images by **exhaustively searching** all the possible motions. This can be computationally extremely expensive. Another possibility is to **iteratively adjust** the motion parameters by minimizing the differences between the overlapping areas. This method leads to a local minimum unless a reliable initial estimate is provided.
- **Frequency domain** techniques compute the image displacement from phase correlation. These methods require significant overlap between the images.

After image alignment, usually the mosaic has to be further processed in order to eliminate remaining distortions and discontinuities. These errors are caused by changes in the illumination, imperfect registration, dynamic scenes, etc. The lighting problem can be reduced using histogram equalization or locally smoothing the mosaic at the intersection lines [52]. For compensating

small errors introduced by motion parallax Szeliski and Shum [52, 48] developed a local alignment (*deghosting*) technique which warps each image based on the results of pairwise local registration.

Image mosaics are easy to build, but the rendered images must satisfy the same constraints as the input images, so it is not possible to create arbitrary new views. Adding more constraints like depth or disparity will overcome this problem.

1.4.2 Depth based reprojection

Depth or disparity is powerful information that can be used in image-based rendering. Depth is evaluated from stereo vision or using other sensors like range-finders or sonars and then combined with color information provided by images to form an image-based model. Disparity is usually computed from image correspondence. Different approaches to this problem have been proposed. Laveau and Faugeras [29] use a collection of fully calibrated images and the disparities between them to compute a novel view using a raytracing process. McMillan and Bishop [35] developed a rendering technique called *plenoptic modeling*. They compute new views from cylindrical panoramic images by directly transferring disparity values (general angular disparities) from the cylindrical models to the virtual view. Debevec [7] combines image-based and geometry-based techniques to produce photorealistic models of architectural scenes. Their system is built from two components: the first is a photogrammetric modeling system that recovers the basic geometry of the scene and the second a model-based stereo algorithm that refines the geometric model to conform with its actual appearance from a set of photographs. For rendering they present a view-dependent texture-mapping that produces new images by warping and composing multiple views of the scene. When dense range data is available from range-finders, it can be combined with the image data to form a detailed image-based model [33], or a 3D model [49] of a large scene. In both approaches range data is registered with the image data using line features (edges).

In order to compensate the occlusion problem in an efficient way, Shade *et al.* [46] introduce *layer depth images* (LDI). A LDI is a view of the scene from a single camera view point, where multiple pixel values are stored for each line of sight. This idea is further improved in [3] by introducing the LDI tree - a combination of a hierarchical space partition scheme with the concept of the LDI. The fixed resolution of the LDI may not provide an adequate sampling rate for every reference image. The LDI tree preserves the sampling rate of the reference images by adaptively selecting an LDI in the tree.

Normally 2D to 2D warping used in texturing models is only physically correct when model facets are true planes. If the texture has a 3D structure texturing with an image alone is at best an acceptable approximation, but in some cases (such as close up views or views at a grazing angle to the surface) it will give a flat, unnatural appearance to the rendering. In 3D warping a depth map is used to correctly “rearrange” the texture image into a new view[34]. Three dimensional warping has further been developed into an efficient method for texturing, *relief texturing*[39]. For instance, using a rectangular geometry model of a house and texturing it with a flat 2D texture would give un-natural renderings for many views. However, using a texture composed of both an image and depth map, and relief-texturing the same geometry recreates the correct views of the house fine structure. In Figure 4 particularly this is evident for the roof and dormers.

By now the reader may have noticed that in using texturing as a means of rendering macro-



Figure 4: (a) Image texture and depth map. (b) Pre-warped relief texture and final textured and rendered house.

scopic structure we have departed from its original meaning and intention (of describing a fine scale surface appearance). This can be illustrated by considering that using 3D textures objects of arbitrary shape can be rendered by enclosing them in an enveloping geometry, a visual hull or bounding box. This can be as simple as the six sides of a cube. In Figure 5 a bust is rendered using six textures with depth maps. The enclosing geometry is drawn for illustration purposes only, and as can be seen the final object appearance has little to do with the geometric model used.

Geometry-based techniques are the most suitable in certain applications e.g. robot navigation and mapping, because they represent geometrically correct models of the scene and new views can be generated from any arbitrary position. One of the biggest problems with image-based systems is that it is hard to model dynamic environments and most of the natural environments are changing. This needs to be overcome by automatically updating the model. This can also improve the spatial sampling in the rendering process, by improving the model resolution from newly acquired images.

1.5 Summary

This section has provided a short overview of image-based modeling and rendering techniques. Depending on the way of representing the data they can be divided into: morphing techniques, interpolation from dense samples, and geometrically based pixel reprojection which includes image mosaics and depth based reprojection models.

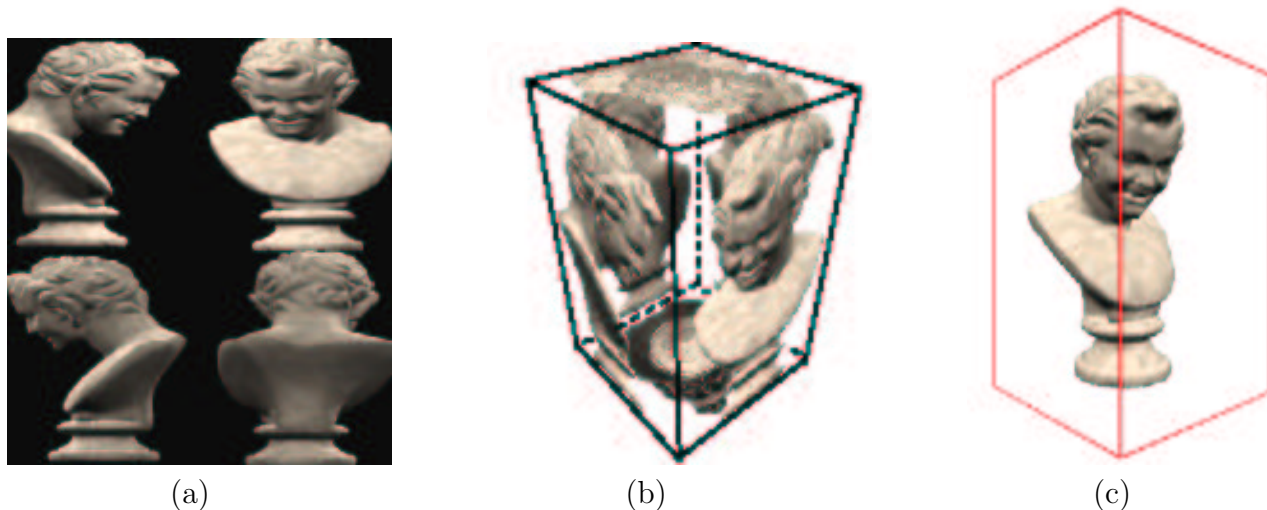


Figure 5: (a) Image textures for four of the six model planes. (b) Two-dimensional flat texturing (c) Three-dimensional relief texturing creates a realistic *image-based object*

2 The Geometry of Image Formation

Zach Dodds

The primary source of information used in image-based modeling is images from one or more cameras. Cameras are remarkably effective sensors, but they are also deceptive: the information they provide about the structure of the observed world may not always be what it seems.

An everyday example of visual ambiguity

Figure 6 shows an ordinary newspaper picture which demonstrates how even the human visual system can incorrectly extract 3d geometric information from an image.

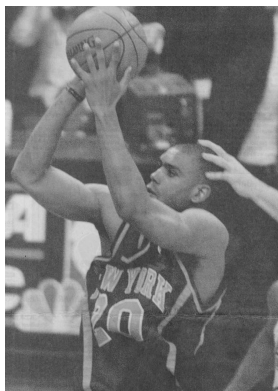


Figure 6: A photograph demonstrating ambiguity of the 3d locations of different parts of an imaged scene [14].

To many observers, the defender’s hand in Figure 6 looks firmly clasped on the back of the shooter’s head. In actuality, the hand is well away from the shooter and very close to the depth of his left forearm. Given this scenario, imagine if a referee had only the information from this image with which to evaluate the position of his hand — he would be unable to determine whether he was committing a foul or merely reaching out for the ball, since either situation yields the same image information.

One approach to disambiguating the different possible worlds which Figure 6 could depict is to use precise models of both the objects imaged and the camera’s geometry. Knowing the exact dimensions of the shooter’s head and defender’s hand, along with an accurate estimate of the pose of the camera with respect to those objects, would suffice to determine the true 3d geometry of the scene. Only under extremely controlled conditions, however, are such object and camera models available. Without these constraints it is still possible to obtain and use geometric models based on visual information. This chapter motivates different levels or *strata* of recoverable geometric structure. These basic principles form the foundation of some of the hybrid model-based and image-based rendering techniques presented in subsequent sections.

2.1 Camera Model

Effectively relating 2d image information with the 3d world drives the study of the geometric properties of vision. As Figure 7 shows, the pinhole camera models image formation as a projection of 3d points through a focal point C known as the camera center. Similar triangles then yield

$$x_c = \frac{X_c}{Z_c} \quad \text{and} \quad y_c = \frac{Y_c}{Z_c}, \quad (1)$$

where $m = [x_c, y_c, 1]^T$ and $f = [X_c \ Y_c \ Z_c \ 1]^T$. Though nonlinear as a mapping from points in \mathfrak{R}^3 to \mathfrak{R}^2 , when expressed in homogeneous coordinates Equation 1 becomes linear. (A motivation and detailed description of homogeneous coordinates is provided below.) Thus, $m = \mathbf{P}_c f$, or componentwise

$$\lambda \begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix} = \begin{bmatrix} u_c \\ v_c \\ w_c \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix}. \quad (2)$$

In Equations 1 and 2 the mapping \mathbf{P}_c assumes canonical coordinate systems for both world points and image points, depicted in Figure 7 (left).

In general, world points might exist in any coordinate system rigidly displaced from the canonical one by a rotation \mathbf{R} and translation t . In addition, image coordinates measured in pixels are modeled as an affine transformation \mathbf{K} of the metric coordinates of Equations 1 and 2. Hence, in these more general frames image projection becomes

$$y = \mathbf{K} [\mathbf{R} \ t] f. \quad (3)$$

\mathbf{R} and t hold six “external” parameters (the pose of the world frame with respect to the camera’s canonical frame). \mathbf{K} is an invertible, upper triangular matrix which models five “internal” camera

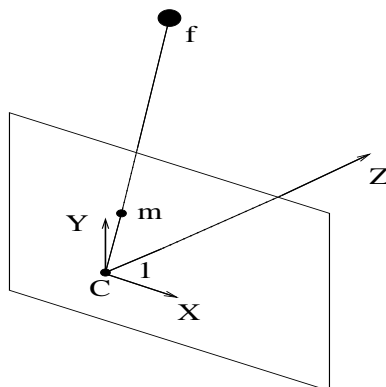


Figure 7: Projection in a pinhole camera. Suppose a world point f is expressed in a camera’s *canonical (Euclidean) frame* with the indicated x , y , and z directions scaled to units of focal length and centered at the camera’s focal point C . Then f ’s projection, y , satisfies Equations 1 and 2 when written in the *metric image frame* — identical to the camera’s canonical frame.

parameters. Because f and y are homogeneous, the lower right entry of \mathbf{K} can be scaled to 1. In this case

$$\mathbf{K} = \begin{bmatrix} a & b & c \\ 0 & d & e \\ 0 & 0 & 1 \end{bmatrix} \quad (4)$$

implies that (c, e) are the coordinates of the principal point, $\cot^{-1}(-\frac{b}{a})$ is the angle θ between the image’s x and y axes, and there are a and $\frac{d}{\sin(\theta)}$ pixels in a single focal length in the x and y image directions, respectively. Cameras with known \mathbf{K} are considered *internally calibrated*, and \mathbf{K} provides the relationship

$$y = \mathbf{K}m \quad (5)$$

between m ’s metric image frame and y ’s *pixel image frame*.

Strata of geometric representation Because of the nonlinearity of the camera transformation (indeed, real camera transformations are even worse than depicted above, as the next section attests), it is difficult to extract Euclidean or metrical structure from image information. One technique for recovering that structure is to *build up to it* from less precise geometric knowledge more readily extracted from an imaged scene. The following sections present a hierarchy of five layers or *strata* [12] of geometric structure, as well as the geometric properties and interrelationships among them.

Injective Geometry

Although an injective transformation of a scene is so general as to seem unhelpful in our goal of modeling objects’ and images’ structure, it is important to keep in mind that real imaging systems don’t start with even the clean perspective projection model described above. There is often considerable distortion caused by the lens system of a camera. With shorter focal lengths

(and wider fields of view), the distortion tends to be more noticeable, as indicated in Figure 8. (Note: this figure and others annotated “HZ” are used with permission from [20].)

6.4 Radial distortion

17

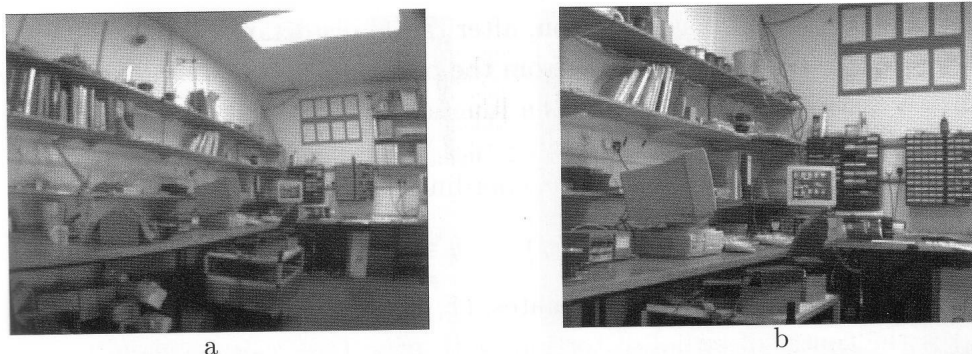


Figure 8: The advantage of a wide field of view is tempered by the presence of significant radial distortion from a linear projection

Calibration (or *lens* calibration) is the process of determining how to undo this type of distortion, and comes in at least two basic varieties: techniques using algebraic coordinates of known fiducial points and techniques that rely on the geometric constraints of the corrected transformation. A common approach to the former, algebraic approach is that of Tsai [58] or Brand, Mohr, and Bobet [2], who model the distortion using both radial and center-displacement terms:

$$\begin{aligned} x' &= x + k_1 \bar{x} r^2 + k_2 \bar{x} r^4 + k_3 \bar{x} r^6 + P_1 (2\bar{x}^2 + r^2) + 2P_2 \bar{x} \bar{y} \\ y' &= y + k_1 \bar{y} r^2 + k_2 \bar{y} r^4 + k_3 \bar{y} r^6 + P_2 (2\bar{y}^2 + r^2) + 2P_1 \bar{x} \bar{y} \\ &\text{where } \bar{x} = x - u_0, \bar{y} = y - u_0, r = \bar{x}^2 + \bar{y}^2 \end{aligned}$$

Figure 9: The advantage of a wide field of view is tempered by the presence of significant radial distortion from a linear projection

The geometric approach relies on the fact that, without distortion, the camera transformation is linear (in homogeneous, not Euclidean, coordinates), meaning that subspace dimensionality is preserved: points map to points and lines (generically) map to lines. Thus, a calibration object that consists of a large number of lines gives rise to a set of constraints that express the collinearity of points recovered from those lines, e.g., Figure 10.

2.2 2d Projective Geometry

Once lens distortion has been accounted for, the camera effects a *projective* linear transformation from 3d to 2d. Because of this relationship, projective geometric structure is the easiest to deduce from and about images. This section provides a brief introduction to both 2d and 3d projective space with a focus on the image-transfer applications of 2d projective planes.

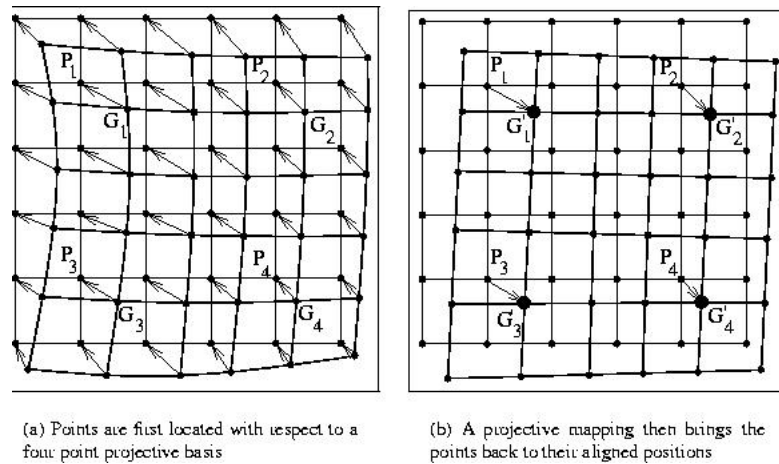


Figure 10: Lens calibration may be found by adjusting distortion parameters so that lines map to lines, as predicted by a linear projective transformation



Figure 11: An example of an image with radial distortion and a resampled image without it.

A metaphor for 2d projective space A camera's retina is usefully modeled as a projective plane. If you place the retina one unit in front of the camera's center parallel to the plane formed to the x - and y - axes of a coordinate frame whose origin is that camera center, every point on the retina (or image plane) has Euclidean coordinates $[x, y, 1]^T$.

Especially in light of the ray-projection used in imaging and rendering, it is natural to associate each image point with the ray passing through that point. Each of these rays, then, models a point in a *projective plane* in which the retinal plane is embedded. When considered a set of rays, the use of 3d coordinates, i.e.,

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = s \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (6)$$

with x , y , and z not all equal to 0, is a natural means for representing projective points. Each ray is distinguished by the 3d (Euclidean) coordinates of *any* point on it. Naturally, $\mathbf{0} = [0, 0, 0]^T$

must be excluded, as it is shared by all of the rays. The convention of these *homogeneous coordinates* requires that coordinate vectors be considered equivalent if they differ by any nonzero scale factor. In addition, this model also provides intuition for why the projective (homogeneous) point $[x, y, z]^T$ is considered equivalent to the Euclidean point $[\frac{x}{z}, \frac{y}{z}]^T$.

Both this geometric analogy and algebraic coordinate convention point to the primary difference between a projective and a Euclidean plane: there is a line of points "at infinity" in the former. These *ideal* points correspond to rays parallel to the image plane and have homogeneous coordinates of $[x, y, 0]^T$. Such points may be considered the limit of the Euclidean points $[x, y, \alpha]^T$ as $\alpha \rightarrow 0$ or the vector in the direction of $[x, y]^T$. Indeed, homogeneous coordinates are important fundamentally because they can handle this line, l_{inf} , of points at infinity. Figure 12 [HZ] depicts this model of the projective plane.

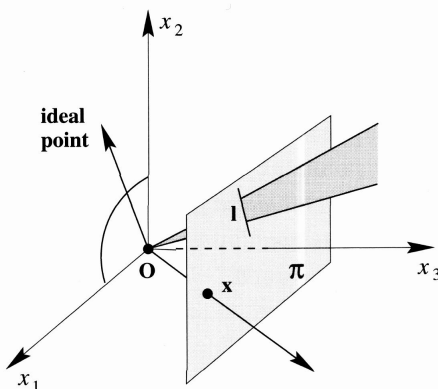


Figure 12: A model for the projective plane as the set of rays emanating from the origin of 3d Euclidean space

In 2d projective space, lines are represented as three homogeneous coordinates as well. If $l = [a, b, c]^T$, and the point $x = [x, y, 1]^T$, x lies on the line l if $ax + by + 1c = 0$. This is true regardless of scale, so that

$$x^T l = l^T x = 0$$

expresses the relationship between the coordinates of a line l and the points x on that line. The symmetry in this equation is important: lines and points are dual in 2d projective space. That is, for any result in the projective plane there holds a dual result that interchanges the role of points and lines.

One example of this duality is in finding the point x that is shared by two lines, l and l' . x is the intersection of l and l' if

$$x = l \times l'$$

where \times represents the cross-product of two three-dimensional vectors. The dual theorem reverses the roles of points and lines: the *line* shared by two *points*, i.e., the line l that passes through both points x and x' , is given by

$$l = x \times x'$$

Transformations of projective planes Keep in mind that the ray-model of Figure 12 uses a canonical coordinate system. A projective plane, of course, need not correspond to a useful world coordinate frame so neatly. In general, a projective plane (i.e., an image plane) will be a projective transformation of some canonical view that we might want to use as a model of (planar) structure. A linear projective transformation H (sometimes called a collinearity, projectivity, or homography) is, in 2d, a 3×3 matrix:

$$s \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Since H is defined up to a scale factor, it has 8 independent degrees of freedom (dof).

The importance of these 2d homographies is that they capture the geometric changes that a planar patch undergoes when imaged. Figure 13 [HZ] provides three examples of this process; in all three cases the coordinates of corresponding points in the two planes are related by a homography. As a result, image-based rendering of a planar surface can be achieved through *pixel transfer*, i.e., using a few points to determine the homography between a stored, model image and a current, desired point of view. Then, that homography maps points in the model to appropriate locations in the new view. Figure 14 shows an example of this.

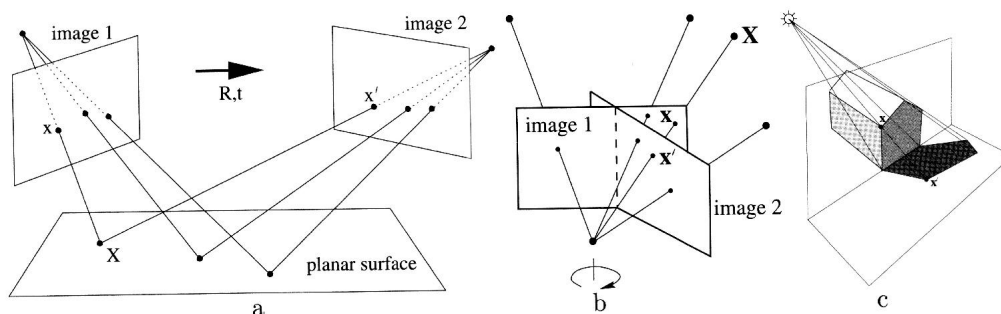


Figure 13: Examples of pairs of images related by a homography: two images of a planar surface, two images of any structure taken by cameras with the same camera center, and the shadow of a surface.

Given known corresponding points in two images, $x_i \leftrightarrow x'_i$, how can we determine the homography H relating them? Consider the fundamental constraint:

$$x'_i = Hx_i$$

Because this holds up to an unknown scale factor, it provides two linear constraints on the entries of H . H has eight degrees of freedom, so that four such constraints suffice to find the entries of the homography. With additional points, the system is overdetermined and a least-squares solution (other than $H = 0$) can be found using the singular value decomposition [20].

A direct application of this algorithm is the creation of image mosaics, where images taken by a rotating camera are stitched together using at least four “tie points” – the points used to

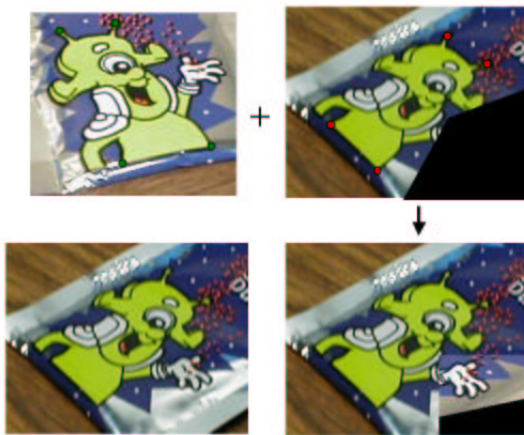


Figure 14: Example of pixel transfer after a four-point homography: note that the geometry of the resulting image is close to the expected geometry, but the lighting is quite different. A dynamically textured approach lessens or removed this problem.

determine the homography H between two views. Figure 15 [HZ] shows an eight-image example as well as a full panormaic view.

2.3 3d Projective Geometry

On the other side of the camera is the world. Just as an image was naturally considered a 2d projective plane, the objects being imaged are naturally considered populating a 3d projective space. A plane of points called the plane at infinity, Π_{inf} , is present (and not distinguished from any other plane) in projective space. Points are represented with four homogeneous coordinates:

$$\mathbf{X} = \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix}$$

In projective space, points and planes are dual, with

$$\mathbf{X}^T \mathbf{\Pi} = 0$$

for points \mathbf{X} on the plane $\mathbf{\Pi}$.

Projective transformations are 4×4 nonsingular matrices identified up to a scale factor and, like their two-dimensional counterparts, preserve dimensionality of features: point-coincidence, collinearity, and coplanarity. In some applications these properties suffice and a projective model of objects is all that is required. For example, many tasks can be phrased as a set of feature alignments among points and lines. Controlling a robotic manipulator to achieve those tasks requires no more than a projective model of the objects being handled. When the sensors available are cameras, this approach to robot control is called *visual servoing* [16]. In a sense, visual servoing is a physically realized version of image-based modeling and rendering: a desired



Figure 15: Eight images and the mosaic resulting from homography-based transfer

rendering of point and line features is achieved by actually moving an object of interest to a goal pose.

However, for virtual reality or augmented reality applications, projective models are too coarse for practical use. They will produce unrealistic images because so much of the objects' expected geometry is not captured in a projective model. Parallelism, volumes, areas, lengths, and angles are all lost at the projective level of specification. the following section describes more precise geometric descriptions that recover these model properties.

Affine and Metric Upgrades

The ability to work with vanishing points (ideal points, points at infinity) as easily as ordinary points is a weakness, as well as a strength of projective space. The fundamental geometric properties of betweenness and parallelism are lost under projective transformations, e.g., Figure 16 [HZ], (In real images, not simply projectively-transformed images, betweenness is preserved.)

Figure 16: An example of a projective transformation that does not preserve "betweenness." Real cameras do not wrap physical objects around the plane or line at infinity because they are one-sided. This transformation requires the comb to occupy portions of the half-spaces in front of and behind the camera's center.

In order to restore these properties, the points (either plane or line) at infinity must be identified. Then, for example, parallel lines are identified as those projective lines whose intersections are points at infinity. Nonparallel lines meet elsewhere. In a static 3d model, the plane at infinity can be determined using precisely this constraint. If sets of parallel lines in three distinct direc-

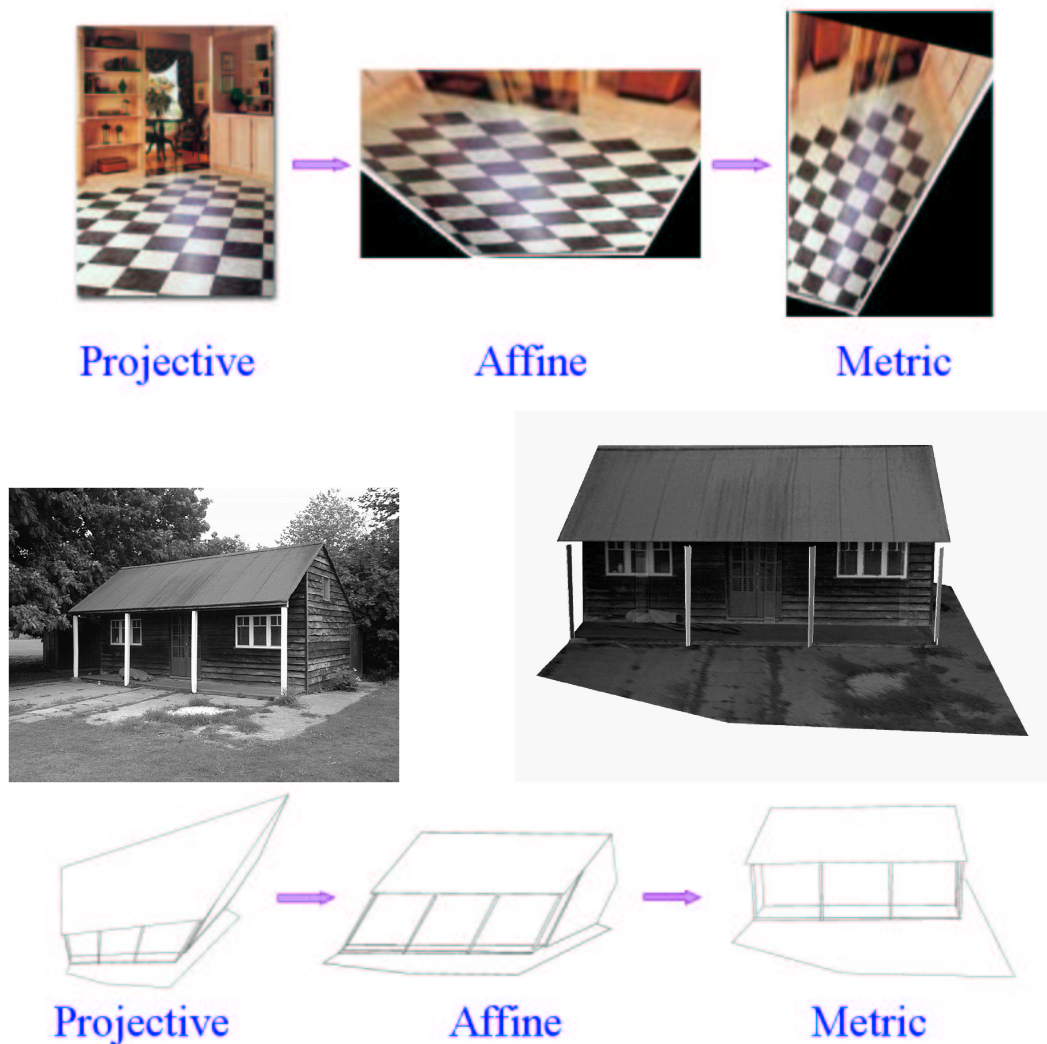


Figure 17: (**Top**) An upgrade through projectively, affinely, and similarity-based transformations of a checkerboard tile pattern. (**Middle**) The initial and final, metrical model of a cabin. (**Bottom**) The strata through which the cabin progressed, each requiring additional a priori knowledge about the structure.

tions are identified, the three points of intersection of those sets determine the plane at infinity. In 2d, two such sets of lines, known to be parallel a priori yield l_{inf} .

In both cases, the structure obtained is called *affine*. In three dimensions, an affine model has 12 degrees of freedom: three for the position of the origin, three for a rigid rotation, one overall scale, two relative scales among the three axes and three representing the angles (skew) among the three coordinate axes. In 2d, there are 6 dof: two for the position of the origin, one for rigid rotation about the origin, one for overall scale, one for relative scale between the two axes, and one for skew (the angle between the two axes).

In order to measure angles and arbitrary length ratios in a 2d or 3d model, a further upgrade of its geometric representation is necessary. In 2d, this upgrade from an affine image requires two pairs of orthogonal lines; in 3d the upgrade requires knowledge of the camera's internal parameters (or consistency of them across multiple images) and/or knowledge of some scene structure. Both result in a metric description of the resulting structure, i.e., a rigid motion followed by a uniform scaling. In order to obtain a Euclidean model with true distance information, a "yardstick" of some sort must be available to define at least one length. From there, all other lengths can be determined relative to that canonical segment. Figure 17 [HZ] show this upgrade process through projective, affine, and metrical representations of 2d and 3d structure.


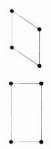


Group	Matrix	Distortion	Invariant properties
Projective 8 dof	$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$		Concurrency, collinearity, order of contact : intersection (1 pt contact); tangency (2 pt contact); inflections (3 pt contact with line); tangent discontinuities and cusps. cross ratio (ratio of ratio of lengths).
Affine 6 dof	$\begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Parallelism, ratio of areas, ratio of lengths on collinear or parallel lines (e.g. midpoints), linear combinations of vectors (e.g. centroids). The line at infinity, l_{∞} .
Similarity 4 dof	$\begin{bmatrix} sr_{11} & sr_{12} & t_x \\ sr_{21} & sr_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Ratio of lengths, angle. The circular points, I, J (see section 1.7.3).
Euclidean 3 dof	$\begin{bmatrix} r_{11} & r_{12} & t_x \\ r_{21} & r_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix}$		Length, area

Figure 18: The hierarchy of geometric strata in two dimensions

Figures 18 and 19 [HZ] summarize the hierarchy of information in various 2d and 3d geometrical models. The modeling and rendering described in the sequel will use a variety of geometric descriptions, depending on both the requirements of the tasks and the information available to support them.

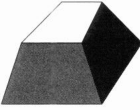
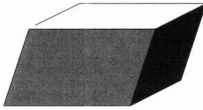

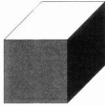
Group	Matrix	Distortion	Invariant properties
Projective 15 dof	$\begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{v}^T & v \end{bmatrix}$		Intersection and tangency of surfaces in contact. Sign of Gaussian curvature.
Affine 12 dof	$\begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$		Parallelism of planes, volume ratios, centroids. The plane at infinity, π_∞ , (see section 2.5).
Similarity 7 dof	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$		The absolute conic, Ω_∞ , (see section 2.6).
Euclidean 6 dof	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix}$		Volume.

Figure 19: The hierarchy of geometric strata in three dimensions

3 Multiple View Geometry and Structure-From-Motion

Dana Cobzas and Martin Jagersand

3.1 Camera models

A camera maps points from the 3D space (object space) to 2D image plane. In this section we present different camera models that mathematically describe this process.

Pinhole camera

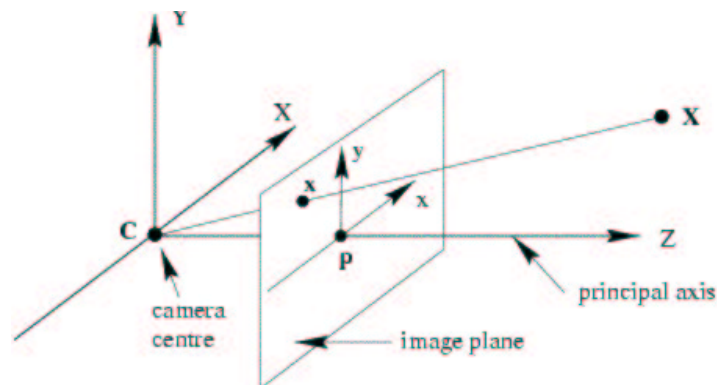


Figure 20: Pinhole camera model

The most basic camera model is the pinhole camera. The image formation process is specified by choosing a *center of projection* and an *image plane* (see figure 20). The projection of a 3D point is obtained by intersecting the ray from the camera center of projection to this point with the image plane. The ray from the camera center perpendicular to the image plane is called *optical axis* or *principal ray*. This model is in general valid for most real cameras. In some cases it can be improved by taking nonlinear effects (e.g. radial distortion) into account.

If the center of projection is chosen as the origin of the world coordinate system, and the image plane is located at a distance f from the origin, the projection equations can be written as follows:

$$(x, y)^T = \left(\frac{fX}{Z}, \frac{fY}{Z} \right)^T$$

Representing points using homogeneous coordinates we can rewrite the projection equation as:

$$\begin{pmatrix} x \\ z \\ 1 \end{pmatrix} \sim \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad (7)$$

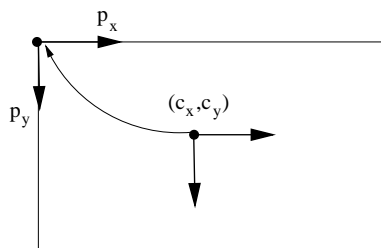


Figure 21: From retinal coordinates to image pixel coordinates

Using this projection the image points will be indexed with respect to the principal point (intersection of image plane with the optical axis). In real cameras image pixels are typically indexed with respect to the top left corner of the image, so a change in coordinate system is required. In addition the coordinates of the pixel do not correspond to the coordinates in the retinal plane but depend on the physical dimensions of the CCD pixels in the camera. With most standard cameras we can model this transformation by scaling along the camera axes. The 2D transformation that transforms retinal points into image pixels is (see figure 21):

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \sim \begin{pmatrix} \frac{1}{p_x} + c_x \\ \frac{1}{p_y} + c_y \\ 1 \end{pmatrix} = \begin{bmatrix} \frac{1}{p_x} & 0 & c_x \\ 0 & \frac{1}{p_y} & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (8)$$

where (c_x, c_y) represents the principal point location (not always the center of the image) and p_x, p_y the 2 pixel scale factors along horizontal and vertical directions respectively. Combining

equation 7 and 8, we obtain the projection equation in component and standard matrix form as

$$\mathbf{x} = \begin{bmatrix} \frac{f}{p_x} & 0 & c_x & 0 \\ 0 & \frac{f}{p_y} & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \mathbf{X}_{cam} = K[I|\mathbf{0}]\mathbf{X}_{cam} \quad (9)$$

Here K is called the calibration matrix and is formed from camera *internal parameters*. More parameters can be introduced to model different real cameras (e.g. skew in the case when the pixels are not rectangular). These parameters are in general related to a particular physical camera and can be determined ahead in the process of camera calibration [58]. In this presentation we mostly assume that the cameras are not calibrated.

Camera motion

World 3D points, *world coordinates*, are normally expressed in their own coordinate frame, different than the camera frame. The camera and world coordinate frames are related by a general Euclidean transformation (*external* or *extrinsic* camera parameters):

$$\begin{aligned} \mathbf{X} &= [R|\mathbf{t}]\mathbf{X}_{cam} \\ \mathbf{X}_{cam} &= [R^T | -R^T\mathbf{t}]\mathbf{X} \end{aligned}$$

Incorporating this into the projection equation 9 we get:

$$\mathbf{x} = K[R^T | -R^T\mathbf{t}]\mathbf{X} = P\mathbf{X} \quad (10)$$

The 3×4 matrix P is called the *camera projection matrix* and has in general 11 DOF. If P can be decomposed like in equation 10 (3×3 left hand submatrix M is nonsingular) it is called *finite camera*. If M is singular the camera is called *infinite camera*.

Some properties of the general projective camera $P = [M|\mathbf{p}_4]$ can be derived without decomposing it into internal and external parameters. For example we can retrieve the camera center \mathbf{C} as the 1D nullspace of P ($P\mathbf{C} = 0$), for finite cameras:

$$\mathbf{C} = \begin{pmatrix} -M^{-1}\mathbf{p}_4 \\ 1 \end{pmatrix}$$

and for infinite cameras:

$$\mathbf{C} = \begin{pmatrix} \mathbf{d} \\ 0 \end{pmatrix}$$

where \mathbf{d} is the nullspace of M . The principal ray (the ray passing through the camera center and which is perpendicular to image plane) can be expressed as:

$$\mathbf{v} = \det(M)\mathbf{m}^3$$

where \mathbf{m}^3 is the last row of M .

Affine cameras

Affine cameras (or parallel projection cameras) is a class of infinite cameras of practical interest. An affine camera has the last row of the projection matrix of the form $(0, 0, 0, 1)$. They are called affine because they map points at infinity to points at infinity. The center of projection is an infinite point so camera projection rays are parallel.

In general an affine camera can be written as:

$$P_{\infty} = K \begin{bmatrix} \mathbf{i} & t_x \\ \mathbf{j} & t_y \\ \mathbf{0}^T & d_0 \end{bmatrix} \quad R = \begin{pmatrix} \mathbf{i} \\ \mathbf{j} \\ \mathbf{k} \end{pmatrix} \quad \mathbf{t} = \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix} \quad d_0 = t_z \quad (11)$$

where d_0 represents the depth of the point along the principal direction and \mathbf{i}, \mathbf{j} are the first two rows of the rotation matrix that related the position of the object with the camera pose.

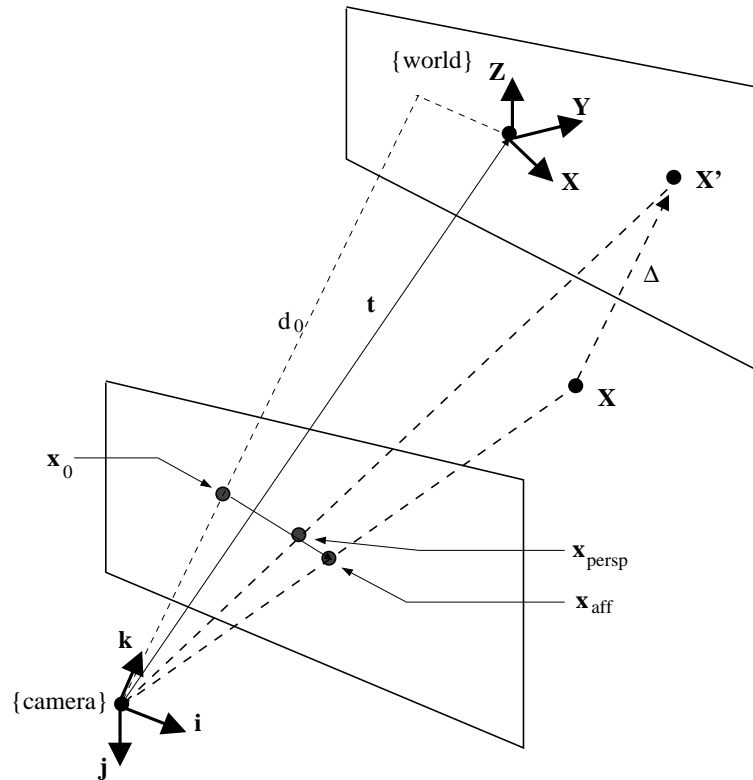


Figure 22: Affine camera approximation

The imaging geometry of an affine camera is illustrated in figure 22. It can be illustrated as a two step projection. Consider a plane that passes through the world coordinate center and is parallel with the image plane. This plane can be thought as approximating the object since for all the points on the plane the affine projection is equivalent to a perspective projection. The 3D point \mathbf{X} is first projected on this plane to get \mathbf{X}' , and then \mathbf{X}' is perspectively projected on the image plane. The deviation of the affine projection with respect to the projective projection

can be expressed as:

$$\mathbf{x}_{aff} - \mathbf{x}_{persp} = \frac{\Delta}{d_0}(\mathbf{x}_{proj} - \mathbf{x}_0) \quad (12)$$

where \mathbf{x}_0 is the principal point and Δ is the distance from the point to the object approximation plane. From this we can see that the affine camera is a good approximation of the imaging geometry when:

- The depth relief (Δ) variation over the object surface is small compared to the average distance to the object (d_0).
- The distances between the object points and the principal ray are small.

A hierarchy of affine cameras

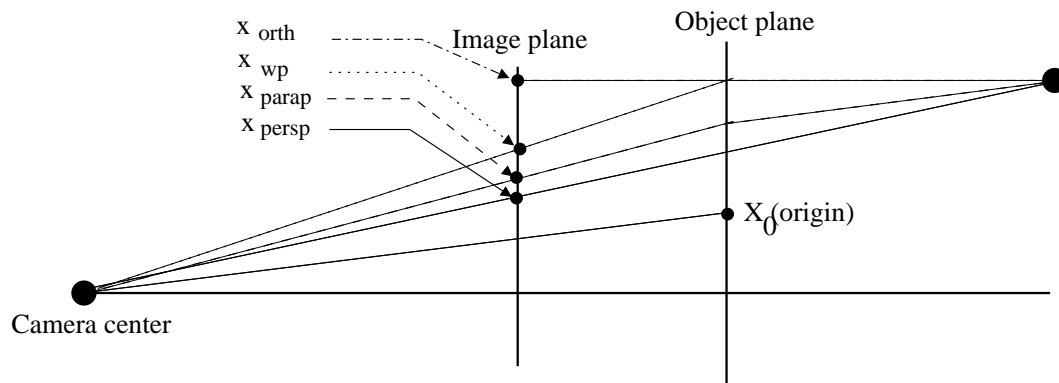


Figure 23: Different types of affine projection.

Different affine models can be derived starting with a very simple model and refining it to better approximate the imaging process. This is illustrated in figure 23

Orthographic camera. If the direction of projection is along Z axis, the camera matrix has the following form:

$$P_{orth} = \begin{bmatrix} \mathbf{i} & t_x \\ \mathbf{j} & t_y \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (13)$$

Weak perspective camera. A better approximation of the perspective camera can be obtained by first projecting orthographically on an approximate object plane and then projectively on the image plane. It is also called scaled orthographic projection and the camera matrix has the form:

$$P_{wp} = \begin{bmatrix} k & 0 & 0 \\ 0 & k & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{i} & t_x \\ \mathbf{j} & t_y \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (14)$$

Paraperspective camera. The weak perspective camera is a good approximation of the perspective distortion only if the object is close to the principal axis. The paraperspective model

compensates this problem by projecting the object point on the model plane with a direction parallel to the ray that projects the center of object coordinate system. It is not an affine camera but a first order approximation of the perspective camera. (The other models can be seen as a zero-order approximation.) The projection equations can be written as:

$$x_{wp} = x_0 + \frac{\mathbf{i} - x_0\mathbf{k}}{t_z} \mathbf{X} \quad (15)$$

$$y_{wp} = y_0 + \frac{\mathbf{j} - y_0\mathbf{k}}{t_z} \mathbf{X} \quad (16)$$

where (x_0, y_0) represents the image projection of the center of the object coordinate system.

Camera calibration

Camera calibration [58] deals with computing the camera matrix given images of a known object and is equivalent to the *resection* problem discussed in subsection 3.2. The calibration object is usually build as a 2D or 3D pattern with easily identified features (e.g. checkerboard pattern). The camera matrix is then decomposed into internal and external parameters, assuming a certain camera model. The linear camera model discussed so far can be a bad approximation for a very wide lens camera and radial distortion has to be considered. The radial distortion is small near the center but increasing toward the periphery, and can be corrected using:

$$\hat{x} = x_c + L(r)(x - x_c) \quad (17)$$

$$\hat{y} = y_c + L(r)(y - y_c) \quad (18)$$

where (\hat{x}, \hat{y}) denotes the corrected point for (x, y) and (x_c, y_c) is the principal point of the camera and $r^2 = (x - x_c)^2 + (y - y_c)^2$. $L(r)$ is the distortion function and can be approximated with $L(r) = 1 + k_1r + k_2r^2 + \dots$. k_1, k_2 , the coefficients of the radial correction are considered part of the internal camera parameters.

3.2 Computation with Camera Models

In practical application we are interested in using camera models computationally to relate the three entities of 3D structure, 2D image projection and projection matrix. Three basic computational problems are:

1. *Resection* Given a 3D structure and its image projection compute the camera pose (extrinsic parameters).
2. *Intersection* Given two or more image projections and corresponding cameras compute a 3D structure giving rise to these images.
3. *Structure and Motion* or *Factoring* Given only the image projections in two or more images find both the camera pose and 3D structure.

Resection

In Resection we are given 3D points $\mathbf{X}_1 \dots \mathbf{X}_n$ and their image projections $\mathbf{x}_1 \dots \mathbf{x}_n$. Under an linear affine camera we seek to determine the eight parameters in an affine projection. Rewriting Eq.11 in affine (non-homogeneous) coordinates the constraint imposed by each image-3D point is given by

$$\mathbf{x}_i = \begin{bmatrix} \mathbf{i} \\ \mathbf{j} \end{bmatrix} \mathbf{X}_i + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \quad (19)$$

Each 2D-3D point correspondence hence imposes two constraints on the eight dimensional affine camera space. We can solve for the camera parameters by extending the above equation for $n \geq 4$ points and grouping the camera elements into a matrix,

$$[\mathbf{x}_1, \dots, \mathbf{x}_n] = \begin{bmatrix} \mathbf{i} | t_x \\ \mathbf{j} | t_y \end{bmatrix} \begin{bmatrix} \mathbf{X}_1, \dots, \mathbf{X}_n \\ 1, \dots, 1 \end{bmatrix} \quad (20)$$

and then transposing the expression and solving for $P = \begin{bmatrix} \mathbf{i} | t_x \\ \mathbf{j} | t_y \end{bmatrix}$ in

$$\begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} \mathbf{X}_1, 1 \\ \vdots \\ \mathbf{X}_n, 1 \end{bmatrix} P^T \quad (21)$$

Since we use affine coordinates we directly minimize geometric error, hence the solution is the ML estimate for measurements perturbed by Gaussian noise.

In the case of a perspective camera model we need to estimate the full 3×4 projective camera matrix in the projection equation, $\mathbf{x}_i \sim P\mathbf{X}_i$ (Eq. 7), where here coordinates are given in homogeneous form, $\mathbf{x} = [u, v, w]^T$ and $\mathbf{X} = [X, Y, Z, v]$. Remember that the projective equivalence “ \sim ” is up to a scale, hence we cannot directly write a linear equation system as in Eq. 20. Instead, the standard solution, Direct Linear Transform (DLT) is based on expressing the equivalence as a vector product $0 = \mathbf{x}_i \times P\mathbf{X}_i$. Let $P^T = [\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3]$ and rewrite the vector cross product on a matrix form we get the following three constraint equations from each point correspondence

$$0 = \begin{bmatrix} \mathbf{0}^T & -w_i \mathbf{X}_i^T & v_i \mathbf{X}_i^T \\ w_i \mathbf{X}_i^T & \mathbf{0}^T & u_i \mathbf{X}_i^T \\ -v_i \mathbf{X}_i^T & u_i \mathbf{X}_i^T & \mathbf{0}^T \end{bmatrix} \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{p}_3 \end{bmatrix} \quad (22)$$

However, only two of the three equations are linearly independent, hence we need six (or more) 2D-3D point correspondences to solve for the projection matrix P. One common way of solving the above is to pick the first two of the three equations for each of the six point and stack them into a 12×12 linear equation system. Another alternative is to solve the overdetermined 18×12 system imposing an additional constraint $\|P\| = 1$. (Without the additional constraint, noise in the measurements will likely increase the rank and give a solution $P=0$.)

Intersection

In Intersection we are given two camera matrices P_1 and P_2 and the corresponding image point projections $\mathbf{x}_1, \mathbf{x}_2$, and seek the 3D point giving rise to these image projections.

In the affine case we can directly rearrange Eq.19 as

$$\begin{bmatrix} u_1 - t_{1,x} \\ v_1 - t_{1,y} \\ u_2 - t_{2,x} \\ v_2 - t_{2,y} \end{bmatrix} = \begin{bmatrix} \mathbf{i}_1 \\ \mathbf{j}_1 \\ \mathbf{i}_2 \\ \mathbf{j}_2 \end{bmatrix} \mathbf{X} \quad (23)$$

and solve the overdetermined system for the 3D world point $\mathbf{X} = [X, Y, Z]$. Again, since the above equations are in pixel space we minimize geometric error, and we don't normally expect problems. (Note however that the system may be ill conditioned due to the two cameras being nearly equal, i.e. \mathbf{i}_1 is nearly co-linear with \mathbf{i}_2 and \mathbf{j}_1 near \mathbf{j}_2)

In the projective case we seek to find a homogeneous 3D point $\mathbf{X} = [X, Y, Z, v]$ that simultaneously satisfies the two equivalences $\mathbf{x}_1 \sim P_1 \mathbf{X}$ and $\mathbf{x}_2 \sim P_2 \mathbf{X}$. Since the equivalence is up to a scale we can write $\lambda_1 \mathbf{x}_1 = P_1 \mathbf{X}$ and $\lambda_2 \mathbf{x}_2 = P_2 \mathbf{X}$ and rewrite into one matrix equation as

$$0 = \begin{bmatrix} P_1 & \mathbf{x}_1 & \mathbf{0}^T \\ P_2 & \mathbf{0}^T & \mathbf{x}_2 \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ \lambda_1 \\ \lambda_2 \end{bmatrix} \quad (24)$$

And solve the homogeneous system for a consistent 3D point \mathbf{X} and scale factors λ_1, λ_2 .

Structure and Motion

In the Structure and Motion (sometimes called Structure From Motion, SFM), using only the n projected points $\mathbf{x}_{i,j}$, $i \in 1 \dots n$, $j \in 1 \dots m$ in m images we seek to find both camera poses P_1, \dots, P_m and a consistent structure $S = [\mathbf{X}_1, \dots, \mathbf{X}_n]$. This is illustrated in Fig. 24.

For image-based modeling, this is the most interesting operations, since we can recover 3D information from uncalibrated images (i.e. without needing to know the camera poses a-priori). Interestingly, fast and robust linear methods based on direct SVD factorization of the image point measurements have been developed for a variety of linear cameras, e.g. orthographic [53], weak perspective [60], and para-perspective [42]. For non-linear perspective projection there is no direct linear factorization method, but iterative methods have been developed [22]. Another solution developed by Sturm and Triggs is to first relate the images pairwise through the fundamental matrix and then factor into structure and camera projections [50]. The factorization methods and algorithms are presented in Sections 3.4.

For two, three and four views image constraints have been formulated that allows extracting camera motion from image points. The 2-view case is presented in the next section.

3.3 Two view geometry

Epipolar geometry and the fundamental matrix

Without having any information about the position of a 3D point, given its projection in one image, restricts its projection in a second image to a line that is called the *epipolar line*. This is the basic geometric constraint for the two view geometry and is illustrated in figure 25 (a)(b). The epipolar line l' can be seen as the projection of the ray going from the camera \mathbf{C} through the

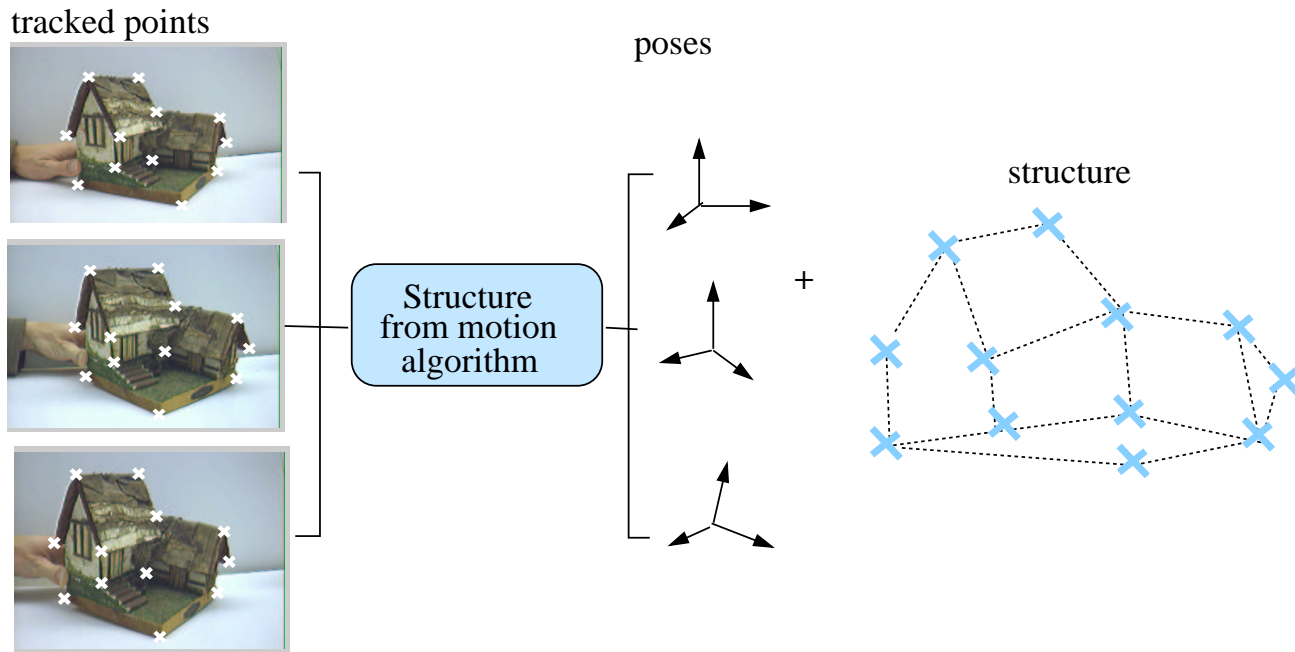


Figure 24: A general structure from motion algorithm extracts the structure and camera poses from a set of tracked points.

image point \mathbf{x} on the second image plane. This is equivalent to intersecting the plane generated by the camera centers \mathbf{C}, \mathbf{C}' and the image point \mathbf{x} (*epipolar plane*) with the second retinal plane. Note that all epipolar lines in an image have a common point - the projection of the second camera center. This point, is called the *epipole* and is denoted \mathbf{e} and \mathbf{e}' respectively for first and second camera in figure 25. Figure 25(c)(d) illustrated an example of four sets of corresponding epipolar lines.

This geometric constraint can be algebraically formulated using the fundamental matrix F [11]. It connects corresponding image points in two views. We present a geometric derivation of the fundamental matrix for image projections \mathbf{x}, \mathbf{x}' of 3D point \mathbf{X} (refer to figure 26). Consider the point \mathbf{X} laying on a plane π not passing through any of the camera centers. This plane induces a 2D projective transformation (homography) H between corresponding image projections, more precisely:

$$\mathbf{x}' = H\mathbf{x}$$

The epipolar line passing through can be written as:

$$\mathbf{l}' = \mathbf{e}' \times \mathbf{x}' = [\mathbf{e}']_{\times} \mathbf{x}' = [\mathbf{e}']_{\times} H\mathbf{x} = F\mathbf{x}$$

where $F = [\mathbf{e}']_{\times} H$ defines the fundamental matrix ($[\mathbf{e}]_{\times}$ denotes the antisymmetric 3×3 matrix representing the vectorial product with \mathbf{e}). It can be easily verified that

$$\mathbf{x}'^T F \mathbf{x} = 0 \quad (25)$$

This result gives a characterization of the fundamental matrix using only image projections. F is a 3 matrix that has rank 2 (the epipole \mathbf{e} is the nullspace of F). A lot of effort has been

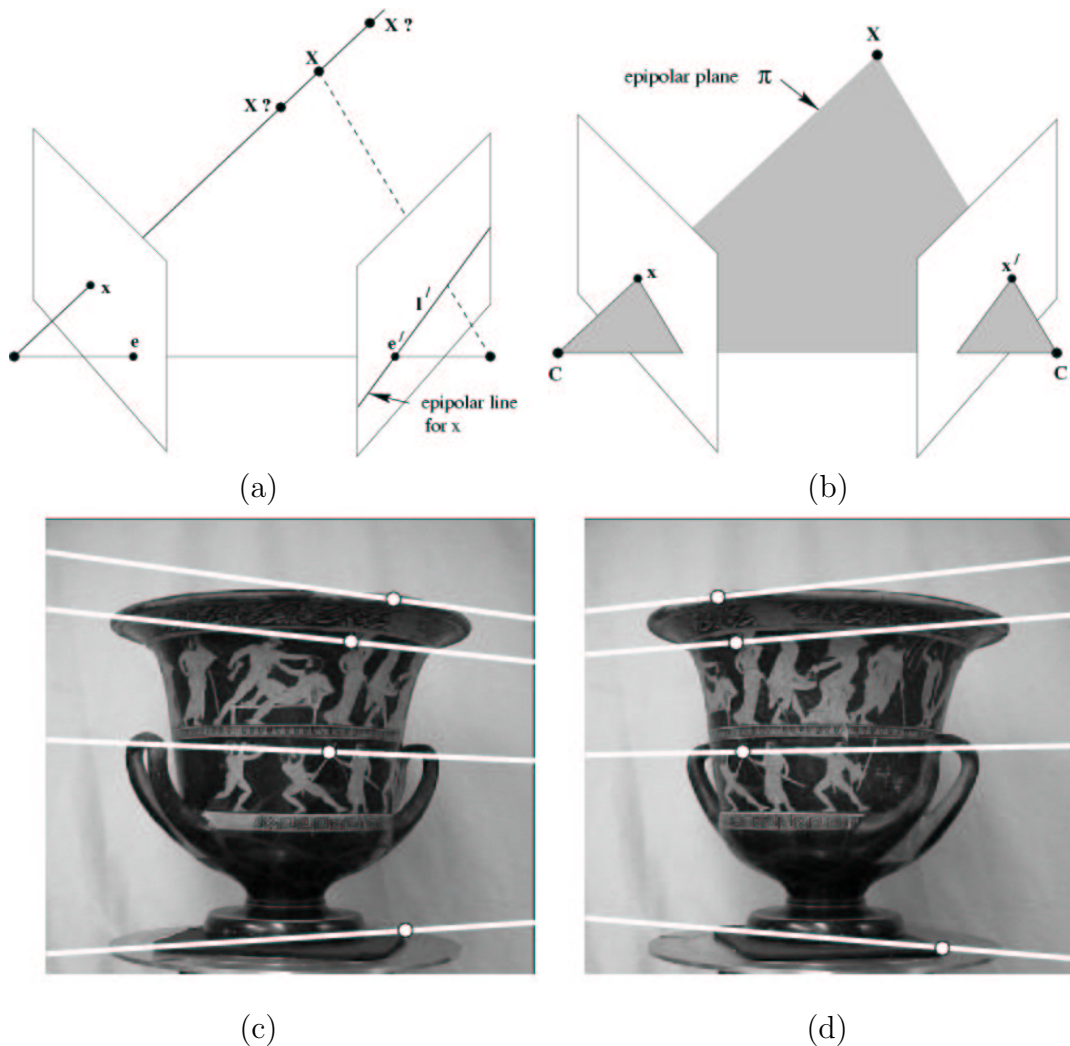


Figure 25: Epipolar geometry: (a) epipolar lines and epipoles (b) epipolar plane (c)(d) example of pairs of epipolar lines

put in robustly estimating F from a pair of uncalibrated images (e.g [54]). It can be estimated linearly given 8 or more corresponding points. A nonlinear solution uses 7 corresponding, but the solution is not unique.

We briefly list some important properties of the fundamental matrix that allows the computation of the epipolar lines and epipoles.

$$\mathbf{l}' = F\mathbf{x} \quad \mathbf{l} = F^T\mathbf{x}' \tag{26}$$

$$F\mathbf{e} = 0 \quad F^T\mathbf{e}' = 0 \tag{27}$$

Equation 25 is a projective relationship between image points and F only depends on projective properties of camera matrices. In general a camera matrix depends on the internal parameters and choice of world coordinate frame (external parameters). F does not depend on the choice of the world frame and is unchanged by a projective transformation of 3D space. So two projective

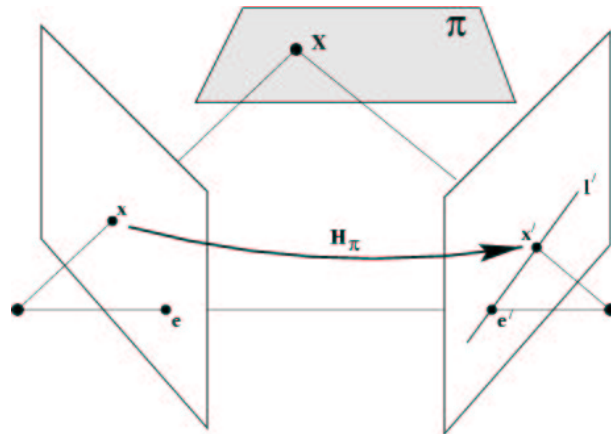


Figure 26: Geometric interpretation of the fundamental matrix

matrices P, P' uniquely determine F but the converse is not true. Given this projective ambiguity, we can formulate a class of projection matrices given a fundamental matrix (canonical cameras):

$$P = [I|0] \quad (28)$$

$$P' = [[\mathbf{e}']_{\times}F + \mathbf{e}'\mathbf{v}^T|\lambda\mathbf{e}'] \quad (29)$$

Calibrated stereo

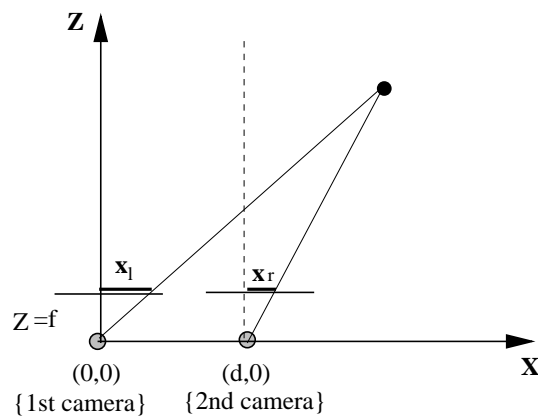


Figure 27: Calibrated stereo geometry

A special case of the two view geometry arises when the cameras are calibrated and aligned (the image planes are parallel with the line connected the camera centers). A 1D illustration of this configuration is presented in figure 27. In this case the epipolar lines correspond to image rows so corresponding points in two images are on the same scanline. The horizontal displacement between two corresponding points is called *disparity*. This special case has practical application in computing dense depth from images. Correspondence problem is simplified (corresponding point is on the same scanline) and there is a simple solution for depth, knowing the focal length

of the cameras f and the distance between the camera centers d :

$$Z = \frac{df}{x_l - x_r}$$

where $x_l - x_r$ represents the disparity (horizontal distance between corresponding calibrated image points). Note that the cameras have to be calibrated and x_l, x_r and normalized with respect to image scale and center. An example of a depth map computed using a trinocular vision system (from Point Gray Research [44]) is presented in figure 28.



Figure 28: Example of a disparity map computed using the Triclops vision system. (a) left intensity image (b) disparity map (c) Triclops device

More than two views

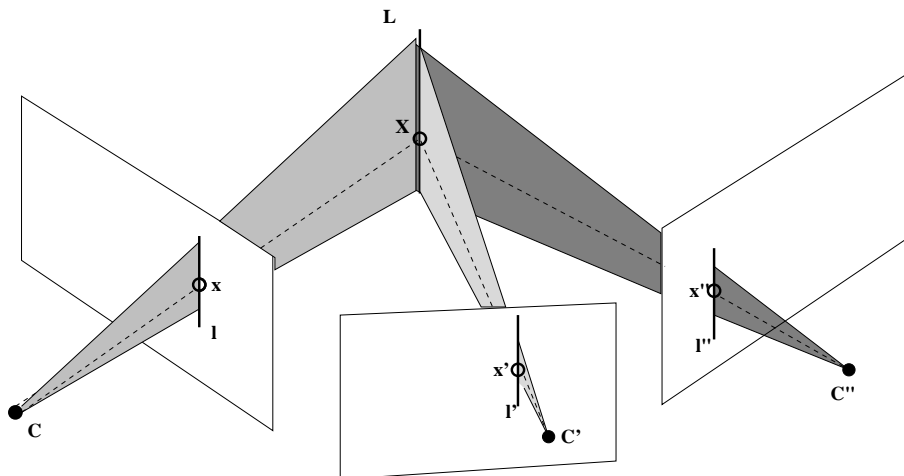


Figure 29: Trilinear constraints

Similar constraints can be defined for 3 and 4 images. The epipolar constraint can be formulated for three images, divided in groups of 2. But there is a stronger constraint that involves all three: the projection of a point in the third image can be computed from corresponding projections in the other two images. A similar constraint hold for lines or combinations of points/lines

(see figure 29). These are called trilinear constraints and can be expressed using the trifocal tensor [55].

Quadrifocal constraints are formulated for 4 images (using the quadrifocal tensor) [56]. An important result is that there are no additional constraints between more than 4 images. All the constraints can be expressed using F , the trilinear tensor and the quadrifocal tensor. A unifying theory uses multiview tensors [23, 19]. Here is a summary of different constraints and the minimal configurations for computing structure from motion:

2 images *epipolar geometry, F*

- linear unique solution from 8 points
- nonlinear solution from 7 points (3 solutions)

3 images *trifocal tensor*

- linear solution from 7 points
- nonlinear solution from 6 points

4 images *quadrifocal tensor*

- linear solution from 6 points

3.4 Multi view geometry

For the case when points are tracked in many images there is a linear formulation of the reconstruction problem, called factorization. The original requires that the image coordinates for all the points are available, so there are no occlusions. More recent extensions deal with missing data. Assuming an affine camera, nonisotropic zero-mean Gaussian noise, the factorization achieves ML affine reconstruction. This method and its extension to the weak perspective camera are presented in the following subsections.

Affine factorization

A general affine camera has the form

$$P_\infty = [M|\mathbf{t}]$$

where M is a 2×3 matrix and \mathbf{t} is a 2D vector (note that we dropped the last row $(0, 0, 0, 1)$ from the projection matrix). The projection equation using homogeneous coordinates can be written as:

$$\begin{pmatrix} x \\ y \end{pmatrix} = M \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \mathbf{t}$$

Having n points tracked in m images we form the $2m \times n$ measurement matrix W and we can write the projection equations in a compact way as:

$$W = \begin{bmatrix} \mathbf{x}_1^1 & \cdots & \mathbf{x}_n^1 \\ \vdots & & \vdots \\ \mathbf{x}_1^m & \cdots & \mathbf{x}_n^m \end{bmatrix} = \begin{bmatrix} M^1 \\ \vdots \\ M^m \end{bmatrix} [X_1 \cdots X_n] + \begin{bmatrix} \mathbf{t}^1 \\ \vdots \\ \mathbf{t}^m \end{bmatrix} = RS + \mathbf{t}\mathbf{1} \quad (30)$$

If the image points are registered with respect to their centroid in the image plane and the center of the world coordinate frame is the centroid of the shape points, the projection equation becomes:

$$\hat{W} = RS \text{ where } \hat{W} = W - \mathbf{t}\mathbf{1} \quad (31)$$

Rank theorem Following [53], in the absence of noise $rank(\hat{W}) = 3$. Under most viewing conditions with a real camera the effective rank is 3. Assuming $2m > n$, \hat{W} can be decomposed $\hat{W} = O_1 \Sigma O_2$, where O_1 is an orthonormal $2m \times n$ matrix, Σ is an $n \times n$ diagonal matrix and O_2 is an $n \times n$ orthonormal matrix (SVD).

Defining

$$\begin{aligned} \hat{R} &= O'_1 \\ \hat{S} &= \Sigma' O'_2 \end{aligned} \quad (32)$$

we can write

$$\hat{W} = \hat{R}\hat{S} \quad (33)$$

O'_1 is formed from the first three columns of O_1 , Σ' is the first 3×3 matrix of Σ and O'_2 contains the first three rows of O_2 (assuming the singular values are ordered in decreasing order).

The factorization calculates the affine shape of the structure \hat{S} and the affine projection matrices \hat{R} .

Weak perspective factorization

The weak perspective camera is a special case of affine camera where

$$M = \begin{bmatrix} s\mathbf{i} \\ s\mathbf{j} \end{bmatrix} \quad (34)$$

The matrices \hat{R} and \hat{S} resulted from the affine factorization, are a linear transformation of the metric scaled rotation matrix R and the metric shape matrix S . There is in general an affine ambiguity in the reconstruction. More specifically there exist a 3×3 matrix Q such that:

$$\begin{aligned} R &= \hat{R}Q \\ S &= Q^{-1}\hat{S} \end{aligned} \quad (35)$$

Normally, to align \hat{S} with an exocentric metric frame the world coordinates, at least four scene points are needed. Assuming no scene information is provided, \hat{S} can be aligned with the pixel coordinate system of the camera row and column. This relates Q to the the components of the scaled rotation R :

$$\begin{aligned} \hat{\mathbf{i}}_t^T Q Q^T \hat{\mathbf{i}}_t &= \hat{\mathbf{j}}_t^T Q Q^T \hat{\mathbf{j}}_t & (= s^2) \\ \hat{\mathbf{i}}_t^T Q Q^T \hat{\mathbf{j}}_t &= 0 \end{aligned} \quad (36)$$

where $\hat{R} = [\hat{\mathbf{i}}_1 \cdots \hat{\mathbf{i}}_m \hat{\mathbf{j}}_1 \cdots \hat{\mathbf{j}}_m]^T$. The first constraint assures that the corresponding rows $s_t \hat{\mathbf{i}}_t^T$, $s_t \hat{\mathbf{j}}_t^T$ of the scaled rotation R in Eq. 34 are unit vectors scaled by the factor s_t and the second equation constrain them to orthogonal vectors. This generalizes [53] from an orthographic to a weak perspective case. The resulting transformation is up to a scale and a rotation of the world coordinate system. To eliminate the ambiguity, the axis of the reference coordinate system is aligned with the first frame and estimate only eight parameters in Q (fixing a scale). This algorithm was adapted from [60].

Projective factorization

A similar idea can be adopted for projective cameras. Several algorithms exist in the literature [50, 22]. The projection equations $\lambda_j^i \mathbf{x}_j^i = P^i \mathbf{X}_j$ for n points and m images can be written as (in homogeneous coordinates):

$$W = \begin{bmatrix} \lambda_1^1 \mathbf{x}_1^1 & \cdots & \lambda_n^1 \mathbf{x}_n^1 \\ \vdots & & \vdots \\ \lambda_1^m \mathbf{x}_1^m & \cdots & \lambda_n^m \mathbf{x}_n^m \end{bmatrix} = \begin{bmatrix} P^1 \\ \vdots \\ P^m \end{bmatrix} [X_1 \cdots X_n] + \begin{bmatrix} \mathbf{t}^1 \\ \vdots \\ \mathbf{t}^m \end{bmatrix} \quad (37)$$

λ_j^i are called the projective depth and are in general unknown. Assuming we know λ_j^i (for example from an initial projective reconstruction), the camera matrices and 3D projective points can be computed using a factorization algorithm similar to the affine one. The measurement matrix has rank four in this case.

If the projective depths are unknown an iterative approach can be adopted. Initially they are all set to 1 and the structure is estimated using factorization. The depths are re-estimated by reprojecting the structure and the procedure is repeated. However, there is no guarantee that the procedure will converge to a global minimum.

Bundle adjustment

Consider that the projective 3D structure $\hat{\mathbf{X}}_j$ and the camera matrices \hat{P}_i had been estimated from a set of image features \mathbf{x}_j^i . The bundle adjustment refines these estimates by minimizing the geometric error

$$\min \sum_{i,j} d(\hat{P}_i \hat{\mathbf{X}}_j, \mathbf{x}_j^i)^2 \quad (38)$$

The name *bundle adjustment* means readjusting bundle of rays between the camera center and the set of 3D points to fit the measured data. The solution looks for the Maximum Likelihood (ML) estimate assuming that the measurement noise is Gaussian. This step is very important in any projective reconstruction and is tolerant to missing data.

3.5 Recovering metric structure

Projective ambiguity

As mentioned before in subsection 3.3, given an uncalibrated sequence of images with corresponding points identified it is possible to reconstruct the structure only up to a projective transformation.

But, there exist a homography H (or a family of homographies) such that the transformed matrices $P^i H$ represent true projection matrices and can be decomposed as: $P^i H = K R^i [I | \mathbf{t}^i]$, where K, R^i, \mathbf{t}^i represent the internal and external parameters of the camera as described in subsection 3.1. The projective structure is upgraded to metric by $H^{-1} X_j$.

The general approach for a metric reconstruction is:

- Obtain a projective reconstruction P^i, X_j

- Determine the rectifying homography H from autocalibration constraints, and transform the reconstruction to metric $P^i H, H^{-1} X_j$

Self-calibration

The theoretical formulation of self-calibration (auto-calibration) for constant camera parameters was first introduced by Faugeras [10]. Present techniques can be divided in two main categories:

- a stratified approach that first determines the affine structure and then the metric structure using the absolute conic;
- a direct approach that recovers the metric structure using the dual absolute quadric.

The rectifying homography can be expressed depending on the plane at infinity $\pi_\infty = (\mathbf{p}^T, 1)^T$ and camera calibration matrix K^1 of the first camera:

$$H = \begin{bmatrix} K^1 & \mathbf{0} \\ -\mathbf{p}^T K^1 & 1 \end{bmatrix} \quad (39)$$

Considering the projection matrices $P^i = [A^i | \mathbf{a}^i]$ the auto-calibration equation are as follows:

$$K^i K^{iT} = (A^i - \mathbf{a}^i \mathbf{p}^T) K^1 K^{1T} (A^i - \mathbf{a}^i \mathbf{p}^T)^T \quad (40)$$

Note that $\omega^{*i} = K^i K^{iT}$, the dual image of the absolute conic (DIAC), and the equation can be rewritten as:

$$\omega^{*i} = (A^i - \mathbf{a}^i \mathbf{p}^T) \omega^{*1} (A^i - \mathbf{a}^i \mathbf{p}^T)^T \quad (41)$$

The auto-calibration methods determine \mathbf{p} and K^1 (8 parameters) based on constraints on K^i such that one of its elements is zero. Triggs [57] introduced the absolute dual quadric for as a numeric device for formulating auto-calibration equations:

$$\omega^{*i} = P^i Q_\infty^* P^{iT} \quad (42)$$

and in this case Q_∞^* is first estimated based on similar constraints on K^i and then decomposed as $Q_\infty^* = H \tilde{I} H^T$ where $\tilde{I} = \text{diag}(1, 1, 1, 0)$ is the canonical configuration of the absolute quadric. Methods that assume constant internal parameters were developed by Hartley [18], Pollefeys and Van Gool [43]. A more flexible self-calibration with varying focal length was proposed by Heyden and Åström [24].

Note that self-calibration is not a general solution to a metric reconstruction. Some critical motions can generate degenerate solutions (e.g. planar motion and constant internal parameters) and the constraints on the internal parameters has to be carefully chosen depending on each real camera. Some external constraints on the scene (if available) like knowledge about parallel lines, angles might improve the robustness. Metric bundle adjustment is recommended as a final step.

3.6 A complete system for geometric modeling from images

Putting together the presented techniques for extracting geometric structure from uncalibrated images we now have a general procedure for calculating the metric structure:

1. Get initial corresponding points.
2. 2,3 view geometry: compute F , trilinear tensor between consecutive frames and recompute correspondences.
3. Initial reconstruction: get an initial projective reconstruction from a subsequence with big baseline (e.g. using chains of fundamental matrices, trilinear tensors or factorization) and bind more frames/points using resection/intersection.
4. Bundle adjustment
5. Self-calibration
6. Metric bundle adjustment

4 Texturing of Image-based Models

Martin Jagersand

With *texture* normally mean fine scale visual or tactile properties of a surface. The word is related to textile, and indeed it was used to describe the particular surface created by the the interwoven threads in a fabric. In computer graphics texturing is the process of endowing a surface with fine scale properties. Often this is used to make the visualization richer and more natural than if only the 3D geometry had been rendered. There are a wide range of computer graphics texturing approaches. Early texturing involved replicating a small texture element over the surface of an object to enrich its appearance. Commonly this is done by warping a small 2D texture element onto the structure but other variations include 3D texturing, where a surface appearance is created by carving away into a 3D texture volume. Texture can also be used to model light, e.g. using a specular highlight texture on a model and reflections.

The focus of this Section is on image texturing. We will study how to compose a texture image suitable for photo-realistic image-based rendering. In this case the texturing element is a comparably large image, and unlike the above mentioned techniques, not repeated over the surface. Hence, we are to some extent transcending out of the original domain of texturing by now not only modeling fine scale structure, but potentially medium and large scale geometry in texture. We will focus on aspects that are specific to image based modeling and rendering, and not treat standard issues such as implementation of 2-D warps, filtering and multi-resolution texturing. The background on basic texturing is covered in the literature[21] and recent text books[36].

In particular, we will describe how to make the texture correctly represent the variation over different viewpoints of a potentially complex underlying geometry. Using the tools of 3D warping, *relief textures* provides an explicit geometric solution to adding 3D structure to a planar texture[39]. However, relief textures require a detailed a-priori depth map of the texture element. This is normally not available in image-based modeling if only uncalibrated camera video is used. An alternative way to represent the image motion caused by depth parallax is by modulating a spatial image basis. Previously, this technique has been used in image (camera) plane encoding of deterministic scene motion[26] and capturing certain quasi-periodic motions[8]. Recently, it has been generalized and made more efficient by parameterizing this motion on a scene geometry instead of the image plane[6]. The following development of *dynamic textures* will closely follow the latter scene geometry based derivation of image variability.

In the previous Section we saw how uncalibrated video can be used to obtain geometric information from a scene. A structure-from-motion (SFM) method starts with a set of m images $I_1 \dots I_m$ from different views of a scene. Through visual tracking or correspondence detection the image projection $x_1 \dots x_n$ of n physical scene points are identified in every image. From this, a structure from motion computes a structure, represented by a set of n scene points $X_1 \dots X_n$, and m view projections $P_1 \dots P_m$ such that (reprojection property):

$$x_{j,i} = P_j X_i \quad i \in 1 \dots n, j \in 1 \dots m \quad (43)$$

Independent of the geometric details and interpretation and central to image based modeling and rendering is that this structure can be reprojected into a new virtual camera and thus

novel views can be rendered. Practically, the structure is divided into Q planar facets (triangles or quadrilaterals are used in the experiments) with the points $x_{j,i}$ as node points. For texture mapping, each one of the model facets are related by a planar projective homography to a texture image. See Fig. 44.

4.1 Texture basis

In conventional texture mapping, one or more of the real images are used as a source to extract texture patches from, and then warped onto the re-projected structure in the new view.

Instead of using an image as a source texture, here we study how to relate and unify all the input sample images into a texture basis. Let $x_{T,i}$ be a set of texture coordinates in one-to-one correspondence to each model point X_i and thus also for each view j with the image points $x_{j,i}$ above. A texture warp function w translates the model vertex to texture correspondences into a pixel-based re-arrangement (or warp) between the texture space I_w to screen image coordinates I .

$$T(x) = I(\mathcal{W}(x)) \quad (44)$$

Common such warp functions are affine, bi-linear and projective warps. The warp function \mathcal{W} acts by translating, rotating and stretching the parameter space of the image, and hence for discrete images a re-sampling and filtering step is needed between the image and texture spaces. Details of these practicalities can be found in [36].

Now if for each sample view j , we warp the real image I_j from image to texture coordinates into a texture image T_j , we would find that in general $T_j \neq T_k$, $j \neq k$. Typically, the closer view j is to k , the smaller is the difference between T_j and T_k . This is the rationale for view-dependent texturing, where a new view is textured from one to three (by blending) closest sample images[7].

In this paper we will develop a more principled approach, where we seek a texture basis B such that for each sample view:

$$\mathbf{T}_j = B\mathbf{y}_j, \quad j \in 1 \dots m. \quad (45)$$

Here, and in the following, \mathbf{T} is a $q \times q$ texture image flattened into a $q^2 \times 1$ column vector. B is a $q^2 \times r$ matrix, where normally $r \ll m$, and \mathbf{y} is a modulation vector. The texture basis B needs to capture geometric and photometric texture variation over the sample sequence, and correctly interpolate new in-between views. We first derive a first order geometric model, then add the photometric variation. For clarity we develop these applied to one texture warp (as in Fig. 36), while in practical applications a scene will be composed by texturing several model facets (as in Figures 44 and 38).

4.2 Connection between intensity change and motion

To capture texture variation due to geometric deformation we seek to derive a spatial basis for the particular small motion present in textures captured from different view-points. At first it may seem contradictory that geometric motion can be generated by linear combinations of a basis. Normally the linear combination, ie blending, of two images dissolves one image into the other. This typically results in a blurry intermediate images with no particular motion perception. However, here we will show that for particular choices of bases and corresponding

blending functions small geometric motions can be modulated. As a very simple motivating example consider that a drifting sine wave grating $I = \sin(u+at)$ can be translated into arbitrary phase by modulating just two basis functions, $I = \sin(u+at) = \sin(u)\cos(at) + \cos(u)\sin(at) = \sin(u)y_1 + \cos(u)y_2$, where y_1 and y_2 are mixing coefficients.

Using a set of filters with different spatial frequencies has been applied to synthesizing image motion in predictive display, a field of robotics where predictive images are synthesized to compensate for delays in the operator-robot control loop[27]. Consider the situation in Fig. 30. A camera is observing the motion of a robotic hand manipulating a block. Exact a-priori modeling

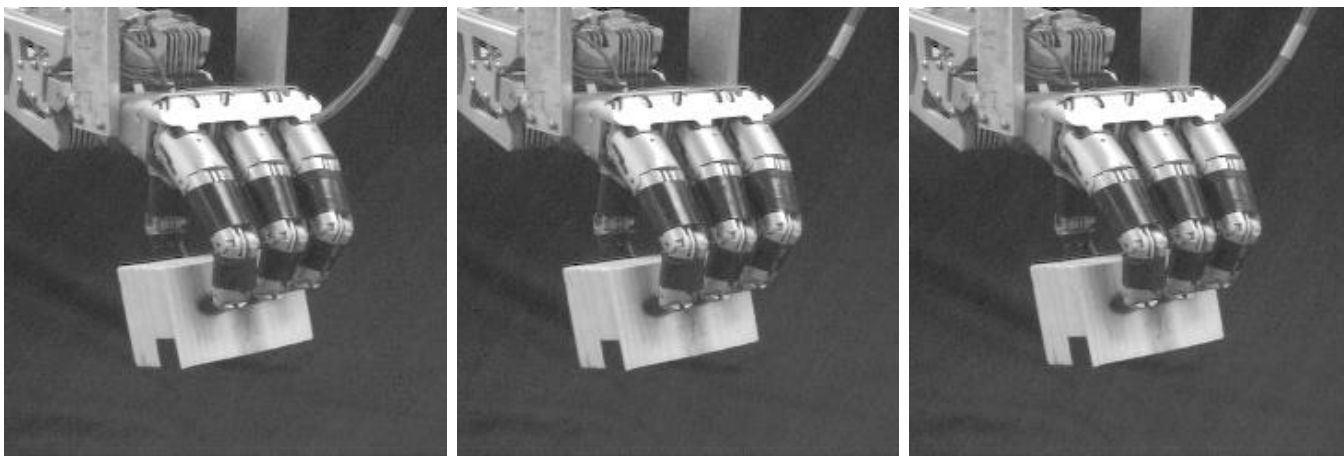


Figure 30: Different poses of a Utah/MIT robotic hand performing fingertip manipulation of a block.

of these motions is difficult due to the complex kinematics of the hand. However, from a short video sequence of motions we can build a spatial basis capturing the image variation. Figure 31 shows the first through sixth basis vector, B_1, \dots, B_6 . As can be seen they have localized patches with roughly the same spatial frequency, but different phase. For instance in the complementary pair B_5 and B_6 the grasped block has a component with approximately 20 pixels period roughly horizontally in both images. The fingers have somewhat higher frequencies, since small finger motions correspond to larger motions of the block. Using only linear combinations of B_5 and B_6 a drifting grating motion effect¹ can be created, see [web-video]. By also adding the low order components $B_1 \dots B_4$ a coherent motion and shape signal can be generated up to the cut off resolution (about 20 pixels here). Notice that these results are different from what has been presented in the face and object recognition literature (e.g. [59]). But our sample set is also very different. In the predictive display application we do not sample different objects or faces, but closely spaced images from the same objects and scene under varying poses.

Modulation of the Spatial Basis Another important characteristic is how the modulation coefficients \mathbf{y} space manifold maps to motions. It has been shown that under varying camera motion the space of the modulation coefficients form a smooth manifold[37]. In Fig. 32 the first

¹The detection of visual motion in the human brain is believed to be well modeled by a linear system with a scale-space hierarchy of eigenfunctions, "gabor patches" each with a localized spatial and frequency response. Hence a stimuli composed of similar functions will maximally stimulate the system

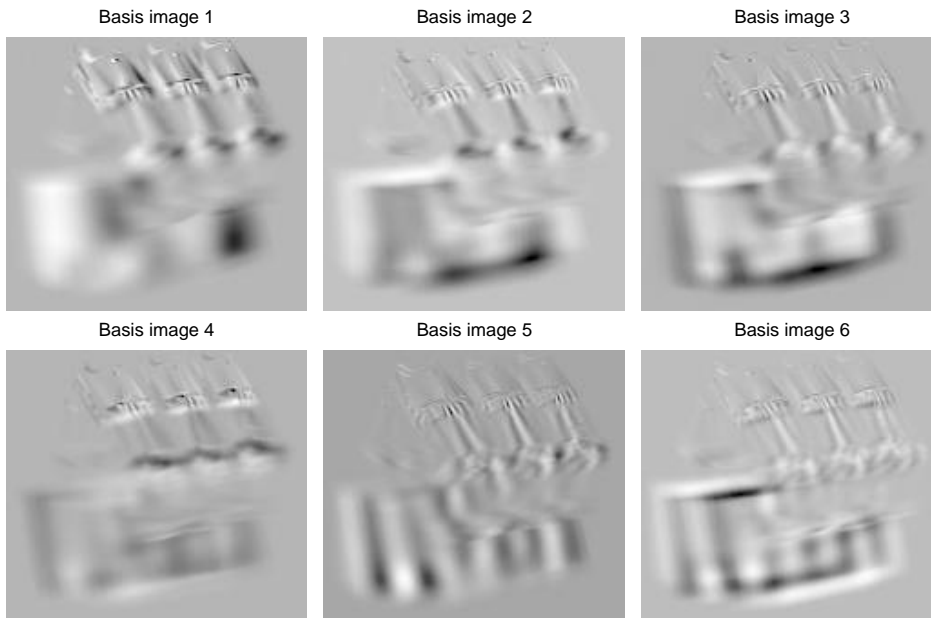


Figure 31: Utah/MIT fingertip manipulation of a block. Two example basis images, \mathbf{u}_5 and \mathbf{u}_6 , having roughly the same spatial frequency content

six components $f_1 \dots f_6$ are plotted against the two first motor components x_1 and x_2 .

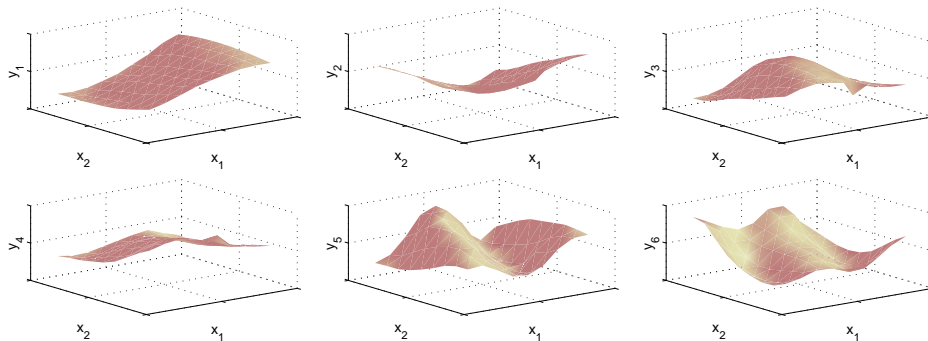


Figure 32: Six first components of the motor-visual function f captured for motions shown in Fig.30.

Small motions are synthesized by linear combinations of basis functions of different phase and direction. Six images from a synthesized animation of hand motion are shown in Fig. 33.

4.3 Geometric texture variation

In the following we will develop a spatial variability basis for the image motion caused by texture warps. While no real physical motion is involved, we can think of the action of moving the texture facet control point as causing a virtual image motion. The starting point for developing a spatial texture basis representing small geometric variations is the well known optic flow constraint,

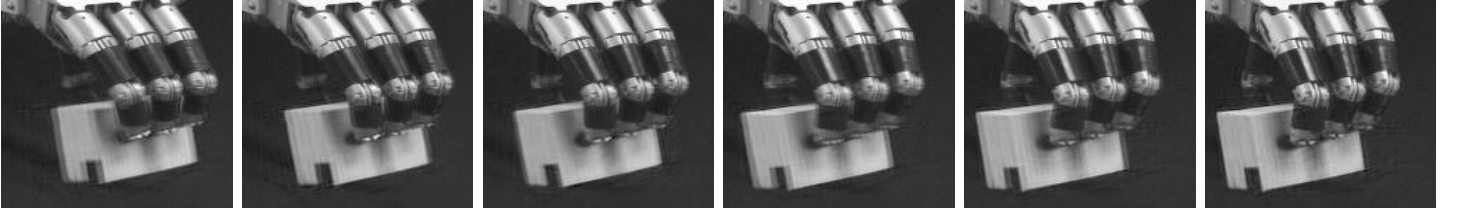


Figure 33: Images from synthesized animation of hand motions.

which for small image plane translations relates texture intensity change $\Delta T = T_j - T_k$ to spatial derivatives $\frac{\partial T}{\partial u}, \frac{\partial T}{\partial v}$ with respect to texture coordinates $\mathbf{x} = [u, v]^T$ under an image constancy assumption[17].

$$\Delta T = \frac{\partial T}{\partial u} \Delta u + \frac{\partial T}{\partial v} \Delta v \quad (46)$$

Note that given one reference texture T_0 we can now build a basis for small image plane translations $B = [T_0, \frac{\partial T}{\partial u}, \frac{\partial T}{\partial v}]$ and from this generate any slightly translated texture $T(\Delta u, \Delta v) = B[1, \Delta u, \Delta v]^T = B\mathbf{y}$

In rendering, we are interested not only in translations but in the effects of parameterized warps. Given a warp function $\mathbf{x}' = \mathcal{W}(a, \mathbf{x})$ we study the residual image variability introduced by the imperfect stabilization achieved by a perturbed warp \hat{w} , $\Delta T = T(\hat{w}, t) - T(\mathcal{W})$. Let $\hat{w} = w + \Delta w$ and rewrite as an approximate image variability to the first order (dropping t):

$$\Delta T = T(w + \Delta w) - T_w = T(w) + \frac{\partial}{\partial w} T(w) \Delta w - T_w = \frac{\partial}{\partial w} T(w) \Delta w \quad (47)$$

Explicitly writing out the components of the inner derivatives (Jacobian) we have:

$$\Delta T = \left[\frac{\partial T}{\partial u}, \frac{\partial T}{\partial v} \right] \left[\begin{array}{c} \frac{\partial u}{\partial a_1} \dots \frac{\partial u}{\partial a_k} \\ \frac{\partial v}{\partial a_1} \dots \frac{\partial v}{\partial a_k} \end{array} \right] \Delta [a_1 \dots a_k]^T \quad (48)$$

The above equation expresses an optic flow type constraint in an abstract formulation without committing to a particular form or parameterization of w . In practice, the function w is usually discretized using e.g. triangular or quadrilateral mesh elements. Next we give examples of how to concretely express image variability from these discrete representations.

Particularly for image based modeling and rendering we warp real source images into new views given an estimated scene structure. Errors between the estimated and true scene geometry cause these warps to generate imperfect renderings. We divide these up into two categories, image plane and out of plane errors. The planar errors cause the texture to be sourced with an incorrect warp.² The out of plane errors arise when piecewise planar facets in the model are not true planes in the scene, and when re-warped into new views under a false planarity assumption will not correctly represent parallax.

²Errors in tracking and point correspondences when computing the SFM, as well as projection errors due to differences between the camera model and real camera both cause model points to be reprojected incorrectly in the sample images

Planar texture variability First we will consider geometric errors in the texture image plane. In most both IBR (as well as conventional rendering) textures are warped onto the rendered view from a source texture \mathbf{T} by means of a affine warp or projective homography.

Affine variation Under a weak perspective (or orthographic) camera geometry, plane-to-plane transforms are expressed using an affine transform of the form:

$$\begin{bmatrix} u_w \\ v_w \end{bmatrix} = \mathcal{W}_a(\mathbf{p}, \mathbf{a}) = \begin{bmatrix} a_3 & a_4 \\ a_5 & a_6 \end{bmatrix} \mathbf{p} + \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \quad (49)$$

This is also the standard image-to-image warp supported in OpenGL. Now we can rewrite the image variability Eq. 48 resulting from variations in the six affine warp parameters as:

$$\Delta T_a = \sum_{i=1}^6 \frac{\partial}{\partial a_i} T_w \Delta a_i = \begin{bmatrix} \frac{\partial T}{\partial u} & \frac{\partial T}{\partial v} \end{bmatrix} \begin{bmatrix} \frac{\partial u}{\partial a_1} & \dots & \frac{\partial u}{\partial a_6} \\ \frac{\partial v}{\partial a_1} & \dots & \frac{\partial v}{\partial a_6} \end{bmatrix} \Delta [a_1 \dots a_6]^T \quad (50)$$

Let $\{T\}_{discr} = \mathbf{T}$ be a discretized texture image flattened along the column into a vector, and let '* \mathbf{u} ' and '* \mathbf{v} ' indicate point-wise multiplication with column flattened camera coordinate u and v index vectors. Rewrite the inner derivatives to get an explicit expression of the six parameter variability in terms of spatial image derivatives:

$$\Delta \mathbf{T}_a = \begin{bmatrix} \frac{\partial \mathbf{T}}{\partial u} & \frac{\partial \mathbf{T}}{\partial v} \end{bmatrix} \begin{bmatrix} 1 & 0 & * \mathbf{u} & 0 & * \mathbf{v} & 0 \\ 0 & 1 & 0 & * \mathbf{u} & 0 & * \mathbf{v} \end{bmatrix} [y_1, \dots, y_6]^T = \begin{bmatrix} \mathbf{B}_1 & \dots & \mathbf{B}_6 \end{bmatrix} [y_1, \dots, y_6]^T = B_a \mathbf{y}_a \quad (51)$$

where $[\mathbf{B}_1 \dots \mathbf{B}_6]$ can be interpreted as a texture variability basis for the affine transform.

Projective variation Under a perspective camera the plane-to-plane warp is expressed by a projective collineation or homography,

$$\begin{bmatrix} u' \\ v' \end{bmatrix} = \mathcal{W}_h(\mathbf{x}_h, \mathbf{h}) = \frac{1}{1 + h_7 u + h_8 v} \begin{bmatrix} h_1 u & h_3 v & h_5 \\ h_2 u & h_4 v & h_6 \end{bmatrix} \quad (52)$$

Rewrite Eq. 48 with the partial derivatives of \mathcal{W}_h for the parameters $h_1 \dots h_8$ into a Jacobian matrix. Let $c_1 = 1 + h_7 u + h_8 v$, $c_2 = h_1 u + h_3 v + h_5$, and $c_3 = h_2 u + h_4 v + h_6$. The resulting texture image variability due to variations the estimated homography is (to the first order) spanned by the following spatial basis:

$$\Delta \mathbf{T}_h = \frac{1}{c_1} \begin{bmatrix} \frac{\partial \mathbf{T}}{\partial u} & \frac{\partial \mathbf{T}}{\partial v} \end{bmatrix} \begin{bmatrix} u & 0 & v & 0 & 1 & 0 & -\frac{u c_2}{c_1} & -\frac{v c_2}{c_1} \\ 0 & u & 0 & v & 0 & 1 & -\frac{u c_3}{c_1} & -\frac{v c_3}{c_1} \end{bmatrix} \begin{bmatrix} \Delta h_1 \\ \vdots \\ \Delta h_8 \end{bmatrix} = \begin{bmatrix} \mathbf{B}_1 & \dots & \mathbf{B}_8 \end{bmatrix} [y_1, \dots, y_8]^T = B_h \mathbf{y}_h \quad (53)$$

Similar expressions can be derived for other warps. E.g. in real time vidusal tracking a four parameter variability from modeling image u , v translations, image plane rotations and scale has shown to be suitable[17].

Non-planar parallax variation While in image-based modeling a scene is represented as piecewise planar model facets, the real world scene is seldom perfectly planar. In rendering this gives rise to parallax errors. Figure 34 illustrates how the texture plane image T changes for

different scene camera centers C . Given a depth map $d(u, v)$ representing the offset between the scene and texture plane, relief texturing [39] can be used to compute the rearrangement (pre-warp) of the texture plane before the final homography renders the new view. In image-based methods, an accurate depth map is seldom available. However we can still develop the analytic form of the texture intensity variation as above. Let $r = [\alpha, \beta]$ be the angle for view P_j between the ray from the camera center C_j to each scene point. The pre-warp rearrangement needed on the texture plane to correctly render this scene using a standard homography warp is then:

$$\begin{bmatrix} \delta u \\ \delta v \end{bmatrix} = \mathcal{W}_p(\mathbf{x}, \mathbf{d}) = d(u, v) \begin{bmatrix} \tan \alpha \\ \tan \beta \end{bmatrix} \quad (54)$$

As before, taking the derivatives of the warp function with respect to a camera angle change and inserting into Eq.48 we get:

$$\Delta \mathbf{T}_p = d(u, v) \begin{bmatrix} \frac{\partial \mathbf{T}}{\partial u} & \frac{\partial \mathbf{T}}{\partial v} \end{bmatrix} \begin{bmatrix} \frac{1}{\cos^2 \alpha} & 0 \\ 0 & \frac{1}{\cos^2 \beta} \end{bmatrix} \begin{bmatrix} \Delta \alpha \\ \Delta \beta \end{bmatrix} = B_p \mathbf{y}_p \quad (55)$$

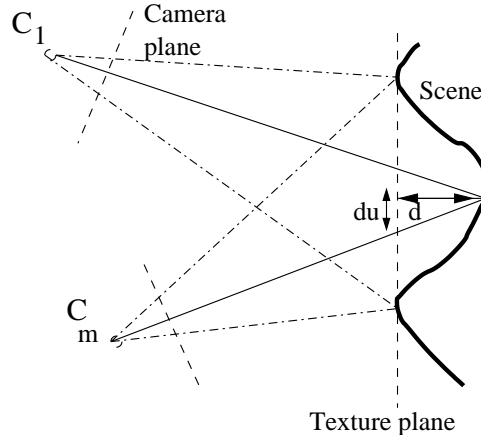


Figure 34: Texture parallax between two views.

Non-rigidity We consider only non-rigidities where the shape change is a function of some measurable quantity $\mathbf{x} \in \mathfrak{R}^n$. In this paper we choose \mathbf{x} from pose and articulation parameters. Let $\mathbf{g}(\mathbf{x}) (= [u, v]^T)$ represent the image plane projection of the non-rigid warp. We can then write the resulting first order image variation as:

$$\begin{aligned} \Delta \mathbf{I}_n &= \left\{ \sum_{i=1}^n \frac{\partial \mathbf{I}}{\partial x_i} \Delta x_i \right\}_{discr} = \\ & \left\{ \begin{bmatrix} \frac{\partial \mathbf{I}}{\partial u} & \frac{\partial \mathbf{I}}{\partial v} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x_1} g(x) \Delta x_1, \dots, \frac{\partial}{\partial x_n} g(x) \Delta x_n \end{bmatrix} \right\}_{discr} = \\ & \begin{bmatrix} \mathbf{B}_1 & \dots & \mathbf{B}_n \end{bmatrix} [y_1, \dots, y_n]^T = B_n \mathbf{y}_n \end{aligned} \quad (56)$$

4.4 Photometric variation

In image-based rendering real images are re-warped into new views, hence the composite of both reflectance and lighting is used. If the light conditions are same for all sample images, there is no additional intensity variability introduced. However, commonly the light will vary at least somewhat. In the past decade several published both empirical studies and theoretical motivations have shown that a low dimensional intensity subspace of dimension 5-9 is sufficient for representing the light variation of most natural scenes[30]. Hence we introduce nine additional freedoms in our variability model to allow for lighting. (Complex scenes may require more, simple (convex lambertian) less).

$$\Delta \mathbf{T}_l = [\mathbf{B}_1 \dots \mathbf{B}_9][y_1 \dots y_9]^T = B_l \mathbf{y}_l \quad (57)$$

4.5 Estimating composite variability

In textures sampled from a real scene using an estimated geometric structure we expect that the observed texture variability is the composition of the above derived planar, parallax and light variation, as well as unmodeled effects and noise $\Delta \mathbf{I}_e$. Hence, total residual texture variability can be written as:

$$\Delta \mathbf{I} = \Delta \mathbf{I}_s + \Delta \mathbf{I}_d + \Delta \mathbf{I}_l + \Delta \mathbf{I}_e \quad (58)$$

Using the basis derived above we can write the texture for any sample view k , and find a corresponding texture modulation vector \mathbf{y}_k :

$$\mathbf{T}_k = [\mathbf{T}_0, B_h, B_p, B_l][1, y_1, \dots, y_{19}] = B \mathbf{y}_k \quad (59)$$

Textures for new views are synthesized by interpolating the modulation vectors from the nearest sample views into a new \mathbf{y} , and computing the new texture $T_{new} = B \mathbf{y}$

Since this basis was derived as a first order representation it is valid for (reasonably) small changes only. In practical image-based modeling the geometric point misalignments and parallax errors are typically within a few pixels, which is small enough.

In IBR typically, neither the dense depth needed to analytically compute B_p , nor light and reflectance models needed for B_l are available. Instead the only available source of information are the sample images $I_1 \dots I_m$ from different views of the scene, and from these, the computed corresponding textures $\mathbf{T}_1 \dots \mathbf{T}_m$.

However, from the above derivation we expect that the effective rank of the sample texture set is the same as of the texture basis B , i.e. $\text{rank}[\mathbf{T}_1, \dots, \mathbf{T}_m] \approx 20$. Hence, from $m \gg 20$ (typically 100-200) sample images we can estimate the best fit (under some criterion) rank 20 subspace using e.g. PCA, SVD, or ICA.

Briefly PCA can be performed as follows. Form a measurement matrix $A = [\mathbf{I}_z(1), \dots, \mathbf{I}_z(m)]$. The principle components are the eigen vectors of the covariance matrix $C = AA^T$. A dimensionality reduction is achieved by keeping only the first k of the eigenvectors. For practical reasons, usually $k \ll M \ll l$, where l is the number of pixels in the texture patch, and the covariance matrix C will be rank deficient. We can then save computational effort by instead computing $L = A^T A$ and eigen vector factorization $L = VDV^T$, where V is an ortho-normal and D a

diagonal matrix. From the k first eigenvectors $\hat{V} = [\mathbf{v}_1 \dots \mathbf{v}_k]$ of L we form a k -dimensional eigenspace \hat{B} of C by $\hat{B} = A\hat{V}$. Using the estimated \hat{B} we can now write a least squares optimal estimate of any intensity variation in the patch as

$$\Delta \mathbf{I} = \hat{B} \hat{\mathbf{y}}, \quad (60)$$

the same format as Eq. 58, but without using any a-priori information to model B . While $\hat{\mathbf{y}}$ captures the same variation as \mathbf{y} , it is not parameterized in the same coordinates. For every training image \mathbf{I}_t we have from the orthogonality of \hat{V} that the corresponding texture mixing coefficients are the columns of $[\hat{\mathbf{y}}_1, \dots, \hat{\mathbf{y}}_m] = \hat{V}^T$. From the factorization of geometric structure we also have the corresponding \mathbf{x}_t .

This yields an estimated texture basis \hat{B} and corresponding space of modulation vectors $\hat{\mathbf{y}}_1, \dots, \hat{\mathbf{y}}_m$ in one-to-one correspondence with the m sample views. From the derivation of the basis vectors in B we know this variation will be present and dominating in the sampled real images. Hence, the analytical B and the estimate \hat{B} span the same space and just as before, new view dependent textures can now be modulated from the estimated basis by interpolating the $\hat{\mathbf{y}}$ corresponding to the closest sample views and modulating a new texture $\mathbf{T} = \hat{B} \hat{\mathbf{y}}$. Practically to estimate the texture mixing coefficients for intermediate poses, we first apply n -dimensional Delaunay triangulation over the sampled poses $\mathbf{x}_{1\dots m}$. Then given a new pose \mathbf{x} we determine which simplex the new pose is contained in, and estimate the new texture mixing coefficients $\hat{\mathbf{y}}$ by linearly interpolating the mixing coefficients of the corner points of the containing simplex.

4.6 Interpretation of the variability basis

In our application, the geometric model captures gross image variation caused by large movements. The remaining variation in the rectified patches is mainly due to:

1. Tracking errors as well as errors due to geometric approximations (e.g. weak perspective camera) cause the texture to be sourced from slightly inconsistent locations in the training images. These errors can be modeled as a small deviation $\Delta[h_1, \dots, h_8]^T$ in the homography parameters from the true homography, and causes image differences according to Eq. 53. The weak perspective approximations, as well as many tracking errors are persistent, and a function of object pose. Hence they will be captured by \hat{B} and indexed in pose \mathbf{x} by f .
2. Depth variation is captured by Eq. 55. Note that projected depth variation along the camera optic axis changes as a function of object pose.
3. Assuming fixed light sources and a moving camera or object, the light variation is a function of relative camera-object pose as well.

From the form of Equations 53 and 55 we expect that pose variations in the image sequence will result in a texture variability described by combinations of spatial image derivatives. In Fig.35 we compare numerically calculated spatial image derivatives to the estimated variability basis \hat{B} .

In synthesizing texture to render a sequence of novel images the function f modulates the filter bank B so that the new texture dynamically changes with pose \mathbf{x} according to $\mathbf{T}_w = B\hat{f}(\mathbf{x}) + \bar{\mathbf{T}}$.

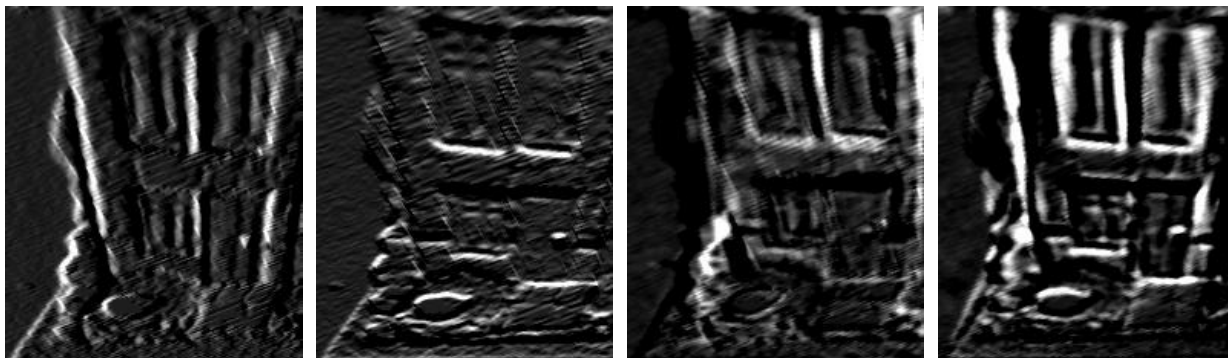


Figure 35: Comparison between spatial derivatives $\frac{\partial T_w}{\partial x}$ and $\frac{\partial T_w}{\partial y}$ (left two texture patches) and two vectors of the estimated variability basis $[\mathbf{B}_1, \mathbf{B}_2]$ (right) for house pictures.

4.7 Examples of Dynamic Texture Renderings

In the first example, a wreath from dried flowers and grains was captured from 128 frames of video with varying camera pose. The geometry used is just one quadrilateral. In Fig. reffig:wreath renderings of the wreath onto a quadrilateral is shown both using a standard static textures and pose varying dynamic texture. On the tutorial web site is an mpeg video of the same wreath rendered under rotating motion.

In the next example, a toy house made from natural materials (bark, wood, pine cone chips) is captured using a better quality Basler A301fc camera with a 12mm lens. This results in a better alignment between captured geometry and real scene points. However, using automatic image point tracking and triangulation the resulting triangles sometimes do not correspond to house walls (Fig. 37 left). Using standard texture mapping on this geometry we get significant geometric distortions (Fig. 37 right). In Fig. 38 these errors have been compensated for by modulating the texture basis to give the correct parallax for the underlying real surfaces.

As an example of a non-rigid object animation we captured the geometry of an arm from the tracked positions of the shoulder and hand using image-plane motions. We recorded a 512 sample images and estimate a texture basis of size 100. The arm motion was coded using the image position of the hand. Figure 40 shows some intermediate position reanimated using the dynamic texture algorithm. To test the performance of our algorithm we generate parallel renderings using static texture. To be consistent with the dynamic texturing algorithm we sampled texture from an equally spaced 100 images subset of the original sequence. Figure 39 illustrates the performance of the two algorithms for re-generating one of the initial positions and for a new position. We also created a movie (`arm.mpg`) that reanimates the arm in the two cases. Notice the geometric errors in the case of static texture algorithm that are corrected by the dynamic texture algorithm.

Quantitative comparison In order to quantitatively analyze how modulating a texture basis performs compared to standard view dependent texturing from a close real image, we produced three image sequences. The first image sequence are 80 real scene images of the wreath viewed under different camera poses from straight on to approximately 50 degrees off axis. The second image sequence is a synthesized rendering of those same poses from the texture basis (Fig. 36

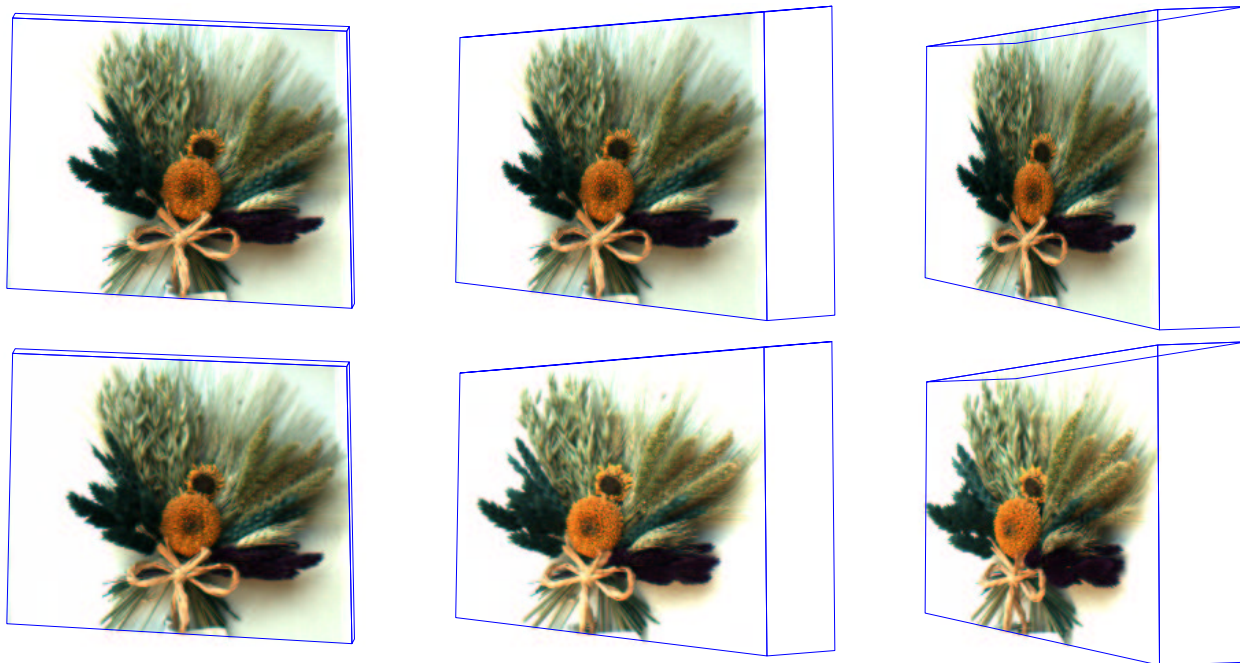


Figure 36: Texturing a rotating quadrilateral with a wreath. Top: by warping a flat texture image. Bottom: by modulating the texture basis B and generating a continuously varying texture which is then warped onto the same quad. Demo on web site

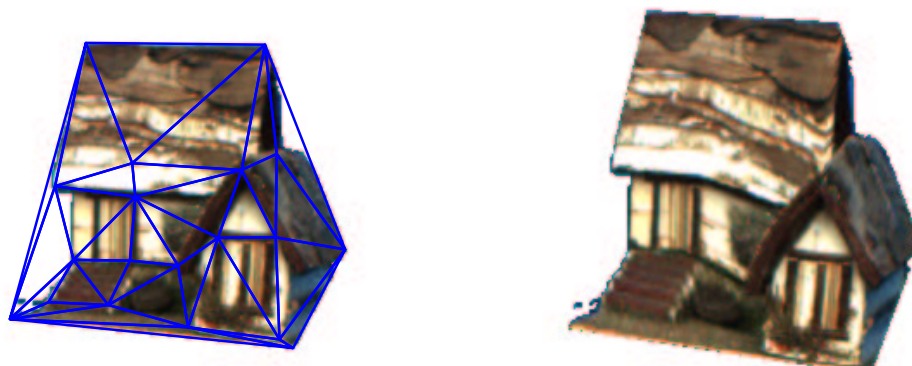


Figure 37: Left: Delauney triangulation of a captured house model. Note that triangles don't correspond well to physical planes. Right: Static texturing of a captured house produces significant errors. Especially note the deformations where the big and small house join due to several triangles spanning points on both houses, and .



Figure 38: Rendered novel views of a house by modulating a texture onto a coarse captured geometric model. Note the absence of geometric distortions compared to the previous figure.

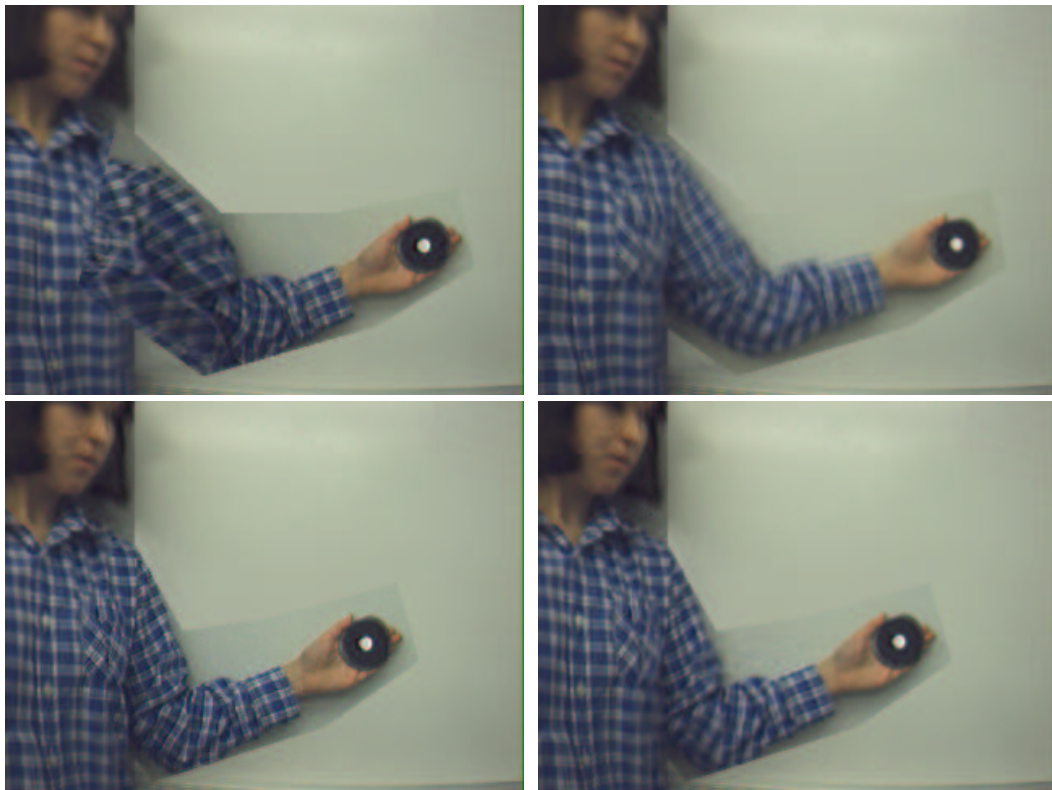


Figure 39: Geometric errors on arm sequence. (top) Renderings of a new position using static and dynamic textures respectively. (bottom) Rerenderings for one frame from the training sequence using static and dynamic textures.



Figure 40: Arm animation

and web page video). The third is the same rendering using standard view dependent textures from 30 sample textures quite close (at most a few degrees from) the rendered pose. The average image intensity error per pixel between rendered and real images was calculated for sequence two and three. It was found that for most views modulating a basis texture we can achieve about half the image error compared to standard view dependent texturing. This error is also very stable over all views, giving real time rendering a smooth natural appearance. The view dependent rendering from sample images did better only when a rendered frame is very close to a sample image, and otherwise gave a jumpy appearance where the error would go up and down depending on the angular distance to a sample view. The error graph for 14 of the 80 views is shown in 41.

For an animation there are global errors through the whole movie that are not visible in one frame but only in the motion impression from the succession of the frames. One important dynamic measurement is motion smoothness. When using static texture we source the texture from a subset of the original images ($k + 1$ if k is the number of texture basis) so there is significant jumping when changing the texture source image. We tracked a point through a generated sequence for the pattern in the two cases and measure the smoothness of motion. Table 1 shows the average pixel jitter.

	Vertical jitter	Horizontal jitter
Static texture	1.15	0.98
Dynamic texture	0.52	0.71

Table 1: Average pixel jitter

4.8 Discussion

Dynamic textures is a texturing method where for each new view a unique view-dependent texture is modulated from a texture basis. The basis is designed so that it encodes a texture intensity spatial derivatives with respect to warp and parallax parameters in a set of basis textures. In a rendered sequence the texture modulation plays a small movie on each model facet, which correctly represents the underlying true scene structure to a first order. This effectively compensates for small (up to a few pixels) geometric errors between the true scene structure and captured model.

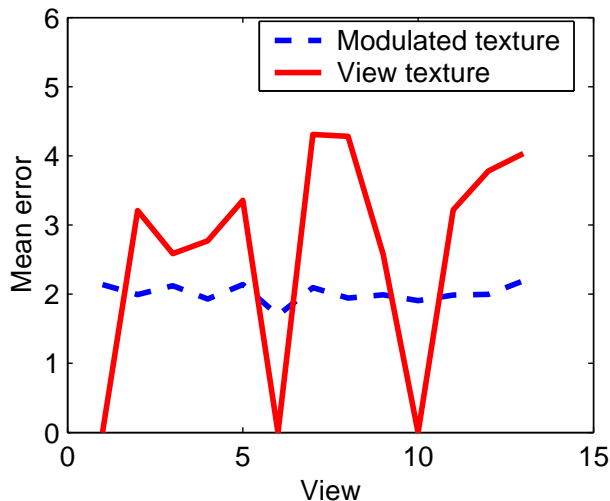


Figure 41: Pixel intensity error when texturing from a close sample view (red) and by modulating the texture basis. For most views the texture basis gives a lower error. Only when the rendered view has the same pose as the one of the three source texture images (hence the IBR is a unity transform) is the standard view based texturing better

The benefit of using *dynamic textures* combined with geometric structure from motion is that it can capture and render scenes with reasonable quality from uncalibrated images alone. Hence, neither a-priori models, expensive laser scanners or extensive human intervention is required. This can potentially enable applications such as virtualized and augmented reality in the consumer market.

5 Real-Time Visual Tracking

Darius Burschka and Greg Hager

5.1 Motivation

Real-time vision is an ideal source of feedback for systems that must interact dynamically with the world. Cameras are passive and unobtrusive, they have a wide field of view, and they provide a means for accurately measuring the geometric properties of physical objects. Potential applications for visual feedback range from traditional problems such as robotic hand eye coordination and mobile robot navigation to more recent areas of interest such as user interfaces, gesture recognition, and surveillance. One of the key problems in real-time vision is to track objects of interest through a series of images. There are two general classes of image processing algorithms used for this task: full-field image processing followed by segmentation and matching and localized feature detection. Many tracking problems can be solved using either approach, but it is clear that the correspondence search in single frames is a challenging and error-prone task. In this sense, precise local feature tracking is essential for the accurate recovery of three-dimensional structure.

The *XVision* is our image processing library providing basic image processing functions and tracking abstractions that can be used to construct application specific, robust tracker designs in a simple way. This software runs on standard commercial hardware allowing a construction of low-cost systems for a variety of robotics and graphics applications. The library provides a variety of tracking primitives that use color, texture, edges or disparity representations of the world as input.

In the following section we will focus on two image domains: texture in monocular systems and disparity domain, where we made significant contributions to the state of the art as presented in the tutorial.

5.2 Visual Tracking in Monocular Images

Visual tracking has emerged as an important component of systems using vision as feedback for continuous control [63, 64, 67, 66], human computer interfaces [68, 69, 70], surveillance, or visual reconstruction [73, 74, 77, 78]. The central challenge in visual tracking is to determine the image position of an object, or target region of an object, as the object moves through a camera's field of view. This is done by solving what is known as the temporal correspondence problem: the problem of matching the target region in successive frames of a sequence of images taken at closely-spaced time intervals.

This problem has much in common with the stereo or motion correspondence problems, but differs in that the goal is not to determine the exact correspondence of each point within the image, but rather to determine, in a gross sense, the movement of the target region. Thus, because all points in the target region are presumed to be part of the same object, we assume – for most applications – that these points move rigidly in space. This assumption allows us to develop low-order parametric models for the image motion of the points within the target region, models that can be used to predict the movement of the points and track the target

region through the image sequence.

The simplest and most common model for the motion of the target region through the image sequence is to assume image translation, which is equivalent to assuming that the object is translating in space and is being viewed orthographically. For inter-frame calculations such as those required for motion analysis, this is typically adequate. However for tracking applications in which the correspondence for a finite size image patch must be computed over a long time span, the translation assumption is eventually violated. As noted by Shi and Tomasi [86], in such cases rotation, scaling, shear and other image distortions often have a significant effect on feature appearance and must be accounted for to achieve reliable matching. A more sophisticated model is to assume affine deformations of the target region, which is equivalent to assuming that the object is planar and again is being viewed orthographically. This model goes along way toward handling more general motions, but again breaks down for object which are not planar or nearly planar.

Recent attempts have been made to model more complex motions or more complex image changes. For example, Black and Yacoob [90] describe an algorithm for computing structured optical flow for recognizing facial expressions using motion models that include affine deformations and simple polynomial distortions. Rehg and Witkin [89] describe a similar algorithm for tracking arbitrarily deforming objects. In both cases, the algorithms track by integrating inter-frame changes, a procedure that is prone to cumulative error. More recent work considers tracking while the target undergoes changes of view by using a subspace of images and an iterative robust regression algorithm [92].

We develop a general framework for tracking objects through the determination of frame to frame correspondence of the target region. The motion is not restricted to pure translation, or even necessarily affine deformations of the target region through the image space. Rather any parametric model for the image motion can be specified. If the shape of the object is known in advance, then one can choose the appropriate model for image motion.

Again in analogy to the stereo and motion correspondence problems, there is the choice of what features of the target region to match through the image sequence. In this paper, we match image intensity for all points within the target region. The advantage of using image intensity is that all the information is used in determining the temporal correspondence. A disadvantage is that the intensity of points within the target region vary with changes in lighting. Thus, many researchers have avoided image intensity in favor of features such as edges or corners [96, 94] or have completely ignored the effects of illumination. Hence, unaccounted for changes in shading and illumination can easily influence the solutions for translation or object geometry, leading to estimation bias and, eventually, mis-tracking. Solutions to illumination problems have been largely confined to simple accommodations for brightness and contrast changes.

We develop a method for tracking image regions that accounts not only for changes in the intensity of the illumination, but also for changes of the pose of the object with respect to the source of illumination. We do this by developing a parametric, linear model for changes in the image of the target region induced by changes in the lighting. As for our parametric models for image motion, our parametric models for illumination are low-order and can be developed either in advance or on the fly. Furthermore, we show how these illumination models can be incorporated into SSD motion estimation with no extra on-line computational cost.

We have both a parametric model for the allowable motion of the image coordinates and a parametric model for the allowable changes in the image due to changes in illumination. To establish the temporal correspondence of the target region across the image sequence, we simultaneously determine the motion parameters and the illumination parameters that minimize the sum of squared differences (SSD) of image intensities. SSD-based methods have been employed in a variety of contexts including stereo matching [84], optical flow computation [85], hand-eye coordination [66], and visual motion analysis [86].

SSD-based methods are sensitive to outliers. If the object becomes partially occluded, then tracking system – in an effort to comply with the SSD constrain – may move away from the partially occluded target and eventually mistrack. To overcome this, we augment our SSD constrain allowing it to discard statistical outliers. This improves performance of the system significantly. Even if the occluded portion of the target region is a significant fraction of the whole target region (e.g., 25% of the object are occluded), the method still allows a robust lock on the target.

5.3 Tracking Moving Objects

In this section, we describe a framework for efficient tracking of a target region through an image sequence. We first write down a a general model for the set of allowable image deformations of the target region across the sequence. This set is treated as a manifold in the target region image coordinate space – we call this manifold the “tracking manifold.” We then pose the tracking problem as the problem of finding the optimal path across a tracking manifold. We define the optimal path as that which minimizes the sum of squared differences (SSD) between the brightness values of the initial and subsequent images of the target and subsequent images of the target warped according to the target coordinates on the tracking manifold. Methods for implementing SSD matching are well-known and can be found in a variety of publications, e.g. [84, 85, 86].

5.3.1 Recovering Structured Motion

First, we consider the problem of describing the motion of a target object from the information contained in a sequence of closely spaced images. Let $I(\mathbf{x}, t)$ denote the brightness value at the location $\mathbf{x} = (x, y)^T$ in an image acquired at time t and let $\nabla_{\mathbf{x}}I(\mathbf{x}, t)$ denote the spatial gradient at that location and time. The symbol t_0 denotes an identified “initial” time and we refer to the image at time t_0 as the *reference image*. Let the set $\mathcal{R} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ be a set of N image locations which define a *target region*. We refer to the brightness values of the target region in the reference image as the *reference template*.

Over time, relative motion between the the target object and the camera causes the image of the target to move and to deform. We stipulate that the deformations over time are well-modeled by a fixed *motion model* which is described as a change of coordinates $\mathbf{f}(\mathbf{x}, \boldsymbol{\mu})$ parameterized by $\boldsymbol{\mu} = (\mu_1, \mu_2, \dots, \mu_n)^T \in \mathcal{U}$, with $\mathbf{f}(\mathbf{x}, \mathbf{0}) = \mathbf{x}$. We assume that \mathbf{f} is differentiable in both $\boldsymbol{\mu}$ and \mathbf{x} . The parameters $\boldsymbol{\mu}$ are the motion information to be recovered from the visual input stream. We write $\boldsymbol{\mu}^*(t)$ to denote the ground truth values of these parameters at time t , and $\boldsymbol{\mu}(t)$ to denote the corresponding estimate. The argument t will be suppressed when it is obvious from context.

Suppose that a reference template is acquired at time t_0 and that initially $\boldsymbol{\mu}^*(t_0) = \boldsymbol{\mu}(t_0) = \mathbf{0}$. Let us initially assume that the only changes in subsequent images of the target are completely described by \mathbf{f} . It follows that for any time $t > t_0$ there is a parameter vector $\boldsymbol{\mu}^*(t)$ such that

$$I(\mathbf{x}, t_0) = I(\mathbf{f}(\mathbf{x}, \boldsymbol{\mu}^*(t)), t) \text{ for all } \mathbf{x} \in \mathcal{R}. \quad (61)$$

This is a generalization of the so-called *image constancy assumption* [97]. It follows that the configuration of the target region at time t can be estimated by minimizing the following least squares objective function:

$$O(\boldsymbol{\mu}) = \sum_{\mathbf{x} \in \mathcal{R}} (I(\mathbf{f}(\mathbf{x}, \boldsymbol{\mu}), t) - I(\mathbf{x}, t_0))^2. \quad (62)$$

For later developments, it is convenient to rewrite this optimization problem in vector notation. At this point, let us consider images as vectors in an N dimensional space. The image of the target region at time t under the change of coordinates \mathbf{f} with parameter $\boldsymbol{\mu}$ is defined as

$$\mathbf{I}(\boldsymbol{\mu}, t) = \begin{bmatrix} I(\mathbf{f}(\mathbf{x}_1, \boldsymbol{\mu}), t) \\ I(\mathbf{f}(\mathbf{x}_2, \boldsymbol{\mu}), t) \\ \dots \\ I(\mathbf{f}(\mathbf{x}_N, \boldsymbol{\mu}), t) \end{bmatrix}. \quad (63)$$

This vector is subsequently referred to as the *rectified image* at time t with parameters $\boldsymbol{\mu}$. Note that with this definition, the image constancy assumption can be restated as $\mathbf{I}(\boldsymbol{\mu}^*(t), t) = \mathbf{I}(\mathbf{0}, t_0)$.

We also make use of the partial derivatives of \mathbf{I} with respect to the components of $\boldsymbol{\mu}$ and the time parameter t . These are defined as:

$$\mathbf{I}_{\mu_i}(\boldsymbol{\mu}, t) = \frac{\partial \mathbf{I}}{\partial \mu_i} = \begin{bmatrix} I_{\mu_i}(\mathbf{f}(\mathbf{x}_1, \boldsymbol{\mu}), t) \\ I_{\mu_i}(\mathbf{f}(\mathbf{x}_2, \boldsymbol{\mu}), t) \\ \vdots \\ I_{\mu_i}(\mathbf{f}(\mathbf{x}_N, \boldsymbol{\mu}), t) \end{bmatrix}, 1 \leq i \leq n, \quad \text{and} \quad \mathbf{I}_t(\boldsymbol{\mu}, t) = \frac{\partial \mathbf{I}}{\partial t} = \begin{bmatrix} I_t(\mathbf{f}(\mathbf{x}_1, \boldsymbol{\mu}), t) \\ I_t(\mathbf{f}(\mathbf{x}_2, \boldsymbol{\mu}), t) \\ \vdots \\ I_t(\mathbf{f}(\mathbf{x}_N, \boldsymbol{\mu}), t) \end{bmatrix}. \quad (64)$$

Using this vector notation, (62) becomes

$$O(\boldsymbol{\mu}) = \|\mathbf{I}(\boldsymbol{\mu}, t) - \mathbf{I}(\mathbf{0}, t_0)\|^2. \quad (65)$$

In general, (65) defines a nonlinear optimization problem, and hence, unless the target region has some special structure, it is unlikely that the objective function is convex. Thus, in the absence of a good starting point, this problem will usually require some type of expensive search or global optimization procedure to solve [98].

In the case of visual tracking, the continuity of motion provides such a starting point. Let us assume that at some arbitrary time $t > t_0$ the target region has the estimated configuration $\boldsymbol{\mu}(t)$. We recast the tracking problem as one of determining a vector of offsets, $\delta\boldsymbol{\mu}$ such that $\boldsymbol{\mu}(t + \tau) = \boldsymbol{\mu}(t) + \delta\boldsymbol{\mu}$ from an image acquired at $t + \tau$. Incorporating this modification into (65), we define a new objective function on $\delta\boldsymbol{\mu}$

$$O'(\delta\boldsymbol{\mu}) = \|\mathbf{I}(\boldsymbol{\mu}(t) + \delta\boldsymbol{\mu}, t + \tau) - \mathbf{I}(\mathbf{0}, t_0)\|^2. \quad (66)$$

If the magnitude of the components of $\delta\boldsymbol{\mu}$ are small, then it is possible to apply continuous optimization procedures to a linearized version of the problem [92, 97, 84, 86]. The linearization is carried out by expanding $\mathbf{I}(\boldsymbol{\mu} + \delta\boldsymbol{\mu}, t + \tau)$ in a Taylor series about $\boldsymbol{\mu}$ and t ,

$$\mathbf{I}(\boldsymbol{\mu} + \delta\boldsymbol{\mu}, t + \tau) = \mathbf{I}(\boldsymbol{\mu}, t) + \mathbf{M}(\boldsymbol{\mu}, t) \delta\boldsymbol{\mu} + \tau\mathbf{I}_t(\boldsymbol{\mu}, t) + h.o.t, \quad (67)$$

where *h.o.t* denotes higher order terms of the expansion, and \mathbf{M} is the *Jacobian matrix* of \mathbf{I} with respect to $\boldsymbol{\mu}$. The Jacobian matrix is the $N \times n$ matrix of partial derivatives which can be written

$$\mathbf{M}(\boldsymbol{\mu}, t) = [\mathbf{I}_{\mu_1}(\boldsymbol{\mu}, t) | \mathbf{I}_{\mu_2}(\boldsymbol{\mu}, t) | \dots | \mathbf{I}_{\mu_n}(\boldsymbol{\mu}, t)]. \quad (68)$$

Here we have explicitly indicated that the values of the partial derivatives are a function of the evaluation point $(\boldsymbol{\mu}, t)$. These arguments will be suppressed when obvious from context.

By substituting (67) into (66) and ignoring the higher order terms, we have

$$O'(\delta\boldsymbol{\mu}) \approx \|\mathbf{I}(\boldsymbol{\mu}, t + \tau) + \mathbf{M} \delta\boldsymbol{\mu} + \tau\mathbf{I}_t - \mathbf{I}(\mathbf{0}, t_0)\|^2. \quad (69)$$

We then observe that $\tau\mathbf{I}_t(\boldsymbol{\mu}, t) \approx \mathbf{I}(\boldsymbol{\mu}, t + \tau) - \mathbf{I}(\boldsymbol{\mu}, t)$. Incorporating this observation and simplifying, we have

$$O'(\delta\boldsymbol{\mu}) \approx \|\mathbf{M} \delta\boldsymbol{\mu} + \mathbf{I}(\boldsymbol{\mu}, t + \tau) - \mathbf{I}(\mathbf{0}, t_0)\|^2. \quad (70)$$

The solution to this optimization problem is

$$\delta\boldsymbol{\mu} = -(\mathbf{M}^T \mathbf{M})^{-1} \mathbf{M}^T [\mathbf{I}(\boldsymbol{\mu}, t + \tau) - \mathbf{I}(\mathbf{0}, t_0)], \quad (71)$$

provided the matrix $\mathbf{M}^T \mathbf{M}$ is invertible. When it is matrix $\mathbf{M}^T \mathbf{M}$ is not invertible, then we are faced with a generalization of the aperture problem, i.e. the target region does not have sufficient structure to determine all of the elements of $\boldsymbol{\mu}$ uniquely.

Defining the *error vector* $\mathbf{e}(t + \tau) = \mathbf{I}(\boldsymbol{\mu}(t), t + \tau) - \mathbf{I}(\mathbf{0}, t_0)$, solution of (65) at time $t + \tau$ given a solution at time t is

$$\boldsymbol{\mu}(t + \tau) = \boldsymbol{\mu}(t) - (\mathbf{M}^T \mathbf{M})^{-1} \mathbf{M}^T \mathbf{e}(t + \tau). \quad (72)$$

It is interesting to note that (72) is in the form of a proportional feedback system. The values in $\boldsymbol{\mu}$ form the state vector of the system and parameterize the change of coordinates \mathbf{f} used to rectify the most recent image based on past estimates. The error term, $\mathbf{e}(t + \tau)$, drives the system to a state where $\mathbf{e}(t + \tau) = 0$. Since the error is formed as a difference with the original reference template, this implies that the system has an inherent stability—the system will adjust the parameter estimates until the rectified image is identical (or as close as possible in a mean-squared sense) to the reference template. It is also interesting to note that if $\boldsymbol{\mu}(t) = \boldsymbol{\mu}^*(t)$ at each estimation step, then $\mathbf{e} \approx \mathbf{I}_t$, (the image temporal derivative) and $\delta\boldsymbol{\mu}$ is a least-squares estimate of the motion field at time $t + \tau$.

Recall that vector inner product is equivalent to computing a cross correlation of two discrete signals. Thus, the vector $\mathbf{M}^t \mathbf{e}$ has n components, each of which is the cross correlation of \mathbf{I}_{μ_i} with \mathbf{e} . The $n \times n$ matrix $(\mathbf{M}^t \mathbf{M})^{-1}$ is a linear change of coordinates on this vector. In particular, if the individual rows of \mathbf{M} are orthogonal, then $(\mathbf{M}^t \mathbf{M})^{-1}$ is a diagonal matrix containing the inverse of the norm of the row vectors. In this case, (71) is computing a vector of normalized cross-correlations.

We can think of each column \mathbf{I}_{μ_i} as a “motion template” which directly represents specific types of target motion in terms of the changes in brightness induced by this motion. These templates, through a cross correlation process with \mathbf{e} are used determine the values of each individual motion component. When the estimation is exact, then the correlation is with \mathbf{I}_t the result is an estimate of inter-frame motion. If the prior estimates are not exact (as is usually the case), then an additional correction is added to account for past estimation error. In particular, if $\boldsymbol{\mu}^* = 0$, then $\mathbf{I}_t = 0$ and it follows that $\|\mathbf{e}\|$ is monotone decreasing and under reasonable conditions $\boldsymbol{\mu} \rightarrow \boldsymbol{\mu}^*$ as $t \rightarrow \infty$.

5.3.2 An Efficient Tracking Algorithm

To this point, we have not specified the contents of the Jacobian matrix \mathbf{M} . This matrix is composed of entries of the form

$$\begin{aligned} \mathbf{M}_{i,j} &= \frac{\partial I(\mathbf{f}(\mathbf{x}_i, \boldsymbol{\mu}), t)}{\partial \mu_j} \\ &= \nabla_{\mathbf{f}} I(\mathbf{f}(\mathbf{x}_i, \boldsymbol{\mu}), t) \cdot \mathbf{f}_{\mu_j}(\mathbf{x}, \boldsymbol{\mu}), \end{aligned} \quad (73)$$

where the last expression follows from an application of the chain rule. Because \mathbf{M} depends on time-varying quantities, in principle it must be recomputed at each time step, a computationally expensive procedure. We now show that it is often possible to factor $\mathbf{M}(t)$ to increase the computational effectiveness of the algorithm.

We first recall that the Jacobian matrix of the transformation \mathbf{f} regarded as a function of $\boldsymbol{\mu}$ is the $2 \times n$ matrix

$$\mathbf{f}_{\boldsymbol{\mu}} = \left[\frac{\partial \mathbf{f}}{\partial \mu_1} \mid \frac{\partial \mathbf{f}}{\partial \mu_2} \mid \cdots \mid \frac{\partial \mathbf{f}}{\partial \mu_n} \right]. \quad (74)$$

Similarly, the Jacobian of \mathbf{f} regarded as a function of $\mathbf{x} = (x, y)^t$ is the 2×2 matrix

$$\mathbf{f}_{\mathbf{x}} = \left[\frac{\partial \mathbf{f}}{\partial x} \mid \frac{\partial \mathbf{f}}{\partial y} \right]. \quad (75)$$

By making use of (74), we observe that the entire i th row of \mathbf{M} can be written

$$\mathbf{M}_i = \nabla_{\mathbf{f}} I(\mathbf{f}(\mathbf{x}_i, \boldsymbol{\mu}), t)^t \mathbf{f}_{\boldsymbol{\mu}}. \quad (76)$$

Let us assume that the constancy assumption (61) holds at time t for the estimate $\boldsymbol{\mu}(t)$. By differentiating both sides of (61) and making use of (75) we obtain

$$\nabla_{\mathbf{x}} I(\mathbf{x}, t_0) = \mathbf{f}_{\mathbf{x}}^t \nabla_{\mathbf{f}} I(\mathbf{f}(\mathbf{x}, \boldsymbol{\mu}), t). \quad (77)$$

Combining this expression with (76), we see that the i th row of \mathbf{M} is given by

$$\mathbf{M}_i = \nabla_{\mathbf{x}} I(\mathbf{x}_i, t_0)^t \mathbf{f}_{\mathbf{x}}^{-1} \mathbf{f}_{\boldsymbol{\mu}}. \quad (78)$$

The complete Jacobian matrix consists of rows obtained by computing this expression for each $1 \leq i \leq N$.

From (78), we observe that, for *any choice* of image deformations, the image spatial gradients need only to be calculated once on the reference template. This is not surprising given that the target at time $t > t_0$ is only a distortion of the target at time t_0 , and so its image gradients are also a distortion of those at t_0 .

The non-constant factor in $\mathbf{M}(t)$ is a consequence of the fact that, in general, \mathbf{f}_x and \mathbf{f}_μ involve components of μ and hence vary with time. If we assume that \mathbf{f} is an *affine* function of image coordinates, then we see that \mathbf{f}_x involves only factors of μ and hence the term $\mathbf{f}_x^{-1}\mathbf{f}_\mu$ involves terms which are linear in the components of \mathbf{x} . It is easy to show that we can write then write $\mathbf{M}(t)$ as

$$\mathbf{M}(t) = \mathbf{M}(t_0) \mathbf{A}(t)^{-1}$$

where $\mathbf{A}(t)$ is a full-rank $n \times n$ matrix.

Let us define

$$\mathbf{\Lambda}(t) = (\mathbf{M}(t)^t \mathbf{M}(t))^{-1} \mathbf{M}(t)^t$$

From the previous definition it then follows that

$$\mathbf{\Lambda}(t) = \mathbf{A}(t) \mathbf{\Lambda}(t_0),$$

and an efficient tracking algorithm for affine image deformations consists of the following steps:

offline:

- Define the target region.
- Acquire and store the reference template.
- Compute and store $\mathbf{\Lambda}(t_0)$ using the reference template.

online:

- Use the most recent motion parameter estimates to acquire and warp the target region in the current image.
- Compute the difference between the warped target region and the reference template.
- Multiply this vector by $\mathbf{A}(t)\mathbf{\Lambda}(t_0)$ and add the result to the current parameter estimate.

We now present three examples illustrating these concepts.

Pure Translation In the case of pure translation, the allowed image motions are parameterized by the vector $\mathbf{u} = (u, v)$ giving

$$\mathbf{f}(\mathbf{x}, \mathbf{u}) = \mathbf{x} + \mathbf{u}.$$

It follows immediately that \mathbf{f}_x and \mathbf{f}_μ are both the 2×2 identity matrix, and therefore

$$\mathbf{M}(t) = \mathbf{M}(t_0) = [\mathbf{I}_x(t_0) \mid \mathbf{I}_y(t_0)].$$

Substituting this definition into (74), we see that the motion templates are then simply the spatial gradients of the target region

$$\mathbf{I}_u(\mathbf{u}, t) = \frac{\partial \mathbf{I}}{\partial u} = \begin{bmatrix} I_x(\mathbf{x}_1 + \mathbf{u}, t) \\ I_x(\mathbf{x}_2 + \mathbf{u}, t) \\ \vdots \\ I_x(\mathbf{x}_N + \mathbf{u}, t) \end{bmatrix} \quad \text{and} \quad \mathbf{I}_v(\mathbf{u}, t) = \frac{\partial \mathbf{I}}{\partial v} = \begin{bmatrix} I_y(\mathbf{x}_1 + \mathbf{u}, t) \\ I_y(\mathbf{x}_2 + \mathbf{u}, t) \\ \vdots \\ I_y(\mathbf{x}_N + \mathbf{u}, t) \end{bmatrix}.$$

Hence, the computed motion values are simply the cross-correlation between the spatial and temporal derivatives of the image as expected.

The resulting linear system is nonsingular if the image gradients in the template region are not collinear.

Translation, Rotation and Scale In [92], it is reported that translation plus rotation and scale are effective for tracking objects. Suppose that the target region undergoes a rotation through an angle θ , a scaling by s , and a translation by \mathbf{u} . The change of coordinates is given by

$$\mathbf{f}(\mathbf{x}, \mathbf{u}, \theta, s) = s\mathbf{R}(\theta)\mathbf{x} + \mathbf{u}$$

where $\mathbf{R}(\theta)$ is

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

We then compute

$$\mathbf{f}_x = s\mathbf{R}(\theta) \tag{79}$$

$$\mathbf{f}_\mu = \left[\begin{array}{c|c|c} 1 & 0 & s \frac{d\mathbf{R}(\theta)}{d\theta} \mathbf{x} \\ 0 & 1 & \mathbf{R}(\theta) \mathbf{x} \end{array} \right] \tag{80}$$

$$\mathbf{f}_x^{-1} \mathbf{f}_\mu = \left[\begin{array}{c|c} \mathbf{R}^t(\theta) \mathbf{x} & \mathbf{R}^t(\theta) \frac{d\mathbf{R}(\theta)}{d\theta} \mathbf{x} \\ \hline & \frac{1}{s} \mathbf{x} \end{array} \right] \tag{81}$$

$$= \left[\begin{array}{c|c} \mathbf{R}^t(\theta) \mathbf{x} & \begin{array}{c} -y \\ x \end{array} \\ \hline & \frac{1}{s} \mathbf{x} \end{array} \right]. \tag{82}$$

By substituting the final expression into (76) and rearranging the result, it can be shown that the i th row of $\mathbf{M}(t_0)$ corresponding to image location $\mathbf{x}_i = (x, y)^t$ with gradient $\nabla_{\mathbf{x}} I = (I_x, I_y)^t$ be expressed as

$$\mathbf{M}_i = (I_x, I_y, -yI_x + xI_y, xI_x + yI_y) \mathbf{A}^{-1}(\theta, s) \tag{83}$$

where

$$\mathbf{A}(\theta, s) = \begin{bmatrix} s\mathbf{R}(\theta) & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix} \quad \text{and} \quad \mathbf{A}^{-1}(\theta, s) = \mathbf{A}(-\theta, 1/s)$$

This result can be intuitively justified as follows. The matrix $\mathbf{M}(0)$ is the linearization of the system about $\theta = 0$ and $s = 1$. At time t the target has orientation $\theta(t)$ and $s(t)$. Image warping

effectively rotates the target by $-\theta$ and scales by $1/s$ so the displacements of the target are computed *in the original target coordinate system*. \mathbf{A} then applies a change of coordinates to rotate and scale the computed displacements from the original target coordinate system back to the actual target coordinates.

Affine Motion The image distortions of planar objects viewed under orthographic projection are described by a six-parameter linear change of coordinates. Suppose that we define

$$\begin{aligned}\boldsymbol{\mu} &= (u, v, a, b, c, d)^t \\ \mathbf{f}(\mathbf{x}; \boldsymbol{\mu}) &= \begin{bmatrix} a & c \\ b & d \end{bmatrix} \mathbf{x} + \begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{A}\mathbf{x} + \mathbf{u}\end{aligned}\quad (84)$$

After some minor algebraic manipulations, we obtain

$$\mathbf{\Gamma}(\mathbf{x}) = \begin{bmatrix} 1 & 0 & x & 0 & y & 0 \\ 0 & 1 & 0 & x & 0 & y \end{bmatrix}\quad (85)$$

and

$$\boldsymbol{\Sigma}(\boldsymbol{\mu}) = \begin{bmatrix} \mathbf{A}^{-1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}^{-1} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{A}^{-1} \end{bmatrix}.\quad (86)$$

Note that $\boldsymbol{\Sigma}$ is once again invertible which allows for additional computational savings as before.

5.4 Illumination-Insensitive Tracking

The systems described above are inherently sensitive to changes in illumination across the target region. Illumination changes over time produce brightness changes that would be interpreted by the machinery of the previous section as motion. This is not surprising, as we are effectively computing a structured optical flow, and optical flow methods are well-known to be sensitive to illumination changes [97]. Thus, shadowing or shading changes across the target object over time lead to bias, or, in the worst case, complete loss of the target.

Recently, it has been shown that often a relatively small number of “basis” images can be used to account for large changes in illumination [100, 80, 82, 81]. Briefly, the reason for this is as follows. Consider a point p on a Lambertian surface and a collimated light source characterized by a vector $\mathbf{s} \in \mathbb{R}^3$, such that the direction of \mathbf{s} gives the direction of the light rays and $\|\mathbf{s}\|$ gives the intensity of the light source. The irradiance at the point p is given by

$$E = a \mathbf{n} \cdot \mathbf{s}\quad (87)$$

where \mathbf{n} is the unit in-wards normal vector to the surface at p and a the non-negative absorption coefficient (albedo) of the surface at the point p [97]. This shows that the irradiance at the point p , and hence the gray level seen by a camera, is linear on $\mathbf{s} \in \mathbb{R}^3$.

Therefore, in the absence of self-shadowing, given three images of a Lambertian surface from the same viewpoint taken under three known, linearly independent light source directions, the

albedo and surface normal can be recovered; this is the well-known method of photometric stereo [101, 100]. Alternatively, one can reconstruct the image of the surface under a novel lighting direction by a linear combination of the three original images [100]. In other words, if the surface is purely Lambertian and there is no shadowing, then all images under varying illumination lie within a 3-D linear subspace of \mathbb{R}^N , the space of all possible images (where N is the number of pixels in the images).

A complication comes when handling shadowing: all images are no longer guaranteed to lie in a linear subspace [82]. Nevertheless, as done in [80], we can still use a linear model as an approximation. Naturally, we need more than three images and a higher than three dimensional linear subspace if we hope to provide good approximation to these effects. However, a small set of basis images can account for much of the shading changes that occur on patches of non-specular surfaces.

Returning to the problem of SSD tracking, suppose now that we have a basis of image vectors $\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_m$ where the i -th element of each of the basis vectors corresponds to the image location $\mathbf{x}_i \in \mathcal{R}$. Let us choose the first basis vector to be the template image, i.e. $\mathbf{B}_1 = \mathbf{I}(\mathbf{0}, t_0)$. To model the brightness changes, let us choose the second basis vector to be a column of ones, i.e. $\mathbf{B}_2 = (1, 1, \dots, 1)^T$.³ Let us choose the remaining basis vectors by performing SVD (singular value decomposition) on a set of training images of the target, taken under varying illumination. We denote the collection of basis vectors by the matrix $\mathbf{B} = [\mathbf{B}_1 | \mathbf{B}_2 | \dots | \mathbf{B}_m]$.

Suppose now that $\boldsymbol{\mu}(t) = \boldsymbol{\mu}^*(t)$ so that the template image and the current target region are registered geometrically at time t . The remaining difference between them is due to illumination. From the above discussion, it follows that inter-frame changes in the current target region can be approximated by the template image plus a linear combination of the basis vectors \mathbf{B} , i.e.

$$\mathbf{I}(\boldsymbol{\mu} + \delta\boldsymbol{\mu}, t + \tau) = \mathbf{I}(\boldsymbol{\mu}, t) + \mathbf{M}\delta\boldsymbol{\mu} + \mathbf{I}_t\tau + \mathbf{B}\boldsymbol{\lambda} + h.o.t \quad (88)$$

where the vector $\boldsymbol{\lambda} = (\lambda_1, \lambda_2, \dots, \lambda_m)^T$. Note that because the template image and an image of ones are included in the basis \mathbf{B} , we implicitly handle both variation due to contrast changes and variation due to brightness changes. The remaining basis vectors are used to handle more subtle variation – variation that depends both on the geometry of the target object and on the nature of the light sources.

Using the vector-space formulation for motion recovery established in the previous section, it is clear that illumination and geometry can be recovered in one global optimization step solved via linear methods. Incorporating illumination into (66) we have

$$O'(\delta\boldsymbol{\mu}, \boldsymbol{\lambda}) = \|\mathbf{I}(\boldsymbol{\mu}(t) + \delta\boldsymbol{\mu}, t + \tau) + \mathbf{B}\boldsymbol{\lambda} - \mathbf{I}(\mathbf{0}, t_0)\|^2. \quad (89)$$

Substituting (88) into (89) and performing the same simplifications and approximations as before, we arrive at

$$O'(\delta\boldsymbol{\mu}, \boldsymbol{\lambda}) = \|\mathbf{M}\delta\boldsymbol{\mu} + \mathbf{B}\boldsymbol{\lambda} + \mathbf{I}(\boldsymbol{\mu}(t), t + \tau) - \mathbf{I}(\mathbf{0}, t_0)\|^2. \quad (90)$$

³In practice, choosing a value close to the mean of the brightness of the image produces a more stable linear system.

Solving $\nabla O'(\boldsymbol{\mu}, \boldsymbol{\lambda}) = \mathbf{0}$ yields

$$\begin{bmatrix} \delta\boldsymbol{\mu} \\ \boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} \mathbf{M}^T \mathbf{M} & \mathbf{M}^T \mathbf{B} \\ \mathbf{B}^T \mathbf{M} & \mathbf{B}^T \mathbf{B} \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{M}^T \\ \mathbf{B}^T \end{bmatrix} \mathbf{e}(t + \tau). \quad (91)$$

In general, we are only interested in the motion parameters. Reworking (91) to eliminate explicit computation of illumination, we get

$$\delta\boldsymbol{\mu} = -(\mathbf{M}^T(\mathbf{1} - \mathbf{B}(\mathbf{B}^T \mathbf{B})^{-1} \mathbf{B}^T) \mathbf{M})^{-1} \mathbf{M}^T(\mathbf{1} - \mathbf{B}(\mathbf{B}^T \mathbf{B})^{-1} \mathbf{B}^T) \mathbf{e}(t + \tau). \quad (92)$$

The above set of equations can be factored as described in the previous section so that \mathbf{M} can be considered time-invariant. Likewise, the illumination basis is time-invariant, and so the matrix pre-multiplying \mathbf{e} can be written as the product of a large time-invariant matrix and a small time-varying matrix is before. Furthermore, the dimensionality of the time-invariant portion of the system depends only on the number of motion fields to be computed, not on the illumination model. Hence, we have shown how to compute image motion while accounting for variations in illumination *using no more on-line computation than would be required to compute pure motion*.

5.5 Making Tracking Resistant to Occlusion

As we track objects over a large space, it is not uncommon that other objects “intrude” into the picture. For example, we may be tracking a target region which is the side of a building when suddenly a parked car begins to occlude a portion of that region, or the object may rotate and the tracked region may “slide off” onto the background. Such occlusions will bias the motion parameter estimates and, in the long term can potentially cause mistracking. In this section, we describe how to avoid such problems. For the sake of simplicity, we develop a solution to this problem for the case where we are only recovering motion and not illumination.

A common approach to this problem is to assume that occlusions create large image differences which can be viewed as “outliers” by the estimation process [92]. The error metric is then modified to be less sensitive to “outliers” by solving a robust optimization problem of the form

$$O_R(\boldsymbol{\mu}) = \sum_{\mathbf{x} \in \mathcal{R}} \rho(I(f(\mathbf{x}, \boldsymbol{\mu}), t) - I(\mathbf{x}, t_0)) \quad (93)$$

where ρ is one of a variety of “robust” regression metrics [102].

It is well-known that optimization of (93) is closely related to another approach of robust estimation—iteratively re-weighted least squares (IRLS) In particular, we have chosen to implement the optimization using a somewhat unusual form of IRLS due to Dutter and Huber [83]. In order to formulate the algorithm, we introduce the notation of an “inner iteration” which is performed one or more times at each time step. We will use a superscript to denote this iteration.

Let $\delta\boldsymbol{\mu}^i$ denote the value of $\delta\boldsymbol{\mu}$ computed by the i th inner iteration with $\delta\boldsymbol{\mu}^0 = \mathbf{0}$, and define the vector of residuals in the i th iteration, \mathbf{r}^i as

$$\mathbf{r}^i = \mathbf{e}(t + \tau) - \mathbf{M}(t) \delta\boldsymbol{\mu}^i. \quad (94)$$

We introduce a diagonal weighting matrix \mathbf{W}^i which has entries

$$\mathbf{W}_{k,k}^i = \eta(\mathbf{r}_k^i) = \rho'(\mathbf{r}_k^i)/\mathbf{r}_k^i. \quad (95)$$

The complete inner iteration cycle at time $t + \tau$ is to perform the estimation step

$$\delta\boldsymbol{\mu}^{i+1} = \delta\boldsymbol{\mu}^i + \boldsymbol{\Lambda}(t)\mathbf{W}(\mathbf{r}^i)\mathbf{r}^i \quad (96)$$

followed by (94) and (95). This process is repeated for k iterations.

This form of IRLS is particularly efficient for our problem. First, it does not require re-computation of $\boldsymbol{\Lambda}$ and, since the weighting matrix is diagonal, does not add significantly to the overall computation time of the algorithm. Second, the error vector \mathbf{e} is fixed over all inner iterations, so these iterations do not require the additional overhead of acquiring and warping images.

As discussed in [83], on linear problems this procedure is guaranteed to converge to a unique global minimum for a large variety of choices of ρ . In this article, ρ is taken to be a so-called “windsorizing” function [83] which is of the form:

$$\rho(r) = \begin{cases} r^2/2 & \text{if } |r| \leq \tau \\ c|r| - c^2/2 & \text{if } |r| > \tau \end{cases} \quad (97)$$

where r is assumed to have unit variance; if not it must be appropriately normalized by dividing by the standard deviation of the data. τ is a user-defined threshold which places a limit on the variations of the residuals before they are considered outliers. This function has the advantage of guaranteeing global convergence of the IRLS method while being cheap to compute. The updating function for matrix entries is

$$\eta(r) = \begin{cases} 1 & \text{if } |r| \leq \tau \\ c/|r| & \text{if } |r| > \tau \end{cases} \quad (98)$$

To this point, we have not specified boundary conditions for \mathbf{W} in the initial estimation step. Given that tracking is a continuous process, it is natural to choose the initial weighting matrix at time $t + \tau$ to be closely related to that computed at the end of the outer iteration at time t . In doing so, two issues arise. First, the fact that the linear system we are solving is a local linearization of a nonlinear system mean that, in cases when inter-frame motion is large, the effect of higher-order terms of the Taylor series expansion will cause areas of the image to “masquerade” as outliers. Second, if we assume that areas of the image with low weights correspond to intruders, it makes sense to add a “buffer zone” around those areas for the next iteration.

Both of these problems can be deal with noting that the diagonal elements of \mathbf{W} are in one-to-one correspondence with image locations of the target region. Thus, \mathbf{W} can also be thought of as an image, where “dark areas” (those locations with low value) are areas of occlusion, while “bright areas” (those with value 1) are the expected target. Thus, classical image-processing techniques can be applied to this “image.” In particular, it is possible to use simple morphology techniques to reduce sensitivity. Define $Q(\mathbf{x})$ to be the nine pixel values in the eight-neighborhood of the image coordinate \mathbf{x} plus the value at \mathbf{x} itself. We define two operators:

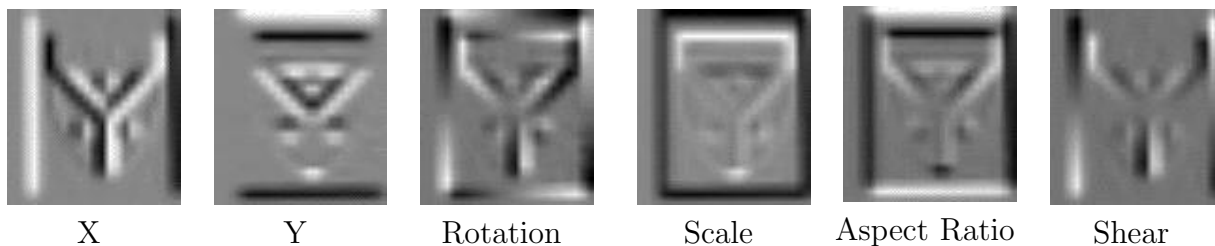


Figure 42: The columns of the motion Jacobian matrix for the planar target and their geometric interpretations.

$$\text{close}(\mathbf{x}) = \max_{v \in Q(\mathbf{x})} v \quad (99)$$

$$\text{open}(\mathbf{x}) = \min_{v \in Q(\mathbf{x})} v \quad (100)$$

If these operators are applied to the image corresponding to the weighting matrix, the former has the effect of removing small areas of outlier pixels, while the latter increases their size. Applying close followed by open has the effect of removing small areas of outliers, and increasing the size of large ones. The former removes isolated areas of outliers (usually the result of linearization error), while the latter increases the area of outlier regions ensuring that outliers are less likely to interfere with tracking in the subsequent step. Thus, we often apply one step of close and two or three steps of open between outer iterations.

5.5.1 Planar Tracking

As a baseline, we first consider tracking a non-specular planar object—the cover of a book. Affine warping augmented with brightness and contrast compensation is the best possible linear approximation to this case (it is exact for an orthographic camera model and purely Lambertian surface). As a point of comparison, recent work by Black and Jepson [92] utilized the rigid motion plus scaling model for SSD-based region tracking. Obviously this reduced model is more efficient and may be more stable since fewer parameters need to be computed, but it does neglect the effects of changing aspect ratio and shear.

We tested both the rigid motion plus scale (RM+S) and full affine (FA) motion models on the same live video sequence of the book cover in motion. Figure 42 shows the set of motion templates (the columns of the motion matrix) for an 81×72 region of a book cover tracked at one third resolution. Figure 43 shows the results of tracking. The upper series of images shows several images of the object with the region tracked indicated with a black frame (the RM+S algorithm) and a white frame (the FA algorithm). The middle row of images shows the output of the warping operator from the FA algorithm. If the computed parameters were physically correct, these images would be identical. However, because of the inability to correct for aspect ratio and skew, the best fit leads to a skewed image. The bottom row shows the output of the warping operator for the RM+S algorithm. Here we see that the full affine warping is much better at accommodating the full range of image distortions. The graph at the bottom of the

figure shows the least squares residual (in squared gray-values per pixel). Here, the importance of the correct geometric model is clearly evident.

5.6 Direct Plane Tracking in Stereo Images

The advances in the hardware development make tracking in disparity domain feasible. Disparity is a direct result of a stereo processing on two camera images. The plane identification and tracking is simplified in the disparity domain because of the additional depth information stored for each pixel of the disparity “image”.

Many methods have been proposed to solve the problem of planar-surface tracking for binocular [122, 121], calibrated and uncalibrated cameras (similar to using sequences of images [111, 110, 113]). A common solution involves a disparity map computation. A disparity map is a matrix of correspondence offsets between two images, often a pair of stereo images [124]. The disparity map calculation employs an expensive neighborhood correlation routines that often yields sparse maps for typical scenes. However, the method makes no assumptions about the environment and has been widely used for the case of general stereo vision. Modern system are able to calculate up to 16 frames/s, which may be sufficient for slow motions in the images, but still not fast enough for fast changing scenes.

In contrast to the general solution with a disparity map, our method exploits a property that planar surfaces exhibit when viewed through a non-verged stereo camera. The disparity, is a linear function whose coefficients are the plane’s parameters. We use a direct method to perform the tracking. Direct methods use quantities that are calculated directly from images values as opposed to feature-based methods discussed earlier [116, 122]. This method should allow a higher processing rate of the images allowing to keep correspondences even in the modern high frame-rate camera systems.

The key component of our work is the plane tracking algorithm that operates directly in the image domain of the acquired images [109].

In the following discussion, let (x, y, z) be a point in world coordinates and (u, v) be a point in pixel coordinates.

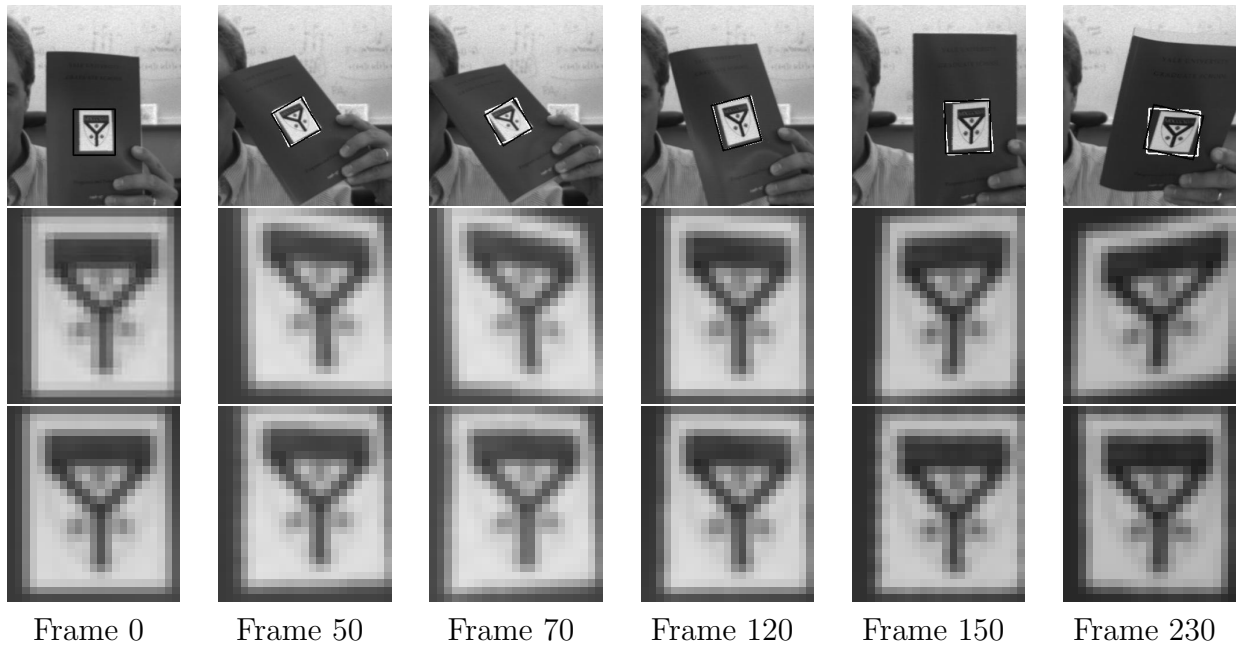
5.6.1 Planar Disparities

To efficiently track a plane, we can make use of a property that planar surfaces exhibit when viewed from non-verged stereo cameras. Namely, a plane becomes a linear function that maps pixels in one image to the corresponding location on the plane in the other image. In indoor environments many surfaces can be approximated with planes \mathcal{E} .

$$\mathcal{E} : ax + by + cz = d \quad (101)$$

In a stereo system with non-verged, unit focal length ($f=1$) cameras the image planes are coplanar. In this case, the disparity value $D(u, v)$ of a point (u, v) in the image can be estimated from its depth z to

$$D(u, v) = \frac{B}{z}, \quad (102)$$



Residuals: Planar Test

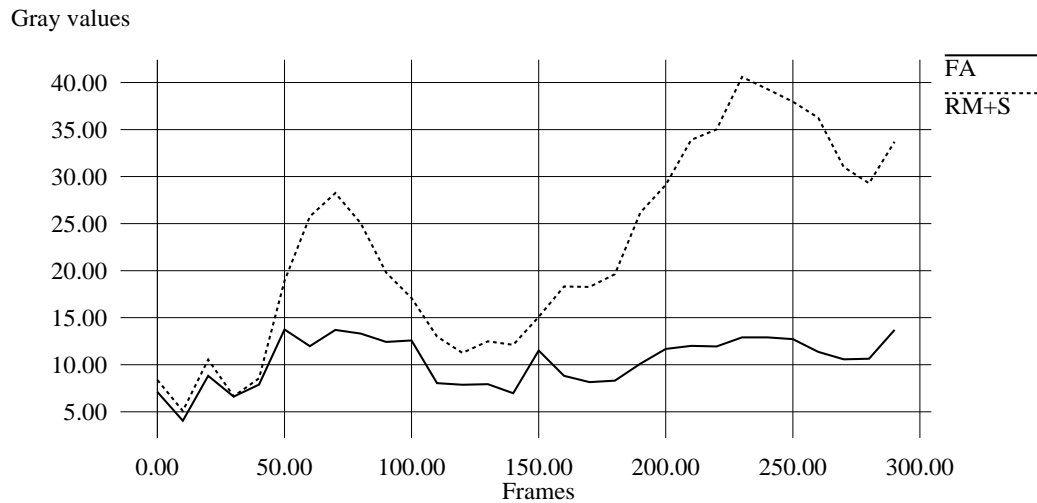


Figure 43: Above, several images of a planar region and the corresponding warped image used by a tracker computing position, orientation and scale, and one computing a full affine deformation. The image at the left is the initial reference image. Below, the graph of the SSD residuals.

with B describing the distance between the cameras of the stereo system [124].

We estimate the disparity $D(u, v)$ of the plane \mathcal{E} at an image point (u, v) using the unit focal length camera ($f=1$) projection to

$$\begin{aligned} \forall z \neq 0 : \quad a \frac{x}{z} + b \frac{y}{z} + c &= \frac{d}{z} \\ au + bv + c &= k \cdot D(u, v) \\ \text{with } u = \frac{x}{z}, v = \frac{y}{z}, k = \frac{d}{B} \end{aligned} \quad (103)$$

The vector $\mathbf{n} = (a \ b \ c)^T$ is normal to the plane \mathcal{E} and describes the orientation of the plane relative to the camera.

The equation (103) can be written in the form

$$\begin{aligned} D(u, v) &= \begin{pmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \end{pmatrix} \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \mathbf{n}^* \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \\ \text{with } \rho_1 &= \frac{a}{k}, \rho_2 = \frac{b}{k}, \rho_3 = \frac{c}{k} \end{aligned} \quad (104)$$

This form uses modified parameters ρ_1, ρ_2, ρ_3 of the plane E relating the image data u, v to $D(u, v)$.

5.6.2 Plane Tracking

From the observation made in Section 5.6.1, we see that tracking the parameters $\mathbf{p} = \rho_1, \rho_2, \rho_3$ of the linear map (104) is equivalent to tracking the planar surface. Thus, assuming inter-frame motion is relatively small and both brightness and contrast shifts can be removed, we pose this problem as one of optimization.

Parameter Update Consider a rectified pair of stereo image, L and R . Based on (104), we relate the images with the following formula. Let $D(u, v) = \rho_1 u + \rho_2 v + \rho_3$.

$$L(u, v) = R(u - D(u, v), v) \quad (105)$$

The plane parameters for the current pair are estimated through the minimization of the following least-squares objective function. To enforce the *brightness constancy constraint* [115], we zero-mean the images: given an image I , $\bar{I} = I - \sum I$.

$$E(\mathbf{p}) = \sum (\bar{L}(u, v) - \bar{R}(u - D(u, v), v))^2 \quad (106)$$

Let $\delta \mathbf{p}$ represent the set of offsets. Assuming a small magnitude for $\delta \mathbf{p}$ we can solve the minimization by linearizing the expression through a Taylor expansion about \mathbf{p} .

$$\begin{aligned} E(\delta \mathbf{p}) &\approx \sum (\bar{L}(u, v) - \bar{R}(u - D(u, v), v) \\ &\quad + u I_x \delta \rho_1 + v I_x \delta \rho_2 + I_x \delta \rho_3)^2 \end{aligned} \quad (107)$$

Here, I_x refers to the spatial gradient of the right image. We neglect the higher order terms of the Taylor series. The system is solved using the Singular-Value Decomposition [123]. It is first convenient to define the error term: $e(u, v) = \overline{L}(u, v) - \overline{R}(u - D(u, v), v)$.

$$\begin{bmatrix} e(u_1, v_1) \\ e(u_1, v_2) \\ \dots \\ e(u_m, v_n) \end{bmatrix} = \begin{bmatrix} u_1 I_{x_{u_1}} & v_1 I_{x_{u_1}} & I_{x_{u_1}} \\ u_1 I_{x_{u_2}} & v_2 I_{x_{u_2}} & I_{x_{u_2}} \\ \dots & \dots & \dots \\ u_m I_{x_{u_m}} & v_n I_{x_{u_m}} & I_{x_{u_m}} \end{bmatrix} \begin{bmatrix} \delta \rho_1 \\ \delta \rho_2 \\ \delta \rho_3 \end{bmatrix} \quad (108)$$

Mask Management Thus far, we have shown how to optimize the parameters of a plane in a static scene. To extend the approach to a tracking setting, we incorporate a mask into the framework. The mask is a binary weighting matrix with an entry per-pixel denoting the pixel's inclusion or exclusion from the current tracked plane. Such a mask removes inaccurate and poor pixel matches from the SVD solution decreasing its processing demand and increasing its stability. Equation (107) is extended to (109) incorporating such a mask; $W(u, v)$ corresponds to the value of the mask at (u, v) .

$$E(\delta \mathbf{p}) \approx \sum \delta_{W(u,v)=1} [(\overline{L}(u, v) - \overline{R}(u - D(u, v), v) + u I_x \delta \rho_1 + v I_x \delta \rho_1 + I_x \delta \rho_2)^2] \quad (109)$$

Each frame, we compute a normalized cross-correlation (\odot) measure to fully recompute the mask matrix based on the current parameters.

$$\eta_{u,v} = \overline{L}(u, v) \odot \overline{R}(u - D(u, v), v) \quad (110)$$

Since (109) is only sensitive to pixels demonstrating horizontal gradients, we also mask pixels with low horizontal variance. To do so, we sample the correlation response along the epipolar line.

$$\begin{aligned} \alpha_{u,v} &= \frac{\overline{L}(u, v) \odot \overline{R}(u - D(u, v) + \delta, v)}{\eta} \\ \beta_{u,v} &= \frac{\overline{L}(u, v) \odot \overline{R}(u - D(u, v) - \delta, v)}{\eta} \end{aligned} \quad (111)$$

$$W(u, v) = (\eta_{u,v} > \tau) \wedge (\alpha_{u,v} > \epsilon) \wedge (\beta_{u,v} > \epsilon) \quad (112)$$

In (112), $0 < \tau < 1$ and $\epsilon > 1$. In (111,112), the selection of δ, τ , and ϵ is dependent on the optics of the system and the scene. To increase the robustness of the mask generated in this manner, we also perform a morphological dilation followed by an erosion [114]. This has the effect of removing noisy correlation response and joining contiguous regions.

6 Implementation of Real-Time Texture Blending and IBMR systems

Keith Yerex and Martin Jagersand

Our renderer manages to do the equivalent of a fairly large matrix multiplication (about $[60000 \times 50] * [50 \times 1]$) each frame easily at interactive framerates. Even with a very fast cpu this is not possible. We accomplish it by taking advantage of texture blending capabilities available on all current graphics accelerators. There are multiple ways to blend textures in hardware, as well as different hardware vendors, and software options for accessing the hardware. We will mostly discuss OpenGL/NVidia implementation, but first, let's briefly look at all the options.

6.0.3 Software choices: OpenGL or DirectX

For software, there is DirectX and OpenGL. OpenGL is available on all platforms, making it our sdk of choice since we use Linux, but DirectX has some advantages. OpenGL uses proprietary extensions to access the latest features of new graphics hardware, which can become confusing since each manufacturer will have different extensions with similar uses (until extensions are standardized by the OpenGL ARB [Architecture Review Board]). DirectX has more uniform ways of accessing different hardware, since proprietary extensions are not allowed. The downside is that with every new generation of hardware released, there is a new version of DirectX to learn.

In hardware, there are two major players in high end consumer graphics accelerators: Ati and NVidia. Ati's Radeon 9700 has been the leader in the market for a while now, with the best performance and features. However, NVidia has recently launched their new GeForce FX line, with slightly better performance, and a far better feature set. Of course Ati will be releasing their new Radeon 9800 soon, so it is pretty hard just to keep up to date with the progression, let alone afford the latest and greatest new card every 6 months. In any case, these days even cheaper hardware like NVidia's GeForce2go laptop chips are easily fast enough to do what we need.

Texture blending can be performed in either a single pass, or multiple passes. Older hardware only supports multi-pass blending, which is the simplest. With multi-pass blending, you first draw some geometry with one texture, then set the blending function (add/subtract/...) and coefficients, and draw the entire geometry again with a second texture. The results are blended together according to the blending function and coefficients. Multi-pass blending can be accessed with the OpenGL 1.3 imaging subset supported by many cards.

Single-pass blending means that multiple textures are loaded simultaneously, along with the parameters for how to combine them, and then the geometry is rendered once. Textures are combined and rendered simultaneously. This method saves the rendering pipeline from processing all the geometric data more than once (camera transforms, clipping etc.), which can be expensive if the geometry is detailed. Primitive single pass blending can be accessed through the OpenGL Multitexturing extension, but to access the all blending features of current graphics cards, we have to use proprietary OpenGL extensions created separately by each vendor. ATI has its fragment shader extension, and NVidia has the register combiners and texture shader extensions

(both accessed through pixel shaders in DirectX). However these issues will be standardized in future ARB extensions. The number of textures that can be processed in a single pass is hardware dependant (NVidia GeForce 3/4 can access 4 textures at once, ATI Radeon 9700 does 8, and the GeForceFX does 16).

In DirectX texture blending is controlled by pixel shaders. These are small programs that execute (on the graphics processor) for each fragment rendered. Pixel shaders are written in an assembly language with instructions for texture accessing, and blending that operate on 4D vector registers. In DirectX 8.0 these are the blending instructions available:

```
add: component addition
cnd: conditional copy [something like this: if(a>0.5)return b; else
return c;]
dp3: 3D dot product
lrp: linear interpolation
mad: scaled addition (multiply and add)
mov: copy
mul: component multiplication
sub: component subtraction
```

Other instructions control loading data from textures into registers. This interface provides a very configurable and yet simple way to control texture blending. There are limitations, however, on the number of instructions of each type allowed per pixel shader. In Pixel Shader version 1.0, 8 arithmetic instructions, and 4 texture access instructions are allowed. To make things more complicated, there are a few instructions that count for more or less than expected, varying with the pixel shader version number

6.0.4 NVIDIA Register Combiners

In our programs, we have used NVidia's register combiners OpenGL extension for texture blending. This interface more closely mirrors how NVidia hardware actually works, making it a little more difficult to work with. With register combiners, there are a number of general combiner stages, followed by a final combinder stage. Each general combiner stage can do one of two operations: two dot products, or two multiplications and one addition. The alpha component, and rgb vector are controled separateley, you can add two scaled numbers in the alpha channel while doing two dot products in on the rgb components. The final combiner stage does two multiplications, and an addition (like this: $final_color3D = a * b + (1 - a)c + d$) on the rgb components, and just loads the alpha component from a register. Register combiners are controlled by setting the number of general combiners active, and then setting the input registers and output registers of each stage. Using the output registers of previous stages as inputs to later stages, you can bassically write a program just like a DirectX pixel shader. The number of general combiner stages available is hardware dependant (2 on GeForce 2Go, 8 on GeForce 3)

For our application, we may need to blend up to 100 textures, which makes it difficult to use single-pass blending. First, we will look at how it can be implemented with standard multi-pass blending.

The basic idea is simply to draw the mean, followed by successively blending in each eigenvector with its coefficient. But it isn't quite that simple. OpenGL blending (GL_BLEND) doesn't support signed numbers, and eigenvectors will be composed of negative and positive pixels. To get around that, we can simply separate the eigenvectors into a positive texture and a negative texture, effectively wasting half the pixels in each texture by filling them with zeroes. Then we can render twice for each eigenvector: once to do the subtraction, and once to do the addition (both scaled by the blending coefficient). Here is the pseudo code for this implementation:

```
for(each  $q$ )
{
  // draw the mean
  BindTexture( $\bar{I}_q$ );
  DrawQuad( $q$ );

  // add basis textures
  for(each  $i$ )
  {
    SetBlendCoefficient( $|y_{qi}(t)|$ );

    BindTexture( $B_{qi}^+$ );
    if( $y_{qi}(t) > 0$ ) SetBlendEquation(ADD);
    else SetBlendEquation(SUBTRACT);
    DrawQuad( $q$ );

    BindTexture( $B_{qi}^-$ );
    if( $y_{qi}(t) > 0$ ) SetBlendEquation(SUBTRACT);
    else SetBlendEquation(ADD);
    DrawQuad( $q$ );
  }
}
```

The multi-pass implementation works reasonably well, but can be significantly improved on (especially in texture memory usage) by taking advantage of the programmability of current graphics hardware while applying multiple textures in each pass. We can blend as many textures as possible in each pass, and still do several passes, but there are more benefits. The register combiners OpenGL extension supports textures in a signed format, so we can use 50% less texture memory already. On top of that, using register combiners, we can convert between color spaces while blending textures. This means that we can store Y,U and V eigenvectors separately, and use more memory and rendering time on Y and less on U and V.

For each pass, we load 4 Y eigenvectors into one RGBA texture in each available texture unit (we have GeForce 3's so that's 4) 4 coefficients are loaded to go with each eigenvector, and then

register combiners are used to do dot products between each element of the eigenvector, and the four coefficients, and then sum the results. Before adding the sum from the current pass to the contents of the framebuffer, it is multiplied by a YUV to RGB conversion matrix. So we can do 16 eigenvectors per pass (with GeForce3). We still have the problem that multipass blending doesn't support signed addition. Because of this, we will still have to render two passes to apply the 16 eigenvectors. In the first pass, we will add, with coefficients as is. results less than zero will be clipped to zero, and positive results will be added. Then we negate the coefficients, and change the blending mode two subtract. Drawing the second pass, results above zero are now the real negatives (since we negated the coefficients) and they are subtracted. In the first pass, the mean must be included, with with a coefficient of 1. The first pass doesn't need to be followed by a subtraction pass, because it should result in a real image, with no negatives. That means if we have 47 eigenvectors for Y, 15 for U, and 15 for V, plus each one has it's mean, then we can render in 11 passes (7 for Y, 2 for u, and 2 for v). If we had a GeForceFX it could be done in only 2 or 3 passes.

6.0.5 Summary

Graphics accelerators are becoming more and more programmable, and increasingly powerful, making it possible to use hardware to accelerate new graphics algorithms that the hardware designers were not even aware of. Now not only classic model based graphics can take advantage of current rendering hardware. Various image-based methods, like ours, can also run on standard consumer graphics cards.

6.1 IBMR System

We have implemented and tested our method using consumer grade PC's for video capture, tracking and rendering. A demo pre-compiled for both Linux and windows and several movies of sample renderings are on the tutorial web site. The tracking and capture part is based on XVision[17]. Image point correspondences $\mathbf{x}_{j,i} = [u, v]$ are obtained for each frame from real-time SSD tracking.

In our method the approximate texture image stabilization achieved using a coarse model reduces the difficulty of applying IBR techniques. The residual image (texture) variability can then be coded as a linear combination of a set of spatial filters. (Figure 44). More precisely, given a training sequence of images I_t and tracked points $[\mathbf{u}_t, \mathbf{v}_t]$, a simplified geometric structure of the scene P and a set of motion parameters $\mathbf{x}_t = (R_t, a_t, b_t)$ that uniquely characterize each frame is estimated from the tracked points using affine structure from motion (section 2). The reprojection of the structure given a set of motion parameters $\mathbf{x} = (R, a, b)$ is obtained by

$$\begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} = RP + \begin{bmatrix} a \\ b \end{bmatrix} \quad (113)$$

The projection of the estimated structure $[\mathbf{u}_t, \mathbf{v}_t]$ into the sample images is divided into Q trian-

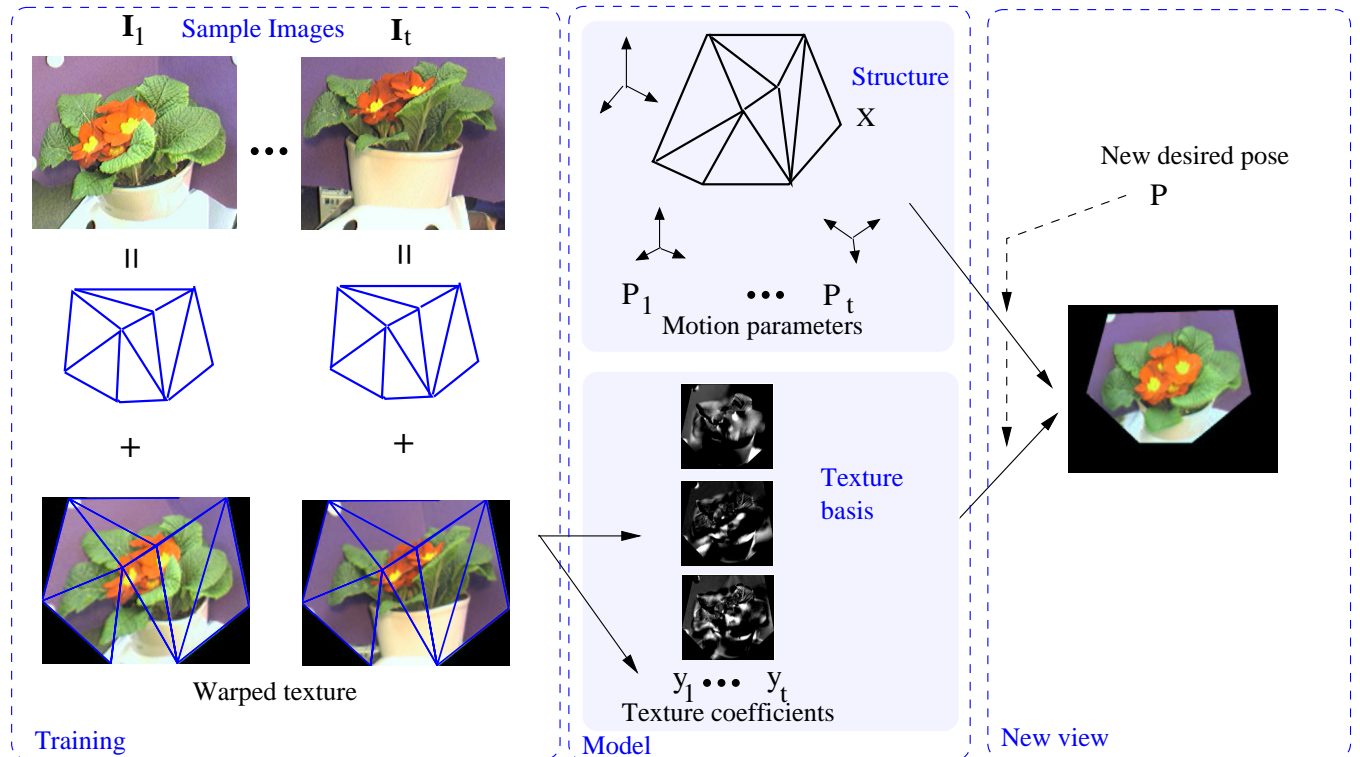


Figure 44: A sequence of training images $I_1 \dots I_t$ is decomposed into geometric shape information and dynamic texture for a set of quadrilateral patches. The scene structure X and views P_j are determined from the projection of the structure using a structure-from-motion factorization algorithm. The view-dependent texture is decomposed into its projection \mathbf{y} on an estimated basis B . For a given desired position, a novel image is generated by warping new texture synthesized from the basis B on the projected structure. On the web site is a compiled demo rendering this flower and some captured movies

gular regions I_{qt} that are then warped to a standard shape I_{wqt} to generate a texture I_{wt} .

$$I_t = \sum_{q=1}^Q I_{qt} \quad (114)$$

$$I_{wqt} = I_{qt}(\mathcal{W}(\mathbf{u}_t, \mathbf{v}_t)) \quad (115)$$

$$I_{wt} = \sum_{q=1}^Q I_{wqt} \quad (116)$$

Using the algorithm described in section 4.1 we then compute a set of basis images B that capture the image variability caused by geometric approximations and illumination changes and the set of corresponding blending coefficients \mathbf{y}_t .

$$I_{wt} = B\mathbf{y}_t + \bar{I} \quad (117)$$

Practically, using HW accelerated OpenGL each frame I_j is loaded into texture memory and warped to a standard shape texture T_j based on tracked positions. (Equations 114,115,116). The standard shape $\mathbf{x}_{T,i}$ is chosen to be the average positions of the tracked points scaled to fit in a square region as shown in Fig. 44. To compute a model we use a hand-held uncalibrated camera to capture a sample sequence of about 100-500 frames of video under varying camera pose. We estimate the geometric model X using structure-from-motion [20] and a texture basis B as in Section 4.1.

New View Animation

1. For each frame in the animation compute the reprojection $[u, v]$ from the desired pose \mathbf{x} as in Equation 113.
2. Estimate texture blending coefficients \mathbf{y} by interpolating the coefficients of the nearest neighbors from the coefficients, and poses from the training data.
3. Compute the new textures in the standard shape using Equation 117, and finally the texture is warped to the projected structure (inverse of Equations 115 and 114). These two operations are performed simultaneously in hardware as described in section 6.0.4

We have designed our method to work well with a range of consumer grade cameras. In the example shown in Fig. 44 a \$100 pyro1394 web cam with significant lens distortion was used to capture a flower. The flower has a complex geometry, which would be difficult to capture in detail. Instead an enveloping geometry with only eight triangular facets was estimated from eight tracked points. A texture basis B is estimated from the normalized texture images, and used to render new views from a mouse controlled viewpoint in the supplied demo program (web-site). Note how a realistic 3D effect is synthesized by modulating the texture basis. If static textures had been used the rendering would look like the 8 facet geometric polygon with glued on pictures.

6.1.1 User interface

The software used in the lab (and available from on the tutorial web site) consists of three main parts:

1. A real-time tracking program “capture_ui” interfaces to a digital camera and uses XVision real time tracking to maintain point correspondences in the video sequence. It also grabs sample frames of intensity images.
2. A structure and texture editor is used to view and verify the geometric model, as well as provide an interface to control the texture selection and generation.
3. A real-time renderer takes a processed models and renders it under varying virtual camera pose controlled by mouse input.

6.1.2 Video capture and tracking

To track and capture video the “capture_ui” interface is used. It can connect to either consumer IEEE 1394 standard web cams in yuv422 mode or to higher quality machine vision cameras, (we use Basler A301fc) in raw Bayer pattern mode. The camera mode is selected on the initial pop-up box. Once the camera is select red, a second real time process “mexv2” will be spawned. This process implements the video pipeline and real time tracking, and will pop up a video window “XVision2”. See Fig. 45. To capture an object select a region to be tracked by clicking “add tracker” in the trackers window, and then click on the desired scene point in the “XVision2” window. Normally, trackers should be added so that the object or scene region of interest is subdivided into roughly planar patches.

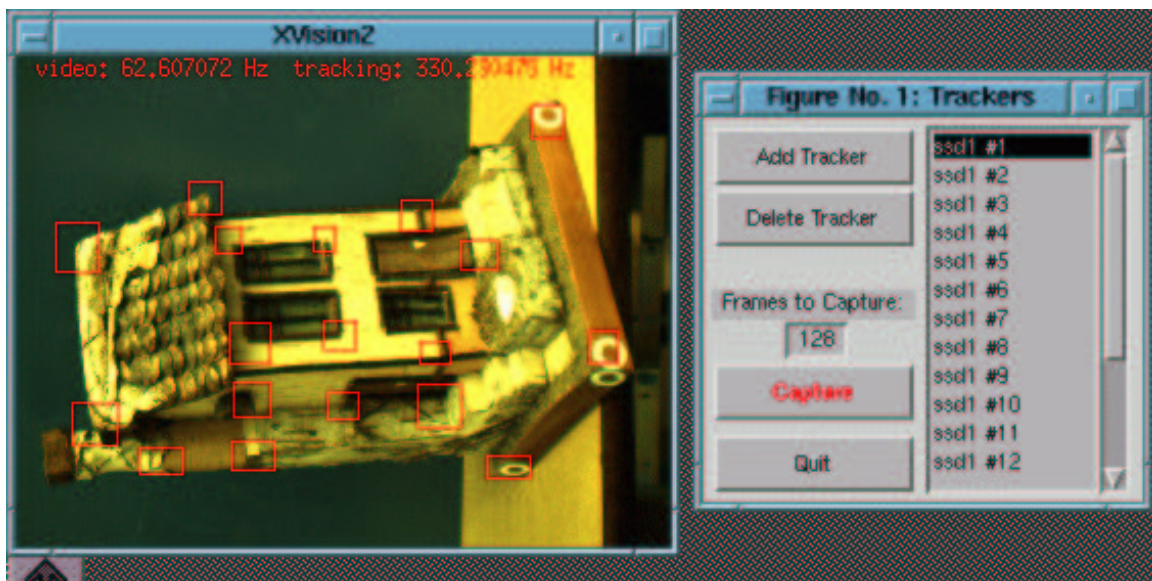


Figure 45: Video capture and tracking user interface

Next select the number of frames to capture. Usually 128-256 is sufficient, but if the scene geometry is more complex than the approximation obtainable from the tracked points more trackers frames can be used. Press the capture button and move the camera (or scene) to

evenly cover the viewing range desired in the model. After capturing the structure editor will be automatically launched.

6.1.3 Structure and texture editor

The “editor” is used to verify the structure and if desired change structure or texture coordinates. After a capture, the editor is launched with the just captured model and sample images. At this point the captured model can be either saved, modified or processed into a format readable by the real time renderer. If started alone, a prompt for a model file name will appear.

The editor has three modes. Toggling between the modes is done using the three buttons “edit”, “triangulation” and “bluescreen”, Fig. 46. In the first “edit” mode, the accuracy of the captured geometry can be evaluated by comparing the tracked points (black circles) to the reprojected points (blue circles), while stepping through the captured images by clicking or dragging the image slider at the bottom. If they differ significantly, (more than a few pixels) the likely cause is that the real time tracking lost the point for some frames. This can be corrected in the affected frames by selecting the point. Click select button, left click on one (or more) point(s). To quit select mode click the middle button. Once selected points can be moved using the move points(frame) button in one or a range of frames. As an alternative to manually moving the points, mistracked points can be corrected using the current structure reprojection by selecting the points, a range of frames and clicking the “correct points” button. This latter only works well if the point was correctly tracked in most of the frames, and hence it’s 3D coordinates can be found accurately. Points that are unnecessary or too difficult to correct can be deleted from the whole sequence with the delete button.

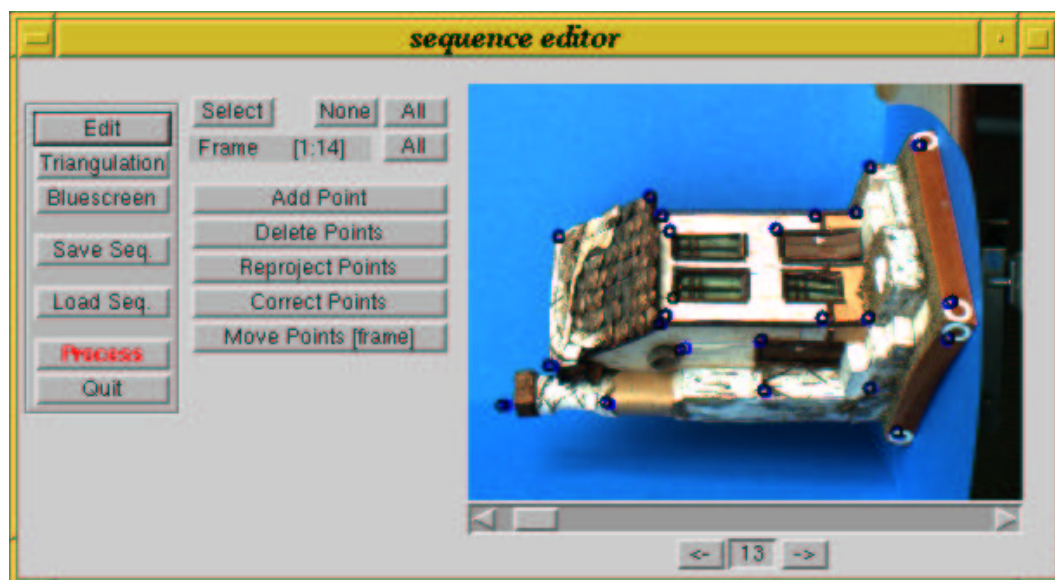


Figure 46: Structure editor mode

In the second mode the triangulation used to extract and represent the dynamic texture can be modified, Fig. 47. A captured structure is initially triangulated using standard Delauney

triangulation. This triangulation often does not put triangles to best correspond to the real scene surfaces. In the triangulation editor unfortunate triangles can be deleted and new triangles can be added by clicking on three points. Additionally, the resolution or relative area of texture given to a particular triangle can be modified by clicking and dragging the triangles in the “texture editor” window. The “opengl” window shows the actual texture representation for the image frame selected.

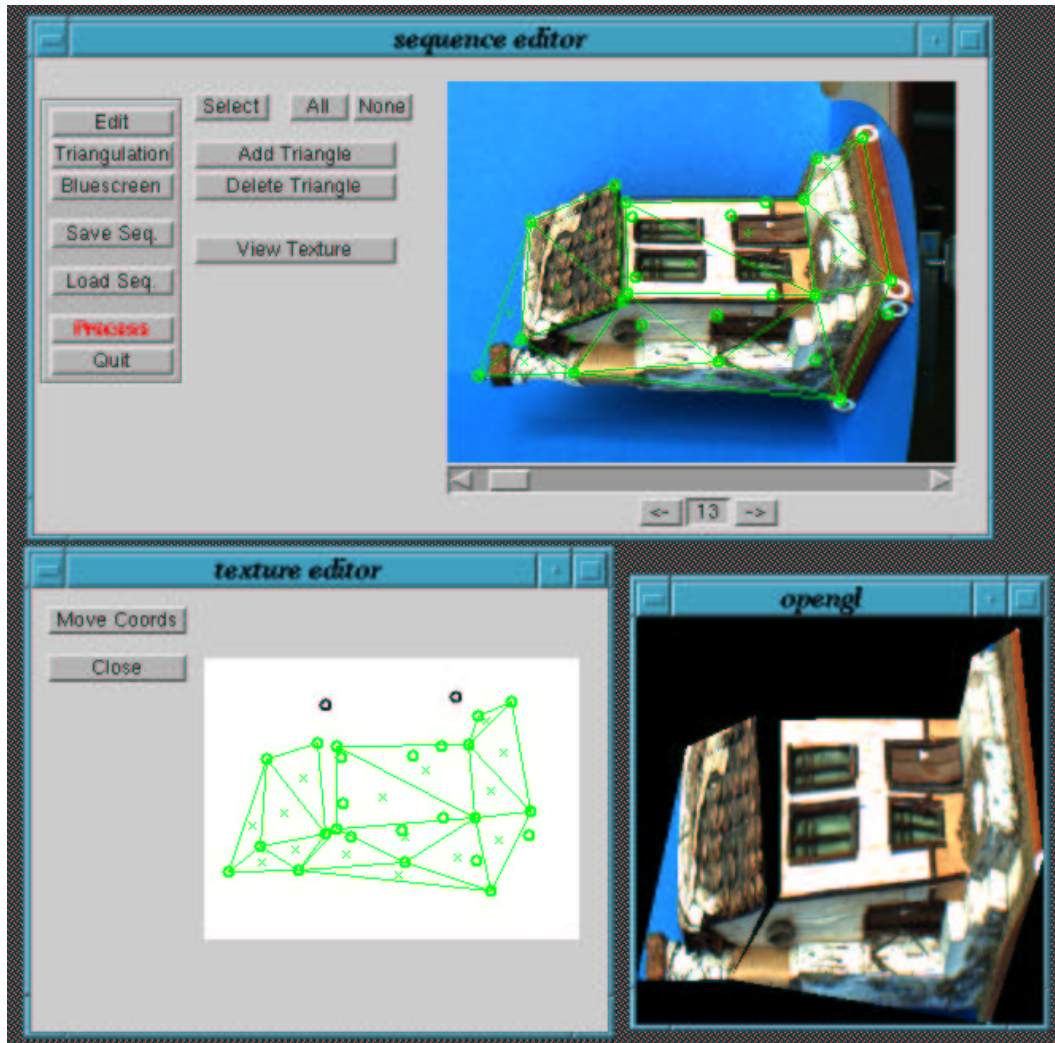


Figure 47: The triangulation and texture editor

In the third mode “blue screen” (not shown), a blue screen color can be selected by clicking in the image. Pixels with this (or close) color will be made transparent.

Once the structure and triangulation are satisfactory, the capture is processed into a format that can be read by the renderer by pressing the “process” button. These files are by default deposited in the `./renderer/` subdirectory.

6.1.4 Real-time renderer

The real time renderer reads several files from the current directory, and starts a glut window, where the scene or object viewpoint can be interactively varied using the mouse. Holding down different mouse button selects which pair of camera pose parameters are mapped to mouse x-y motion.

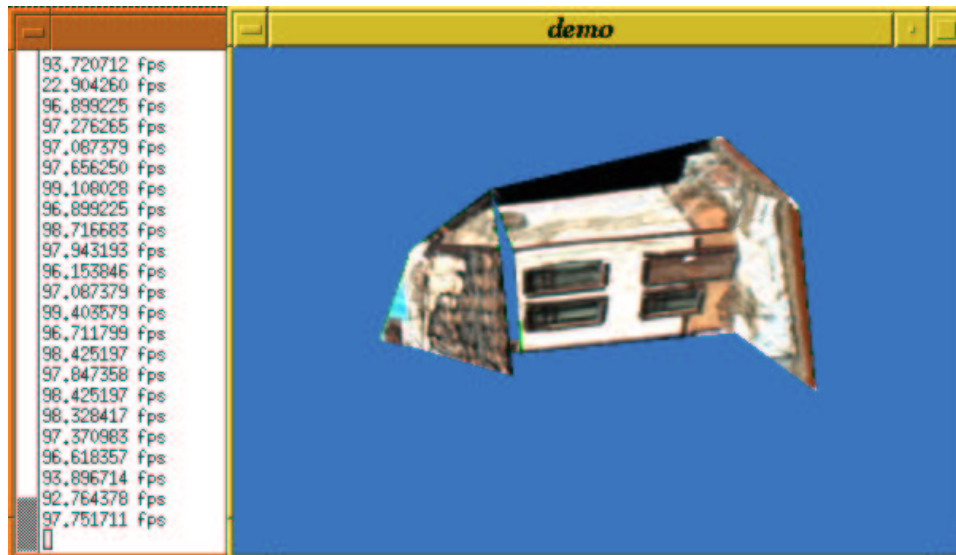


Figure 48: Real time renderer

7 Discussion and Outlook

Image Based Modeling and Rendering is a relatively new field, in the intersection between Computer Graphics and Computer Vision. In the past, the approach to research has been somewhat different in these two fields. In computer vision, researchers have often taken a fundamental principled approach, and so far relatively little of computer vision research has found widespread application. In computer graphics on the other hand the focus of the field has been on developing methods and algorithms of immediate practical use. One reason for this difference is that in the past, cameras and video digitizing hardware was expensive and rarely seen outside research labs. On top of that, to do any real-time video processing special purpose computers such as “Datacube” were needed. For graphics there has been a potential output device since the video display unit replaced the teletype. On the side of personal computers, even the first ones stimulated interest in graphics programming, and while resources were primitive, programmers managed to come up with hacks and tricks to implement worthwhile graphics for e.g. computer games.

Recently, rapid progress in computer performance combined with the availability of inexpensive digital video cameras, from \$100 ieee1394 web cams to decent quality camcorders, has made possible real-time video input and processing on average consumer PC’s. Combined with easy to use capture and rendering software this could potentially take consumer and small business image editing from 2D picture editing to 3D capture, modeling and rendering. Potential applications are plentiful. People could capture and send “3D photos”, joint as tele-present life-like figures in virtual meetings, and include themselves visually accurate as characters in multi-player computer games. One step up form the home consumer market, in architecture and design real objects and scenes can be captured and combined with both real and synthetic models into new designs. Institutions such as museums can build virtual exhibitions, where artifacts now located in different parts of the world can be brought together in their original scenes.

Overall, widespread use of image-based modeling would require easy to use systems. While early approaches to both ray-set and geometry based capture needed calibrated cameras, recent work has shown that uncalibrated video suffices. Hence, the time where capturing and using a 3D model is as easy as using a cam-corder to shoot a video clip may not be very far away.

Martin Jagersand (editor)

References

- [1] T. Beier and S. Neely. Feature-based image methamorphosis. In *Computer Graphics (SIGGRAPH'92)*, pages 35–42, 1992.
- [2] P. Brand, R. Mohr, and Ph. Bobet. Distortion optique : correction dans un modele projectif. *Actes du 9eme Congres AFCET de Reconnaissance des Formes et Intelligence Artificielle*, pages 87–98, 1994.
- [3] C.-F. Chang, G. Bishop, and A. Lastra. Ldi tree: a hierarchical representation for image-based rendering. In *Computer Graphics (SIGGRAPH'99)*, 1999.
- [4] S. Chen. Quicktime VR - an image-based approach to virtual environment navigation. In *Computer Graphics (SIGGRAPH'95)*, pages 29–38, 1995.
- [5] S. Chen and L. Williams. View interpolation for image synthesis. In *Computer Graphics (SIGGRAPH'93)*, pages 279–288, 1993.
- [6] D. Cobzas, K. Yerex, and M. Jagersand. Dynamic textures for image-based rendering of fine-scale 3d structure and animation of non-rigid motion. In *Eurographics*, 2002.
- [7] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from phtographs. In *Computer Graphics (SIGGRAPH'96)*, 1996.
- [8] G. Doretto and S. Soatto. Editable dynamic textures. In *ACM SIGGRAPH Sketches and Applications*, 2002.
- [9] F. Dornaika and R. Chung. Image mosaicing under arbitrary camera motion. In *Asian Conference on Computer Vision*, Taipei, Taiwan, 2000.
- [10] O. Faugeras. Camera self-calibration: theory and experiments. In *ECCV*, pages 321–334, 1992.
- [11] O. D. Faugeras. *Three Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, Boston, 1993.
- [12] O. D. Faugeras. Stratification of 3D vision: Projective, affine, and metric representations. *Journal of the Optical Society of America, A*, 12(7):465–484, 1995.
- [13] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1997.
- [14] Agence France-Presse. Photograph accompanying "It's Up, It's Good: Houston Sends Knicks to Round 2". *New York Times*, CXLVIII(51,525):D1, 5/17/1999.
- [15] S. J. Gortler, R. Grzeszczuk, and R. Szeliski. The lumigraph. In *Computer Graphics (SIGGRAPH'96)*, pages 43–54, 1996.

- [16] G. D. Hager. A modular system for robust positioning using feedback from stereo vision. *IEEE Transactions on Robotics and Automation*, 13(4):582–595, August 1997.
- [17] Gregory D. Hager and Peter N. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(10):1025–1039, 1998.
- [18] R. Hartley. Euclidian reconstruction from uncalibrated views. *Application of Invariance in Computer Vision LNCS 825*, pages 237–256, 1994.
- [19] R. Hartley. Multilinear relationships between coordinates of corresponding image points and lines. In *Sophus Lie Symposium, Norway*, 1995.
- [20] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2000.
- [21] P. Heckbert. *Fundamentals of Texture Mapping*. Msc thesis. Technical Report No. UCB/CSD 89/516, University of California, Berkeley, 1989.
- [22] A. Heyden. Projective structure and motion from image sequences using subspace methods. In *SCIA*, pages 963–968, 1997.
- [23] A. Heyden. Algebraic varieties in multiview geometry. In *ICCV*, pages 3–19, 1998.
- [24] A. Heyden and K. Åström. Euclidian reconstruction from image sequences with varying and unknown focal length and principal point. In *CVRP*, 1997.
- [25] M. Irani, P. Anandan, and S. Hsu. Mosaic based representation of video sequences and their applications. In *Proc. of the Fifth International Conference on Computer Vision*, pages 605–611, 1995.
- [26] M. Jagersand. Image based view synthesis of articulated agents. In *Computer Vision and Pattern Recognition*, 1997.
- [27] M. Jagersand. Image based predictive display for tele-manipulation. In *Int. Conf. on Robotics and Automation*, 1999.
- [28] S.B. Kang and R. Szeliski. 3D scene data recovery using omnidirectional multibaseline stereo. In *Proc. of the IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR'96)*, pages 364–370, 1996.
- [29] S. Laveau and O.D. Faugeras. 3-D representation as a collection of images. In *Proc. of the IEEE Int. Conf. on Pattern Recognition (CVPR'97)*, pages 689–691, Jerusalem, Israel, 1994.
- [30] Kuang-Chih Lee, Jeffrey Ho, and David Kriegman. Nine points of light: Acquiring subspaces for face recognition under variable lighting. In *Computer Vision and Pattern Recognition*, 2001.

- [31] M. Levoy and P. Hanrahan. Light field rendering. In *Computer Graphics (SIGGRAPH'96)*, pages 31–42, 1996.
- [32] R. A. Manning and C. R. Dyer. Interpolating view and scene motion by dynamic view morphing. In *Proc. of the IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR'99)*, pages 388–394, 1999.
- [33] D. K. McAllister, L. Nyland, V. Popescu, A. Lastra, and C. McCue. Real-time rendering of real world environments. In *Proc. of Eurographics Workshop on Rendering, Spain, June 1999*.
- [34] L. McMillan. *An Image-Based Approach to Three-Dimensional Computer Graphics*. Ph.D. Dissertation. UNC CS TR97-013, University of North Carolina, 1997.
- [35] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Computer Graphics (SIGGRAPH'95)*, pages 39–46, 1995.
- [36] Tomas Möller and Eric Haines. *Real-time Rendering*. A.K. Peterson, 2002.
- [37] H. Murase and S. Nayar. Visual learning and recognition of 3d objects from appearance. *International Journal of Computer Vision*, 14:5–24, 1995.
- [38] S. Nayar. Catadioptric omnidirectional camera. In *Proc. of the IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR'97)*, pages 482–488, 1997.
- [39] Manuel M. Oliviera, Gary Bishop, and David McAllister. Relief texture mapping. In *Computer Graphics (SIGGRAPH'00)*, 2000.
- [40] S. Peleg and M. Ben-Ezra. Stereo panorama with a single camera. In *Proc. of the IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR'99)*, 1999.
- [41] S. Peleg and J. Herman. Panoramic mosaics by manifold projection. In *Proc. of the IEEE Int. Conf. on Computer Vision and Pattern Recognition (CVPR'97)*, pages 338–343, 1997.
- [42] C. Poelman and T. Kanade. A paraperspective factorization method for shape and motion recovery. In *ECCV*, pages 97–108, 1994.
- [43] M. Pollefeys and L. Van Gool. Self-calibration from the absolute conic on the plane at infinity. *LNCS 1296*, pages 175–182, 1997.
- [44] Point Grey Research. <http://www.ptgrey.com>.
- [45] S. M. Seitz and C. R. Dyer. View morphing. In *Computer Graphics (SIGGRAPH'96)*, pages 21–30, 1996.
- [46] J. Shade, S. Gortler, L. He, and R. Szeliski. Layered depth images. In *Computer Graphics (SIGGRAPH'98)*, 1998.

- [47] H.-Y. Shum and L.-W. He. Rendering with concentric mosaics. In *Computer Graphics (SIGGRAPH'99)*, pages 299–306, 1999.
- [48] H.-Y. Shum and R. Szeliski. *Panoramic image mosaics*. Technical Report MSR-TR-97-23, Microsoft Research, 1997.
- [49] I. Stamos and P. K. Allen. Integration of range and image sensing for photorealistic 3d modeling. In *Proc. of IEEE Int. Conf. on Robotics and Automation*, pages 1435–1440, 2000.
- [50] Peter Sturm and Bill Triggs. A factorization based algorithm for multi-image projective structure and motion. In *ECCV (2)*, pages 709–720, 1996.
- [51] R. Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, pages 22–30, March 1996.
- [52] R. Szeliski and H.-Y. Shum. Creating full view panoramic image mosaics and environment maps. In *Computer Graphics (SIGGRAPH'97)*, pages 251–258, 1997.
- [53] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: A factorization method. *International Journal of Computer Vision*, 9:137–154, 1992.
- [54] P. Torr. *Motion segmentation and outliers detection*. PhD thesis, University of Oxford, 1995.
- [55] P. Torr and A. Zisserman. Robust parametrization and computation of the trifocal tensor. *Image and Visual Computing*, 15:591–605, 1997.
- [56] W. Triggs. The geometry of projective reconstruction i: Matching constraints and the joint image. In *ICCV*, pages 338–343, 1995.
- [57] W. Triggs. Auto-calibration and the absolute quadric. In *CVRP*, pages 609–614, 1997.
- [58] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Transactions and Robotics and Automation*, 3(4):323–344, 1987.
- [59] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3:71–86, 1991.
- [60] D. Weinshall and C. Tomasi. Linear and incremental acquisition of invariant shape models from image sequences. In *Proc. of 4th Int. Conf. on Computer Vision*, pages 675–682, 1993.
- [61] D. N. Wood, A. Finkelstein, J. F. Hughes, C. E. Thayer, and D. H. Salesin. Multiperspective panoramas for cell animation. In *Computer Graphics (SIGGRAPH'97)*, 1997.
- [62] J. Y. Zheng and S. Tsuji. Panoramic representation for route recognition by a mobile robot. *International Journal of Computer Vision*, 9(1):55–76, 1992.

- [63] P. Allen, B. Yoshimi, and A. Timcenko, "Hand-eye coordination for robotics tracking and grasping," in *Visual Servoing* (K. Hashimoto, ed.), pp. 33–70, World Scientific, 1994.
- [64] S. Hutchinson, G. D. Hager, and P. Corke, "A tutorial introduction to visual servo control," *IEEE Trans. Robot. Automat.*, vol. 12, no. 5, 1996.
- [65] G. D. Hager and P. Belhumeur, "Efficient Region Tracking With Parametric Models of Geometry and Illumination", *EEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20(10), pp.1125-1139, 1998.
- [66] N. Papanikolopoulos, P. Khosla, and T. Kanade, "Visual tracking of a moving target by a camera mounted on a robot: A combination of control and vision," *IEEE Trans. Robot. Automat.*, vol. 9, no. 1, 1993.
- [67] E. Dickmanns and V. Graefe, "Dynamic monocular machine vision," *Machine Vision and Applications*, vol. 1, pp. 223–240, 1988.
- [68] A. F. Bobick and A. D. Wilson, "A state-based technique for the summarization of recognition of gesture," in *Proceedings of the ICCV*, pp. 382–388, 1995.
- [69] T. Darrell, B. Moghaddam, and A. Pentland, "Active face tracking and pose estimation in an interactive room," in *Proc. IEEE Conf. Comp. Vision and Patt. Recog.*, pp. 67–72, 1996.
- [70] D. Gavrilu and L. Davis, "Tracking humans in action: A 3D model-based approach," in *Proc. Image Understanding Workshop*, pp. 737–746, 1996.
- [71] R. Howarth and H. Buxton, "Visual surveillance monitoring and watching," in *Proc. European Conf. on Computer Vision*, pp. II:321–334, 1996.
- [72] T. Frank, M. Haag, H. Kollnig, and H.-H. Nagel, "Tracking of occluded vehicles in traffic scenes," in *Proc. European Conf. on Computer Vision*, pp. II:485–494, 1996.
- [73] R. C. Harrell, D. C. Slaughter, and P. D. Adsit, "A fruit-tracking system for robotic harvesting," *Machine Vision and Applications*, vol. 2, pp. 69–80, 1989.
- [74] D. Reynard, A. Wildenberg, A. Blake, and J. Marchant, "Learning dynamics of complex motions from image sequences," in *Proc. European Conf. on Computer Vision*, pp. I:357–368, 1996.
- [75] E. Bardinnet, L. Cohen, and N. Ayache, "Tracking medical 3D data with a deformable parametric model," in *Proc. European Conf. on Computer Vision*, pp. I:317–328, 1996.
- [76] P. Shi, G. Robinson, T. Constable, A. Sinusas, and J. Duncan, "A model-based integrated approach to track myocardial deformation using displacement and velocity constraints," in *Proc. Internal Conf. on Computer Vision*, pp. 687–692, 1995.
- [77] E. Boyer, "Object models from contour sequences," in *Proc. European Conf. on Computer Vision*, pp. II:109–118, 1996.

- [78] L. Shapiro, *Affine Analysis of Image Sequences*. Cambridge Univ. Press, 1995.
- [79] C. Tomasi and T. Kanade, "Shape and motion from image streams under orthography: A factorization method," *Int. J. Computer Vision*, vol. 9, no. 2, pp. 137–154, 1992.
- [80] P. Hallinan, "A low-dimensional representation of human faces for arbitrary lighting conditions," in *Proc. IEEE Conf. on Comp. Vision and Patt. Recog.*, pp. 995–999, 1994.
- [81] R. Epstein, P. Hallinan, and A. Yuille, " 5 ± 2 Eigenimages suffice: An empirical investigation of low-dimensional lighting models," Technical Report 94-11, Harvard University, 1994.
- [82] P. N. Belhumeur and D. J. Kriegman, "What is the set of images of an object under all possible lighting conditions," in *Proc. IEEE Conf. on Comp. Vision and Patt. Recog.*, pp. 270–277, 1996.
- [83] R. Dutter and P. Huber, "Numerical methods for the nonlinear robust regression problem," *J. Statist. Comput. Simulation*, vol. 13, no. 2, pp. 79–113, 1981.
- [84] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. Int. Joint Conf. Artificial Intelligence*, pp. 674–679, 1981.
- [85] P. Anandan, "A computational framework and an algorithm for the measurement of structure from motion," *Int. J. Computer Vision*, vol. 2, pp. 283–310, 1989.
- [86] J. Shi and C. Tomasi, "Good features to track," in *Proc. IEEE Conf. Comp. Vision and Patt. Recog.*, pp. 593–600, IEEE Computer Society Press, 1994.
- [87] J. Rehg and T. Kanade, "Visual tracking of high DOF articulated structures: An application to human hand tracking," in *Computer Vision – ECCV '94*, vol. B, pp. 35–46, 1994.
- [88] C. Bregler, "Learning and recognizing human dynamics in video sequences," in *Proc. IEEE Conf. Comp. Vision and Patt. Recog.*, pp. 568–574, 1997.
- [89] J. Rehg and A. Witkin, "Visual tracking with deformation models," in *Proc. IEEE Int. Conf. Robot. and Automat.*, pp. 844–850, 1991.
- [90] M. Black and Y. Yacoob, "Tracking and recognizing rigid and non-rigid facial motions using local parametric models of image motion," in *Proceedings of the ICCV*, pp. 374–381, 1995.
- [91] H. Murase and S. Nayar, "Visual learning and recognition of 3-D objects from appearance," *Int. J. Computer Vision*, vol. 14, no. 5–24, 1995.
- [92] M. Black and A. Jepson, "Eigentracking: Robust matching and tracking of articulated objects using a view-based representation," in *Proc. European Conf. on Computer Vision*, pp. 329–342, 1996.
- [93] M. Isard and A. Blake, "Contour tracking by stochastic propagation of conditional density," in *European Conf. on Computer Vision*, pp. I:343–356, 1996.

- [94] D. G. Lowe, "Robust model-based motion tracking through the integration of search and estimation," *Int. Journal of Computer Vision*, vol. 8, no. 2, pp. 113–122, 1992.
- [95] D. B. Gennery, "Visual tracking of known three-dimensional objects," *Int. J. Computer Vision*, vol. 7, no. 3, pp. 243–270, 1992.
- [96] A. Blake, R. Curwen, and A. Zisserman, "A framework for spatio-temporal control in the tracking of visual contour," *Int. J. Computer Vision*, vol. 11, no. 2, pp. 127–145, 1993.
- [97] B. Horn, *Computer Vision*. Cambridge, Mass.: MIT Press, 1986.
- [98] M. Betke and N. Makris, "Fast object recognition in noisy images using simulated annealing," in *Proceedings of the ICCV*, pp. 523–530, 1995.
- [99] R. Szeliski, "Image mosaicing for tele-reality applications," in *Proceedings of the Workshop on Applications of Computer Vision*, pp. 44–53, 1994.
- [100] A. Sashua, *Geometry and Photometry in 3D Visual Recognition*. PhD thesis, MIT, 1992.
- [101] R. Woodham, "Analysing images of curved surfaces," *Artificial Intelligence*, vol. 17, pp. 117–140, 1981.
- [102] P. Huber, *Robust Statistics*. New York, NY: John Wiley & Sons, 1981.
- [103] R. M. Haralick and L. G. Shapiro, *Computer and Robot Vision*. Addison Wesley, 1993.
- [104] G. D. Hager and K. Toyama, "XVision: A portable substrate for real-time vision applications," *Computer Vision and Image Understanding*, vol. 69, no. 1, 1998.
- [105] S. McKenna, S. Gong, and J. Collins, "Face tracking and pose representation," in *British Maching Vision Conference*, 1996.
- [106] G. D. Hager and K. Toyama, "The "X-vision" system: A general purpose substrate for real-time vision applications," *Comp. Vision, Image Understanding.*, vol. 69, pp. 23–27, January 1998.
- [107] P. Belhumeur and G. D. Hager, "Tracking in 3D: Image variability decomposition for recovering object pose and illumination," in *Proc. Int. Conf. Pattern Anal. Applic.*, 1998. Also available as Yale Comp. Sci #1141.
- [108] S. Ullman and R. Basri, "Recognition by a linear combination of models," *IEEE Trans. Pattern Anal. Mach. Intelligence*, vol. 13, pp. 992–1006, 1991.
- [109] Jason Corso and Gregory D. Hager. Planar surface tracking using direct stereo. Technical report, The Johns Hopkins University, 2002. CIRL Lab Technical Report.
- [110] F. Dellaert, S. Seitz, C. Thorpe, and S. Thrun. Structure from motion without correspondence. Technical Report CMU-RI-TR-99-44, Carnegie Mellon University, 1999.

- [111] F. Dellaert, C. Thorpe, and S. Thrun. Super-resolved texture tracking of planar surface patches. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robotic Systems*, 1998.
- [112] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. In *Proc. IEEE Int. Conf. on Robotics and Automation*, 1999.
- [113] V. Ferrari, T. Tuytelaars, and L. Van Gool. Real-time affine region tracking and coplanar grouping. In *IEEE Computer Soc. Conf. on Computer Vision and Pattern Recognition*, 2001.
- [114] G. Hager and P. Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *IEEE Transactions of Pattern Analysis and Machine Intelligence*, 20(10):1125–1139, 1998.
- [115] B. Horn. *Robot Vision*. The MIT Press, 1986.
- [116] M. Irani and P. Anandan. All about direct methods. Technical report, <http://link.springer.de/link/services/series/0558/bibs/1883/18830267.htm>, 2002.
- [117] K. Kanatani. Detection of surface orientation and motion from texture by stereological technique. *Artificial Intelligence*, 24:213–237, 1984.
- [118] K. Kanatani. Tracing planar surface motion from a projection without knowing the correspondence. *Computer Vision, Graphics, And Image Processing*, 29:1–12, 1985.
- [119] D. Keren, S. Peleg, and R. Brada. Image sequence enhancement using sub-pixel displacements. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1988.
- [120] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings DARPA Image Understanding Workshop*, 1981.
- [121] S. Pei and L. Liou. Tracking a planar patch in three-dimensional space by affine transformation in monocular and binocular vision. *Pattern Recognition*, 26(1):23–31, 1993.
- [122] G. Stein and A. Shashua. Direct estimation of motion and extended scene structure from a moving stereo rig. *IEEE Conference on Computer Vision and Pattern Recognition*, 1998.
- [123] G. Strang. *Linear Algebra and Its Applications*. Saunders HBJ, 1988.
- [124] Emanuele Trucco and Alessandro Verri. *Introductory Techniques for 3-D Computer Vision*. Prentice Hall, 1998.

A Biographies

Darius Burschka graduated from Technische Universität München, Germany, was a post doc at Yale University, and is now a research scientist in the Computational Interaction and Robotics Lab (CIRL) at the Johns Hopkins University. His research centers around sensing systems and methods, particularly real-time vision, tracking and three-dimensional reconstruction, for the use in human-computer interfaces and mobile robotics.

Zachary Dodds received the PhD from Yale University in 2000. He is now an assistant professor of computer science at Harvey Mudd College in Claremont, CA. His research centers around the geometry of imaging and hand-eye coordination. He has investigated the specification of alignments based stereo camera information. Theoretically he has shown what alignments are verifiable under different levels of camera calibration. Practically, he has implemented a number of complete specification languages for visual servoing tasks.

Dana Cobzas is a PhD student at the University of Alberta. Her interest are in the area of image based modeling, with applications in graphics rendering and image-based mobile robot navigation. She has contributed several new methods for registration of camera and 3D laser range sensory data, and applied these to an image-based model acquisition system for indoor mobile robot localization and navigation. Lately she has also been researching and implementing Structure-From-Motion (SFM) methods used to build 3D models from uncalibrated video alone.

Gregory D. Hager received the PhD in computer science from the University of Pennsylvania in 1988. He then held appointments at University of Karlsruhe, the Fraunhofer Institute and Yale University. He is now a full professor in Computer Science at Johns Hopkins University and a member of the Center for Computer Integrated Surgical Systems and Technology. His current research interests include visual tracking, vision-based control, medical robotics, and human-computer interaction.

Martin Jagersand graduated from the University of Rochester in 1997, was then a post doc at Yale University and research scientist at Johns Hopkins University. He is now an assistant professor at the University of Alberta. His research centers around dynamic vision – how the visual image changes under different camera or object/scene motions. He has applied this to robotics, developing a novel method for vision-based motion control of an uncalibrated robot and vision system. On the vision and graphics side he has applied the same kind of modeling to synthesize novel animations of camera motion or articulated agent actions.

Keith Yerer graduated from the University of Alberta in 2002 and is now a researcher at Virtual Universe Corporation. His interests are in the intersection of computer graphics and vision. Particularly, he has shown how to combine geometry and texture obtained from uncalibrated video and efficiently use this in real-time image-based rendering. He is also interested in image-based approaches to lighting and animation of non-rigid motion.