Rendering from Unstructured Collections of Images

by

Christopher James Buehler

B.S., Electrical Engineering
B.S., Computer Science
University of Maryland at College Park, 1996

S.M., Computer Science and Electrical Engineering
Massachusetts Institute of Technology, 1998

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in Computer Science and Electrical Engineering
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
May 2002
(June 7 GO2)

Author . . . . . .
Department of Electrical Engineering and Computer Science
May 3, 2002

Certified by . . . . . .
Leonard McMillan
Associate Professor
Thesis Supervisor

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

Rendering from Unstructured Collections of Images

by

Christopher James Buehler

Submitted to the Department of Electrical Engineering and Computer Science
on May 3, 2002, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Electrical Engineering

Abstract

Computer graphics researchers recently have turned to image-based rendering to achieve the goal of photorealistic graphics. Instead of constructing a scene with millions of polygons, the scene is represented by a collection of photographs along with a greatly simplified geometric model. This simple representation allows traditional light transport simulations to be replaced with basic image-processing routines that combine multiple images together to produce never-before-seen images from new vantage points.

This thesis presents a new image-based rendering algorithm called *unstructured lumigraph rendering* (ULR). ULR is an image-based rendering algorithm that is specifically designed to work with unstructured (i.e., irregularly arranged) collections of images. The algorithm is unique in that it is capable of using any amount of geometric or image information that is available about a scene.

Specifically, the research in this thesis makes the following contributions:

- An enumeration of image-based rendering properties that an ideal algorithm should attempt to satisfy. An algorithm that satisfies these properties should work as well as possible with any configuration of input images or geometric knowledge.
- An optimal formulation of the basic image-based rendering problem, the solution to which is designed to satisfy the aforementioned properties.
- The unstructured lumigraph rendering algorithm, which is an efficient approximation to the optimal image-based rendering solution.
- A non-metric ULR algorithm, which generalizes the basic ULR algorithm to work with uncalibrated images.
- A time-dependent ULR algorithm, which generalizes the basic ULR algorithm to work with time-dependent data.

Thesis Supervisor: Leonard McMillan
Title: Associate Professor

Acknowledgments

Completing this thesis would not have been possible without the help and support of many people. I would like to thank the following people for making the experience fun and rewarding.

- My advisor Leonard McMillan, for being a great advisor and giving me the freedom to pursue my own ideas while gently guiding me in the right direction.
- My thesis committee members Steven Gortler, who has been a source of many ideas and inspiration, and Eric Grimson, who provided a refreshingly different point-of-view.
- All of the members, past and present, of MIT's Computer Graphics Group.
- All of my friends at Microsoft Research, especially Michael Cohen, whom I consider an unofficial thesis committee member.
- My good friends Michael Carr, Michelle Carr, and Lisa Kozsdiy, who have been helpful and supportive throughout my entire graduate career, even when thousands of miles away.
- My roommates J.P. Grossman and Michael Bosse, who are great guys and good cooks.
- My mother and father, who are always there for me, and my brother Tom, for being a great little brother.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

For the past three decades, computer graphics researchers have strived for photorealism, the elusive goal of producing synthetic images that have the appearance of photographs. Great progress has been made: computers have become powerful enough to process the tens of millions of polygons used to represent scenes, complex shading languages have been created to describe the surface properties of these scenes, and sophisticated light transport simulations have been used to predict the appearance of these scenes under any imaginable lighting conditions. However, despite these advances, true photorealism arguably has not been achieved.

As a consequence, researchers recently have turned to a different approach in achieving the goal of photorealism: image-based rendering (IBR). Instead of representing a scene as a set of millions of polygons, the scene is represented by a collection of photographs. Surfaces are no longer described by a carefully programmed shading procedure; they are implicitly described by their appearance in multiple photographs taken from different positions. Time-consuming light transport simulations are replaced with simple image-processing routines. These routines combine multiple images together to produce never-before-seen images from new vantage points.

Image-based rendering has a number of advantages over traditional computer graphics. First, image-based rendering "solves" the photorealism problem. Novel images are composed of real photographs, which by definition are photorealistic. Second, image-based rendering largely eliminates the modeling problem. Image-based modeling consists of taking photographs rather than arranging millions of triangles. Third, image-based rendering is very efficient. In most cases, it is possible to generate output images at greater than 60 frames per second.

Of course, image-based rendering is not without its faults. One of the biggest obstacles to the widespread use of image-based rendering techniques is restrictions placed on the configuration of the input images. For example, it is common for an algorithm to require that cameras face forward and lie on a regular grid, or that cameras face radially inward from the surface of a sphere. Such restrictions make the algorithms easier to implement on the computer, but they also make the algorithms much more difficult to use in practice. Difficulties arise because it is nearly impossible to place real cameras in precise locations. To combat this problem, the rendering algorithms presented in this thesis are designed to operate with almost any configuration of input images. This flexibility allows the algorithms to be used in a wide variety of situations, including some that were never before possible.

1.1 Previous Work

Modern image-based rendering techniques started appearing in the computer graphics community less than a decade ago. Much of the earliest work involves techniques for warping, or "reprojecting," a single image to simulate a novel view of a scene. Later, researchers turned their attention to multi-image methods, which sidestep some single-image problems and enable greater algorithm flexibility and simplicity. These methods roughly fall into two classes: *sparse* multi-image methods, which use a small number of images, and *dense* multi-image methods, which use a large number of images. Of course, the distinction between "small" and "large" can be rather arbitrary; the terms wide-baseline and small-baseline are perhaps more applicable. Sections 1.1.1, 1.1.2, and 1.1.3 highlight some of the

details of the various methods.

In largely parallel work, the computer vision community has developed techniques for rendering novel views from sets of images. This work tends to emphasize techniques that use uncalibrated cameras instead of calibrated cameras, which are preferred in the graphics literature. Section 1.1.4 surveys these methods.

1.1.1 Single-Image Methods

(a) (b)

Figure 1-1: (a) A spherical panoramic image of MIT's lobby 7. (b) A view of the panorama as seen from an interactive viewing application. Images courtesy of Michael Bosse.

The simplest single-image methods use wide field-of-view (e.g., 360 degree cylindrical or spherical) panoramic imagery (see Figure 1-1a). Typically, these panoramic images are created by using image mosaicking [Szeliski 1996; Szeliski and Shum 1997], a technique in which a single high-resolution, wide field-of-view image is created from multiple low-resolution, small field-of-view images. More recently, panoramic lenses and mirrors have been developed to capture panoramas with a single exposure [Nayar 1997].

Panoramic imagery is generally very high-resolution, and it immerses the viewer in the environment. However, the available control over the desired view is very limited. The viewer can look in any direction and change the zoom (see Figure 1-1b). However, the viewer can not change the position of the virtual camera. That is, translation of the virtual camera is not allowed. Nevertheless, panoramic images have been the most successful image-based rendering technique to date, having been commercially deployed in many

products, including Apple's QuicktimeVR [Chen 1995] and Interactive Picture Corporations's IPIX.

In order to allow virtual camera navigation, the single image can be augmented with auxiliary geometric information about the scene. A natural representation for this information is a depth map, which stores a single depth value per image pixel. An image with an associated depth map is often called an *image-with-depth*. A good example is McMillan's plenoptic modeling system [McMillan and Bishop 1995], which uses cylindrical panoramic images that have been augmented with per-pixel depth information. This system provides an immersive experience like a panorama, and it also allows for virtual camera translation. The pixels in the image-with-depth are reprojected into the virtual view using a forward mapping approach.

Figure 1-2: A multiple-center-of-projection image. Both sides of the elephant can be represented in a single image. Image courtesy of Paul Rademacher.

One of the problems associated with this approach is the appearance of "holes" when regions that are occluded in the original image become visible in the virtual view. The Layered Depth Image (LDI) [Shade et al. 1998] combats this problem by storing multiple depth and color values at each pixel in the original image (LDIs are generally used with synthetic data). The extra depth layers help fill in holes in the virtual view. Another approach is multiple-center-of-projection (MCOP) images [Rademacher and Bishop 1998]

(see Figure 1-2). In MCOPs, each column of pixels contains color and depth data that are acquired from a different vantage point (i.e., a multi-perspective image). MCOPs can be created, for example, by moving a line camera (and depth sensor) along a linear path. In doing so, MCOPs can store the image of a scene from many different positions, which allows for greater sampling of potentially visible surfaces.

Another problem with the image-with-depth approach is deducing the depth map. For convincing renderings, the depth map needs to be reasonably faithful to the true geometry of the scene. Some of the highest quality single-image techniques use completely manual specification of the scene depth. In the "Tour into the Picture" system [Horry et al. 1997], the user specifies the depth map as a "spidery mesh" using vanishing points and lines. Foreground objects are manually segmented, and occluded regions are manually painted. A similar approach is used in [Oh et al. 2001], which further allows for scene editing and relighting with only a single image.

1.1.2 Sparse Multi-Image Methods

Some of the best single-image techniques use manual painting of occluded regions. The problem of occlusion can also be attacked in more automatic ways using additional views of the scene. However, using multiple images introduces the interesting problem of how best to combine multiple source images together into one output image.

View interpolation [Chen and Williams 1993] represents a scene with multiple images-with-depth. Virtual image sequences are created by interpolating frames from an original image to a virtual view. In-between images are created by linearly interpolating two-dimensional motion vectors from the source pixels in the original images to the destination pixels in the virtual images. The motion vectors are computed from depth maps, which are acquired from a ray-tracer.

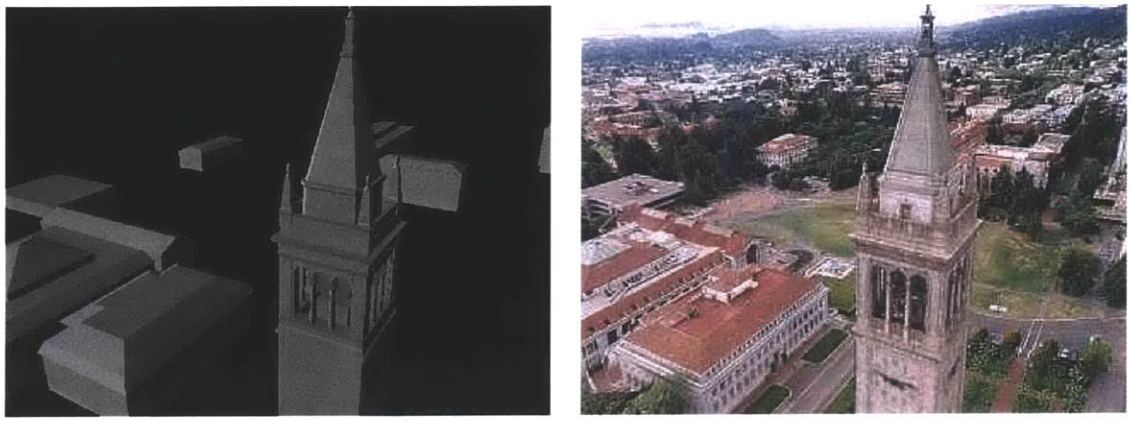
Holes in the interpolated frames are filled in by using a weighted combination of multiple source images. The weights are determined by a spatial graph connecting the positions of the source images. For example, a triangular mesh is used to connect source images arranged in a two-dimensional plane. The weights for a given virtual view are taken to be

the barycentric coordinates of that virtual view in the triangular mesh. This approach can be generalized to cameras arranged in one-dimensional or three-dimensional spaces.

View interpolation forms a linear combination of source images using a *single* weight per image. This weight is calculated based on the physical distance between the virtual view and the source images. A similar approach is taken in [Pulli et al. 1997] and [Pighin et al. 1998], except that the image weights are calculated based on the *angular* distance between the virtual view and the source images. A single "view direction" is computed for each image, typically from the camera position to a target point, and the angles between these directions and the virtual view direction are used to weight the images. Source images with smaller angles are assigned larger weights. In both of these methods, texture-mapped triangular meshes (acquired from laser range scans) are rendered multiple times, once for each source image with a non-zero weight. The renderings are blended together according to their weights to arrive at the final image.

The approach in [Pighin et al. 1998] also includes other factors in the image weighting process. Interestingly, these additional weights are applied at the per-pixel level rather than at the per-image level as with the view-direction weight. The first additional weight attenuates a source image's contribution for pixels close to the edge of its field-of-view. This weight reduces discontinuities at the boundaries of invisible regions, where one image takes over from another. The second additional weight attenuates a camera's contribution for polygons that are obliquely oriented to the camera. This weight penalizes poor sampling densities, and it is intended to reduce blurriness in the rendered image.

The per-image view-direction weight is based on the notion that viewing direction is a better measure of "image closeness" than physical proximity. However, the "view direction" is different for every pixel in the virtual view, so this weight is best computed on a per-pixel basis as well. Per-pixel angular weighting is the approach taken in the Façade system described in [Debevec et al. 1996]. In that system, simple architectural models are textured with a small number of source images (see Figure 1-3). In regions that multiple source cameras see, the cameras' contributions are weighted according to their angles relative to the viewing ray from the virtual view. This approach, called view-dependent texture mapping (VDTM), favors the camera that views the scene point at an angle close to that



(a) (b)

Figure 1-3: (a) The architectural model constructed in the Façade system. (b) A view-dependent rendering of the model using 16 photographs. Images courtesy of Paul Debevec.

of the virtual camera, and it may use all source images in different regions of the virtual image.

VDTM can produce photorealistic images, provided the scene geometry is reasonably good. The main drawbacks of the approach are (1) it is slow and (2) its particular angular weighting scheme only works well with small numbers of images. The first problem has been addressed in [Debevec et al. 1998], which describes a real-time algorithm for implementing VDTM. In this algorithm, image blending weights are computed on a per-polygon basis, which works fine for small numbers of images or finely tesselated models. The second problem is addressed in later chapters of this thesis.

1.1.3 Dense Multi-Image Methods

One common characteristic of sparse multi-image methods is that they assume a fairly accurate geometric model of the scene. However, it is possible to generate convincing views of a scene *without* a model if enough densely-spaced views are available. This notion is nicely quantified in [Chai et al. 2000] by the *minimum sampling curve*, which demonstrates the inverse relationship between number of input images and geometric model complexity.

The light field [Levoy and Hanrahan 1996] and lumigraph [Gortler et al. 1996] papers originate the idea of using large numbers of images to render without a geometric model.

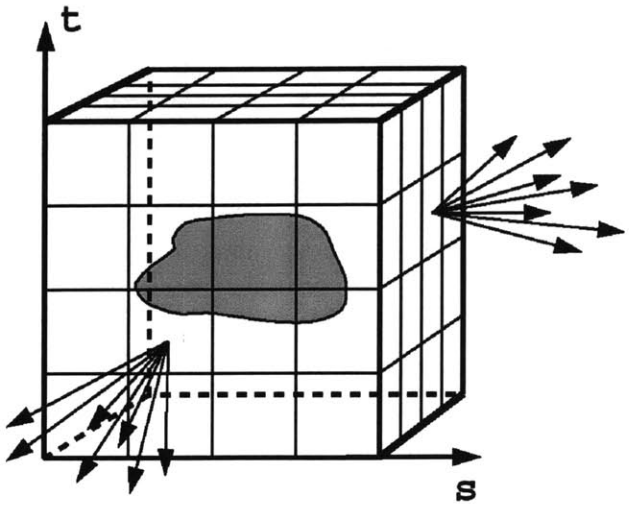


Figure 1-4: The light field and lumigraph techniques represent all radiance entering and leaving a volume, a four dimensional entity. Image courtesy of Steven Gortler.

The basic approach is to represent all of the radiance (i.e., color) in a three-dimensional volume as a four-dimensional function (see Figure 1-4). This four-dimensional radiance representation is called a *light field.* Images of the scene within the volume are simply two-dimensional slices through the four-dimensional light field. The process of generating a virtual view is then seen as a signal reconstruction process: first a continuous representation of the light field is reconstructed from two-dimensional samples (i.e., the source images) and then virtual views are extracted from this continuous light field. To facilitate this signal reconstruction, a regular sampling of the four-dimensional light field is assumed (see Figure 1-5). This assumption is in contrast to most sparse multi-image techniques, which place few restrictions on the camera configurations.

Many extensions and modifications to the basic light field/lumigraph techniques have been proposed. Some authors propose different ray parameterizations [Camahort et al. 1998], dynamic parameterizations [Isaksen et al. 2000], and lower dimensional representations [Shum and He 1999].

However, even with a large number of images (greater than 1000), "pure" light field techniques exhibit objectionable artifacts such as blurring and image ghosting. The lumigraph authors sought to reduce to these artifacts by modifying the signal reconstruction

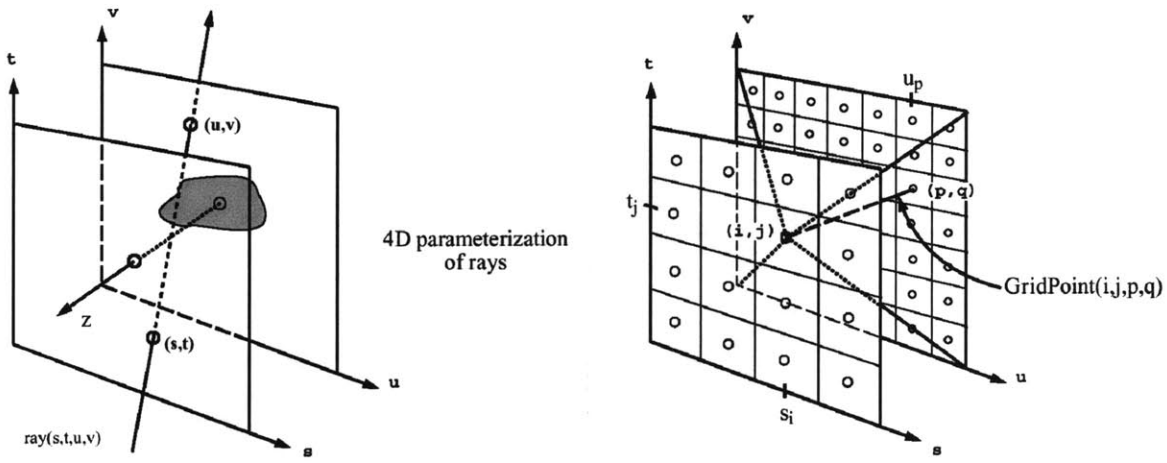


Figure 1-5: (a) In both the light field and lumigraph techniques, rays are parameterized by their intersections with two parallel planes. (b) These planes are uniformly sampled to facilitate signal reconstruction. The points on the *st* plane correspond to camera positions, while the points on the *uv* plane correspond to pixels in the cameras' images. Images courtesy of Steven Gortler.

filters with geometric information about the scene (so-called depth correction). For this reason, the term "lumigraph" is often reserved for light fields that have been augmented with geometric information, which is the terminology adopted in this thesis. The idea of using scene geometry recalls the sparse multi-image techniques, and suggests that lumigraph rendering is simply a dense VDTM technique (or, conversely, that VDTM is a sparse lumigraph technique).

Both interpretations are fundamentally correct, but the details of the algorithms prevent either from being used in place of the other. For example, the basic VDTM blending strategy does not scale well to many images because its blending strategy causes too much blurring . The problem with light field and lumigraph techniques is that they assume that the data is regularly sampled, or, equivalently, that the input images are arranged in a regular grid structure. In fact, this grid structure requirement turns out to be a major impediment to the practical use of light field and lumigraph techniques. Satisfying it generally requires the use of a computer-controlled camera gantry (see Figure 1-6), which limits light field subjects to static scenes inside of a laboratory. Recently, the Digital Michelangelo project [Levoy et al. 2000] has acquired regularly sampled light fields of famous statues in

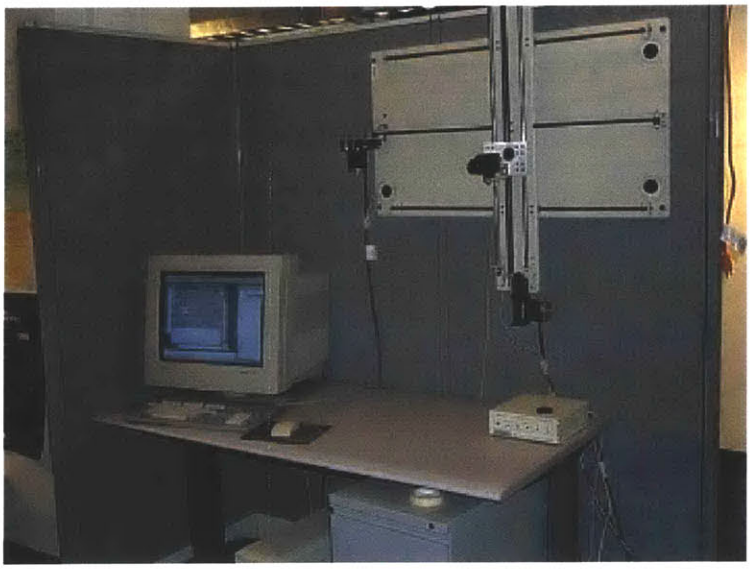


Figure 1-6: MIT's single-camera light field capture device.

Italy, although the work entailed considerable effort and expense.

Interestingly, the original lumigraph paper describes a procedure called *rebinning* that converts an unstructured set of images into a grid-structured set. The rebinning procedure, as described in [Gortler et al. 1996], is undesirable because it introduces degradations to the lumigraph data by resampling the data and by filtering it with a non-linear algorithm to fill in gaps.

It turns out that rebinning is not even necessary. In an attempt to accelerate lumigraph rendering, Sloan et al. [Sloan et al. 1997] demonstrated that lumigraphs can be rendered directly even if the *st* (camera) plane is sampled irregularly. The basic technique uses an arbitrary triangulation of the *st* plane to derive simple linear basis functions for the lumigraph reconstruction. Taking this idea one step further, Heigl et al. [Heigl et al. 1999] generalize the camera plane to a non-planar two-dimensional manifold of cameras. This camera manifold approach allows the use of lumigraph techniques for rendering drectly from video sequences, simplifying the acquisition of lumigraph data.

Unfortunately, camera manifold techniques do not work in all situations. For example, a static non-planar manifold may fold over on itself when viewed from different vantage points. Heigl et al. deal with this particular problem by dynamically recomputing the cam-

era manifold from the virtual camera's point-of-view. But this stop-gap solution ignores the bigger problem: what if the cameras do not naturally lie on a two-dimensional manifold? Dealing with cameras in general position is a central problem of this thesis.

1.1.4 Uncalibrated Methods

The previously discussed methods all assume that the images are calibrated. That is, the camera focal lengths (single- and multi-image methods) and relative camera positions (multi-image methods) are known in advance. Such information is generally obtained by using ray-traced source images or by applying three-dimensional computer vision techniques. However, the need for calibrated images often makes an algorithm difficult to use in practice, since the calibration information is not always easily available. Thus, uncalibrated rendering methods are a popular topic of research.

In the work of Faugeras and Laveau [Faugeras and Laveau 1994], novel views are generated with only knowledge of the fundamental matrix between pairs of views. The fundamental matrix between two views is a weak form of calibration, which is generally far simpler to obtain than a strong calibration. Their approach highlights one of the biggest difficulties of working with uncalibrated images: specifying a virtual camera view. In their work, the viewer specifies the virtual view by constraining the positions of four image points. This form of virtual navigation is unintuitive at best, and it may lead to non-rigid transformations of the scene elements.

A similar technique is described in [Avidan and Shashua 1997], except that trilinear tensor relationships are used instead of fundamental matrices. The trilinear tensor for three views is analogous to the fundamental matrix for two views. In the trilinear tensor technique, the virtual view specification is simplified by roughly guessing the cameras' focal lengths.

Other uncalibrated methods sidestep the virtual navigation problem by constraining the virtual camera motion to easily specified positions. For example, view morphing [Seitz and Dyer 1996] produces physically valid novel views from a pair of uncalibrated source images. The virtual views are constrained to lie on the line connecting the two source

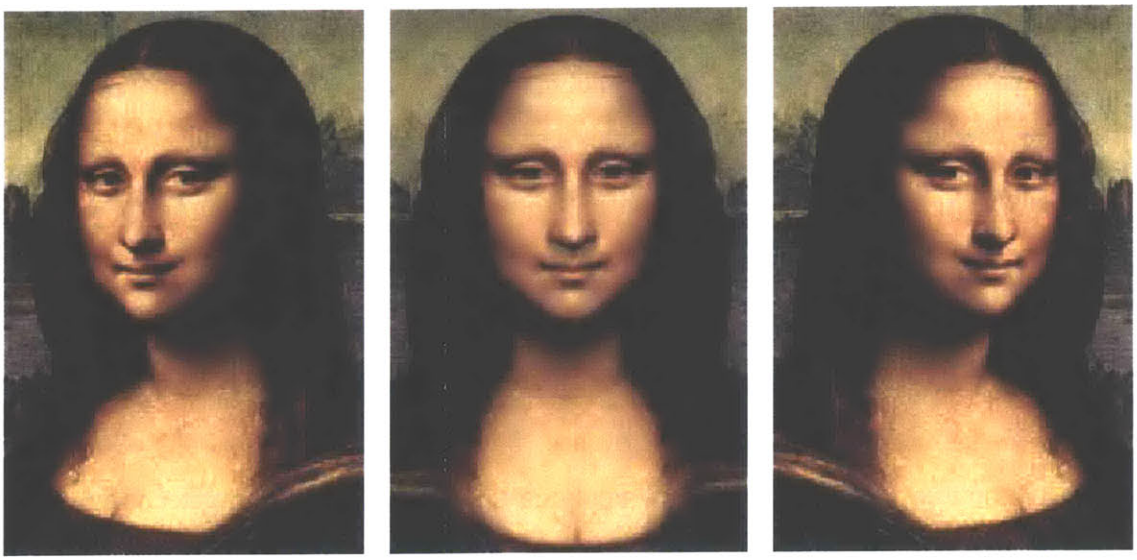


Figure 1-7: A view morph of the Mona Lisa. The left image and right image (reflected) are the source images. The center image is the morphed image halfway between the source images. Images courtesy of Steven Seitz.

cameras' positions, which enables the physically valid morph. An example view morph is shown in Figure 1-7.

A similar approach is taken in [Lhuillier and Quan 1999], except that the interpolated images only approximate a physically valid camera. Scharstein [Scharstein 1996] generalizes the view morphing approach to virtual camera motions that are in the plane connecting three source cameras.

In his Ph.D. thesis, McMillan [McMillan 1996] generalizes his image warping algorithm to use uncalibrated cameras. In his method, he assumes that the virtual view has the same focal length as the source image and derives a parameterized family of valid image warps. Using one of these warps results in a convincing rendering. Chang [Chang and Zakhor 1997] also starts with uncalibrated images and then upgrades to a pseudo-calibrated state that is sufficient for convincing rendering.

1.2 The Contributions of this Research

This thesis presents a new image-based rendering algorithm called *unstructured lumigraph rendering* (ULR). As its name suggests, ULR is a lumigraph-style rendering algorithm that is specifically designed to work with unstructured (i.e., irregularly arranged) collections of images. The algorithm is unique in that it is capable of using any amount of geometric or image information that is available. Thus, the algorithm operates similarly to view-dependent texture mapping when the images are sparse and the geometry is good. At the other extreme, the algorithm behaves like a light field renderer when many images are available and the geometry is unknown.

Specifically, the research in this thesis makes the following contributions:

- A set of image-based rendering properties that an ideal algorithm should attempt to satisfy. An algorithm that satisfies these properties should work as well as possible with any configuration of input images or geometric knowledge (Chapter 3).
- An optimal formulation of the basic image-based rendering problem, the solution to which is designed to satisfy the aforementioned properties (Chapter 4).
- The unstructured lumigraph rendering algorithm, which is an efficient approximation to the optimal image-based rendering solution (Chapter 5).
- A non-metric ULR algorithm, which generalizes the basic ULR algorithm to work with uncalibrated images (Chapter 6).
- A time-dependent ULR algorithm, which generalizes the basic ULR algorithms to work with time-dependent images. Specifically, the time-dependent extensions assume that the images are calibrated in time, although the time dimension may be irregularly sampled (Chapter 7).

1.3 Thesis Organization

Chapter 2 introduces the computer graphics and computer vision background necessary to understand the content of this thesis. The formal definition of an image is presented as

well as the mathematical constructs for manipulating images and geometry. Mathematical notation and terminology are introduced.

Chapter 3 introduces the basic image-based rendering problem that is addressed in this thesis. It also presents nine properties of "ideal" solutions to this problem. It is proposed that an algorithm that satisfies (at least) these nine properties will generate maximal image quality given any configuration of input images and geometry. No existing algorithm satisfies all nine properties, although each of the properties has been addressed by at least one previous algorithm.

Chapter 4 presents an optimal solution to the problems introduced in Chapter 3. The solution is optimal in the sense of minimizing the expected square error in color values at each pixel of a virtual view. The solution requires knowledge of the image autocorrelation function, which is derived using statistical image models. A generalized autocorrelation function is proposed, which accommodates more of the desirable properties from Chapter 3.

Chapter 5 describes the unstructured lumigraph rendering algorithm, which is an efficient approximation to the optimal solution of Chapter 4. ULR accelerates the rendering to real-time performance through a series of five optimizations. Numerous examples demonstrate ULR's effectiveness on a variety of scenes.

Chapter 6 extends the ULR algorithm to uncalibrated cameras. The aspects of ULR that assume calibrated cameras are replaced with alternative techniques that do not require calibration. The effectiveness of the approach is demonstrated with a video stabilization example.

Chapter 7 extends the ULR algorithm to scenes containing time-dependent aspects. The basic approach is to augment each image with a timestamp, essentially calibrating the images in both space and time. The technique is demonstrated with a special class of time-dependent lumigraphs: time-periodic lumigraphs. Time-periodic lumigraphs are interesting because they can be acquired with a single continuous video sequence from one video camera. The resulting lumigraphs are irregularly sampled in both space and time.

Chapter 8 concludes the thesis and presents areas for future research.

CHAPTER 2

Background

Image-based rendering draws heavily from the field of computer vision. In many cases, traditional computer vision problems need to be solved before the actual graphics problems can be approached. For example, in order to use the unstructured lumigraph rendering algorithm, it is necessary to know the camera positions and orientations that produced the input images. Also important is the related problem of finding the positions of objects in the world. Fortunately, calculating camera and object positions from images is a classical computer vision problem that has been addressed in many computer vision papers.

This chapter introduces the mathematical background that is necessary to understand the image-based rendering techniques presented in this thesis. This treatment covers basic computer vision and computer graphics topics, such as camera calibration and projective texture mapping. It also introduces the mathematical notation that is used throughout the rest of the thesis. Readers that are already familiar with the basics of three-dimensional computer vision and graphics can safely skip this chapter.

2.0.1 Overview

The chapter begins by presenting the most basic image-based rendering primitive: the image. Next, the relevant details of real image formation are covered, with particular emphasis on the pinhole projection camera model and the various parameterizations used in this thesis. Then the discussion moves on to other useful concepts, including inverse projection matrices, homographies, and epipoles. Next, the distinction between projective, affine, and Euclidean scene reconstructions is explained. Finally, the topic of camera calibration is briefly covered, focusing primarily on the techniques that are used to calibrate the examples presented later in this thesis.

2.1 Images

The term "image" means different things to different authors. In this thesis, an image is defined to be all of the radiance through a single point in space, called the center of projection. This definition excludes, for example, an MCOP image since MCOPs represent the radiance through multiple centers of projection.

Radiance is a measure of power per unit solid angle per unit area in a particular direction. Thus, an image typically has two dimensions, corresponding to the degrees of freedom of directions through a point. In this thesis, images are denoted by the letter I with an identifying subscript. The center of projection is generally understood from the context.

Images can be treated like a two-dimensional function from angles to radiance, so the notation

$$R = I(\theta, \phi)$$

means that the radiance in the direction specified by θ and ϕ through the center of projection associated with I is equal to R. While there are exceptions [Debevec and Malik 1997], in practice images generally contain quantized RGB color values instead of actual radiance measurements. So $R = (R_{red}, R_{green}, R_{blue})$, where the red, green, and blue values are between 0 and 255 inclusive. As a consequence, the terms "radiance" and "color" are used interchangeably.

Real images are formed through an imaging process, typically using some sort of camera. In most cases, the scene radiance is focused through the center of projection onto a planar imaging surface, where it is recorded. The recorded radiance is digitized into a two-dimensional array of pixels for processing on a computer. In this thesis, the two-dimensional image plane is parameterized by the variables u and v, so the notation $I(u,v)$ refers to the radiance at pixel (u,v) in image I.

The imaging process can be characterized by the camera's projection function. The projection function maps directions (θ,ϕ) to pixel coordinates (u,v):

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} p_u(\theta,\phi) \\ p_v(\theta,\phi) \end{pmatrix}.$$

The inverse projection function is simply in the inverse of the above relationship. Projection functions mapping three-dimensional points (rather than directions) to pixel coordinates are also commonly used. In this case, the inverse function is not uniquely defined.

The projection function can be any mapping, even a non-linear one. For example, the projection function for the image in Figure 1-1 maps most of a unit sphere into a rectangular image. However, one of the simplest and most commonly used projection functions belongs to the simple pinhole camera, which is examined in more detail in the following section.

2.2 Pinhole Cameras

The pinhole camera model is central to image-based rendering. Real cameras have complex optical properties, but for computational simplicity they are generally represented by a simple mathematical model known as the *pinhole projection model.* In the pinhole projection model, a camera simply consists of a center of projection (i.e., the pinhole) and a planar imaging surface, or image plane. The camera forms a planar image, which consists of the radiance along all rays that pass through the center of projection and intersect the image plane. These rays are called viewing rays. One viewing ray is considered special: that is the ray that passes through the center of projection and is perpendicular to the image plane. This ray is called the optical axis of the camera, and the point at which it intersects the image plane is called the principal point (see Figure 2-1).

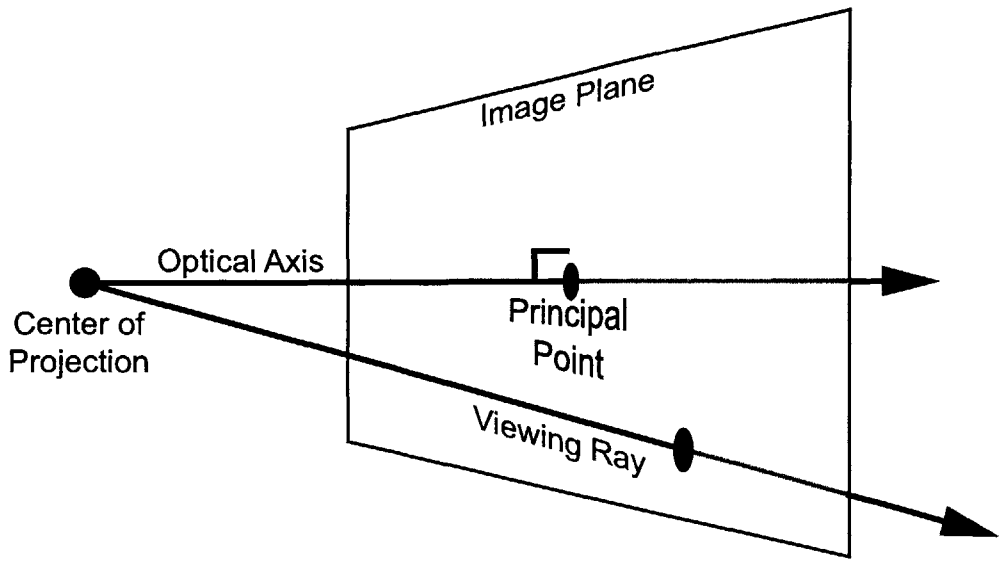


Figure 2-1: An illustration of a pinhole camera showing the center of projection, the optical axis, the principal point, and an example viewing ray.

2.2.1 Projection Matrix

The pinhole camera can be concisely represented with a single 3×4 matrix, called a projection matrix. The projection matrix transforms a three-dimensional point in the "world" to its two-dimensional coordinates in the camera's image plane:

$$\mathbf{u} \doteq \mathbf{PX}. \tag{2.1}$$

Here, lowercase symbols represent three-dimensional points (e.g., image plane coordinates) and uppercase symbols represent four-dimensional points (e.g., world coordinates) or matrices. These points are represented with homogeneous coordinates, which add an extra dimension to each point.

The symbol $\doteq$ represents *equality up to scale*. It is important to note that projection matrices are projective quantities, and that equality holds in equation 2.1 only up to an arbitrary scale factor. Correspondingly, the three-dimensional world point $\mathbf{X}$ is represented (up to a scale factor) with four coordinates $(X,Y,Z,W)^T$, and the two-dimensional image point $\mathbf{u}$ is represented with three coordinates $(u,v,w)^T$. One can think of the "extra" degree-of-freedom as representing the scale factor.

The projection matrix transforms all world points, even those "behind" the camera, to

image points. The only exception is the camera's center of projection, which transforms to $(0,0,0)^T$ (the zero vector is never a valid projective point). That is, the center of projection is the null-space of the projection matrix, so one can find the center of projection of any projection matrix by computing its null-space.

2.2.2 Parameterizations

A projection matrix has twelve elements but only 11 degrees of freedom because of the arbitrary scale factor. Thus, one simple parameterization is simply the eleven elements of the matrix with the twelfth element arbitrarily set to 1. This representation works for all projection matrices, except those whose twelfth element is zero, in which case some other element can be set to 1. However, this representation is neither intuitive nor easy to work with, so other parameterizations are generally used.

The parameterization most often used in this thesis factors the projection matrix into two components:

$$\mathbf{P} = \mathbf{A}\begin{pmatrix}\mathbf{R} & \mathbf{t}\end{pmatrix}, \tag{2.2}$$

where $\mathbf{A}$ is a 3×3 matrix that represents internal properties of the camera, and $\begin{pmatrix}\mathbf{R} & \mathbf{t}\end{pmatrix}$ is a 3×4 matrix that represents the position and orientation of the camera in the world. Implicit in the use of this parameterization is the assumption that the world is represented in a Euclidean space. When working with non-Euclidean spaces, it is necessary to use the more general parameterization. These issues are discussed further in section 2.4.

The position and orientation constitute six degrees of freedom. As shown in the next section, the matrix $\mathbf{A}$ contains five degrees of freedom, which results in a total of 11 degrees of freedom.

Camera Intrinsics

The $\mathbf{A}$ matrix is called the *instrinsics* matrix for the camera, and it encapsulates physical properties of the camera such as focal length and pixel aspect ratio. The $\mathbf{A}$ matrix is an

upper-diagonal matrix with five parameters:

$$\mathbf{A} = \begin{pmatrix} f_u & s & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.3}$$

The diagonal elements, f_u and f_v, are the horizontal and vertical focal lengths of the camera. For cameras with typical lenses, f_u equals f_v (in the case of digital cameras, the camera is said to have square pixels). In cases where f_u does not equal f_v, the fraction $\frac{f_v}{f_u}$ is called the *aspect ratio* of the camera. Cameras with anamorphic lenses have non-unity aspect ratios. Other common examples include miniDV camcorders, which have aspect ratio $\frac{8}{9}$.

The elements c_u and c_v are the coordinates of the camera's principal point in the image plane. In typical cameras, this point is close to the center of the camera's image. However, this is not always the case, especially for skewed-frustum cameras or other "camera-like" devices such as video projectors.

The final intrinsics parameter s is called the skew parameter. A non-zero skew parameter indicates non-orthogonality of the image plane coordinate axes. In almost all cases, the skew parameter is zero, which means the axes are perpendicular.

Since many cameras have unit aspect ratio and zero skew, it is common to use a reduced parameter version of $\mathbf{A}$ consisting of parameters f_u, c_u, and c_v. A single parameter version is also possible by assuming that the principal point lies at the center of the image plane, but this assumption is generally less accurate.

The entries of $\mathbf{A}$ are measured in the same units as the two-dimensional coordinate system of the image plane. In this thesis, the image plane has its origin in the upper-left corner of the image. The positive u-axis is to the right, and the positive v-axis is down. The coordinates are specified in units of pixels. Thus, for an image of $W \times H$ pixels, the upper-left pixel has coordinates $(0,0)$, the lower-right pixel has coordinates $(W-1, H-1)$, and the principal point generally has coordinates close to $(W/2, H/2)$.

Often it is more convenient to specify the field-of-view of the camera rather than the focal length. The focal length of the camera is directly related to the camera's field-of-view.

The simple relation is

$$f = \frac{W}{2\tan(\frac{\theta_{FOV}}{2})},$$

where θ_{FOV} is the field-of-view of the camera measured in radians.

Camera Extrinsics

The second matrix of the parameterization, $(\mathbf{R}\ \mathbf{t})$, is simply a rigid transformation that maps points in the world to the camera's coordinate system. The 3×3 submatrix $\mathbf{R}$ is an orthonormal rotation matrix and the 3×1 subvector $\mathbf{t}$ is the origin of the world coordinate system represented in the camera's frame. Note that this formulation of the projection matrix assumes that three-dimensional points in the world are represented by (possibly scaled versions of) $(X, Y, Z, 1)^T$.

The camera coordinate system used in this thesis is a right-handed system. The origin is located at the center of projection, and the positive z-axis is along the optical axis of the camera. Thus, after transformation, points with positive z values are in front of the camera. When looking along the optical axis, the positive x-axis is to the right, and the positive y-axis is down.

2.2.3 Radial Distortion

The pinhole camera is generally a good approximation to a real camera with typical lenses. However, it fails to model some imaging effects that may be important for improved rendering quality. One of these non-pinhole effects is radial distortion, which is a nonlinear stretching along radial directions in the image. Radial distortion has the effect of making straight lines in the image appear curved. In typical images, radial distortion becomes apparent only near the edge of the image, although with wide-angle lenses it can be quite apparent.

The relationship between undistorted image points $\mathbf{u}'_u$ and distorted image points $\mathbf{u}'_d$ is

generally expressed with a series approximation:

$$\begin{aligned} u'_u &= u'_d + u'_d \sum_{i=1}^{\infty} \kappa_i r_d^i, \\ v'_u &= v'_d + v'_d \sum_{i=1}^{\infty} \kappa_i r_d^i. \end{aligned}$$

The constants κ_i are called the radial distortion coefficients, and $r_d = u'^2_d + v'^2_d$ is the squared radius of the point. In most cases, only the first one or two radial coefficients are needed.

Note that the pixel coordinates $\mathbf{u}'_u$ and $\mathbf{u}'_d$ are not actual pixel coordinates. For these primed coordinates, the principal point has been subtracted and a non-unity aspect ratio has been corrected. All of these issues are easily handled by pre-transforming the pixel coordinates by the inverse of the intrinsics matrix

$$\mathbf{u}'_d = \mathbf{A}^{-1}\mathbf{u}_d,$$

and rescaling the result such that the third coordinate is equal to one. Of course, the radial coefficients κ_i must be calculated with this transformation in mind. The actual pixel coordinates of the undistorted points can then be recovered by transforming with the intrinsics matrix,

$$\mathbf{u}_u = \mathbf{A}\mathbf{u}'_u.$$

Given the radial distortion parameters, it is possible to undistort the image so that it fits the standard pinhole model. In this thesis, it is assumed that all images have been corrected for radial distortion, unless noted otherwise.

2.2.4 Inverse Projection Matrix

The projection matrix maps world points to image coordinates. Logically, the inverse of this matrix should map world coordinates to world points. However, this inverse is not unique, which is clearly seen by noting that multiple world points (i.e., those along the same viewing ray) project to the same image coordinates. Thus, an inverse projection matrix is defined to be any 4×3 matrix $\mathbf{P}^+$ that results in identity when multiplied by its corresponding projection matrix:

$$\mathbf{P}\mathbf{P}^+ \doteq \mathbf{I}.$$

Inverse projection matrices are denoted with a superscript $+$.

A simple way to compute an inverse projection matrix is to first augment the projection matrix $\mathbf{P}$ with a fourth row:

$$\mathbf{P}_{aug} = \begin{pmatrix} \mathbf{P} \\ \Pi \end{pmatrix}.$$

Then, invert the augmented matrix and take the first three columns of the result as an inverse projection matrix:

$$\mathbf{P}_{\Pi}^{+} = \mathbf{P}_{aug,4\times 3}^{-1}.$$

This procedure has an intuitive interpretation. The augmented row Π can be interpreted as the equation of a plane. The resulting inverse projection matrix maps image coordinates to world points on this plane Π. This mapping is unique, except in the case when the plane Π contains the center of projection. However, in this case, the plane Π is a linear combination of the other three rows of the projection matrix, which results in a singular matrix $\mathbf{P}_{aug}$ that cannot be inverted.

When working with Euclidean projection matrices, the inverse projection matrix can be computed in the same way. There is one interesting case when $\Pi = (0,0,0,1)$, which is the equation of the plane at infinity. In this case, the inverse projection matrix is

$$\mathbf{P}_{\infty}^{+} = \begin{pmatrix} \mathbf{R}^{-1}\mathbf{A}^{-1} \\ 0 \quad 0 \quad 0 \end{pmatrix},$$

which maps image coordinates to *directions* in the world coordinate system.

2.3 Multiple Cameras

The previous sections covered mathematical notions concerning a single camera. In the following sections, the relationships between two or more cameras are explored.

2.3.1 Epipoles

Given the projection matrices of two or more cameras, it is possible to compute the image of one camera as seen by another. If $\mathbf{C}_0$ is the center of projection of camera $\mathbf{P}_0$, then its

image in camera $\mathbf{P}_1$ is

$$\mathbf{e}_{10} \doteq \mathbf{P}_1 \mathbf{C}_0.$$

This point is called the *epipole*, and it is useful in many contexts.

2.3.2 Homographies

A homography is a 3×3 matrix that performs a projective change of basis between 2 two-dimensional projective spaces. Intuitively, a homography simply maps a two-dimensional projective point to another. In this thesis, homographies are used to map points from the image plane of one camera to the image plane of a second camera. Unlike an inverse projection matrix, which maps image points to world points, a homography maps from image points to image points.

Planar Homographies

Of particular interest is a class of homographies known as planar homographies. Given two projection matrices and a plane equation, the planar homography relating them is defined as follows:

$$\mathbf{H}_{01,\Pi} \doteq \mathbf{P}_1 \mathbf{P}^{+}_{0,\Pi}.$$

This homography maps points from the image plane of $\mathbf{P}_0$ onto the plane Π, and then to the image plane of $\mathbf{P}_1$. Planar homographies are interesting because they form the basis of projective texture mapping, which is an important graphics tool for image-based rendering.

Projective Texture Mapping

In traditional texture mapping, an artist or designer specifies an affine mapping between an image and the planar surfaces of a geometric model. Then, a rendering process forms an image of the textured model, usually through a perspective projection matrix. Thus, the entire process of texture mapping can be seen as an affine map from the texture coordinate system to the model's surface followed by a projective map from the model's surface to the final image plane. Not surprisingly, this image-to-image mapping can be concisely represented with a single homography.

Projective texture mapping takes this process one step further. Instead of using an affine map from the texture image to the planar surface, projective texture mapping uses a full projective map (e.g., an inverse projection matrix). The homography for a projective texture map is specified by equation 2.3.2, in which $\mathbf{P}_0$ is the projection matrix associated with the texture image, Π is the plane equation of the surface, and $\mathbf{P}_1$ is the projection matrix of the viewing camera.

Projective texture mapping has an intuitive interpretation. One can imagine that the texture is "projected" onto a planar "screen", much like a slide from a slide projector. This screen is then viewed from another vantage point. Image-based rendering algorithms commonly use projective texture mapping to map photographs onto a three-dimensional model, which is then viewed from arbitrary viewpoints. Since almost all modern graphics hardware supports projective texture mapping, it is an attractive procedure for real-time implementations.

2.4 Cameras and Scenes

The previous sections have considered the representation of cameras in terms of projection matrices. Equally important is the representation of the environment, or scene, that the cameras populate. One factor that greatly influences the design of rendering algorithms is the scene representation.

Consider a typical computer vision scenario: one has a set of images I_i, each with a list of point image features $\mathbf{u}_{ij}$. The task is to recover projection matrices and three-dimensional world points (often called *structure points*) such that the projections of the points match the images features,

$$\mathbf{u}_{ij} \doteq \mathbf{P}_i \mathbf{X}_j, \tag{2.4}$$

for all images and points. Assuming perfect data, a valid solution, also called a *reconstruction* of the scene, will satisfy equation 2.4.

However, equation 2.4 does not provide enough constraints to uniquely specify the scene reconstruction. For example, there is an inherent scale ambiguity; it is not possible to tell if the images view the "true" scene or a miniature version of it. Also, the choices of

origin and coordinate axes are of course arbitrary, which leads to different representations of the same scene. In general, a reconstruction of a scene can be transformed to yield a different, but equivalent, representation of the scene.

2.4.1 Euclidean Reconstruction

When the reconstructed scene differs from the "true" scene by a rigid transformation (and possibly a uniform scale), the reconstruction is said to be Euclidean. It is also known as a metric reconstruction or a strongly calibrated scene. A Euclidean reconstruction corresponds to how we perceive the real world. Distances measured from the reconstruction correspond to the distances in the true scene, and angles between elements of the reconstruction reflect the true angles. Euclidean reconstructions are preferred for image-based rendering because they mesh well with traditional computer graphics. They are also intuitive to work with and easy to visualize.

Unfortunately, from a computer vision standpoint, accurate Euclidean reconstructions are also the most difficult to achieve. Computer vision researchers distinguish three different types of reconstructions (in order of difficulty to compute): projective, affine, and Euclidean. The three types differ in the types of transformations needed to convert the reconstruction to a Euclidean one. Appropriately, an affine reconstruction can be converted to a Euclidean one by applying an appropriate affine transformation, and a projective reconstruction can be converted by applying an appropriate general projective transformation. Non-Euclidean reconstructions are referred to as non-metric reconstructions or weakly calibrated scenes.

2.4.2 Non-metric Reconstructions

Consider a Euclidean reconstruction consisting of projection matrices $\mathbf{P}_i$ and structure points $\mathbf{X}_j$. Since the reconstruction is Euclidean, the parameterization in Equation 2.2 can be used:

$$\mathbf{u}_{ij} \doteq \mathbf{A}_i \begin{pmatrix} \mathbf{R}_i & \mathbf{t}_i \end{pmatrix} \mathbf{X}_j.$$

Converting this Euclidean reconstruction to projective is easily done by transforming the projection matrices and the structure points with an arbitrary, non-singular 4×4 matrix $\mathbf{T}$:

$$\mathbf{u}_{ij} \doteq (\mathbf{A}_i \begin{pmatrix} \mathbf{R}_i & \mathbf{t}_i \end{pmatrix} \mathbf{T})(\mathbf{T}^{-1}\mathbf{X}_j). \tag{2.5}$$

It is obvious that the projection equation is still satisfied, although the projection matrices and structure points are now represented in a projective space. That is, the reconstruction is now a projective one.

Generally, one does not convert a Euclidean reconstruction into a non-metric one. Non-metric reconstructions (both affine and projective) satisfy the basic projection relation of equation 2.4, but provide little other knowledge about the scene. For example, distances and angles measured in these spaces are not meaningful. They also cause problems for computer graphics: virtual cameras are difficult to control in non-metric spaces, and z-buffers become useless. However, because non-metric reconstructions are often easier to obtain, it is useful to understand their limitations and how they can be used in graphics.

To better understand the difference between Euclidean, affine, and projective reconstructions, it is instructive to examine how to convert a projective reconstruction back into a Euclidean one. Given a projective reconstruction, the task is to recover the unknown projective transformation $\mathbf{T}$. Without loss of generality, assume that $\mathbf{P}_0$ has $\mathbf{R}_0$ equal to the identity matrix and $\mathbf{t}_0$ equal to the zero vector. Then the transformed $\mathbf{P}_0$ becomes

$$\mathbf{P}_0\mathbf{T} = \mathbf{A}_i \begin{pmatrix} \mathbf{T}_1^T \\ \mathbf{T}_2^T \\ \mathbf{T}_3^T \end{pmatrix},$$

where $\mathbf{T}_1^T$, $\mathbf{T}_2^T$, and $\mathbf{T}_3^T$ are the first three rows of the unknown projective transformation. Assuming that $\mathbf{A}_i$ is known, the first three rows of the unknown projective transformation are readily available. The difficulty lies in determining the fourth row, which is the equation of the plane at infinity Π_∞ in the projective space. Once it is found, however, the

reconstruction can be transformed by the projective transformation

$$\mathbf{T}_\infty \doteq \begin{pmatrix} \pi_4 & 0 & 0 & 0 \\ 0 & \pi_4 & 0 & 0 \\ 0 & 0 & \pi_4 & 0 \\ & & -\Pi_\infty & \end{pmatrix},$$

where π_4 is the fourth element of Π_∞.This transformation annihilates the fourth row of the projective transformation, resulting in an affine transformation that has the form

$$\mathbf{TT}_\infty \doteq \begin{pmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Of course, this affine transformation can be further annihilated to achieve a Euclidean reconstruction.

In practice, converting a projective reconstruction to a Euclidean one is not so simple. First, the camera intrinsics matrix **A** is not always known, although techniques exist for estimating Π_∞ in the absence of **A** [Hartley et al. 1999; Pollefeys et al. 1999]. Second, estimating the plane at infinity, with or without **A**, is very difficult to do. The process of obtaining a Euclidean reconstruction from a collection of images falls under the topic of camera calibration, discussed in the next section.

2.5 Camera Calibration

In this thesis, the entire process of obtaining a reconstruction of a scene is referred to as camera calibration. Given a collection of images, the goal is to obtain a Euclidean reconstruction of the scene. A large number of methods for solving this problem have been documented in the computer vision literature. This section is not intended to be a survey of calibration techniques; it merely outlines the basic approach that is used to calibrate the image collections in this thesis.

Calibration proceeds in three steps. First, the intrinsic parameters of the camera are determined using multiple images of a checkerboard pattern. Figure 2-2 shows example

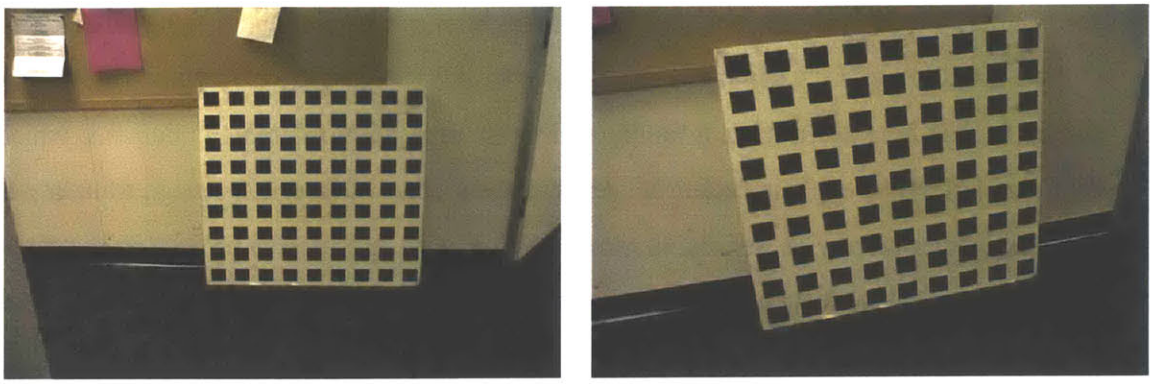


Figure 2-2: Example calibration images.

calibration images. The corners of the checkerboard pattern are automatically tracked, and the resulting features are used in Zhang's calibration algorithm [Zhang 1998]. Zhang's algorithm computes the five intrinsic parameters of the camera as well as two radial distortion coefficients.

The second step is to obtain a projective reconstruction of the scene. An initial solution is computed using a projective factorization technique [Triggs 1996]. The solution is refined using standard robust bundle adjustment techniques [Triggs et al. 2000]. For large image collections, the solution is partitioned into overlapping subsets of images, for which independent solutions are computed. The independent solutions are registered by computing pairwise projective transformations that map each solution into a common projective frame. While this method does not always give a globally consistent solution, it works well enough in most situations.

Finally, the reconstruction is upgraded to a Euclidean one by using the known intrinsic camera parameters and a procedure for estimating the plane at infinity [Pollefeys et al. 1999]. Unfortunately, this procedure occasionally fails, in which case the images can not be used or must be used in a weakly calibrated sense (see Chapter 6).

2.6 Summary

This chapter has presented the basic mathematical background needed to understand the algorithms in this thesis. The most basic primitive is the image, and the most commonly

used camera model is the pinhole model (with perhaps some radial distortion parameters).

The pinhole model is concisely represented by a 3×4 projection matrix. This matrix has no unique inverse; instead a family of inverse projection matrices is defined, which is parameterized by a plane equation. An important class of image-to-image transforms, the planar homography, is defined in terms of projection and inverse projection matrices. Planar homographies form the basis of projective texture mapping, a key computer graphics technique for image-based rendering.

The distinction between Euclidean, affine, and projective representations is explained, with the Euclidean representation being the easiest to work with and the most difficult to obtain. Under a Euclidean representation, the projection matrix can be factored into intrinsic and extrinsic camera parameters, an intuitive and useful form.

A stratified approach to camera calibration is described. First the camera's instrinsic parameters are recovered, followed by a projective reconstruction of the scene. Then, using the camera intrinsic parameters, the plane at infinity is estimated and used to upgrade the reconstruction from projective to Euclidean.

CHAPTER 3

Properties of Image-Based Rendering Algorithms

In recent years, researchers have developed many different image-based rendering algorithms. All of the algorithms attempt to solve the same basic image-based rendering problem: given a collection of images from known viewpoints, generate images from unknown viewpoints.

These algorithms have many similarities and differences. Although all algorithms purport to solve the same problem, in many cases these differences preclude the use of one algorithm in favor of another. For example, standard VDTM algorithms do not perform well (in terms of output image quality) with large numbers of input images, while light field techniques only perform well in this case. One could attribute this behavior to violating the input assumptions of the VDTM algorithm. However, this particular limitation is peculiar, as it seems reasonable to assume that any IBR algorithm should perform better as more images are available. Understanding these differences is key to developing new algorithms that produce high-quality images with a wide range of input configurations.

3.0.1 Overview

This chapter investigates these differences between algorithms. It begins by formally presenting the particular version of the image-based rendering problem that is considered in this thesis. The discussion then turns to three properties that help maximize the flexibility and utility of the IBR algorithms. To illustrate these properties, the chapter develops a *hypothetical* IBR algorithm as a thought experiment. This algorithm is not intended for actual use, and in fact, it is shown to be deficient because it does not properly exploit the *empty-space assumption*. The empty-space assumption simplifies the solution of the *radiance reconstruction problem*, which is a key sub-problem that most modern image-based rendering algorithms address. The chapter concludes with a discussion of six properties of good solutions to the radiance reconstruction problem.

3.1 The Image-Based Rendering Problem

IBR algorithms attempt to solve the following problem:

> Given a collection of calibrated images of a static scene and a specification for an unknown view, generate the image of that scene as seen from that unknown view.

Of course, the above definition contains a number of assumptions that make the problem more tractable. A "calibrated" image has known internal and external camera parameters, which are specified in a Euclidean reference frame. The "static scene" assumption specifies that there are neither motion changes nor lighting changes visible in the reference images. Note that the actual scene does not need to be static, as the static scene assumption can be satisfied with, for example, multiple synchronized cameras.

3.2 Desirable Properties for Algorithm Flexibility

Just from the definition of the IBR problem, it is possible to identify certain properties that enable some algorithms to be more flexible than others. A more flexible algorithm can, for

example, handle a wider range of inputs or generate a wider range of outputs.

3.2.1 Property #1: Unstructured Input

It is desirable for an image-based rendering algorithm to accept input images from cameras in general position. One barrier to the use of most existing IBR algorithms is the common restriction that their input images must come from cameras arranged in very specific spatial structures.

For example, the original light field method assumes that the cameras are arranged at evenly spaced positions on a single plane. This limits the applicability of this method since it requires a special capture gantry that is both expensive and difficult to use in many settings [Levoy et al. 2000]. Other algorithms require linear, circular, cylindrical, spherical, or other camera positions that are equally difficult to achieve precisely.

Some researchers suggest "regularizing" the images before applying a rendering algorithm that requires regular inputs. For example, the lumigraph paper describes an acquisition system that uses a hand-held video camera to acquire unconstrained input images [Gortler et al. 1996]. Instead of rendering directly from these images, they apply a preprocessing step, called *rebinning*, that resamples the input images to virtual source cameras situated on a regular grid. Rebinning has at least two drawbacks. First, the rebinning process adds an additional reconstruction and sampling step to lumigraph creation. This extra step tends to degrade the overall quality of the representation. Second, the process of rebinning really does not solve the problem. Rebinning a lumigraph is equivalent to rendering a grid of regularly arranged novel views from unstructured input images. In essence, to rebin a lumigraph one needs to know how to render from an unstructured collection of images, which is the problem that rebinning is intended to circumvent.

Algorithms could be made even more flexible by removing the requirement for strongly calibrated cameras. Since it is generally easier to obtain weak (or no) calibration, such algorithms could be used with data for which calibration is not available or difficult to obtain. Unfortunately, using weak or no calibration strongly impacts the property discussed in the following section, "Natural Navigation." Nonetheless, rendering with weakly calibrated

cameras is useful, and it is discussed in detail in Chapter 6.

3.2.2 Property #2: Natural Navigation

One often-overlooked aspect of the image-based rendering problem is the specification of the output view. In order for an algorithm to be generally useful, it should be easy and natural to specify the output view. Computer graphics researchers are accustomed to controlling the position, look direction, and field-of-view of the rendering camera, and the IBR algorithm should not restrict this freedom.

Generally, navigation only becomes a problem when dealing with weakly calibrated cameras. The scene may be represented by a non-Euclidean reconstruction, in which case it is difficult to specify a desired camera projection matrix that actually corresponds to a realistic camera.

The view morphing algorithm [Seitz and Dyer 1996] is a good example of an algorithm with a restrictive navigation mode. View morphing uses only weakly calibrated cameras (just a fundamental matrix is required), but it limits the position of the virtual camera to be on the line connecting the two input cameras. As a result, view morphing can synthesize convincing linear motions, but it has limited suitability to other tasks.

3.2.3 Property #3: Real-Time Performance

It is also desirable that the image-based rendering algorithm run at interactive rates. While this property is desirable for almost all computer graphics algorithms, it is especially true for image-based rendering algorithms, which have been billed as real-time alternatives for photorealistic graphics.

Typical applications of IBR algorithms require high performance. For example, image-based algorithms are often targeted at immersive, virtual reality applications in which responsiveness is very important. They are also useful for three-dimensional displays and video processing applications, both of which require real-time performance.

Furthermore, most existing image-based algorithms run an interactive rates. It is reasonable to expect new algorithms to ensure that images are still computed efficiently.

3.3 A Hypothetical Algorithm

To illustrate the above properties, consider a hypothetical image-based rendering algorithm. The input to this algorithm is a collection of images I_i and their corresponding projection matrices:

$$\mathbf{P}_i = \mathbf{A}_i \begin{pmatrix} \mathbf{R}_i & \mathbf{t}_i \end{pmatrix}.$$

It is assumed that the input images are scattered randomly throughout the three-dimensional region of space in which navigation is desired. The virtual image is specified with another projection matrix,

$$\mathbf{P}_{out} = \mathbf{A}_{out} \begin{pmatrix} \mathbf{R}_{out} & \mathbf{t}_{out} \end{pmatrix}.$$

In this hypothetical algorithm, the virtual image is formed as a linear combination of four warped input images. The algorithm first selects four images that are "close" to the virtual image. Then these images are warped so that their orientations are compatible with the virtual image. Finally, the four images are blended together to form the virtual image.

Formally, the virtual image is defined by the following equation:

$$I_{out} = w_i \mathbf{H}_i I_i + w_j \mathbf{H}_j I_j + w_k \mathbf{H}_k I_k + w_l \mathbf{H}_l I_l,$$

where I_x are the four images, $\mathbf{H}_x$ are homographies that warp the images, and w_x are weighting factors that sum to one.

The four "closest" input images I_i, I_j, I_k, and I_l are determined by using a tetrahedral decomposition of space. First, the three-dimensional positions of the input cameras are connected in a tetrahedral mesh (much like a triangulation of two-dimensional points in the plane). The desired virtual camera can then be localized to within one tetrahedral cell in the mesh (in this algorithm, virtual views are limited to within the convex hull of the input cameras). The four cameras forming the vertices of this cell are used for the image interpolation, and the four weighting factors are just the barycentric coordinates of the virtual camera's location within the cell.

Before forming the linear combination, it is necessary to warp the input images so that their orientations and internal parameters match those of the virtual view. Assuming planar

projection cameras, this warp is accomplished by applying the following homography:

$$\mathbf{H}_i = \mathbf{A}_{out}\mathbf{R}_{out}\mathbf{R}_i^{-1}\mathbf{A}_i^{-1}.$$

Intuitively, this algorithm simply blends together four input images that "surround" the desired virtual image. Of the four, the images that are physically closer to the virtual camera contribute more than those that are farther away.

In terms of the previously discussed properties, this algorithm does very well. It accommodates cameras in general position because it uses a tetrahedral mesh to determine interpolation neighborhoods rather than a fixed regular structure. It also handles differences in camera orientation by using a homography to warp images before interpolation. The virtual camera can be completely specified using a simple Euclidean projection matrix, and the algorithm would be trivial to implement in real-time on modern graphics hardware.

However, under the surface, this algorithm has serious flaws. These flaws are primarily due to the fact that the algorithm does not exploit the empty-space assumption.

3.3.1 The Empty-Space Assumption

A typical scene consists of opaque objects that are arranged in empty space. Of course, the space is not really empty; it contains air, which is generally assumed to be a *non-participating medium*. This assumption means that the color of a point in the scene remains the same no matter the distance from which it is viewed. In other words, the transmission medium (air) does not change the radiance that travels through it.

This assumption about empty space (i.e., that it is non-participating) is very important for image-based rendering algorithms. It turns out that there are very few scenes for which this assumption does not at least partially hold. For example, scenes with participating media, such as smoke or water, often violate the static scene assumption, and thus they are already considered invalid. In other cases, such as outdoor scenes with distant haze, the effect may not be noticeable in the desired navigation region.

Exploiting the empty-space assumption gives an image-based rendering algorithm two advantages. First, it allows the algorithm to utilize potentially *all* input images to form the highest quality virtual image. As an example, consider rendering an virtual image from

a collection of N input images. The virtual image can be considered to be a collection of viewing rays (say, one for each pixel in the image). The task is to determine the radiance, or color, along each of the viewing rays in the virtual image. A special subset of these viewing rays connect the center of projection of the virtual camera to the centers of projection of the N input cameras. The empty space assumption allows the colors along these N viewing rays to be determined exactly, because these colors have been directly observed by the N input cameras. Note that as N increases, the colors along more viewing rays can be determined exactly. Thus, the "error" in the virtual image (e.g., measured as the squared difference from the unknown true image) decreases as the number of input images increases. In the limit as $N \rightarrow \infty$, the exact virtual image can be recovered.

The second, and most significant, advantage of the empty space assumption is that it allows for a reduction in dimensionality of the rendering algorithm's input data. In the hypothetical algorithm, the input data consists of two-dimensional images scattered throughout three-dimensional space, a five-dimensional data structure. By utilizing the empty space assumption, the dimensionality can be reduced to four. Consider a collection of N cameras that are scattered along a two-dimensional surface that encloses a convex volume of space. For any virtual view inside the volume, the cameras along the surface contribute N viewing rays to image. As N increases, more viewing rays are available. In the limit as $N \rightarrow \infty$, the exact output image can again be recovered, but in this case the cameras populate a two-dimensional surface instead of a three-dimensional volume.

Of course, regardless of the organization of the input cameras, practical rendering systems can not expect an infinite number of input cameras. Current systems typically work with images numbering in the hundreds to thousands, but sometimes much less. Typical output images can have a million pixels, which means that at best only 0.1% of the pixels can be colored exactly. The remaining 99.9% of the pixels need to recovered from the rest of the input data by solving the *radiance reconstruction problem.*

3.4 The Radiance Reconstruction Problem

The radiance reconstruction problem is defined as follows:

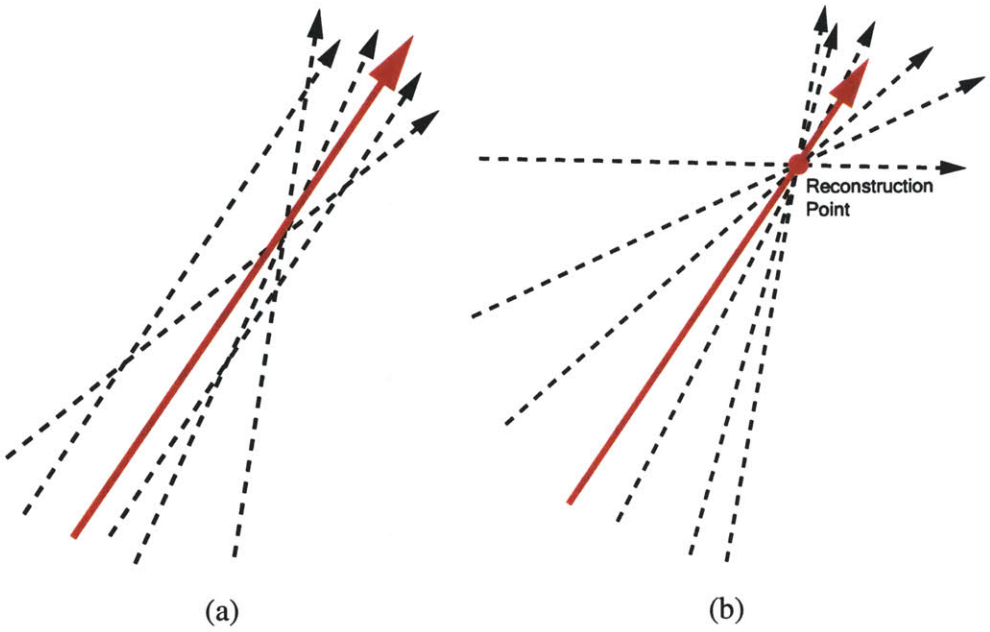


Figure 3-1: (a) Unknown radiances (bold arrows) can be determined from known radiances along any set of "nearby" rays (dotted arrows). (b) It is simpler to use only *corresponding* rays, which are defined to be rays that intersect in a common point on the unknown ray.

> Given a collection of calibrated images of a static scene and a specification for an unknown ray, determine the radiance along that ray.

The radiance reconstruction problem is closely related to the image-based rendering problem. In fact, given a solution to the radiance reconstruction problem, it is simple to solve the IBR problem: simply determine the radiance along all viewing rays in the desired output image. Many IBR algorithms take this approach, including the ones developed in this thesis.

There are many possible ways to attack the radiance reconstruction problem. For example, the hypothetical algorithm of section 3.3 forms the radiance along an unknown ray as the weighted sum of radiances from parallel rays in four neighboring input images. In general, any set of rays from the input images could be used to solve the problem (see Figure 3-1a).

To keep the analysis simpler, the techniques described in this thesis determine the unknown radiance as a weighted sum of radiances of *corresponding* viewing rays in the input images. In this context, viewing rays correspond if they all intersect at a single point in

space. Equivalently, this simplification can be seen as choosing a special *reconstruction point* along the unknown ray around which the radiance is estimated (see Figure 3-1b).

Restricting attention to a single point on the ray has a number of advantages. First, a set of radiances through a single point is simply an image, so existing image analysis and reconstruction techniques can be brought to bear on the problem. Second, when the point happens to lie on the surface of an object, the radiance variation obeys certain well-studied rules relating to the surface reflectance and the illumination of the scene. These rules can be used to improve the reconstruction. Third, the input rays that intersect this point can be ordered very simply according to their angles relative to the unknown viewing ray. This fact turns out to be important because rays with smaller angles tend to have the best estimates for the unknown radiance.

3.5 Desirable Properties for Radiance Reconstruction

There are a number of properties that a good solution to the radiance reconstruction problem should have. Following these guidelines can help ensure that a radiance reconstruction algorithm gives the best possible results in a wide variety of situations.

3.5.1 Property #4: Use of Correspondence

When pixel correspondence is known, it should be exploited to determine the reconstruction point on the unknown ray (see Figure 3-2). This pixel correspondence can be specified in a variety of ways, such as a geometric model, a depth map, or an optical flow field. Since the IBR problem assumes a static scene, pixel correspondence generally implies some sort of fixed geometry, which is the representation most commonly used in this thesis.

In this thesis, such approximate geometric information is called a *proxy*. The term proxy is used to emphasize that the geometry is just a stand-in for the true geometry, and that the proxy may in fact be an extremely coarse approximation. As shown in [Isaksen et al. 2000; Chai et al. 2000], the proxy need not be exact when many input images are available. Conversely, when few images are available, the proxy requires more fidelity.

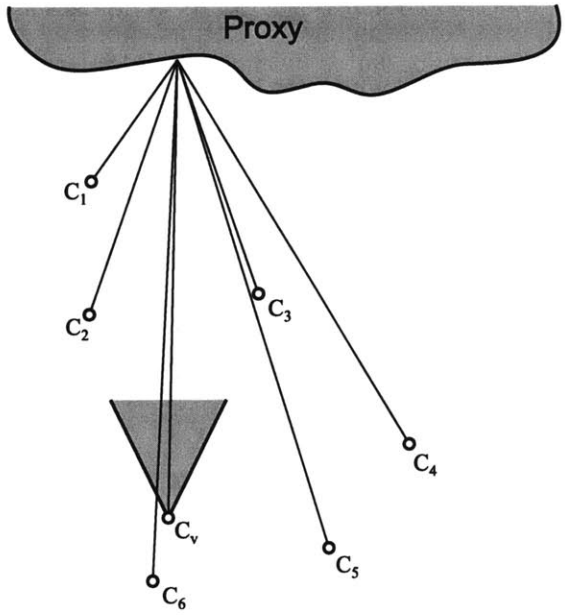


Figure 3-2: When available, approximate geometric information should be used to determine which source rays correspond well to a desired ray. Here $C_1 \ldots C_6$ denote the positions of reference cameras, and C_v is the virtual viewpoint whose field-of-view is shown as a gray triangle. The proxy is represented with a gray shape.

To understand the role of the proxy in radiance reconstruction, consider the case of a perfectly Lambertian scene. In this case, it is well-known that the color of a scene point does not depend on the viewing direction. Thus, if the reconstruction point is on the proxy, then all of the known viewing rays observe the same radiance, and only one observation is needed to reconstruct the correct radiance.

In the case of an imperfect proxy, the reconstruction point may lie in front of or behind the true scene surface. The known viewing rays then do not intersect the true geometry in a single point, but rather in an area on the surface. If this surface area is small, then it is more likely that it is has a nearly uniform color. If the intersection area is large, then the known viewing rays may actually intersect multiple regions of different colors (see Figure 3-3). In the former case, the radiance reconstruction is likely to be good despite the incorrect geometry. In the latter case, the reconstruction will be poor. In both cases, the area of intersection can be made smaller by improving the proxy, leading to a better radiance reconstruction.

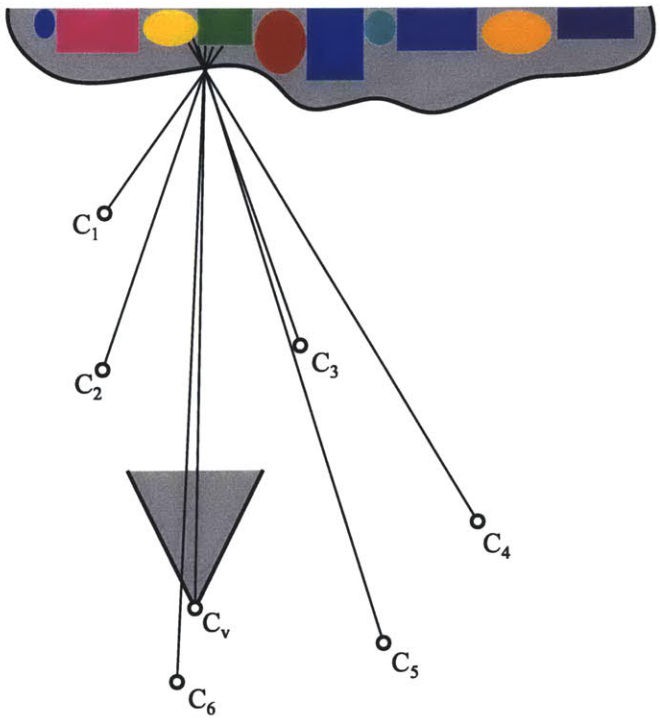


Figure 3-3: The proxy is a coarse approximation to the true scene geometry, which in this case consists of multi-colored boxes and ovals. Since the proxy is not exact, a point on the proxy may be seen with different colors from different reference cameras. In this case, some cameras see the green color while others see the yellow color.

3.5.2 Property #5: Angle-Based View-Dependence

The area of intersection can be made smaller in another way: by reducing the angles between the known viewing rays and the unknown ray. Rays with smaller angular differences are much less sensitive to errors in the proxy, as they "spread out" less as the proxy deviates from the true geometry. The behavior helps explain the inverse relationship between number-of-images and quality-of-geometry. As the number of images increases, there are more known viewing rays that are angularly close to the unknown ray.

The angular closeness of viewing rays is even important when the scene geometry is known. Consider the case of a non-Lambertian scene and precise geometry. The radiance observed at the reconstruction point may vary with the viewing angle (e.g., because of specular highlights or other reflections). In general, one should reconstruct the radiance using rays that view the reconstruction point at an angle close to that of the unknown ray,

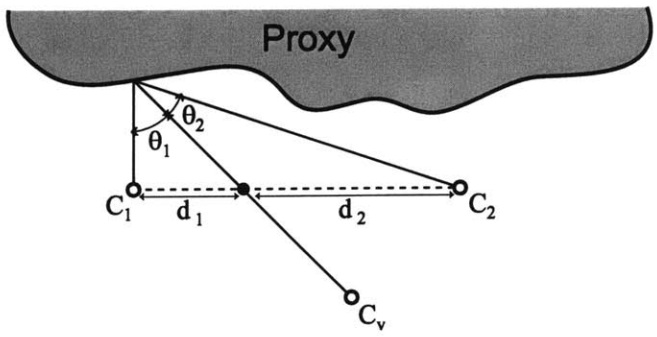


Figure 3-4: The "closest" ray measured by distances d_1 and d_2 is not necessarily the closest one when measured by angles θ_1 and θ_2.

just as in the case with a poor geometry proxy.

Clearly in a more realistic setting, such as a non-Lambertian scene with unknown geometry, the radiance reconstruction algorithm should consider very strongly the angular deviations of the known viewing rays from the unknown one. That is, the algorithm should try to reconstruct the radiance in a view-dependent way. View-dependence has two aspects: first, known viewing rays that are close in viewing angle to the unknown ray should be weighted more heavily in the reconstruction. Second, these weights should fall off very quickly as the angular difference increases. Including too many radiance measurements in the reconstruction can lead to excessive blurring and reduction of view-dependent effects such as highlights and reflections.

Interestingly, the light field, lumigraph, and other "camera manifold" rendering algorithms that select rays based on how close the ray passes to a source camera manifold do not always favor the angularly closest radiance measurements. As shown in figure 3-4, the "closest" ray measured by distances d_1 and d_2 is not necessarily the closest one when measured by angles θ_1 and θ_2. While this problem is not very noticeable with traditional light field techniques, it can become a problem with irregular camera manifolds.

3.5.3 Property #6: Epipole Consistency

Related to the notion of view-dependence is the idea of epipole consistency, which was alluded earlier in the discussion of the hypothetical rendering algorithm. When a desired ray passes through the center of projection of a source camera it can be trivially reconstructed

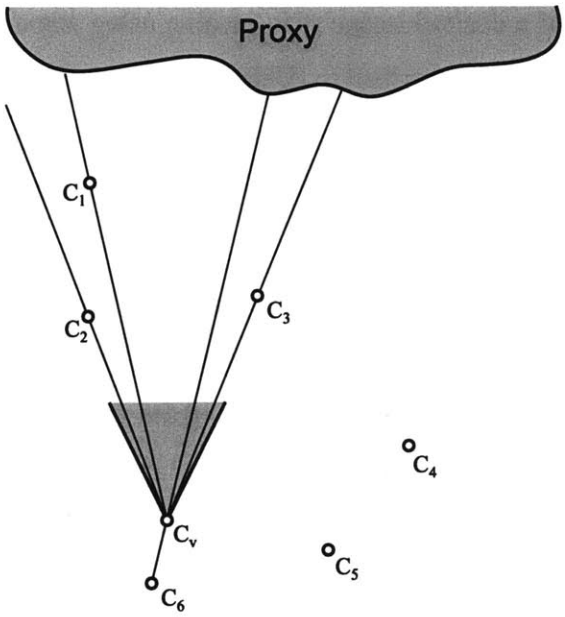


Figure 3-5: When a virtual viewing ray passes through a reference camera center, that reference camera should contribute the exact color to the virtual image. Here this case occurs for cameras C_1, C_2, C_3, and C_6.

(assuming a sufficiently high-resolution input image and the ray falls within the camera's field-of-view) (see Figure 3-5). In this case, an ideal algorithm should return the exact color from the source image. This property is called epipole consistency because the pixels for which colors can be exactly reconstructed are simply the epipoles of the input cameras as seen by the virtual camera.

An algorithm with epipole consistency can reconstruct these special rays correctly without any geometric information (the angular difference is zero). With large numbers of source cameras, algorithms with epipole consistency can create accurate output images with essentially no geometric information. Light field and lumigraph algorithms are designed specifically to maintain this property, which is why they are well-suited to large image sets.

Surprisingly, many real-time VDTM algorithms do not ensure this property, even approximately, and therefore, will not work properly when given poor geometry. The algorithms described in [Pulli et al. 1997; Darsa et al. 1997] reconstruct all of the rays in a fixed desired view using a fixed selection of three source images but, as described in Section 3.3,

proper reconstruction of a desired image may involve using some rays from each of the source images. The algorithm described in [Debevec et al. 1998] always uses three source cameras to reconstruct all of the desired pixels on a single polygon (rather than a single point) of the geometry proxy. This approach departs from epipole consistency if the proxy is coarse.

Generally, satisfying the view-dependence property implies that an algorithm satisfies epipole consistency, at least approximately. However, it is possible for an algorithm to satisfy epipole consistency without strictly satisfying the view-dependence property. This situation occurs in the algorithm of Heigl et al. [Heigl et al. 1999]. This algorithm uses a point on the proxy to determine corresponding rays. However, instead of measuring angular differences relative to this point, it measures angular differences relative to the position of the desired camera. While this procedure works with some camera configurations, it does not work with arbitrary camera configurations.

3.5.4 Property #7: Radiance Consistency

Through any empty region of space, the ray along a given line-of-sight should be reconstructed consistently, regardless of the viewpoint position (See Figure 3-6). This property states that the radiance reconstruction algorithm should enforce the empty space assumption. This property often holds for algorithms that always choose the same reconstruction point on the unknown ray (e.g., the point of intersection with the proxy), but not always.

As mentioned in the previous section, the algorithm of Heigl et al. [Heigl et al. 1999] uses the current desired camera location as the point for measuring angular differences (the algorithm actually uses a measurement in the image plane of the desired camera, which is equivalent to an angle measure). Figure 3-6 illustrates the problem: two desired cameras that share a desired viewing ray have different "closest" cameras, therefore giving different reconstructions. As a result, the algorithm does not satisfy radiance consistency.

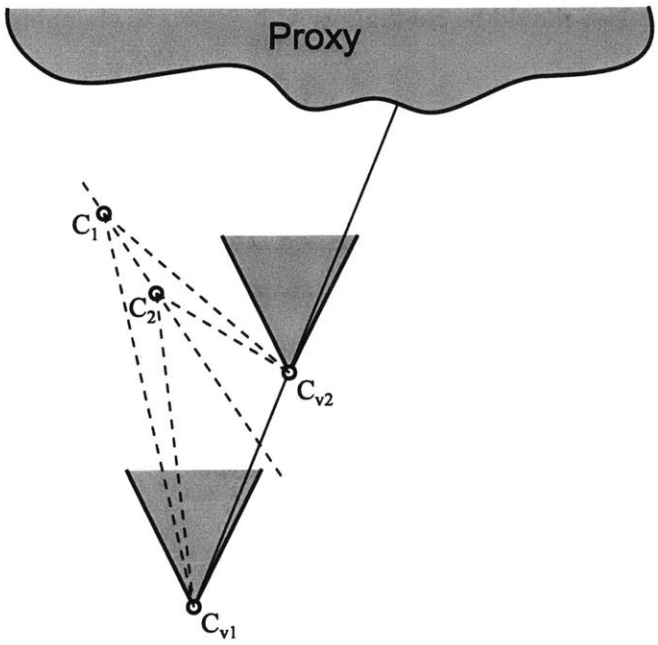


Figure 3-6: When ray angle is measured in the desired view, one can get different reconstructions for the same ray. The algorithm of Heigl et al. would determine C_2 to be the closest camera for C_{v1}, and C_1 to be the closest camera for C_{v2}. The switch in reconstructions occurs when the desired camera passes the dotted line.

3.5.5 Property #8: Continuity

Reconstruction continuity is very important in image-based rendering for avoiding rendering and animation artifacts. There are two notions of continuity: spatial and temporal. Spatial continuity refers to continuity in the reconstruction of a single image. Individual pixel color values should be reconstructed according to underlying continuous basis functions. Temporal continuity refers to the evolution of the reconstructed images over time. If the desired camera moves in a continuous manner, then the image reconstructions should evolve continuously as well. In most applications, minimal C^0 continuity is sufficient for pleasing results.

Spatial and temporal continuity follow directly from the continuity of the radiance reconstruction algorithm. Consider two points in space: a desired camera location and a geometric proxy point. These two points define a viewing ray for which the radiance is to be reconstructed. To ensure spatial continuity, the radiance reconstruction procedure should be continuous with respect to small changes in the proxy point. To ensure temporal

continuity, the procedure should be continuous with respect to changing the virtual camera location.

Some algorithms do not guarantee continuity of reconstruction. The VDTM algorithm of [Debevec et al. 1998] uses a triangulation of directions to source cameras to pick the "closest three" cameras for radiance interpolation. This procedure does not provide spatial continuity when evaluated at different points on the proxy. Nearby points on the proxy can have very different triangulations of the "source camera view map" resulting in very different reconstructions. While this objective is subtle, it is nonetheless important, since lack of such continuity can introduce noticeable artifacts.

3.5.6 Property #9: Sensitivity to Non-Ideal Effects

The previous discussions of the empty space assumption and radiance reconstruction make some "ideal" assumptions. For example, it is implicitly assumed that cameras have infinite resolution, 360 degree field-of-views, and that they perfectly sample the radiance of the environment. Unfortunately, real cameras have finite resolution, finite field-of-views, and they actually integrate radiance over the area of a pixel.

These non-ideal effects often violate the empty space assumption. For example, a low-resolution camera may observe a different color than a high-resolution one, even along the same direction. Or, a camera may not observe the color at all because it falls outside of its field-of-view. Furthermore, the empty space assumption can be violated by occlusion interactions. Two cameras may observe different colors because of an intervening opaque object between them.

While these effects are often so minor as to be ignored, a good algorithm should be sensitive to their impact on the image quality. In addition, it is important to maintain continuity while handling these issues.

Resolution

In reality, image pixels are not really measures of a single ray, but instead an integral over a set of rays subtending a small solid angle (See Figure 3-7). For example, if a source camera

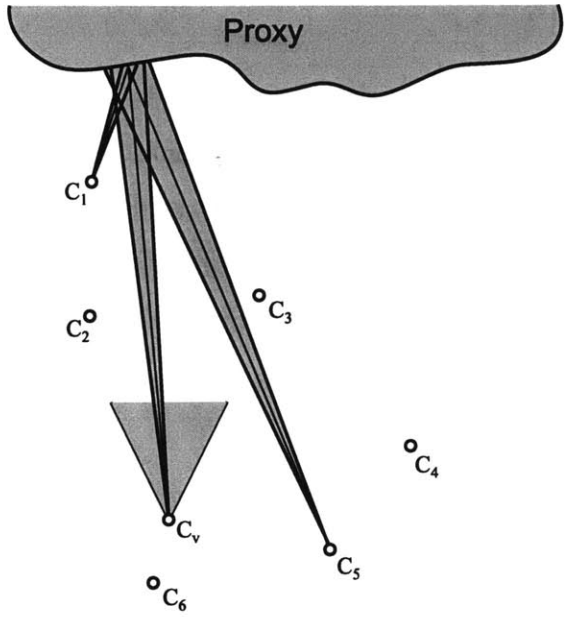


Figure 3-7: When cameras have different views of the proxy, their resolution relative to the virtual view differs. Here cameras C_1 and C_5 have different resolutions because of their distances from the proxy.

is far away from an observed surface, then its pixels represent integrals over large regions of the surface. If these ray samples are used to reconstruct a ray from a closer viewpoint, an overly blurred reconstruction will result (assuming the desired and reference rays subtend comparable solid angles). Resolution sensitivity is an important consideration when combining source rays from cameras with different focal lengths, or when combining rays from cameras with varying distance and obliqueness relative to the imaged surface. It is seldom considered in traditional light field and lumigraph rendering, since the source cameras usually have common focal lengths and are located roughly the same distance from any reconstructed surface. However, when using unstructured input cameras, a wider variation in camera-to-surface distances can arise, and it is important to consider image resolution in the radiance reconstruction process. To date, no image-based rendering approaches have dealt with this problem.

Field-of-View

Some cameras may not see the reconstruction point and consequently should not be used in the radiance reconstruction algorithm. This situation is easy to check for, but care must be taken that the field-of-view is accounted for in a way that preserves reconstruction continuity. For example, the contribution due to any particular camera should fall gradually to zero as one approaches the boundary of its field-of-view [Debevec et al. 1996].

Visibility

With a highly accurate geometric model, the visibility of any surface point relative to a particular source camera can also be determined. If a camera's view of the point is occluded by some other point on the geometric model, then that camera should not be used in the reconstruction of the desired radiance. When possible, image-based algorithms should consider visibility in their reconstruction. Again, it is key to incorporate visibility information in such a way as to not violate the continuity requirement, as in [Raskar et al. 1999].

3.6 Summary

This chapter has introduced the two problems that are tackled in this thesis. The the image-based rendering problem is the basic problem that the algorithms presented in this thesis solve. The radiance reconstruction problem is a key sub-problem whose successful solution can be used to solve the larger image-based rendering problem.

Many researchers have proposed algorithms for these problems. However, many of these algorithms have restrictions on the form of the inputs, restrictions on the type of outputs, or sub-optimal image quality. In light of this situation, this chapter outlines a set of nine properties that an IBR algorithm should have in order to be usable with a wide array inputs and outputs while maintaining high image quality. Table 3.1 summarizes how existing IBR algorithms stack up against the desired properties. The properties are summarized in Table 3.2.

Algorithm	Unstructured Input	Natural Navigation	Real-time Performance	Use of Correspondence	Angle-Based View Dependence	Epipole Consistency	Radiance Consistency	Continuity	Non-ideal: Field-of-view	Non-ideal: Resolution	Non-ideal: Visibility
[Chen and Williams 1993]	✓	✓	✓	✓				✓	✓		✓
[Faugeras and Laveau 1994]	✓			✓		✓	✓	✓			
[Levoy and Hanrahan 1996]		✓	✓			✓	✓	✓			
[Gortler et al. 1996]	✓[1]	✓	✓	✓		✓	✓	✓			
[Debevec et al. 1996]	✓	✓		✓	✓[2]	✓	✓	✓	✓		✓
[Seitz and Dyer 1996]	✓[3]			✓		✓	✓	✓			
[Scharstein 1996]	✓[3]			✓		✓	✓	✓			
[Pulli et al. 1997]	✓	✓	✓	✓	✓		✓	✓	✓	✓[4]	
[Chang and Zakhor 1997]	✓			✓					✓		✓
[Debevec et al. 1998]	✓	✓	✓	✓	✓		✓		✓		✓
[Pighin et al. 1998]	✓	✓	✓	✓	✓		✓	✓			
[Heigl et al. 1999]	✓	✓	✓[5]	✓		✓		✓			
[Lhuillier and Quan 1999]	✓		✓	✓		✓	✓	✓			
[Wood et al. 2000]	✓[1]	✓	✓	✓[6]	✓	✓	✓	✓	✓		✓

Table 3.1: Properties of existing multi-image IBR algorithms. [1]The input images are regularized in a pre-processing stage. [2]The angular weighting scheme does not handle dense image collections. [3]The images must be rectifiable. [4]Resolution mismatch is not measured relative to the virtual view. [5]A real-time implementation is suggested but not demonstrated. [6]Extremely accurate geometry is required.

Property	Description
Unstructured Input	Allows the algorithm to use images and geometry in any arrangement.
Natural Navigation	Allows for flexible specification of the desired virtual view.
Real-time Performance	Allows the algorithm to be used in interactive applications.
Use of Correspondence	Improves radiance reconstruction when the density of images is low.
Angle-Based View-Dependence	Improves radiance reconstruction when the pixel correspondence (e.g., geometry) is poor.
Epipole Consistency	Ensures that the algorithm reproduces its inputs.
Radiance Consistency	Ensures that the algorithm exploits the empty-space assumption.
Continuity	Minimizes artifacts by ensuring spatial and temporal continuity in radiance reconstruction.
Sensitivity to Non-ideal Effects	Allows the violation of properties 4–8 to deal with field-of-view limitations, finite resolution, etc.

Table 3.2: Nine desirable properties for an image-based rendering algorithm.

CHAPTER 4

Optimal Radiance Reconstruction

The radiance reconstruction problem is important for image-based rendering, and it lies at the heart of the new unstructured lumigraph rendering algorithm described in Chapter 5 of this thesis. This chapter investigates different "optimal" solutions to the radiance reconstruction problem. These solutions differ in how faithfully they adhere to the desired properties outlined in Chapter 3. Ultimately, approximations to these optimal solutions form the basis of the unstructured lumigraph rendering algorithm.

4.0.1 Overview

The chapter begins by considering a linear minimum mean-squared-error estimator for the unknown radiance. This problem is simple to solve if the correlation function of the data is known. In the IBR case the exact correlation function is not known, but a reasonable approximation can be taken from the image modeling literature. This solution to the radiance reconstruction problem satisfies many ideal properties from Chapter 3, but it does not handle non-ideal effects, such as finite fields-of-view and resolution mismatches between cameras.

To deal with this deficiency, the image correlation function is generalized to account

for differences in field-of-view and resolution. Simple dissimilarity measures for field-of-view and resolution are proposed, and the correlation function is modified to take them into account. This modification results in a radiance reconstruction procedure that can handle non-ideal effects and that is easily generalized to other non-ideal effects.

4.1 Linear Minimum Mean Squared Error Estimation

The radiance reconstruction problem is difficult to solve. In general, it is possible to construct pathological cases in which the unknown radiance is completely unrelated to any observed radiances. However, these cases do not occur frequently, and it is reasonable to consider estimating the unknown radiance as a linear combination of known radiances, which is the approach taken in this thesis. This approach generally works well in practice, especially when the reconstruction point is chosen such that the radiance through it is a smooth function.

Consider a reconstruction point in space. This point and all the radiance that passes through it constitute an unknown image I. By parameterizing directions with two angles θ and ϕ, I can be considered a function $I(\theta,\phi)$ from directions to radiance.

Under the empty space assumption, a set of N known images I_j provides N radiance samples of the unknown image I. These samples can be represented by the direction (θ_j,ϕ_j) from the unknown image to the known one. Thus, given an arbitrary direction (θ,ϕ), the task is to determine a set of linear weights w_j such that

$$\tilde{I}(\theta,\phi) = w_1 I(\theta_1,\phi_1) + w_2 I(\theta_2,\phi_2) + \cdots + w_N I(\theta_N,\phi_N)$$

is the "best" estimate of the unknown radiance $I(\theta,\phi)$.

There are many possible notions of "best," and the particular choice depends on the application. One commonly used criterion is to minimize the average squared error between the estimate and unknown quantity. Since the unknown quantity is, of course, not known, the mean-squared-error (MSE) is defined in a statistical sense using the notion of expected value:

$$e_{MSE}(\theta,\phi) = \mathrm{E}\left[\left(I(\theta,\phi) - \sum_{j=1}^{N} w_j I(\theta_j,\phi_j)\right)^2\right].$$

The error $e_{MSE}(\theta,\phi)$ is simply a quadratic expression in the unknown weighting factors w_j. This error can be minimized by taking derivatives with respect to the unknown weights and exploiting the linearity of the expected value operator $\mathrm{E}[\cdot]$. The following system of linear equations is found,

$$\begin{pmatrix} \mathrm{E}[I_1 I_1] & \mathrm{E}[I_1 I_2] & \cdots & \mathrm{E}[I_1 I_N] \\ \mathrm{E}[I_2 I_1] & \mathrm{E}[I_2 I_2] & \cdots & \mathrm{E}[I_2 I_N] \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{E}[I_N I_1] & \mathrm{E}[I_N I_2] & \cdots & \mathrm{E}[I_N I_N] \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix} = \begin{pmatrix} \mathrm{E}[I_1 I] \\ \mathrm{E}[I_2 I] \\ \vdots \\ \mathrm{E}[I_N I] \end{pmatrix}, \tag{4.1}$$

where the shorthand notation $\mathrm{E}[I_j I_k]$ stands for $\mathrm{E}[I(\theta_j,\phi_j)I(\theta_k,\phi_k)]$ and $\mathrm{E}[I_j I]$ stands for $\mathrm{E}[I(\theta_j,\phi_j)I(\theta,\phi)]$.

In order to solve this equation, it is necessary to know the function

$$R_I(\theta_1,\phi_1,\theta_2,\phi_2) = \mathrm{E}[I(\theta_1,\phi_1)I(\theta_2,\phi_2)], \tag{4.2}$$

which is known as the *correlation function* of image I. Intuitively, the correlation function describes how similar two "pixels" (or radiance values) in image I are expected to be. Thus, the matrix on the left hand side of equation 4.1 can be seen as a *similarity matrix* $\mathbf{S}$ measuring the expected similarity of all pairs of known radiance samples. Likewise, the vector on the right hand side of equation 4.1 measures the expected similarity between the unknown radiance and the known radiances.

The solution to equation 4.1 results in the optimal weights for radiance reconstruction in the minimum mean squared error (MMSE) sense. In many cases, it is desirable to have a solution in which the weights sum to one. This property is useful, for example, for maintaining constant intensity levels during rendering.

Lagrange multipliers can be used to compute the optimal weights subject to the constraint that $\sum_j w_j = 1$. Doing so leads to an equation of the form

$$\mathbf{S}\mathbf{w}_1 = \mathbf{y} + \lambda,$$

where $\mathbf{S}$ is the similarity matrix, $\mathbf{w}_1$ is the new weight vector, $\mathbf{y}$ is the original right-hand side, and λ is a constant added to each element of $\mathbf{y}$. The constant λ is given by

$$\lambda = \frac{1 - \sum_j w_j}{\sum_{i,j} s_{i,j}},$$

where $s_{i,j}$ is the $(i,j)^{th}$ element of $\mathbf{S}^{-1}$. This procedure gives optimal weights subject to the constraint, but it requires that $\mathbf{S}^{-1}$ be explicitly computed. Although sub-optimal, it is often much easier to simply renormalize the weights so that they sum to one. The renormalization approach is used in this thesis with good results.

4.1.1 Correlation Functions for Images

The correlation function for an image is generally unknown. However, a few assumptions can be made about its form. First, it is generally assumed that the distribution of radiance values in an image is stationary. Mathematically, stationarity implies that

$$R_I(\theta_1, \phi_1, \theta_2, \phi_2) = R_I(0, 0, \theta_2 - \theta_1, \phi_2 - \phi_1),$$

or that the correlation function depends only on the difference in its arguments. In the case of images parameterized by (θ, ϕ), stationarity means that changing the "yaw" and "pitch" angles of the camera does not change the statistics of the image. This assumption reduces the dimensionality of the correlation function by two.

Furthermore, it is generally assumed that the two-dimensional correlation function is rotationally invariant, reducing its dimensionality to one. Intuitively, this means that changing the "roll" angle of the camera does not affect the image statistics either. Now the correlation function has the form $R_I(\theta)$, which measures the expected similarity in radiance between two pixels separated by the angle θ.

Even in its simplified form, the correlation function is not trivial to determine. If a large collection of representative images is available, then the correlation function can be estimated. However, such a correlation function is valid only for images "similar" to those in the original collection. This poses a problem for radiance reconstruction, since the reconstruction point generally does not correspond to a known image (i.e., it is generally a point on the geometric proxy). In fact, it is likely that reconstruction points lie very close to the scene geometry, which would result in a set of images with drastically different statistics from the input images. Thus, a different source for the correlation function is needed, such as a parametric image model.

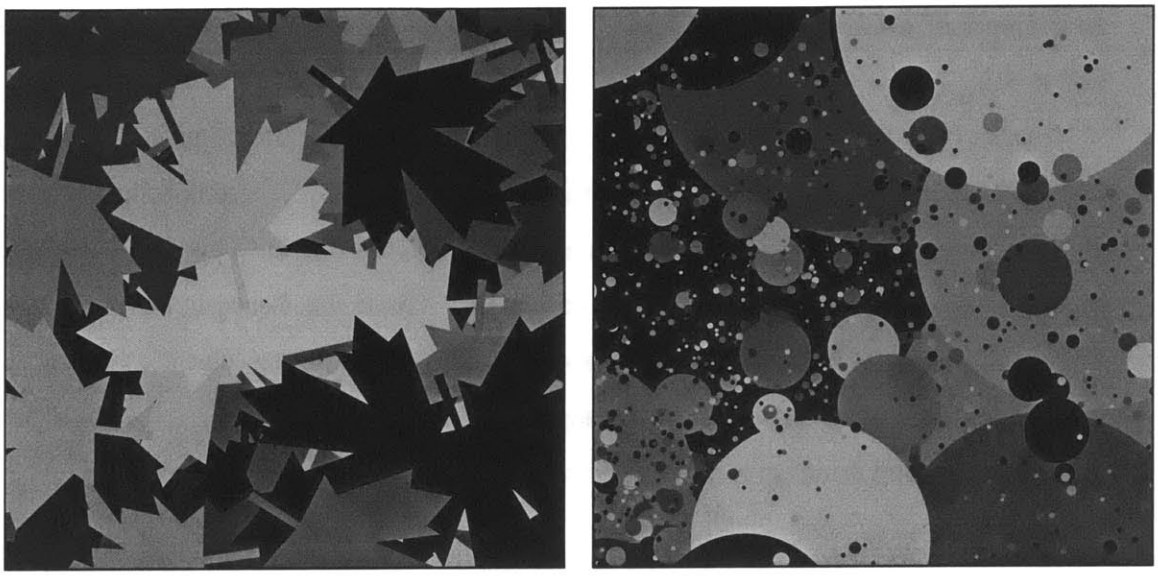


Figure 4-1: Two example images generated from the falling-leaves image model.

4.1.2 An Image Model for Radiance Reconstruction

Image models offer a concise way to describe the statistics of an image. With a small number of parameters, an image model can describe the statistics (e.g., correlation function, joint co-occurrence function, etc.) of an image, assuming the appropriate parameters are used. In this thesis, a simple image model is used to derive an analytical expression for the correlation function of an image. This image model, an instance of the more general *falling-leaves model* [Cowan and Tsang 1994], has two parameters that can be varied to accommodate a wide range of behaviors.

The general falling-leaves model is a constructive image model. That is, the images it models are described constructively rather than mathematically. The falling-leaves model is effective because this construction process mimics the process by which real images are made.

The falling-leaves model models images that are formed in the following way. Imagine an infinite image plane. Flat two-dimensional objects are dropped onto the plane at random locations and orientations, overlapping previously dropped objects. The objects can have random sizes and colors. After a while, the image plane is entirely covered, and the balance between new objects and old reaches a steady-state equilibrium. The resulting mosaic is

an example of a falling-leaves image. Two sample falling-leaves images are shown in Figure 4-1.

Images from the falling leaves model consist of different regions of pixels. Within a region, the colors of pixels are highly correlated. Between regions, there is no correlation. One can think of these correlated regions as surface patches that have approximately the same reflectance properties and lighting conditions. Given the discussion of the view-dependence property in Section 3.5.2, it is reasonable that the falling-leaves image model is appropriate for image-based rendering applications.

The correlation function of a falling-leaves image has an extremely simple form,

$$R_I(\theta) = C_0 P_{same}(\theta), \tag{4.3}$$

where C_0 is a constant and $P_{same}(\theta)$ is the probability that two pixels separated by angle θ belong to the same object in the image. Equation 4.3 assumes that the image I has mean pixel value zero (i.e., the equation actually represents the covariance function). Although most images do not have mean value zero, this is not a problem since the mean is left unchanged by requiring that the linear weights sum to unity.

The specific form of $P_{same}(\theta)$ depends on the details of the "leaves" in the model. In the simple case used in this thesis, the falling objects are uniformly colored circles. The sizes of the circles are distributed according to a $\frac{1}{r^3}$ law, where r is the radius of the circle (the radii are measured in angles). This distribution has been found to result in statistics that mimic those of real images [Lee et al. 1999]. The circles are constrained to have radii greater than r_{min} and less than r_{max}. These two extremal radii constitute the only model parameters, which must be chosen manually.

The image on the right in Figure 4-1 is an example of the type of image generated by the model used in this thesis. Although they look nothing like a "real" images, these types of images model quite closely the correlation functions of real-world imagery.

This circular-leaf model has been extensively studied [Lee et al. 1999; Ruderman 1997], and it has been shown that, to a close approximation,

$$P_{same}(\theta) = \frac{B(\theta)}{2\ln\frac{r_{max}}{r_{min}} - B(\theta)}. \tag{4.4}$$

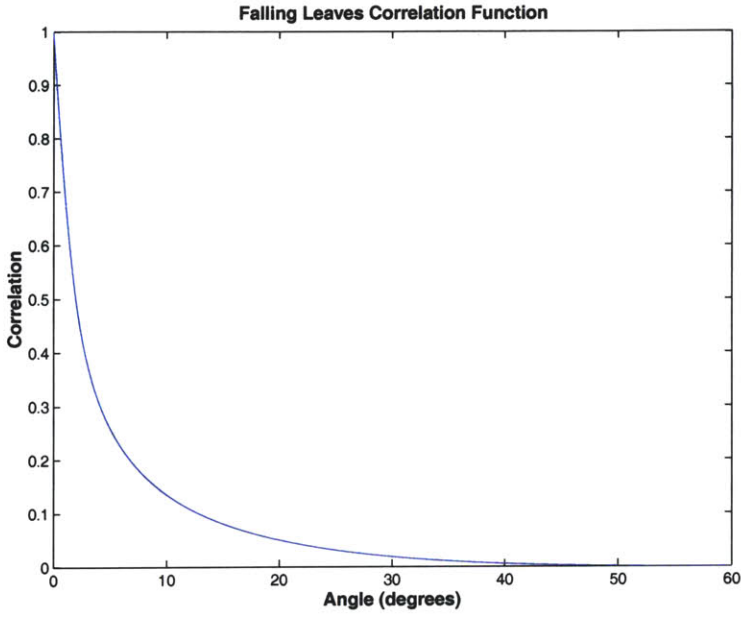


Figure 4-2: An example correlation function from the falling-leaves model.

The function $B(\theta)$ has three cases:

$$B(\theta) = \begin{cases} 0 & \text{if } \theta > 2r_{max}, \\ \frac{a_3}{3}(s^3 - u^3) + \frac{a_2}{2}(s^2 - u^2) + a_1(s-u) + a_0 \ln\left(\frac{r_{max}}{r_{min}}\right) & \text{if } \theta < 2r_{min}, \\ \frac{a_3}{3}(8 - u^3) + \frac{a_2}{2}(4 - u^2) + a_1(2-u) + a_0 \ln\left(\frac{2}{u}\right) & \text{otherwise,} \end{cases} \tag{4.5}$$

where $s = \frac{\theta}{r_{min}}$, $u = \frac{\theta}{r_{max}}$, $a_0 = 1.0$, $a_1 = -0.61$, $a_2 = 0.051$, and $a_3 = 0.052$.

For most settings of r_{min} and r_{max}, the shape of $R_I(\theta)$ follows a power law,

$$R_I(\theta) = -|A| + |B|\theta^{-|\eta|}, \tag{4.6}$$

as illustrated in Figure 4-2.

In turns out that the basic shape of this function is more important than the specifics of A, B, or η. The shape reinforces the notion that radiance samples that are close in angle to one another are highly correlated, while those farther away quickly become less important– a reiteration of the view-dependence principle from Section 3.5.2.

4.1.3 Example I

This section demonstrates the optimal radiance reconstruction algorithm on real data. The data consists of 262 images of a scene that contains various items in front of a black background. Figure 4-3a shows a sample image from the data set.

The images are arranged in a semi-structured manner. They were taken with a camera mounted on a tripod and attached to a FARO digitizing arm. The digitizing arm was used to determine the position and orientation of each image. Multiple images were taken at each tripod location by raising the tripod between exposures. As a result, clusters of images tend to lie along linear paths, although no attempt was made to adjust or place the tripod in a regular way.

Figures 4-3b through 4-3d show the camera configuration from three different angles. Each input camera is represented by a small blue pyramid. The apex of the pyramid is located at the camera's position, and the sides of the pyramid show the field-of-view of the camera. The red camera is the virtual view for which the output image is generated. The large gray triangle is the geometric proxy that is used for determining rough pixel correspondence. In this example, the proxy is simply a planar surface.

The desired view is generated in a ray-tracing fashion. For each pixel in the desired view, the corresponding viewing ray is intersected with the proxy geometry (in this example, a single triangle). This intersection point serves as the radiance reconstruction point for this pixel. The similarity matrix and right-hand side of Equation 4.1 are constructed by evaluating the correlation function (Equation 4.3) for each pair of cameras. In this example, $r_{min} = 0.001°$, $r_{max} = 5°$, and the similarity matrix has dimensions 262×262. The weight vector is then obtained by solving the system of 262 linear equations. Finally, the pixel color is taken to be a weighted sum of colors from each of the input images.

Figure 4-4a shows the image obtained for the virtuald camera from Figure 4-3. The image has a photorealistic quality, and it is not readily apparent that it is a blend of 262 other images. The image is less sharp than any one of the input images, which results from using an extremely crude proxy geometry.

In order to better understand how the output image is formed, it is useful to visualize

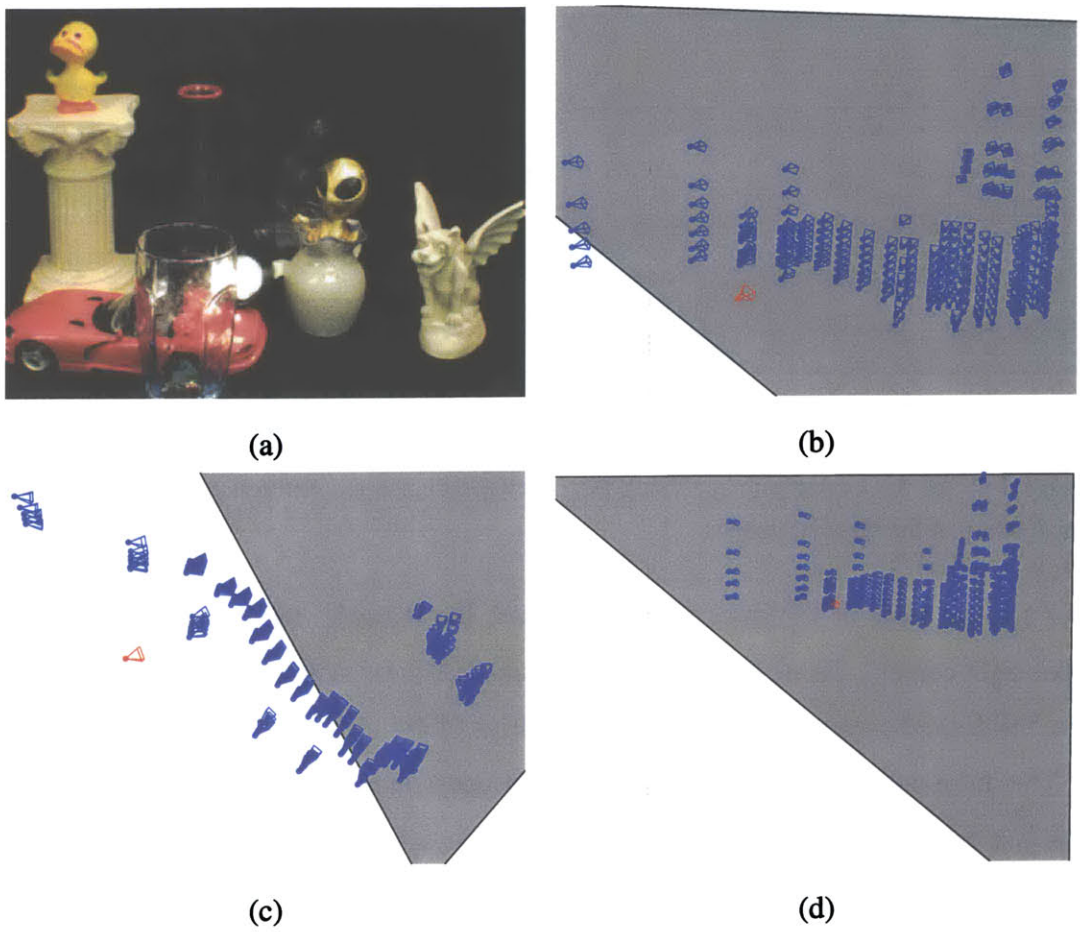
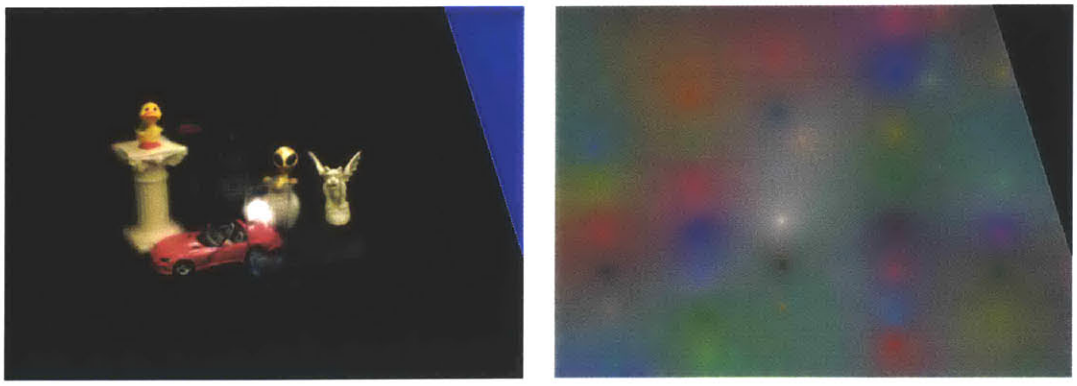


Figure 4-3: A set of 262 images used to demonstrate the radiance reconstruction procedure. (a) An example from the image collection. (b)-(d) Three different views of the input camera configuration. The virtual camera is shown in red.

how the input images combine to form the output image. One simple way to do this is to assign each input image a unique, random color. Then, when forming the estimated color for a pixel, these assigned colors can be blended together instead of the actual colors from the images. This process results in a "false color" image that more clearly indicates the contributions of each of the input images.

Figure 4-4b shows the false color visualization for the image in Figure 4-4a. The bright spots in the visualization correspond to camera epipoles. The colors for these pixels come from a single camera (recall the epipole consistency property of Chapter 3). The pixels near the epipoles tend to be colored similarly to the epipole itself, which indicates that the cameras have large influence on pixels near their corresponding epipoles. Some epipoles



(a) (b)

Figure 4-4: (a) An example image using the optimal radiance reconstruction procedure. (b) A visualization of the blended images.

have smaller areas of influence than others (see, for example, the three epipoles in the upper-right corner). Generally, epipoles with smallers areas of influence are farther away from the desired camera than those with large areas of influence.

This false-color visualization greatly aids in understanding the image formation process. It is especially helpful for visualizing the effects of non-ideal issues, such as field-of-view limitations or resolution mismatches between images, as discussed in the next section.

4.2 MMSE with Generalized Similarity Matrix

The correlation functions derived from image models handle the purely angle-dependent aspects of the radiance reconstruction problem. However, as mentioned in Section 3.5.6, a good radiance reconstruction procedure should be able to accommodate non-ideal effects such as field-of-view limitations or resolution mismatches.

One way to handle these effects is by generalizing the notion of similarity between radiance samples. By augmenting the similarity matrix of Equation 4.1 to reflect differences in, for example, field-of-view or resolution, it is possible to achieve a more flexible radiance reconstruction procedure.

There are a variety of ways to generalize the similarity matrix. Perhaps the simplest way is to modify the correlation function $R_I(\theta)$ to include dependencies on field-of-view

and resolution mismatches, resulting in an augmented function $R_{I,gen}(\theta, f, r)$, where f and r measure differences in field-of-view and resolution, respectively. For example, such dependencies could be represented by using a separable representation of the correlation function:

$$R_{I,gen}(\theta, f, r) = R_I(\theta) R_{fov}(f) R_{res}(r),$$

where the functions $R_{fov}(f)$ and $R_{res}(r)$ attenuate the original correlation function based on differences in field-of-view and resolution.

An alternative approach is to use the field-of-view and resolution measures to modify the input parameter of the original correlation function. The correlation function becomes $R_I(\theta')$, where θ' is a perturbed version of θ. The generalized correlation function is then

$$R_{I,gen}(\theta, f, r) = R_I(h(\theta, f, r)),$$

where $h(\theta, f, r)$ is a function that modifies θ based on the field-of-view and resolution mismatches. This approach (rather than the separable one) is followed in this thesis. The function $h(\theta, f, r)$ is considered a "generalized angle," and it is simply a linear combination of the variables,

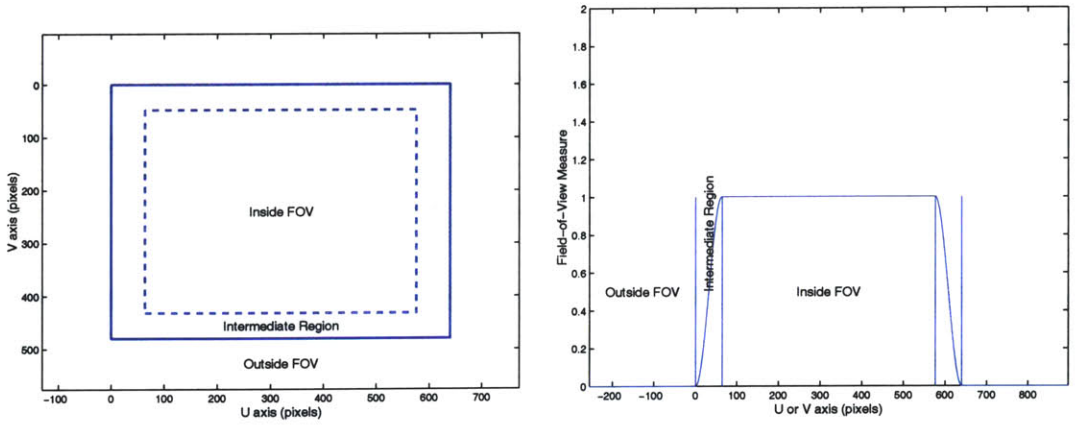
$$h(\theta, f, r) = \alpha\theta + \gamma f + \beta r. \tag{4.7}$$

The constants α, γ, and β control the relative importance of the input variables. For example, γ can be set to zero to ignore field-of-view issues. The linear combination assumes that the three measures are independent.

The variables f and r are assumed to be measures of field-of-view and resolution dissimilarity–that is, they are zero for perfect matches and greater than zero for mismatches. The following two sections describe ways to compute these measures.

4.2.1 Field-of-View Dissimilarity Measure

The field-of-view pairwise dissimilarity measures whether a radiance sample is inside or outside the fields-of-view of two cameras that observe it. For example, if a point is inside both fields-of-view, the the dissimilarity is small. Likewise, the dissimilarity is small if the point is outside both of the fields-of-view. If a point is inside one field-of-view but outside the other (or vice versa), then the dissimilarity is large.



(a) (b)

Figure 4-5: (a) The field-of-view measure is zero outside the field-of-view, one inside the inner field-of-view, and between zero and one in the intermediate region. (b) A cross-section of the field-of-view measure. The values in the intermediate region are determined from a raised cosine function.

A simple way to compute this dissimilarity is to compute a value f_i for each camera that measures how far outside (or inside) the field-of-view the point lies. This value is determined by dividing the image plane of the camera into three regions: (1) outside the field-of-view, (2) inside a smaller, inner field-of-view, and (3) inside an intermediate region between the first two regions (see Figure 4-5a). The value is zero for viewing rays in region (1), it is one for rays in region (2), and it varies continuously between zero and one in region (3). A raised cosine function is used for determining the values in the intermediate region (see Figure 4-5b). Values for both the x- and y-dimensions are multiplied together to arrive at the final field-of-view value f_i.

Given the values f_1 and f_2 for two cameras, the field-of-view dissimilarity is simply

$$f = 2r_{max}|f_1 - f_2|,$$

where r_{max} is the constant used in Equation 4.5. Scaling by this constant causes the correlation to fall to zero at maximum field-of-view dissimilarity. Note that the dissimilarity measure is symmetric, and that it is equal to zero when both cameras are the same. When one of the cameras is the unknown camera (for which the point is assumed to be within its

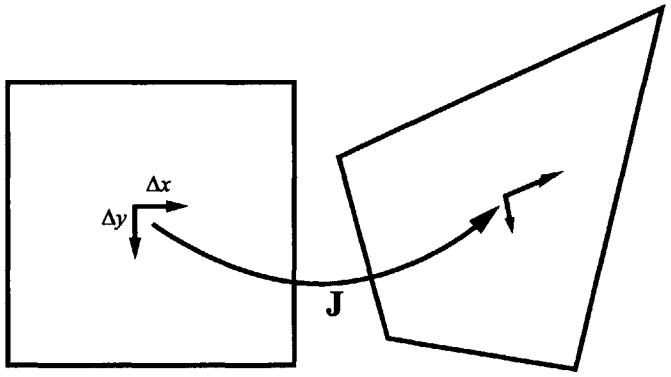


Figure 4-6: The Jacobian matrix describes how small increments in one image are mapped to small increments in another image.

vield-of-view, i.e., $f_1 = 1$), the measure is

$$f = 2r_{max}|1 - f_2|.$$

This expression is zero when the point is within the observer camera's field-of-view and large when it is not.

4.2.2 Resolution Dissimilarity Measure

The resolution dissimilarity compares the sampling densities of two cameras that observe the same reconstruction point. Two cameras that sample the radiance at about the same sampling rate should observe similar radiances, while two cameras with vastly different resolutions may observe different radiances, even along the same viewing direction.

There are a variety of reasons for resolution mismatches. First of all, the camera's distance from the reconstruction point influences the sampling density. The farther the distance, the lower the effective resolution of the observing camera. Also, if the point is on a surface, the surface obliqueness can also affect the sampling density. Cameras best observe surfaces that are oriented perpendicular to their image planes.

All of these resolution concerns can be described in a simple homography [McMillan 1996; Shade et al. 1998]. The resolution of an observer camera is measured relative to the desired camera, a reconstruction point, and a surface normal at this point. The reconstruction point and normal define a plane Π. This plane combined with the desired camera

and an observer camera define a planar homography $\mathbf{H}_{\Pi,j}$, where j is the index of the observer camera. This homography maps points in an observer camera to points in the desired camera. The Jacobian matrix of this mapping (a 2×2 matrix), evaluated at a pixel (x,y), describes how small increments $(\Delta x, \Delta y)$ in an observer image map to increments in the desired camera image (see Figure 4-6). If the entries of $\mathbf{H}_{\Pi,j}$ are

$$\mathbf{H}_{\Pi,j} = \begin{pmatrix} A & B & C \\ D & E & F \\ G & H & I \end{pmatrix},$$

then the Jacobian matrix is given by

$$\mathbf{J}_{\Pi,j}|_{x,y} = \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{pmatrix} = \frac{1}{Gx+Hy+I} \begin{pmatrix} (AH-GB)y+AI-GC & (GB-AH)x+BI-HC \\ (DH-GE)y+DI-GF & (GE-DH)x+EI-HF \end{pmatrix}.$$

The singular values σ_1 and σ_2 of this matrix can be used to measure the resolution mismatch between the two cameras. Intuitively, the singular values indicate the amount of "stretch" that occurs in the transformation from an observer camera to the desired camera. Large singular values (those greater than one) indicate undersampling–a small step in an observer image maps to a large step in the desired image. That is, an observer camera views the point at a lower resolution than the desired camera. Excessive amounts of undersampling can lead to blurriness in the desired image. Small singular values (those less than one) indicate oversampling–an observer camera views the point at a higher resolution than the desired camera. This situation can lead to aliasing in the desired image.

In general, undersampling is worse than oversampling, since the missing information in the undersampled observer images can never be recovered. On the other hand, oversampling can be circumvented either by pre-filtering the observer images or by supersampling the desired image.

In light of this observation, resolution dissimilarity is determined by examining the largest singular value (i.e., worst-case undersampling) of the Jacobian matrix, $\sigma = \max(\sigma_1, \sigma_2)$. This singular value is transformed into a resolution measurement value r_i by using a function similar to that used for the field-of-view measure (see Figure 4-7). The shape of this function can be controlled to favor images within a certain resolution range of the desired

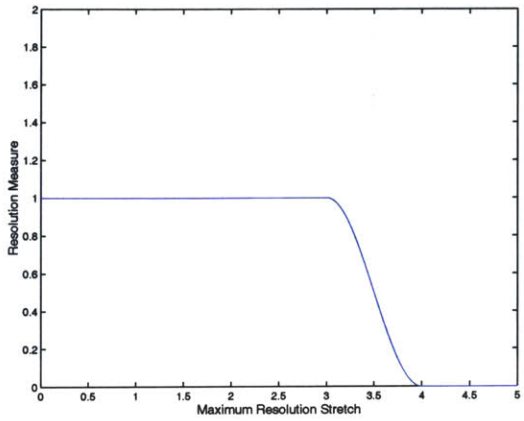


Figure 4-7: An example resolution measure. Cameras that observe the scene point at lower resolution receive smaller measures.

camera. For example, the function shown in Figure 4-7 penalizes cameras whose resolutions differ by more than a factor of three along any dimension. While this may seem like an excessively large range, it is necessary to allow some resolution mismatch in order to extrapolate virtual views away from the observer images.

Given the resolution values for two cameras, the resolution dissimilarity measure is

$$r = 2r_{max}|r_1 - r_2|, \tag{4.8}$$

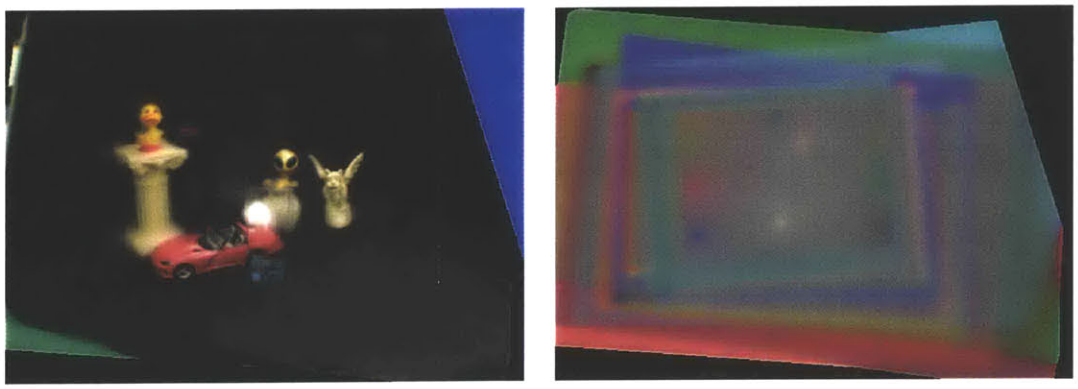
where r_{max} is the constant used in Equation 4.5. When one of the cameras is the desired camera, the dissimilarity is given by

$$r = 2r_{max}|1 - r_2|,$$

in which the resolution value for the desired camera, relative to itself, is one.

4.2.3 Example II

The field-of-view and resolution measures can be demonstrated using the same example image from Section 4.1.3. The output images are produced in exactly the same manner as before, except that the field-of-view and resolution issues are accommodated by adjusting α, γ, and β from Equation 4.7.



(a) (b)

Figure 4-8: (a) An example image using the angle and field-of-view measures ($\alpha = 0.5$ and $\gamma = 1$). (b) A visualization of the blended images.

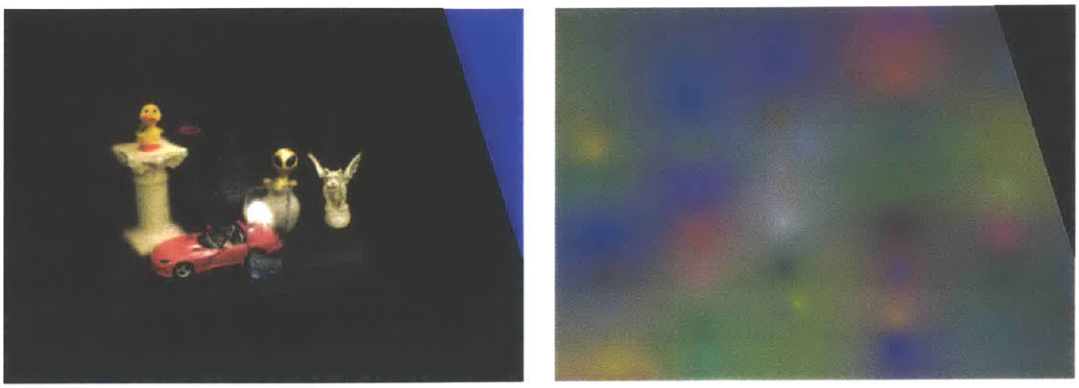
Field-of-View Example

Figure 4-8a shows the image that results from using the field-of-view dissimilarity measure ($\alpha = 0.5$ and $\gamma = 1$). The center of the image is unchanged from that in Figure 4-4, but more of the scene is visible around the perimeter. In particular, the green background, which was invisible in Figure 4-4, is now visible on the left-hand side of the image. Also, more of the brown cloth on the lower right-hand side of the image is visible.

The false color visualization in Figure 4-8b reveals how the image is formed. In the image center, where the angularly close images actually see the proxy, the image blending is the same as in Example I. However, around the perimeter of the image, pixels are taken from images that are not the closest angularly. Instead, the colors are taken from the angularly closest images that actually see the proxy at the reconstruction point. In essence, the algorithm interpolates the output color only from those images that can see the reconstruction point. Of those images, the ones that are closest angularly receive the largest weights.

Resolution Example

The effect of the resolution measure is not as apparent in the image shown in Figure 4-9a, since the resolution-affected areas fall within the black background region. However, the effect is apparent in the false color image shown in Figure 4-9b. In comparison with



(a) (b)

Figure 4-9: (a) An example image using the angle and resolution measures ($\alpha = 0.5$ and $\beta = 1$). (b) A visualization of the blended images.

Figure 4-4b, the resolution sensitive image has a slightly different set of epipoles. In particular, the three epipoles in the upper-right corner of Figure 4-4b are absent from Figure 4-9b. These epipoles correspond to cameras that are farther away from the desired view and thus have a larger difference in resolution. Their influence has been suppressed by the use of the resolution measure. The influence of the resolution measure is much more apparent in an example shown in the next chapter.

4.3 Problems with the Optimal Approaches

The generalized radiance reconstruction procedure described in this chapter satisfies most of the properties of Chapter 3. However, it fails miserably on property #3: real-time performance. The procedure, as described, necessitates the solution of a linear system of hundreds of equations *at each pixel* of the desired output image. Generating one image takes hours on a high-end computer.

In order to meet the real-time constraint, some key changes are needed in the radiance reconstruction procedure. The algorithms described in this chapter coupled with the necessary modifications for real-time performance constitute the heart of the unstructured lumigraph rendering algorithm, which is described in detail in the next chapter.

4.4 Summary

This chapter has presented optimal (in the MMSE sense) approaches to the radiance reconstruction problem. The first approach considers only the angular differences between radiance observations. This problem can be solved if the image correlation function is known. Although difficult to measure, this function can be analytically derived from a falling-leaves statistical image model, which predicts a power-law shape for the image correlation function. This useful result not only reinforces the notion of angle-based view-dependence from Chapter 3, but it gives a concrete form for the relative importance between different radiance observations.

The second optimal approach generalizes the image correlation function to take into account non-ideal effects such as field-of-view limitations and resolution mismatches. As a simplification, it is assumed that the generalized correlation function has the same power-law shape as the original version. This assumption allows the generalized function to be expressed as a simple modification of the original image correlation function. It is shown that this generalized function has the desired behavior (disregarding performance) for image-based rendering applications.

CHAPTER 5

Unstructured Lumigraph Rendering

The generalized radiance reconstruction procedure described in Chapter 4 satisfies most of the desirable properties of an image-based rendering algorithm. As a result, it is very effective at producing high-quality renderings from unstructured collections of images. However, it fails to satisfy one property: real-time performance. Because of this failing, the optimal radiance reconstruction procedure is not suitable for many rendering tasks including virtual reality simulations, games, or interactive scene walk-throughs.

This chapter presents a series of optimizations that can accelerate the radiance reconstruction task to real-time performance. The resulting algorithm, called *unstructured lumigraph rendering*, is the core development of this thesis.

5.0.1 Overview

The chapter begins by describing five different optimizations to the radiance reconstruction procedure. Two of the optimizations deal with simplifying the optimal radiance reconstruction equations. By making some reasonable assumptions, it is possible to reduce the amount of computation considerably. Two other optimizations reduce computation further by limiting the radiance reconstruction to only select pixels and by reducing the number

of cameras that are used in each reconstruction. A final optimization is a technique for blending the images together using graphics hardware.

Following independent descriptions of the five optimizations, the complete unstructured lumigraph rendering algorithm is presented. The operation of the algorithm is demonstrated with a number of diverse examples, which highlight the flexibility and quality of the algorithm.

5.1 Radiance Reconstruction Optimizations

In order to develop a real-time rendering algorithm based on the optimal radiance reconstruction procedure from Chapter 4, the rendering time per frame must be reduced from hours to milliseconds. This section describes a series of optimizations that achieve this goal without sacrificing overall image quality.

5.1.1 Optimization #1: Simplified Similarity Matrix

The single slowest aspect of the optimal radiance reconstruction approach is the need to solve a large system of linear equations (typically hundreds) for each output pixel. Written in matrix notation, the system of equations is

$$\mathbf{Sw} = \mathbf{y},$$

where $\mathbf{S}$ is the generalized similarity matrix, $\mathbf{y}$ is the correlation vector between the unknown radiance sample and the known radiance samples, and $\mathbf{w}$ is the system's solution, the unknown weight vector.

Systems of linear equations such as this one can be solved much more quickly if the matrix $\mathbf{S}$ has a known and sparse structure. In this case, $\mathbf{S}$ has ones on the diagonal since the correlation function $R_I(\theta)$ equals one for $\theta = 0$. The correlation function also falls to zero rapidly for $\theta > 0$. Thus, the matrix $\mathbf{S}$ could be simplified by quantizing small elements to zero, leading to a much sparser system of equations. Typically, it is even reasonable to approximate $\mathbf{S}$ with the identity matrix. This leads to the system of equations,

$$\mathbf{Iw} = \mathbf{y},$$

which has the trivial solution

$$\mathbf{w} = \mathbf{y}.$$

Unstructured lumigraph rendering assumes that the similarity matrix is the identity matrix, which eliminates the need to solve a large system of equations. The algorithm only needs to compute the vector $\mathbf{y}$ and renormalize the result using one of the methods discussed in 4.1.

5.1.2 Optimization #2: Simplified Resolution Measure

Computing the correlation vector $\mathbf{y}$ requires evaluating the generalized correlation function once for each observer camera. The angle, field-of-view, and resolution measures for each camera are required for this computation. Of the three measures, the resolution measure is by far the most expensive to compute. Simplifying this measure results in considerable speed-up.

In the simplest case, the resolution measure can be completely bypassed by setting β to zero. This approach works well for a large class of data sets, especially traditional light fields and lumigraphs in which the images are all arranged at roughly the same distance from the scene. However, in unstructured image collections it is still important to handle resolution mismatches.

In special cases, the resolution measure can be approximated by examining the distances of the cameras from the geometry proxy. This approximation assumes that all of the cameras have the same focal lengths and orientations.

Given a reconstruction point $\mathbf{p}$, its distance to any camera is easily computed. The simplified resolution measure is simply the ratio of the observer camera's distance to virtual camera's distance. This ratio approximates the resolution "stretch" measure from Chapter 4, and it can be mapped through a weight function shaped like that shown in Figure 4-7 to arrive at a simplified resolution measure. Equation 4.8 is used as before.

5.1.3 Optimization #3: Sparse Sampling

Even with simplifications in both the similarity matrix and the resolution measure, computing the weight vector at every pixel of the desired view is still too slow for real-time performance. However, by exploiting the smoothly varying nature of the weight vector, it is possible to drastically reduce the number of pixels at which the weight vector is computed.

As mentioned in Chapter 4, false color rendering is a useful way to visualize the variation of the weight vector across the desired image plane. From the visualization in Figure 4-4b, it is apparent that the weight vector for that particular image varies slowly across the image plane. The only exceptions occur at the epipoles, which stand out as discontinuities in the radiance reconstruction. It turns out that this variation of the weight vector is typical of a large class of synthesized images, and it can be exploited for a performance gain.

Since the weight vector varies slowly across the image plane, it is possible to reconstruct it at every pixel from a small set of weight vectors that are sparsely sampled across the image plane. The only potential problems occur near the epipoles, where the weight vector changes more rapidly. These epipole areas can either be ignored for simplicity, or the sampling density can be increased in these areas.

The approach used in unstructured lumigraph rendering is to sample the weight vector at a small set of selected pixels in the desired view. These sampling locations are then triangulated to form a tessellation of the desired image plane. The weight vector for a pixel inside of a triangle is taken to be a linear combination of the weight vectors at each vertex of the triangle. The coefficients of the linear combination are simply the barycentric coordinates of the pixel within the triangle.

Using this approach, the weight vector for every pixel in the desired view can be interpolated after performing the full computation for just a small percentage of pixels. In practice, good results can be obtained by carrying out the full calculation for less than one percent of the pixels in the desired view. This optimization becomes even more significant in light of optimization #4, which demonstrates how to use graphics hardware to do the

barycentric weight interpolation.

5.1.4 Optimization #4: Use of Graphics Hardware

The triangle-based weight-vector interpolation enables the use of graphics hardware for blending images together. Modern graphics hardware is sophisticated enough to perform both the weighted image blending and the proxy-based pixel correspondence simultaneously.

Image Blending

Image blending is done using the hardware's built-in alpha blending capability. Each triangle vertex can be assigned a transparency value (called the alpha value) from zero to one. Zero indicates the vertex is perfectly transparent, while one means the vertex is completely opaque. The graphics hardware automatically interpolates these alpha values across the face of the triangle. When drawing the triangle, any textures applied to the triangle are modulated with the barycentrically interpolated alpha values. This modulation has the effect of attenuating the triangle's texture in transparent areas while preserving it in opaque areas. The attenuated texture is then added into the frame buffer, which stores the accumulated color values. Many textures can be linearly combined by drawing the same triangle multiple times with different textures and alpha values.

This hardware function maps well onto the unstructured lumigraph rendering problem. Alpha values correspond exactly to radiance reconstruction weights, and the final output image is made by rendering triangles once for each reference image.

As an example, consider rendering a desired view from a set of N images. First, the weight vectors are sampled at M locations, and those locations are triangulated. Each vertex v_i of the resulting triangulation has associated with it a vector of N weights $w_{i,j}$, where $1 \leq i \leq M$ and $1 \leq j \leq N$. The pixels in the interior of a triangle can now be colored by accumulating the results of drawing the triangle N times. Each time, the texture is set to image j, and the alpha values are set to $w_{a,j}$,$w_{b,j}$, and $w_{c,j}$, where v_a,v_b, and v_c are the vertices of the triangle.

Pixel Correspondence

When using graphics hardware, it is still necessary to blend corresponding pixels together. As mentioned in Chapter 4, correspondence is established by a geometric proxy. When the proxy is represented by planar polygons, this correspondence is easily done using the projective texture mapping capabilities of modern graphics hardware.

When a triangle is textured, three-dimensional points on the triangle are mapped into a texture image to determine their colors. Typically, this mapping is a simple affine transformation. However, modern graphics hardware has the ability to use an arbitrary projective transformation (called the *texture transform*) to map points on the triangle to points in the texture image. Thus, by setting the texture transform to be the projection matrix of the observer image, the hardware can determine the colors of three-dimensional points by projecting them into the image of the observer camera. This projection is precisely how corresponding pixels are determined in Chapter 4. Thus, as long as the proxy is represented with polygons, the graphics hardware can efficiently blend corresponding pixels.

5.1.5 Optimization #5: k-Nearest Camera Weighting

The final optimization improves the efficiency of the hardware-accelerated blending algorithm. As described in the previous section, a triangle must be drawn N times, once for each observer image. For typical values of N (> 100), this approach is impractical for a few reasons.

First, the graphics hardware does not have the bandwidth to draw each pixel hundreds of times. Drawing a pixel multiple times is called "overdraw," and even the best hardware can only overdraw every pixel a couple dozen times.

Second, for any given triangle, many of the image contributions and their corresponding vertex weights are very close to zero. In these cases, many textures may not add anything to the output for that particular triangle. Thus, drawing these textures wastes CPU time and contributes nothing to the image.

In light of these issues, the unstructured lumigraph algorithm limits the number of non-zero weights at each vertex to a small, fixed number k that is much less than N. By limiting

the number of non-zero weights in this manner, the algorithm can bound the worst-case computational load. Note that the number k is fixed, and only the cameras with the k largest weights are used even if other cameras have comparably sized weights.

An alternative approach might be to only use cameras whose weights fall above a certain threshold value. The difficulty of this approach lies in choosing a proper threshold. If the threshold is too small, then too many cameras may be used and performance suffers. If the threshold is too large, then it is possible that no cameras may be selected. In fact, when faced with unstructured input data, both of these cases may occur during the rendering of a single image. Thus, the k-nearest camera weighting can be seen as a "variable" threshold technique, in which the threshold is chosen dynamically such that only k cameras are selected.

Using only the k-nearest cameras does introduce some problems with regard to the continuity of the reconstruction. As the weight vector varies over the image plane, the radiance reconstruction experiences discontinuities whenever the set of k-nearest cameras changes. This discontinuity occurs because weights may enter and leave the set of k-nearest with non-zero values. Thus, the algorithm modifies the weights to ensure that they always fall to zero at the $k+1st$ camera. This attenuation of the weight vector is accomplished by "windowing" the correlation function $R_I(\theta)$.

Consider a correlation function, such as that described in Chapter 4, which has the form

$$R_I(\theta) = -A + \frac{B}{\theta^\eta},$$

where A, B, and η are positive constants chosen such that the function is greater than zero for all θ of interest. Given a set of N cameras and a reconstruction point, the angles θ_j (or generalized versions taking into account field-of-view and resolution) can be computed and sorted such that $\theta_j \geq \theta_{j+1}$. In general, the unmodified correlation function is non-zero for $\theta \geq \theta_{k+1}$. In order to force the function to be zero for these values of θ, the correlation function can be modulated with a windowing function that is non-zero at the origin and falls to zero at $\theta = \theta_{k+1}$. One such window function is a hat function:

$$W(\theta) = 1 - \frac{\theta}{\theta_{k+1}}.$$

Using this window function, the modified correlation function becomes

$$\tilde{R}_I(\theta) = W(\theta)R_I(\theta) = -A + \frac{A}{\theta_{k+1}}\theta + B\theta^{-\eta} - \frac{B}{\theta_{k+1}}\theta^{1-\eta}.$$

It has been found that the constants $A = 0$, $B = 1$, and $\eta = 1$ work well in practice, and they lead to a particulary simple form of the modified correlation function:

$$\tilde{R}_I(\theta) = \frac{1}{\theta} - \frac{1}{\theta_{k+1}}. \tag{5.1}$$

This function is simpler to evaluate than Equation 4.4, which makes it appropriate for a real-time implementation. The only issue is dealing with $\tilde{R}_I(0)$, which results in a division by zero. In practice this case is not a problem, since it occurs at epipole locations for which the weight vector can be trivially computed, without evaluating the correlation function.

5.2 Real-Time Unstructured Lumigraph Rendering

The operation of the real-time unstructured lumigraph renderer proceeds as follows. First, the algorithm selects a set of points (i.e., pixels) in the desired view at which to evaluate the weight vector. The blending weights are evaluated at each of these points using the k-nearest camera weighting and Equation 5.1.

Next, the sampling points are triangulated to form a tessellation of the desired view, perhaps adding new sampling points in the process. After triangulation, the windowed weight vectors are computed for all sampling points using the optimized radiance reconstruction procedure. Finally, the resulting triangles are blended together using the graphics hardware's alpha blending and projective texture mapping capabilities. The pseudocode for the algorithm appears in Figure 5-1, and the following sections describe the main procedures in more detail.

5.2.1 Selecting Weight Vector Sample Points

The weight vectors are sampled at a sparse set of points in the desired image plane. These points correspond to desired viewing rays. A number of heuristics are used when selecting which rays to sample. First, the rays to all of the geometric proxy vertices are used. These

```
Clear frame buffer to zero
Select weight vector sample points
Triangulate sample points
for each sample point j do
   for each input image i do
      Evaluate generalized θ_{i,j}
   end for
   Sort θ_{i,j}
   Construct windowed correlation function using k+1st θ_{i,j} and Equation 5.1
   for each input image i do
      Evaluate k non-zero w_{i,j}
   end for
end for
for each input image i do
   Set current texture to image i
   Set current texture transform matrix to P_i
   for each sample point j do
      Set vertex alpha values to w_{i,j}
   end for
   for each triangle t do
      if at least one vertex has non-zero alphas then
         Backproject t onto proxy surface
         Draw t, accumulating result in frame buffer
      end if
   end for
end for
```

Figure 5-1: The pseudocode for the real-time, unstructured lumigraph rendering algorithm.

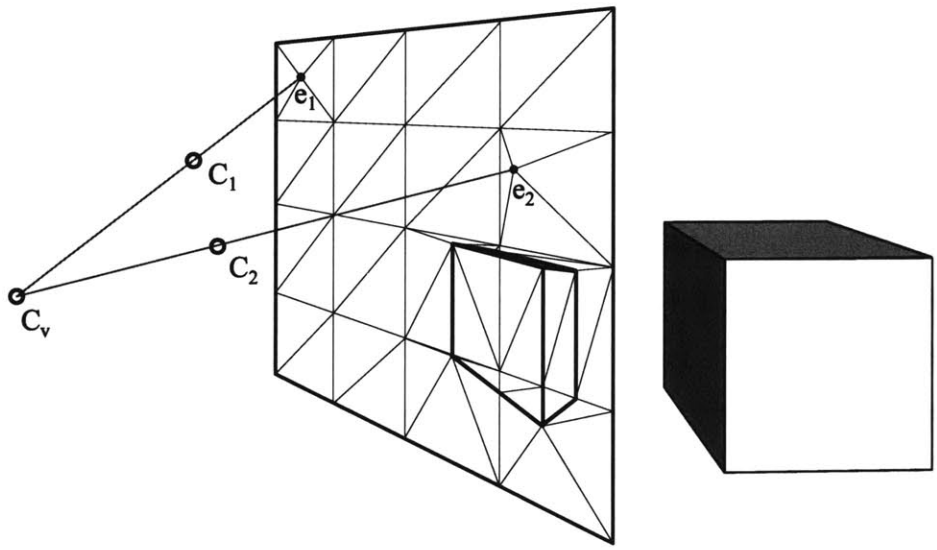


Figure 5-2: The real-time renderer uses the projection of the proxy, the projection of the source camera centers, and a regular grid to triangulate the image plane. In this figure, the proxy is a cube, the camera centers are labeled C_x, and their projections are labeled e_x.

rays are selected so that the tessellation of the sample points does not span more than one planar facet of the geometric proxy (see the next section for more details). This is done because the graphics hardware's projective texture mapping ability only works with a single plane (i.e., it uses a simple planar homography).

Second, to assure epipole consistency, rays to every source camera are also used. These rays correspond to the epipoles, and they should be included in the sampling to maintain exact epipole consistency. However, they can be often omitted with little perceptible loss of image quality since other nearby samples generally reflect the strong influence of the epipole. So, if more performance is needed, the epipole samples can be safely ignored.

Finally, a regular grid of viewing rays is included to obtain a sufficiently dense sample set. These extra samples help capture the interesting spatial variation of the weight vectors. They also contribute samples to areas that are potentially undersampled by the previously selected sample points. In some cases, especially when the proxy contains many polygons, these extra grid samples are unnecessary. Figure 5-2 shows the sample points selected for an unstructured lumigraph with two cameras and a cubical proxy.

Generally, a uniform sampling of the virtual image plane provides the best results.

Thus, the number of grid samples is related to the number and density of proxy and epipole samples. If the proxy and epipole samples are uniformly distributed over the image plane, then few or no grid samples are necessary. On the other hand, if the proxy or epipole samples are very sparsely or non-uniformly distributed, then grid samples should be used to even out the uniformity of the sampling. For large numbers of non-uniformly distributed cameras, the required number of grid samples may be impractical from a performance standpoint. In these cases, the number of grid samples should be chosen to meet the performance requirement. For most of the examples in this thesis, a 16×16 grid of samples is sufficient, although up to 32×32 can be used without much performance degradation.

5.2.2 Triangulating Sample Points

Triangulating the sample points must be done with some care. It is not sufficient simply to construct, for example, a Delaunay triangulation of the selected sample points. This approach fails because of the constraint that the sample point triangles must "see" only one geometry proxy plane. More precisely, the projections of edges from a geometry proxy polygon must not cross any edges in the virtual image plane tessellation. Consider the example shown in Figure 5-2. The projected proxy edges are shown in bold, while the additional tessellation edges are shown in gray. None of the bold edges cross the gray edges (without the addition of a new vertex). Note the bottom edge of the projected cube; an extra vertex has been inserted where it crosses a grid edge.

Further, given a tessellation that conforms to this requirement, it is necessary to know which geometry proxy polygon the sampling triangle sees. Fortunately, a *constrained* Delaunay triangulation can be used to overcome these problems.

Constrained Delaunay Triangulation

A constrained Delaunay triangulation is a Delaunay triangulation in which certain edges are forced to exist in the triangulation. Thus, the problem of spanning multiple geometry polygons can be easily solved by requiring that the projected edges of the geometry proxy appear in the final triangulation. Enforcing this constraint is made possible by including the

projections of the geometry vertices in the set of sample points. In addition, the edges of the regular sample grid can also be included. These grid constraints are useful because regular grids have ambiguous Delaunay triangulations that change randomly and cause temporal artifacts.

Given this set of vertices and constraint edges, the constrained Delaunay triangulation of the sample points is computed using Shewchuk's software [Shewchuk 1996]. This code automatically inserts new vertices at all constraint edge-edge crossings. These new vertices become additional sample points at which the weight vectors must be evaluated.

Associating Image Plane Triangles with Proxy Planes

When rendering the unstructured lumigraph, the vertices of the sample tessellation must be associated with three-dimensional points on the surface of the geometry proxy. The graphics hardware needs the three-dimensional position of the triangle to compute the planar homography for projective texture mapping. However, the triangles from the sample point tessellation are two-dimensional triangles situated in the desired image plane. Before drawing a triangle, it must be backprojected onto the surface of the proxy. This backprojection is easily done using an inverse projection matrix, assuming that the equation of the proxy plane is known. Thus, every triangle in the constrained Delaunay triangulation must be associated with a plane of the geometry proxy.

Fortunately, Shewchuk's software makes this association fairly simple. His implementation allows edges and regions of the input to be labeled. These labels are then propagated to the edges and regions of the output triangulation. By labeling the input geometry constraint edges, it is possible to deduce which triangles of the tessellation correspond to which planes of the proxy. This assocation process is done using a simple graph labeling procedure that runs in time linear in the number of triangles.

Unfortunately, the above labeling process only works if the geometry proxy has unit depth complexity, that is, if every viewing ray from the desired camera intersects the proxy exactly once. In the case that the proxy has depth complexity greater than one, it is necessary to generalize the labeling procedure. Instead of associating each sampling triangle with a single proxy plane, each sampling triangle is associated with a set of proxy planes,

one for each surface of the proxy which is visible through the sampling plane. This generalization changes the labeling procedure in a trivial way, and it still runs in linear time.

5.2.3 Drawing Triangles

Before a triangle can be drawn, it must first be backprojected onto the surface of the proxy. As mentioned in the previous section, this backprojection is done using the plane equation that is associated with the sampling triangle.

After the triangle is backprojected onto the proxy, it is drawn multiple times using different images and sets of alpha values each time. Previous algorithms have used graphics hardware for similar blending strategies, but they typically blend only three images per triangle (i.e., each image is opaque at one vertex and transparent at the others). It is important to note that in unstructured lumigraph rendering, more than three images may be blended across each triangle.

Suppose that there are a total of m unique cameras ($k \leq m \leq 3k$) with non-zero blending weights at the three vertices of a triangle. Then this triangle is rendered m times, using the texture from each of the m cameras. Thus, a triangle is textured with a minimum of k images and potentially with as many as $3k$ images.

If a triangle has more than one proxy plane associated with it, then it is rendered once for each plane. The graphics hardware's z-buffer resolves visibility.

5.3 Examples

In this section, the performance of unstructured lumigraph rendering is demonstrated with a number of examples. First, the example from Chapter 4 is revisited. This example serves to illustrate the impact of the approximations made by the unstructured lumigraph algorithm. The remaining three examples demonstrate the flexibility of the unstructured lumigraph algorithm.

Two of the remaining examples are based on video sequences. The first video example comes from a hand-held video camera. The camera positions are computed using feature-tracking and structure-from-motion techniques. The second video example is from a video

camera mounted on an instrumented robot. Camera positions are derived from the wheel encoders and inertial motion sensors of the robot. In both of the video examples, the proxy is composed of a small number of planes.

The final example demonstrates the algorithm with a more complex geometric proxy. While the image collection has only 36 images, the proxy consists of 500 polygons, which makes up for the lack of images. The algorithm handles this data as easily as the others.

5.3.1 Example #1

The sequence of images in Figure 5-3 illustrates the impact of the optimizations used in the unstructured lumigraph rendering algorithm. The first row of images recalls the results from Chapter 4. These images come from the optimal radiance reconstruction procedure using angle and resolution measures ($\alpha = 1$ and $\beta = 1$) (field-of-view is ignored as none of the ULR optimizations change this measure).

The second row of images in Figure 5-3 shows the result of optimization #1: assuming the similarity matrix is the identity. Both the image and the false-color visualization look similar to the first row. However, it is evident from the false-color image that the epipoles are less distinct. This fuzziness is a direct result of assuming that the similarity matrix is the identity. Using the proper similarity matrix removes the "contamination" (i.e., contributions from other cameras) from the epipole samples. However, even with the epipole contamination, the image quality suffers little.

The third row shows the result of optimization #2: using a simplified resolution measure. Since the simplified resolution measure is less conservative than the homography-based measure, a slightly different range of "resolution stretch" is used for these images. The second row allows resolution differences ranging from a factor of 0.5 to 2, while the third row uses 0.7 to 2. In this case, the difference between the two measures is primarily due to the fact that the desired camera and the observer cameras have different focal lengths, which results in a systematic error in the simplified measure. However, after adjusting for this error, the resulting images are very similar.

The fourth row shows the effect of optimization #5: using only the k-nearest cameras.

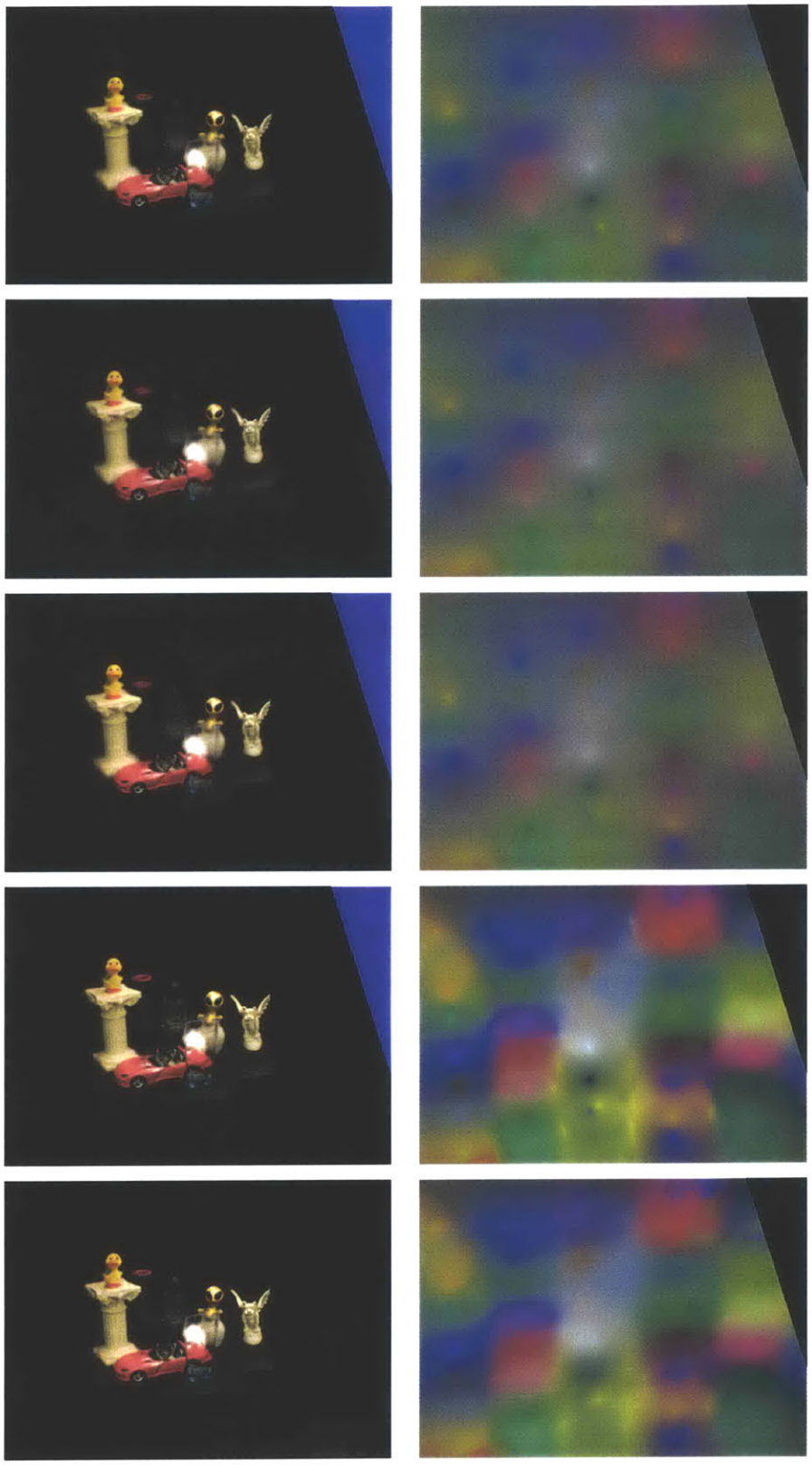


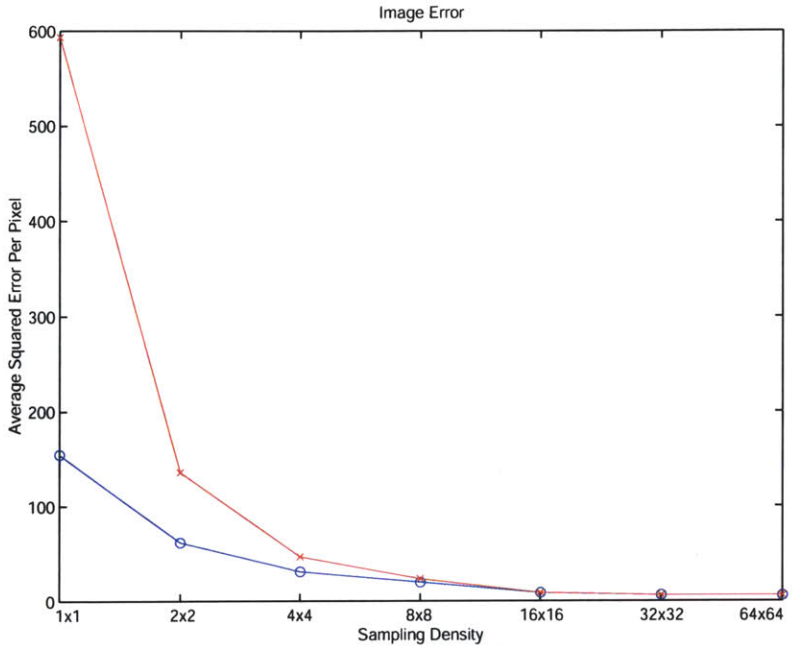
Figure 5-3: A sequence of images showing the effect of the ULR optimizations on image formation. Top to bottom: original, simplified correlation matrix, simplified resolution measure, k-nearest weighting, and sparse sampling.

This optimization has a large effect on the image formation, as shown by the false-color image. The colored regions are much more pronounced, which is a result of fewer cameras contributing to each pixel. In a typical image generated using the optimal approach, up to 30 or 40 cameras can have non-trivial weights at a pixel (i.e., greater than $\frac{1}{256}$, the quantization level of the frame buffer). The images in the fourth row use $k = 5$ images at each pixel. However, the final output image is not appreciably changed, since only the cameras with the largest weights are retained.

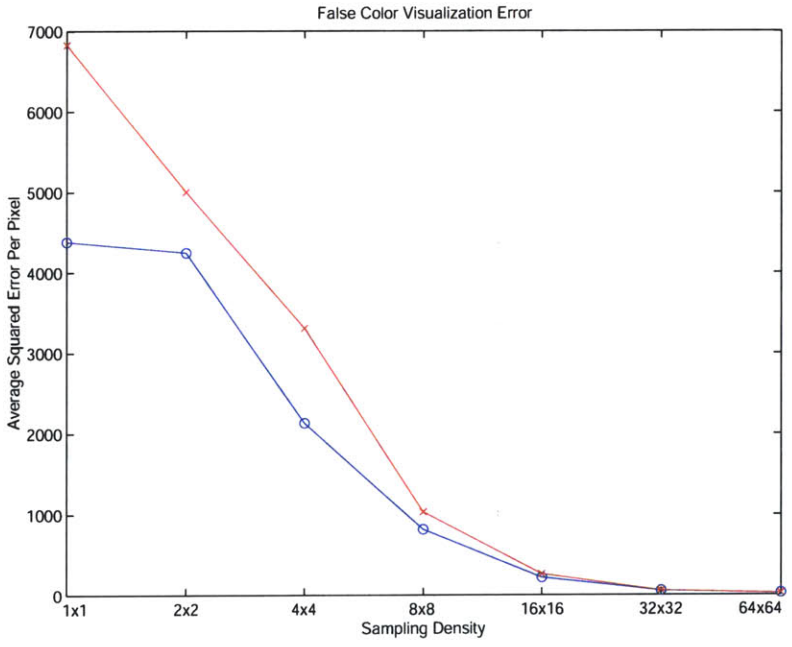
The final row of Figure 5-3 demonstrates optimizations #3 and #4: sparse sampling and hardware rendering. In this case, the weight vectors are sampled on a 25×25 grid covering the desired view. The image and the false-color visualization are generated using graphics hardware. While the false-color visualization shows slight differences in the interpolated weight vectors, the resulting image does not exhibit noticeable artifacts. This fact is remarkable, as the length of time to generate the image has been reduced from 5 hours to less than 33 milliseconds through the use of the 5 optimizations.

The choice of sampling density involves a trade-off between rendering performance and reconstruction fidelity. A very small number of samples results in a very fast rendering time, but it may also negatively impact the image quality. On the other hand, an extremely large number of samples may unnecessarily slow the algorithm with little benefit in quality.

The impact of sampling density on image quality is shown in the two plots in Figure 5-4. Figure 5-4a shows the error in the sparsely sampled image as compared to the ray-traced image (i.e., one sample per pixel). The error is expressed as the average squared error per pixel, which is the squared error of each color channel summed over all pixels and divided by the total number of pixels. Two error curves are shown; one that includes epipole samples (blue curve with circles) and one that does not include epipole samples (red curve with crosses). Clearly, there is more error at lower sampling densities, and the error seems to level off at about 16×16 samples. Using epipole samples always improves the error (this is always the case, since the epipole samples are additional samples), but does not appreciably improve the error after 8×8 samples. Thus, for this configuration of images, a 16×16 sampling density, without epipole samples, is sufficient for quality rendering. The system remains interactive up to 32×32 samples, so a 16×16 grid results



(a)



(b)

Figure 5-4: Plots of the error caused by using a sparse sampling of image reconstruction weights. (a) The error in the actual image. The blue curve with circles shows error with epipole sampling, and the red curve with crosses shows without. (b) The error in the false-color visualization.The blue curve with circles shows error with epipole sampling, and the red curve with crosses shows without.

in quality, interactive rendering.

The plot in Figure 5-4b shows the same error curves applied to false-color visualizations. Although the false-color visualization is not the desired output of the system, it is useful for investigating the "worst-case" performance of the rendering. Some of the error in Figure 5-4 is hidden by the fact that many regions in the images are almost uniform in color across many cameras. Thus, reconstruction errors may be masked because an incorrectly interpolated pixel happens to be the same color as the correctly interpolated pixel. Since the false-color visualization assigns unique colors to each input image, this error hiding is much less likely to occur. As expected, the error in Figure 5-4b is higher than that in Figure 5-4a. In this case, the errors level off and coincide at 32×32 samples, which also allows for interactive rendering.

Of course, this analysis is only applicable to this particular collection of images viewed from this particular virtual viewpoint. However, for all of the image collections considered in this thesis, it has been found that a 16×16 sampling grid provides good quality images at interactive rendering rates.

5.3.2 Example #2

The second example comes from a video sequence shot with a hand-held video camera at the Staples center in Los Angeles. The camera intrinsics and extrinsics are determined by tracking image features and by solving a structure-from-motion problem as described in Section 2.5. The camera and proxy configurations are shown in the first row of Figure 5-5. In this example, the proxy consists of two planar quadrilaterals. One corresponds to the ground plane, and the other roughly aligns with the background objects. Each column of Figure 5-5 represents a different desired camera. The desired cameras are shown in red in the first row of images.

The second row of images shows the output of the unstructured lumigraph rendering algorithm. The ragged tops and bottoms of the images reveal the rhythmic camera motion, which corresponds to the walking gait of the person holding the camera. The rendered images themselves, of course, do not exhibit this motion.

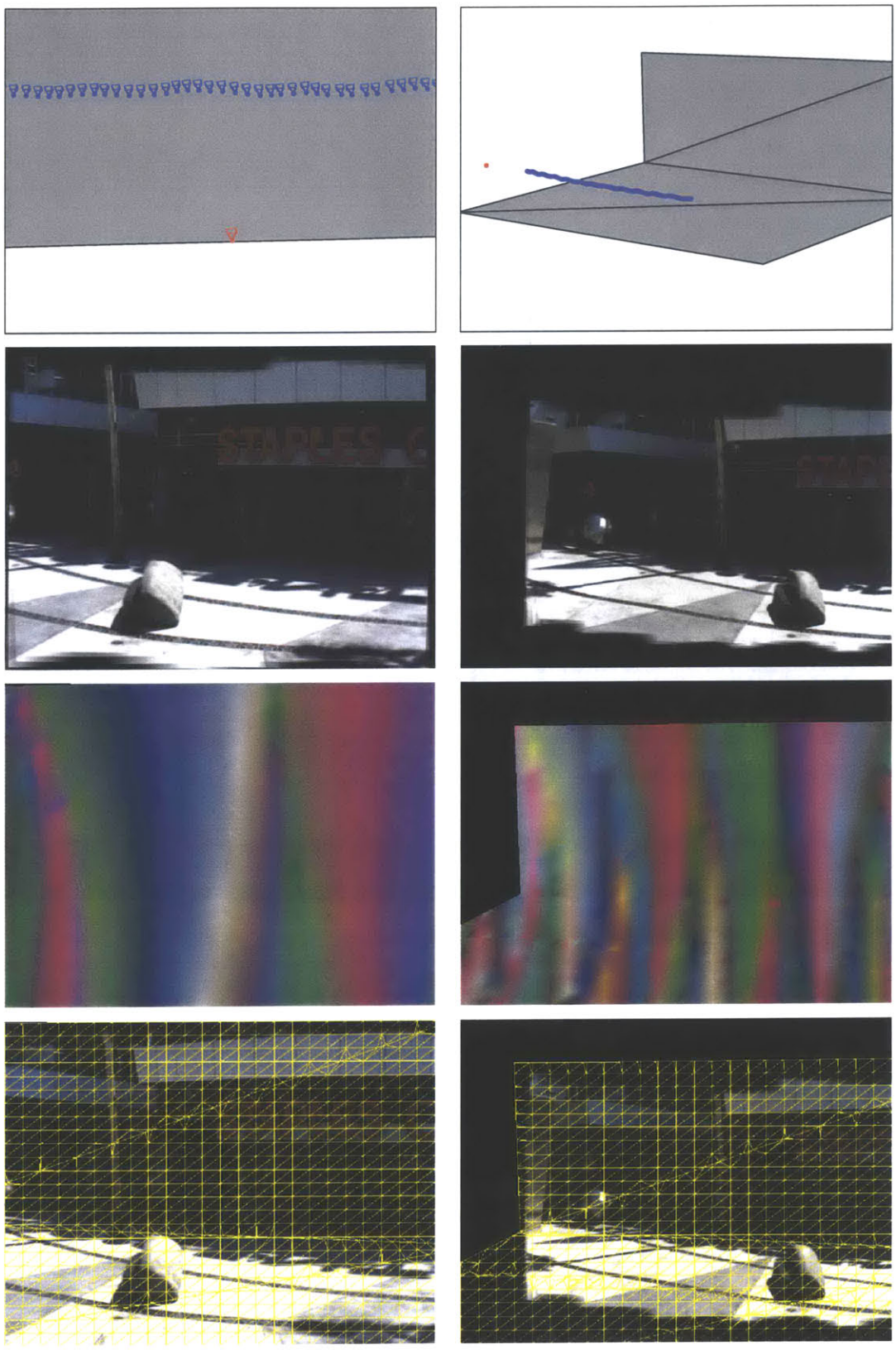


Figure 5-5: Images rendered from a video taken at the Staples center in Los Angeles. Top to bottom: data configuration, rendered images, false-color visualizations, and image-plane tessellations.

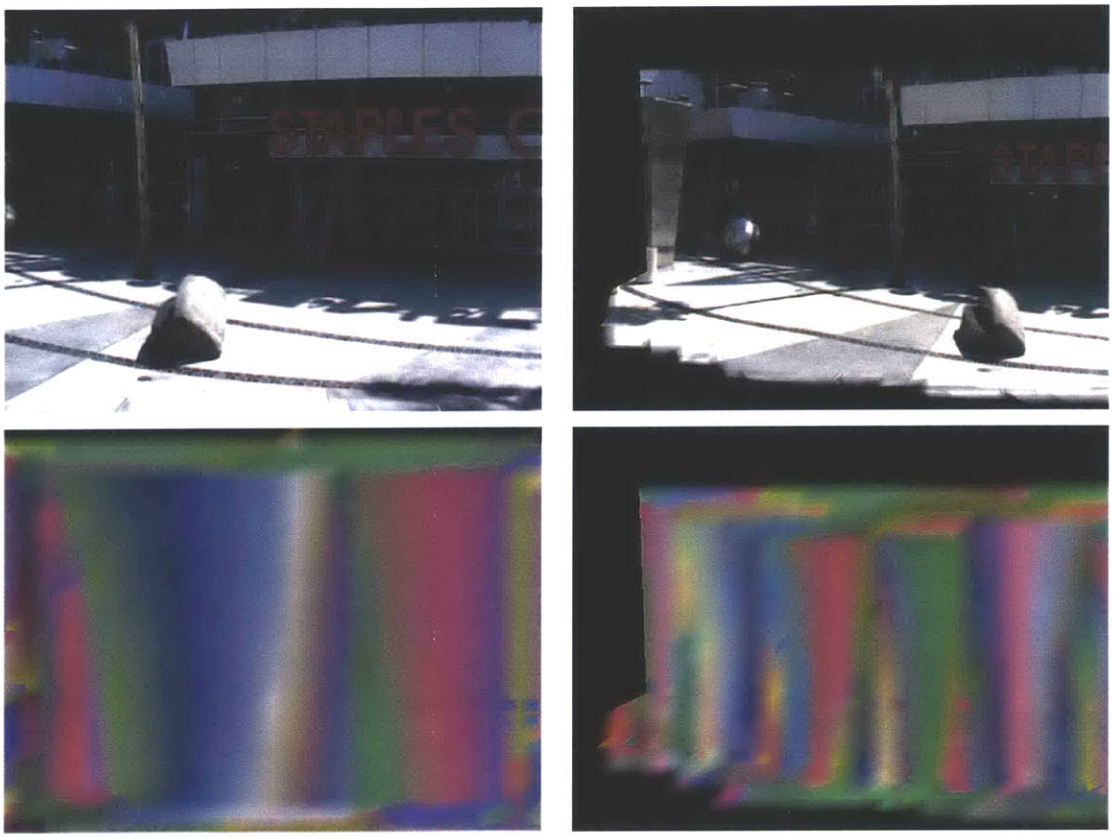


Figure 5-6: Images showing the effect of field-of-view consideration. Rendered images (top) and false-color visualizations (bottom).

The third rows of images shows the tessellation of the image plane. The shape of the proxy is clear in the image on the right. A 25×25 grid of samples is used for these images, although a smaller grid results in similar quality images.

The fourth row of Figure 5-5 shows the false-color visualizations for the two desired views. The view on the left is closer to the original camera path, which results in fewer cameras contributing to the final image. The view on the right is farther from the original path, so more cameras contribute to the image, although each camera contributes a smaller portion.

Figure 5-6 shows the effect of considering field-of-view. Here the images are rendered again, although with $\gamma = 1$, which weights cameras less if they can not see a portion of the scene. For the view on the left, almost the entire view is filled. Regions that are invisible in Figure 5-5 are filled in with angularly close cameras that observe the region. For the view

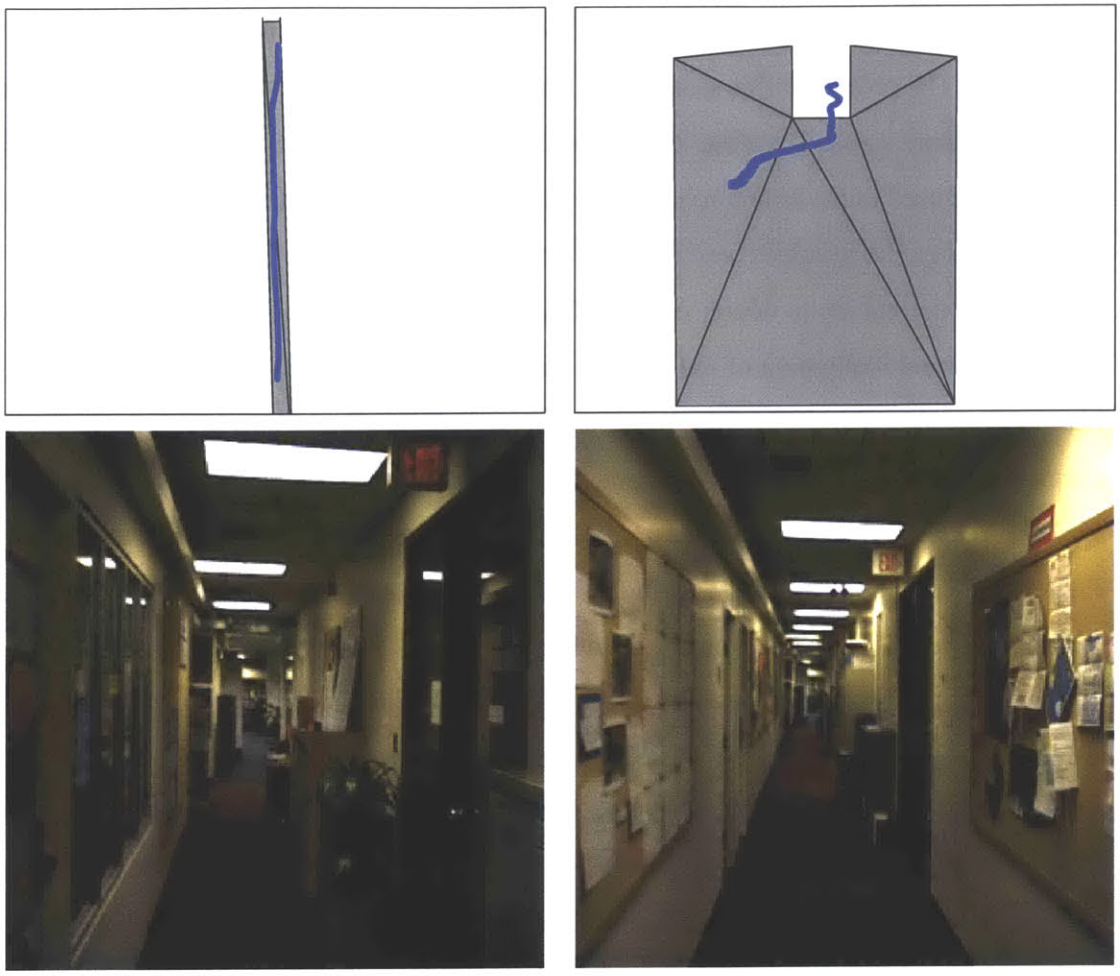


Figure 5-7: Camera configuration geometry proxy (top), and example images (bottom) from the hallway example. Three planes (front, back, and top) have been removed from the proxy for visualization purposes.

on the right, the image is expanded somewhat, although there are still many regions that are not seen by any camera.

The images in this example all view the proxy at approximately the same distance, so including a resolution measure does not change the image quality.

5.3.3 Example #3

This example is constructed from a long video sequence in which the camera moves forward down a hallway. The camera is mounted on an instrumented robot that records its position as it moves. Such forward camera motion is not handled well by previous image-

based rendering techniques, but it is processed by the ULR algorithm with no special considerations. The proxy for this scene is a six sided rectangular tube that is roughly aligned with the hallway walls. Since the video contains many frames (1096), only every 5^{th} frame (219 total) is used in the lumigraph. This reduction in the number of images allows the data to fit entirely on the graphics card for best performance.

None of the cabinets, doors, or other features in the hallway are explicitly modeled. However, virtual navigation of the hallway gives the impression that the hallway is populated with actual three-dimensional objects. The camera configurations and some example camera images from the image collection are shown in Figure 5-7.

Two rendered images and their corresponding false-color visualizations are shown in Figure 5-8. Because of the unusual input camera configuration, the false-color visualization looks unlike any blending pattern seen in other algorithms. It consists of two sets of circular regions that meet in the middle of the image. One of the regions corresponds to cameras that are in front of the desired view, while the other region corresponds to cameras behind the desired view (in this case, the desired view is in the middle of the hallway). Smaller regions belong to cameras that are physically farther away from the desired view. Their distance causes the angular measure to weight these cameras less, except around the epipoles, which are located roughly at the centers of the two circular regions. As the desired camera moves down the hallway, one region "expands" while the other "contracts."

This example also demonstrates the need for both field-of-view and resolution consideration. Figure 5-9 shows an extreme view of the hallway from a viewpoint at which it is physically impossible to place a camera. When the field-of-view is not considered, then a large portion of the image is invisible (top row). When field-of-view is considered, then the invisible regions are filled in from near-by images.

The top row of Figure 5-10 shows the types of blurring artifacts that can occur if resolution is ignored. The second row shows the result of using the simple resolution measure ($\beta = 1$). Low resolution images are penalized, and the wall of the hallway appears much sharper, with a possible loss of view-dependence where the proxy is poor. From the false-color visualization, it is apparent that the resolution-sensitive rendering uses fewer images on the left hand side of the image, which is where the original rendering had most problems

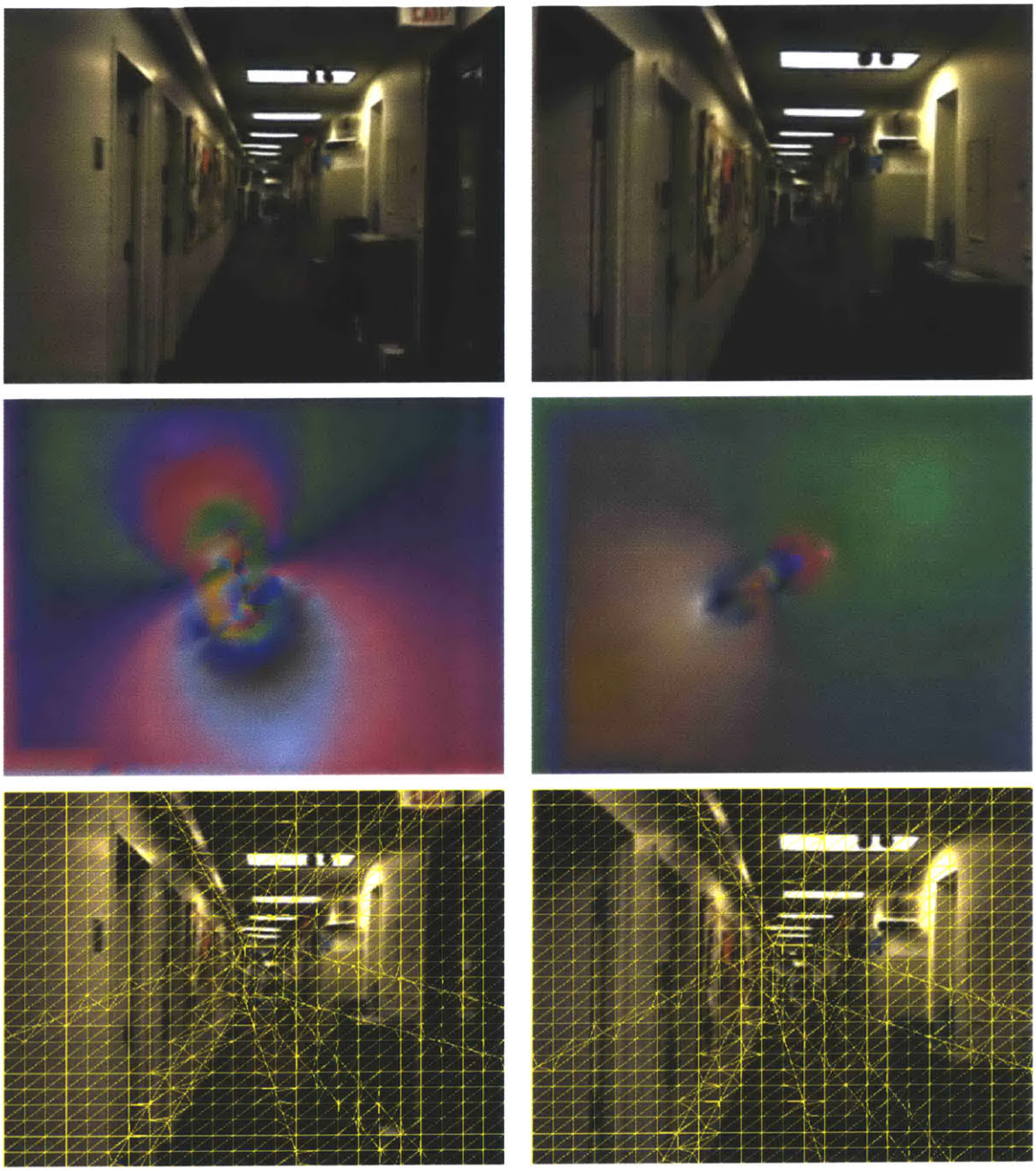


Figure 5-8: Two rendered images from a hallway at MIT (top), the false-color visualizations for each (middle), and the image plane tessellations (bottom).

with excessive blurring. In this case, the attenuated cameras are too far behind the desired view.

As mentioned previously, the hallway lumigraph uses about 219 frames from an original collection of 1096 frames. Calibration information is available for all 1096 frames, which

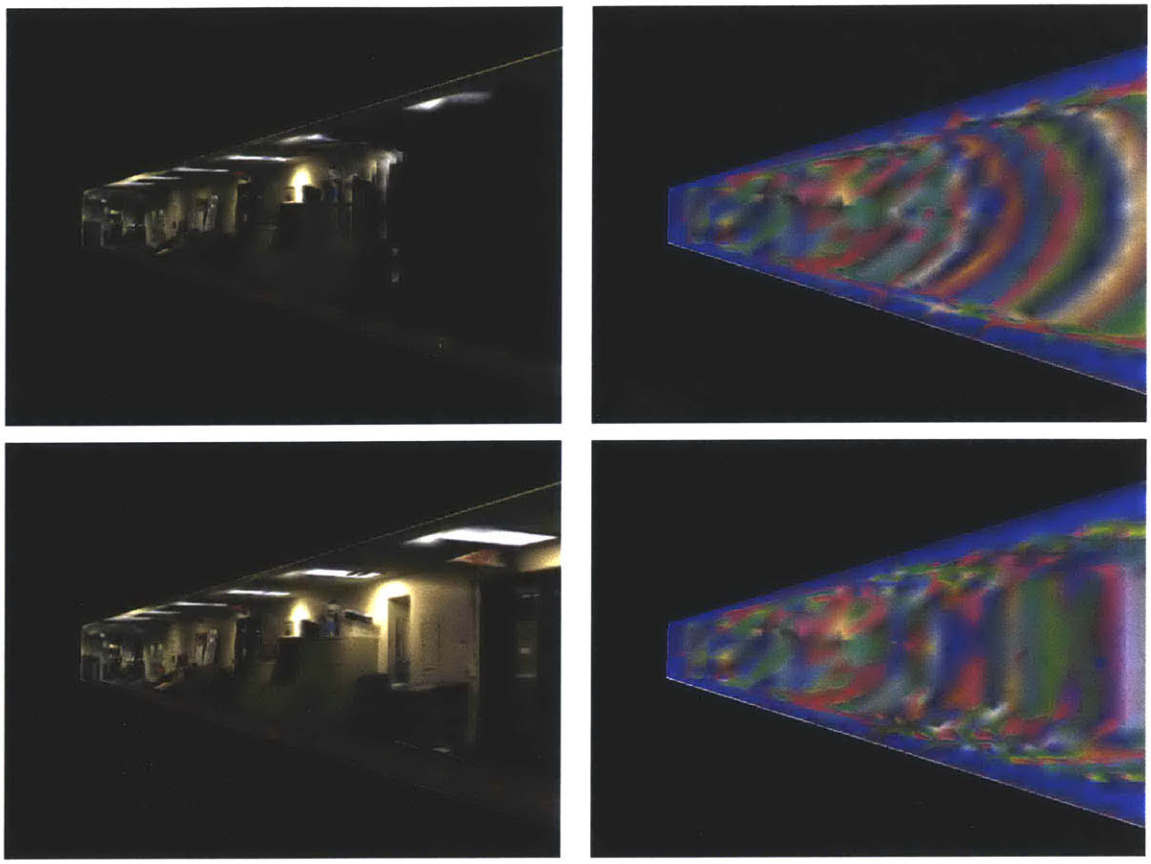


Figure 5-9: Images demonstrating the impact of field-of-view on the hallway example. The top images ignore field-of-view, resulting in black areas where cameras do not see anything.

allows for a comparison of images produced by the rendering algorithm to actual images that are not within the lumigraph. Figure 5-11 shows a comparison for three frames from the original sequence but not used in the lumigraph rendering. The first column shows the original frames, the second column shows the unstructured lumigraph rendering, and the third column shows a difference image.

Qualitatively, the images compare very favorably, except that the virtual images are less sharp than the originals. The difference images reveal two common errors: errors around the edges of objects and errors near specular reflections and highlights. The first type of error is caused by the approximate geometry proxy. The second type of error is due to angular differences between the original view and the views in the lumigraph. A denser set of views in the lumigraph would better reproduce view-dependent effects such as highlights and reflections.

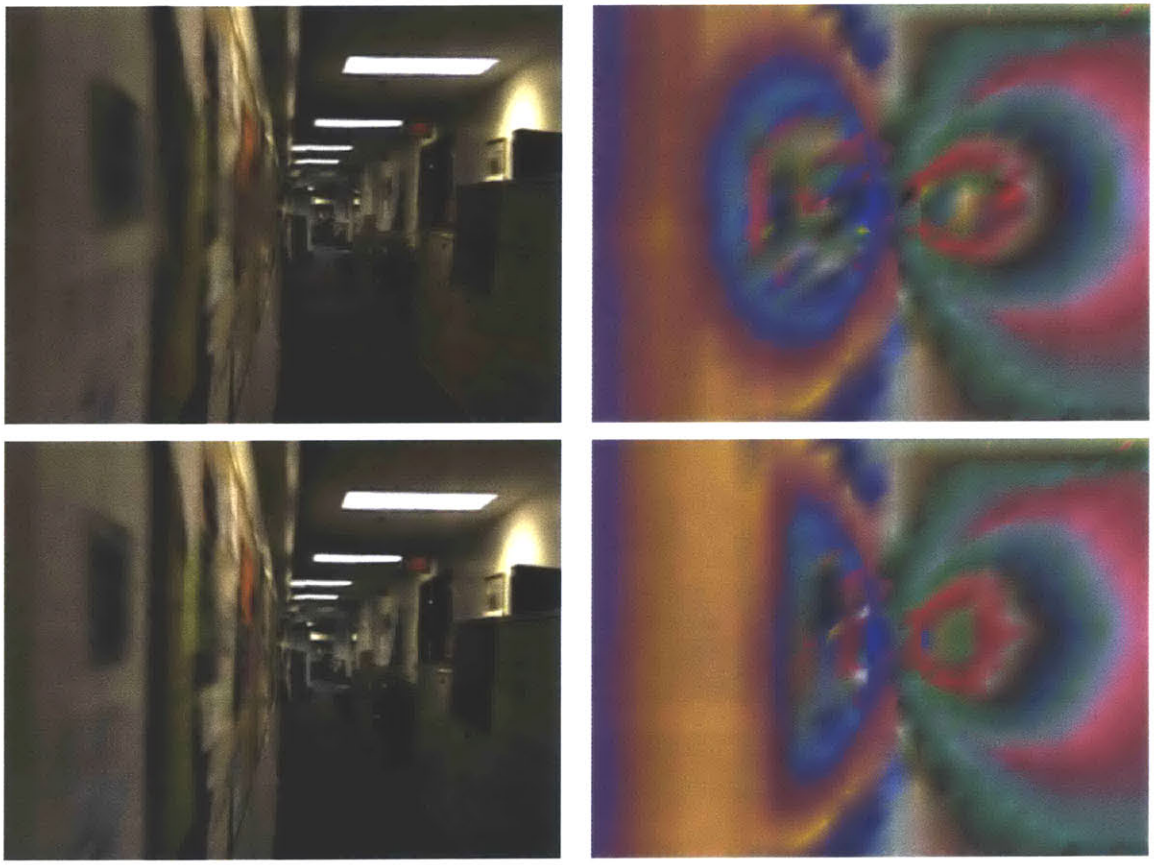


Figure 5-10: Images demonstrating the impact of resolution on the hallway example. Note the orange paper on the left wall.

Figure 5-11: A comparison of original images (first column) to rendered images (second column). The third column shows the absolute value of differences between the images in the first two columns.

5.3.4 Example #4

While the previous examples primarily occupy the light field end of the image-based rendering spectrum, this example demonstrates the view-dependent texture mapping aspects of the algorithm. This unstructured lumigraph consists of only 36 images of a car and a 500 face polygonal geometric proxy. The images are arranged in 10 degree increments along a circle around the car. The images are from an "Exterior Surround Video" (similar to a QuicktimeVR object) database found on the carpoint.msn.com website.

The original images have no calibration information. Instead, it is simply assumed that the cameras are on a perfect circle looking inward. Using this assumption, the proxy is made by constructing a rough visual hull model of the car. Since the true focal lengths are unknown, the camera focal lengths are optimized by hand to give the best reconstruction. The model is simplified to 500 faces while maintaining the hull property using the progressive mesh variation described in [Sander et al. 2000].

Figure 5-12 shows two rendered views of the car. The top row shows the input camera configurations, the proxy, and the desired views in red. The second row shows the rendered virtual views, and the third row shows the false-color visualizations. In this case, since there are so few images, each desired image is composed of a small number (three or four) of input images. The fourth row shows the tessellation of the geometry proxy. In this example, the proxy is sufficiently complex and the number of cameras is sufficiently small, so that the camera weighting need only be sampled at the mesh vertices. This reduction in the number of sampling locations increases the efficiency of the algorithm.

Note that the geometric proxy is significantly larger than the actual car, and it also has noticeable polygonal silhouettes. However, when rendered using the ULR algorithm, the rough shape of the proxy is largely hidden. In particular, the silhouettes of the rendered car are determined by the images and not the proxy, resulting in a smooth contour (see Figure 5-13).

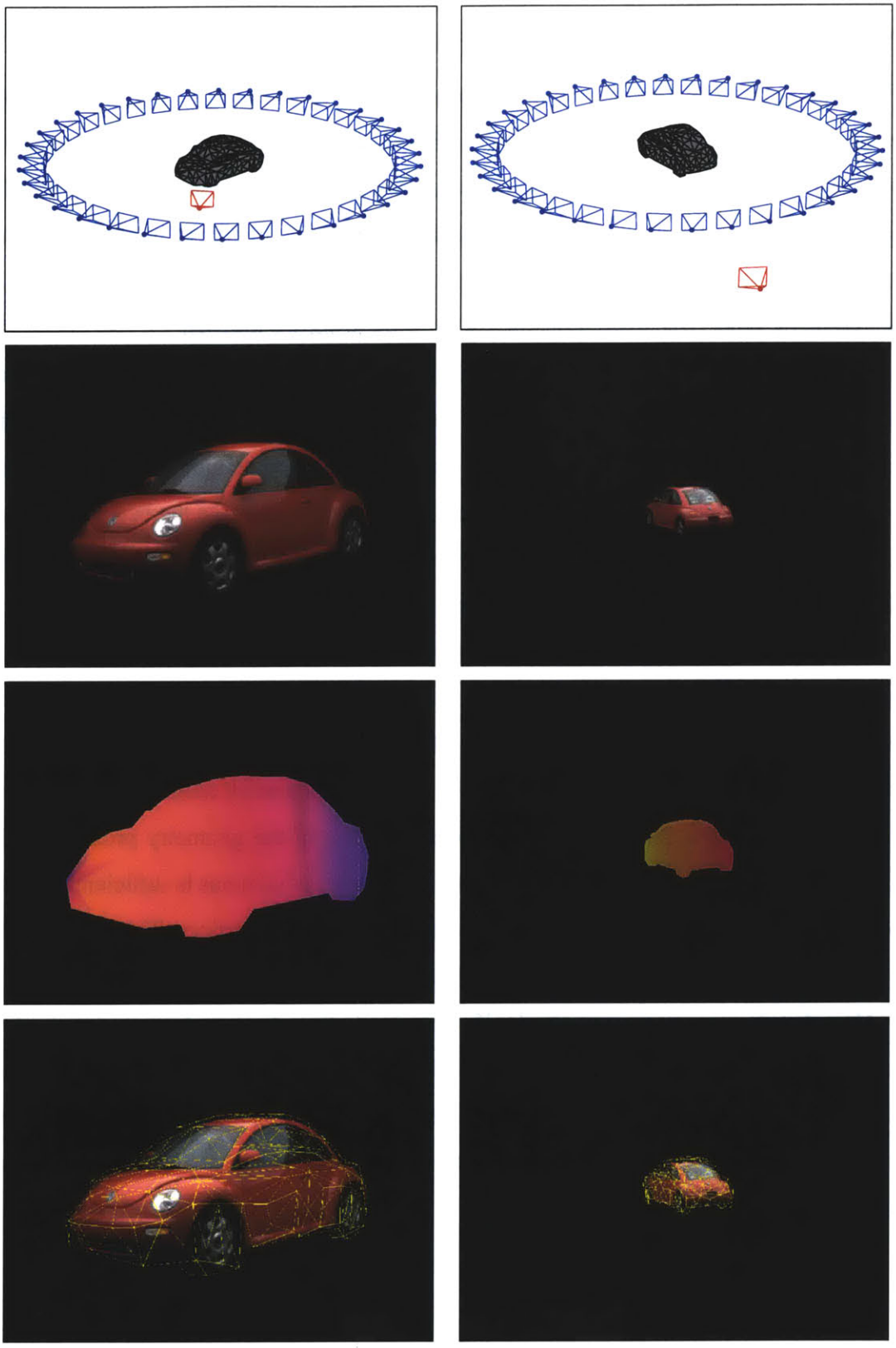


Figure 5-12: Virtual images from the car example. Top to bottom: data configuration, rendered views, false-color visualizations, and image-plane tessellations

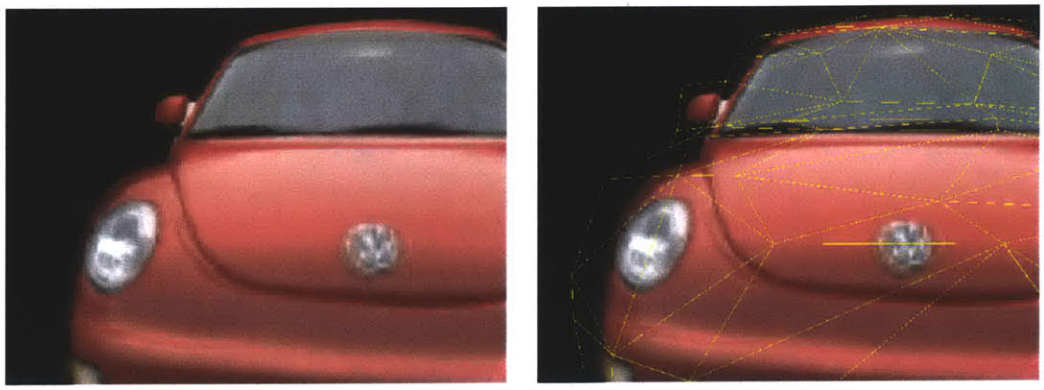


Figure 5-13: A closeup showing how the car silhouette is determined by the images (left) and not the geometry (right).

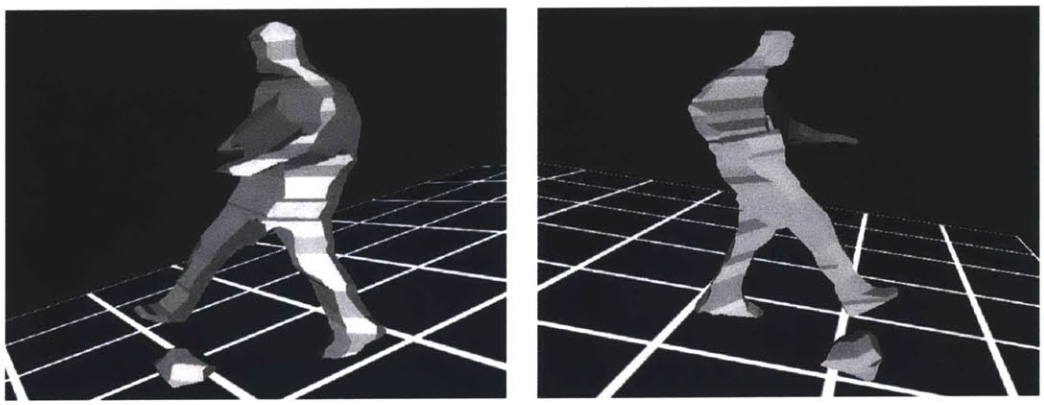
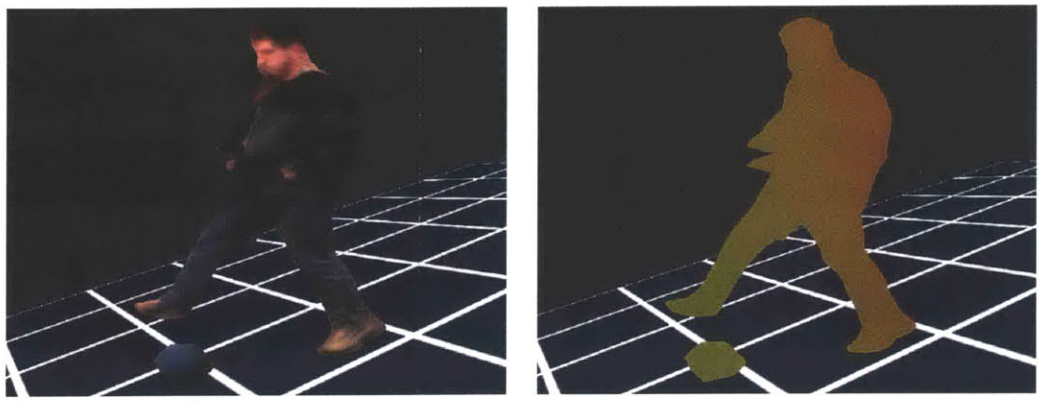


Figure 5-14: Two views of a flat-shaded proxy constructed with the polyhedral visual hull system.

5.3.5 Example #5

This example also exhibits a VDTM application of unstructured lumigraph rendering. In this case, the proxy is a coarse geometric model of a person (see Figure 5-14). This model is a snapshot from the polyhedral visual hull system [Matusik et al. 2001], an interactive visualization system that constructs and renders three-dimensional models of objects in real-time. The system works by constructing a three-dimensional model from multiple silhouettes of an object. The silhouettes are obtained by segmenting four video streams.

The system uses the unstructured lumigraph rendering algorithm to provide a fast, view-dependent visualization of the three-dimensional models. The lumigraphs have reasonably good proxy geometry, but have only four images roughly arranged in a 180 degree semi-circle around the object. Because of the highly tessellated geometry and small number of



(a) (b)

Figure 5-15: (a) The textured polyhedral visual hull. The unstructured lumigraph contains only 4 images. (b) The associated false-color visualization. Note that only two images (colored red and green) contribute most of the textures.

images, the weight vectors are only sampled at the proxy vertices. The textured visual hull and its associated false color visualization are shown in Figure 5-15.

5.4 Summary

This chapter has presented the unstructured lumigraph rendering algorithm, a real-time approximation of the radiance reconstruction routines described in Chapter 4. By exploiting five optimizations, the running time of the algorithm is reduced from 5 hours per image to less than 33 milliseconds per image (i.e., more than 30 frames per second).

The algorithm is demonstrated with a number of examples. Some of the examples use hand-held video, which exploits the ability of the algorithm to handle unstructured inputs. The hallway example uses images that exhibit forward motion, a common configuration of input images that had never been effectively used before in an image-based rendering algorithm. Unstructured lumigraph rendering handles this case because it is based on an angle-based view-dependent rendering strategy that properly handles large numbers of images.

Other examples use higher-fidelity geometry proxies with fewer images, which demonstrate the algorithm's ability to accommodate a wide range of inputs. One of these examples

comes from the polyhedral visual hull system, which is a real-time rendering system that requires real-time performance.

CHAPTER 6

Non-Metric Unstructured Lumigraph Rendering

The unstructured lumigraph rendering algorithm outlined in the previous two chapters allows for rendering from a wide variety of image collections. There are no restrictions on the arrangement of input cameras, nor on the complexity of scene geometry. However, there is one implicit restriction: the input cameras must be strongly calibrated. In other words, the cameras' intrinsic parameters are known, and a Euclidean representation of the scene (cameras and geometry) is available. Relaxing this restriction is the focus of this chapter.

Extending unstructured lumigraph rendering to non-metric image collections widens the algorithm's applicability. In many cases, it is very difficult to obtain a strong calibration. For example, the camera and lens may not be available, which makes calibrating the camera focal length difficult. Many self-calibration techniques (i.e., calibrating the camera from arbitrary images) have been proposed [Faugeras et al. 1992; Pollefeys et al. 1999], but these techniques are fragile and do not work in all situations. The car example from Chapter 5 presents another approach: guess the calibration. However, guessing is very difficult in situations in which the camera parameters may be changing, such as when the camera zooms.

In light of these problems, an algorithm that works directly with non-metric scene rep-

resentations is very useful.

6.0.1 Overview

The first half of this chapter enumerates the aspects of unstructured lumigraph rendering that need to be modified in order to accommodate non-metric data. It turns out that there are four aspects of the ULR algorithm that make Euclidean assumptions. Each of these problem areas are analyzed, and non-metric substitutions are proposed. The resulting non-metric algorithm satisfies almost all of the desired properties outlined in Chapter 3, with some noted exceptions.

The second half of the chapter applies the non-metric ULR algorithm to a common problem: video stabilization. It is shown that non-metric ULR is a natural solution to this problem, since the weaknesses of the non-metric approach are offset by the specific needs of the video stabilization problem. The validity of the approach is demonstrated with three examples.

6.1 Problems with Non-Metric Rendering

Recall that with a non-Euclidean (non-metric) scene reconstruction, distances and angles between points and lines are not meaningful. Non-metric distances and angles certainly do not reflect the "true" (i.e., Euclidean) values, but they also do not preserve relative magnitudes or ordering.

The assumption of a Euclidean reconstruction enters into the unstructured lumigraph rendering algorithm in multiple places. These places are described in more detail in the following sections.

6.1.1 Angle Measure

The image correlation function from Section 4.1.1 is indexed by the angle between two rays. This angle is the natural measure of angular difference between rays, but some other measure is needed for a non-metric algorithm.

6.1.2 Resolution Measure

The simplified resolution measure from Section 5.1.2 is based on Euclidean distances between cameras. Again, these distances can not be used, so some other resolution measure must be employed.

6.1.3 Geometry Proxy

The geometry proxy used in unstructured lumigraph rendering typically consists of a small set of planes that are "close" to the scene structure. Since the idea of closeness assumes Euclidean distance, some other way of specifying the scene proxy is needed.

6.1.4 Navigation

Unstructured lumigraph rendering specifies the desired camera in terms of standard computer graphics techniques, which make Euclidean assumptions. In a Euclidean framework, camera positions can be specified by setting a translation vector, a rotation matrix, and a field-of-view. In a non-metric setting, the unknown intrinsic and extrinsic camera parameters can not be decoupled in this way. Camera position, orientation, and field-of-view must be specified differently, or constrained in such a way as to be realizable. This requirement is the most difficult to satisfy in a non-metric setting.

6.2 Non-Metric Modifications to ULR

A non-metric ULR algorithm can be obtained by modifying the Euclidean ULR algorithm so that the angle measure, resolution measure, proxy geometry, and navigation are specified in a non-metric setting.

Begin by assuming that a projective reconstruction of a scene is known, and that it has been obtained from a set of corresponding point features. The scene is represented as a collection of 3×4 projection matrices $\mathbf{P}_i$ and 4×1 structure points $\mathbf{M}_j$, which project onto the corresponding point features $\mathbf{m}_{ij}$. If the scene has not been obtained from point features, then assume that corresponding point features have been selected and that the appropriate

structure points have been derived from these features (if the features truly correspond, then this computation is a trivial matter).

The projective reconstruction specifies only that

$$\mathbf{m}_{ij} \doteq \mathbf{P}_i \mathbf{M}_j,$$

where $\mathbf{m}_{ij}$ is a point feature (represented $(u, v, 1)^T$) in image I_i, and $\doteq$ denotes equality up to scale. Note that a projective reconstruction is uniquely defined up to an arbitrary projective transformation $\mathbf{T}$. That is, transforming the projection matrices and the structure points with $\mathbf{T}$ results in an equivalent projective reconstruction:

$$\mathbf{m}_{ij} \doteq (\mathbf{P}_i \mathbf{T}^{-1})(\mathbf{T}\mathbf{M}_j) = \mathbf{P}'_i \mathbf{M}'_j.$$

This unknown projective transformation makes it impossible to compare angles and distances in the non-metric space. However, it is clear that the *projections* of quantities into the image space are unaffected by $\mathbf{T}$. Thus, image-to-image transformations, such as planar homographies, are unaffected by the projective distortion introduced by $\mathbf{T}$. In light of this observation, non-metric unstructured lumigraph rendering uses direct image-to-image transforms and image-space measures to avoid measuring quantities in the projective space. However, it does so at the cost of some of the desired properties from Chapter 3.

6.2.1 Angle Measure

For each individual pixel in the desired view, the ULR algorithm blends between corresponding pixels from multiple reference views. Given a set of corresponding pixels $\mathbf{p}_i$ in the reference views, the final color of the pixel $\mathbf{p}_{des}$ in the desired view is computed as a weighted average of these reference pixel colors. The colors are inversely weighted based on the angular difference between the reference viewing rays and the desired viewing ray (see Figure 6-1). In fact, the actual angles are not that important; rather, it is the relationships between angles (e.g., which angle is bigger than another) that matter most.

Consider measuring relative angle sizes using the following measure defined in image space. Compute the vanishing points $\mathbf{v}_i$ of all the corresponding observer rays as seen in the desired view. The distance $d_i = \|\mathbf{v}_i - \mathbf{p}_{des}\|$ is then a relative measure of the angular distance

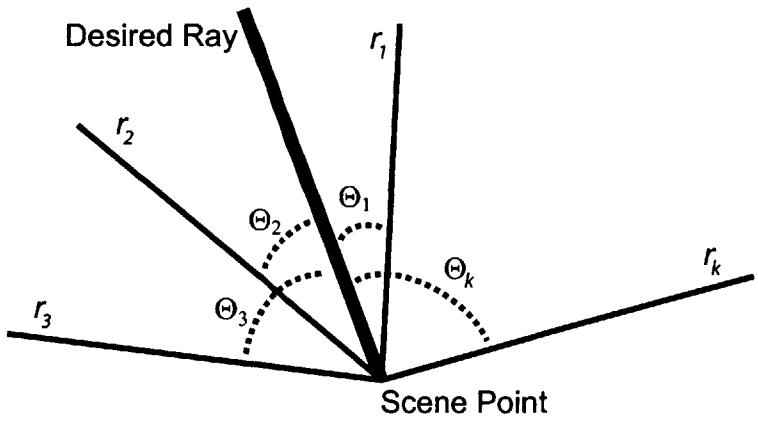


Figure 6-1: The Euclidean ULR algorithm uses the angular distance from the desired ray.

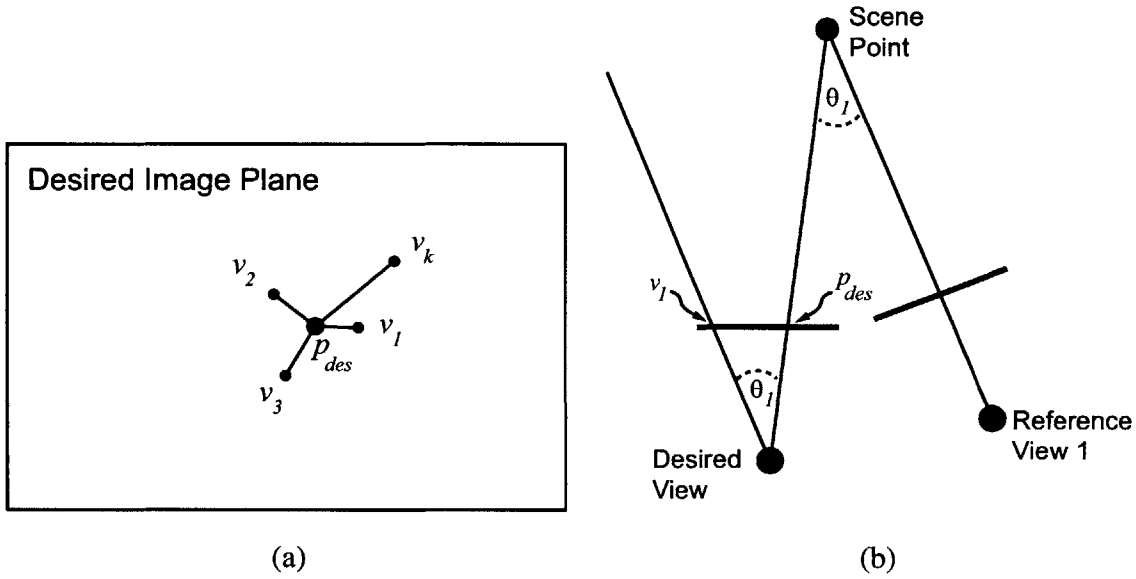
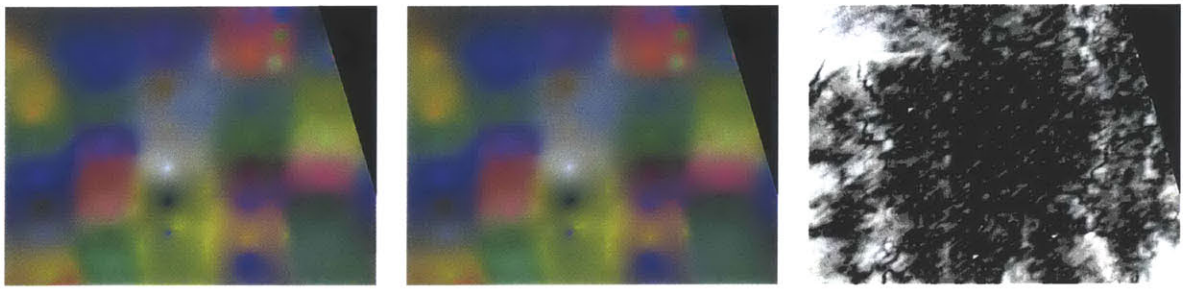


Figure 6-2: Vanishing point distance measurement. (a) An alternative angle measure is the distance of the vanishing points $\mathbf{v}_i$ from the projection of the scene point $\mathbf{p}_{des}$. (b) The vanishing point shows which ray in the desired view is parallel to the observer ray.

between the desired viewing ray and the reference viewing ray (see Figure 6-2a). This measure behaves enough like the true angle (i.e., smaller angles have smaller measures) to compute interpolation weights. Note that in the case of known camera intrinsics, the vanishing point distance can be trivially converted into an actual angle measurement. The



(a) (b) (c)

Figure 6-3: A comparison of false-color visualizations that use true angles (a), and vanishing point approximations (c). The differences between (a) and (b) are shown greatly exaggerated in (c).

diagram in Figure 6-2b demonstrates how this construction works. The vanishing point tells which ray in the desired view is parallel to the observed ray. If the observed ray is parallel to the desired ray (i.e., they are the same ray) then the vanishing point equals $\mathbf{p}_{des}$ and the distance is zero.

The validity of this approach is demonstrated in Figure 6-3. This figure shows two false-color visualizations for Example #1 from Chapter 5. The visualization on the left uses the standard angle measure (recall that a Euclidean reconstruction is available for these images), while the visualization in the middle uses the proposed vanishing point measure. The two images are slightly different, particularly around the image borders, as shown in the exaggerated difference image on the right. It is clear that the vanishing point measure provides a good approximation to the true angle measure.

Of course, computing vanishing points requires knowledge of the plane at infinity Π_∞, which is unknown in a projective reconstruction. If the plane at infinity is found, upgrading the reconstruction from projective to affine, then vanishing points could be used to measure relative angle magnitudes. Unfortunately, computing the plane at infinity is one of the main reasons why obtaining a Euclidean reconstruction is so difficult. Finding the exact plane is very sensitive to noisy data and the input camera configuration.

However, it is not critical to have the *true* plane at infinity to use this vanishing point measure. It suffices to use a plane that satisfies cheirality constraints [Hartley 1993] to approximate plane at infinity. The cheirality constraints require that all data (cameras and

structure points) lie on one side of the chosen plane. Further, choosing the furthest such plane gives an improved approximation [Hartley et al. 1999]. Using such a plane, called $\tilde{\Pi}_\infty$, results in a quasi-affine reconstruction of the scene [Hartley 1993]. Thus, non-metric unstructured lumigraph rendering requires a quasi-affine scene reconstruction, which is generally easier to obtain than a full Euclidean reconstruction.

Given this plane $\tilde{\Pi}_\infty$, vanishing points can be computed using a planar homography mapping from points in image I_j, onto the approximate plane at infinity, and back to vanishing points in image I_i. This homography is given by

$$\mathbf{H}_{ij\tilde{\infty}} \doteq \mathbf{P}_i \mathbf{P}^+_{j\tilde{\infty}}, \tag{6.1}$$

where $\mathbf{P}_i$ is the projection matrix for image I_i, and $\mathbf{P}^+_{j\tilde{\infty}}$ is the inverse projection matrix mapping points in image I_j onto the approximate plane at infinity.

Note that since this angle measure violates the radiance consistency property, as it depends on the projection matrix of the desired view. For example, virtual views with different field-of-views or orientations may result in different reconstructions for the same radiance.

6.2.2 Resolution Measure

The simplified resolution measure described in Section 5.1.2 uses the relative distances of cameras from the scene point as an estimate of resolution similarity. Since it is based on distances, this measure is unsatisfactory in a non-metric setting.

However, recall that the full resolution measure, described in Section 4.2.2 is based on the Jacobian of the planar homography relating two cameras and a plane in the scene. Since this measure is based on an image-to-image transform, it is immune to the effects of the unknown projective transformation. Thus, the original resolution measure, while slower than the simplified one, is suitable for use in the non-metric algorithm.

6.2.3 Geometry Proxy

The ULR algorithm uses correspondence based on planar polygons. This correspondence implies a series of homographies, one per polygon, that relate one camera to another.

Given a polygon k, its associated plane Π_k, and two camera projection matrices $\mathbf{P}_i$ and $\mathbf{P}_j$, it is possible to compute the planar homography $\mathbf{H}_{ijk}$ relating the pixels in the two cameras:

$$\mathbf{H}_{ijk} \doteq \mathbf{P}_i \mathbf{P}_{jk}^{+},$$

where $\mathbf{P}_{jk}^{+}$ is the 4×3 inverse projection matrix that maps pixels in image I_j onto the plane Π_k. These homographies $\mathbf{H}_{ijk}$ establish a correspondence between image I_i and image I_j through plane Π_k.

This type of correspondence is well-suited to a non-metric algorithm since it is based on the image-to-image mapping $\mathbf{H}_{ijk}$. The only difficulty is specifying the polygonal proxy in the first place. The typical approach of choosing polygons that are "close" to the scene does not work, since it is difficult to define the closeness measure. However, it is possible to define polygons that pass *exactly* through the known structure points. Since projective transformations do preserve incidence, a set of polygons that passes through the projective structure points also passes through the "true" Euclidean structure points.

The proxy polygons are defined by triangulating the projections of the structure points. Given some desired view with projection matrix $\mathbf{P}_{des}$, the structure points $\mathbf{M}_j$ are projected into that view. The Delaunay triangulation of these image points results in a tessellation of the image plane. The topology of this triangulation is transferred to the projective structure points to arrive at a polygonal proxy.

6.2.4 Navigation

Navigation is perhaps the most difficult problem when dealing with non-metric spaces. In a projective setting, the projection matrix of a novel view can be computed by specifying the desired image projections of the structure points [Faugeras and Laveau 1994]. However, this method of navigation is not very intuitive and may lead to non-rigid camera motions or other improbable motions. For example, one could move two vertices of a square and leave the other two fixed, resulting in a distorted image of a square in the desired view. This distortion may or may not correspond to a rigid camera motion.

To combat unwanted (e.g., non-rigid) image transformations, it is useful to use an *a*

priori motion model when positioning the projections of the structure points. The motion model constrains the structure points to motions that are realizable within the model. For example, under a linear motion model, the projections of the structure points are constrained to move in straight lines. If the desired motion is not linear, then other models may be used. For example, one might use a pure rotational model, a rotation-about-a-point model, or something even more exotic. Two example motion models are discussed further in Section 6.4.2.

This type of navigation, while a bit cumbersome, is well-suited to the problem of video stabilization. By fitting the observed features motions to a smooth motion model, it is possible derive a sequence of projection matrices that correspond to a stabilized video sequence. The details of the video stabilization procedure are described in the following sections.

6.3 Non-metric ULR for Video Stabilization

An unstabilized video is an image sequence that exhibits unwanted variations in the apparent image motion. The goal of video stabilization is to remove these variations while preserving the dominant motions in the image sequence.

Most video destabilization is due to physical motions of the camera. Thus, many solutions to the problem involve hardware for damping the motion of the camera, such as a Steadicam rig or gyroscopic stabilizers. This equipment works well in practice, but it is very expensive. Recently, many consumer grade video cameras have been equipped with video stabilization features. However, these methods are often not good enough to stabilize gross motions of the camera. Because of these reasons, software solutions are attractive.

6.3.1 Other Approaches to Video Stabilization

Many previous software approaches to video stabilization assume little to no knowledge of the actual three-dimensional camera motion, and instead work to minimize image space motions directly. A common approach is to estimate a dominant planar homography that

stabilizes a large planar region in the video [Irani et al. 1994] or to use simple 2D translations to lock onto an object.

One basic problem with these approaches is that pure image transformations often do not correspond to reasonable camera transformations. For example, stabilizing a video by using pure translation of the video frames is equivalent to varying the camera's principal point, which is unlikely to be the true cause of the destabilization. Homography-based schemes do a better job, but they only stabilize planar scenes or rotational camera motions. Highly non-planar scenes or translational camera motions are not stabilized.

6.3.2 The IBR Approach to Video Stabilization

This thesis proposes a software video stabilization algorithm based on image-based rendering (IBR). The basic premise of the approach is simple. Assume that IBR can generate novel views from some set of known reference images. Assume further that the unstabilized camera trajectory (i.e., camera positions and orientations) is also known. Then, video stabilization can proceed in two steps:

1. Remove unwanted motions from the known camera trajectory through some sort of filtering or smoothing procedure.

2. Using the IBR algorithm, render a new image sequence along the stabilized camera trajectory.

IBR stabilization methods potentially avoid the problems of two-dimensional approaches because they can allow for virtual camera navigation in the scene. Thus, the virtual camera can be moved on a smooth path, removing the video destabilization at its source and resulting in a stable video for all image regions. Not only can an IBR method remove unwanted rotational and translational motions, it can also stabilize the velocity of the camera and the variation in the camera's focal length. IBR methods can also merge information from multiple video frames to synthesize a completely new view, so they are not limited to simple image warping.

The IBR approach defines two problems to solve: (1) filtering the camera trajectory and (2) rendering new views given the original views.

In the method outlined here, non-metric unstructured lumigraph rendering is used to generate novel views. The filtering procedure is implemented by fitting the observed data to *a priori* motion models. Both of these tasks are done without requiring a Euclidean reconstruction of the scene, which makes this approach generally applicable to a wide range of video sequences.

6.4 Stabilizing Video

The first step in the proposed video stabilization procedure is to obtain a projective reconstruction of the video sequence. It is not important how this reconstruction is obtained, just that it conforms to the description given in Section 6.2.

6.4.1 Computing a Projective Reconstruction

Briefly, the projective reconstruction technique used in this thesis proceeds as follows. First, image features are determined using off-the-shelf tracking software that implements the algorithm described in [Shi and Tomasi 1994]. At least 300 features per frame are tracked, because the resulting features generally contain many bad or poorly tracked features. Occlusions, view-dependent effects (reflections and highlights), motion blur, and other uncontrollable factors all contribute to poor feature tracking. Eliminating these bad features is key to obtaining a quality reconstruction. As a first step, all features that exist for less than 30 frames (i.e., one second) are eliminated. Short feature tracks often correspond to bad features.

Next, an initial projective reconstruction is computed using a projective factorization technique [Triggs 1996]. This factorization approach results in a rough solution that may be contaminated by the remaining bad features. At this point, a bad feature can be identified by examining its reprojection error, the distance between the feature's tracked location and the projection of its corresponding structure point. If the reprojection error is large (e.g., greater than 20 pixels), then a feature is considered bad. After culling more bad features in this way, the solution can be recomputed using projective factorization to obtain a better initial guess.

This initial guess is still not an optimal solution (in terms of minimizing reprojection error) even if all the features are good. The solution can be improved by using robust bundle adjustment techniques [Triggs et al. 2000]. Bundle adjustment is simply a non-linear optimization that minimizes the reprojection error of all the features in the projective reconstruction. Bundle adjustment works by iteratively adjusting the parameters of the projective reconstruction until the sum of reprojection errors reaches a local minimum. The parameters of the projection reconstruction are simply the 12 elements of each projection matrix and the 4 elements of each structure point. Bundle adjustment generally results in a high-quality solution, even with this non-minimal parameterization.

For long image sequences, this process needs to be modified because the projective factorization approach requires that all features be visible in all images. This restriction is not satisfied in most video sequences. To deal with this problem, the image sequence is divided into overlapping sub-sequences with independent solutions for each sub-sequence. The independent solutions are then mapped into a common projective frame by computing pair-wise projective transformations. While this method does not always give a globally consistent solution, it is generally sufficient for the video stabilization task.

As a final step, the projective reconstruction is upgraded to a quasi-affine reconstruction by approximating a plane at infinity. The procedure detailed in [Hartley et al. 1999] provides a suitable plane for this purpose. This approximate plane is used to compute vanishing point homographies with Equation 6.1.

6.4.2 Camera Trajectory Filtering

Once a quasi-affine reconstruction of the original video sequence is available, the camera trajectory must be filtered to remove unwanted motions. However, since only a non-metric reconstruction is available, it is impossible to filter the three-dimensional camera trajectory directly. Instead, the filtering is indirectly accomplished by filtering the two-dimensional motion of the observed image features. These filtered image features are then used to derive a sequence of stabilized projection matrices $\mathbf{P}'_i$ that generate a stabilized image sequence.

This feature filtering starts by computing target locations for the tracked image features

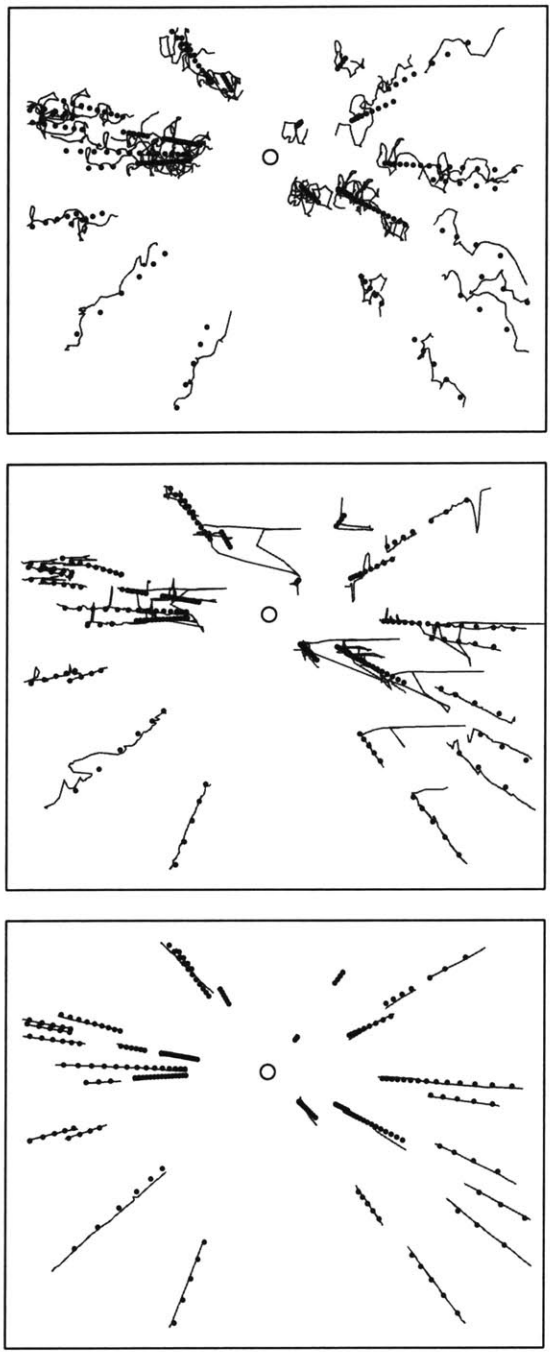


Figure 6-4: Three iterations of the feature filtering procedure. The initial feature locations are drawn as solid lines, and the target locations are shown as dots. The desired focus of expansion is marked with a circle. (top) Before optimization. (middle) After one iteration. (bottom) After all iterations.

in each frame. These target locations specify where the projections of the structure points should appear in a stabilized video sequence.

Given the target locations, the stabilized projection matrices are computed by slightly adjusting the unstabilized projection matrices so that they project the structure points onto the target locations (instead of the original feature locations). This adjustment is done using a non-linear optimization very similar to bundle adjustment (described in Section 6.4.1). In this case, the parameterization consists of only 12 parameters for each projection matrix, and the optimization is not allowed to adjust the positions of the structure points. The initial parameter values are set to the unstabilized projection matrices. The minimized error is the reprojection error between the projections of the structure points and the target locations.

After running this optimization, the quality of the solution can be evaluated by examining the reprojection errors. Large reprojection errors indicate that the target locations have been poorly selected. In such cases, the target locations can be re-estimated from the new solution and the optimization run again. It has been found empirically that this iterative process quickly converges to a good solution.

Figure 6-4 illustrates the evolution of the stabilized projection matrices during the optimization. The initial feature tracks (i.e., the curves that the features trace out over time) are shown as solid lines. The target locations are dots. Here the target locations correspond to a linear motion model, which is described in the next section.

Computing the stabilized target locations is the most difficult step. One way to stabilize the locations of the features is to low-pass filter the spatial variation of the features over time, and then map the original features to points on the new path. This technique reasonably filters the feature locations, but it enforces no constraints on the implied motion of the stabilized camera. To enforce rigid-motion (and other) constraints, it is advantageous to apply a prior model to the motion of the desired feature locations.

Using a motion model helps ensure that the new projection matrices correspond to rigid motions. Some possible motion models include linear translation, fronto-parallel translation, circular motion about a fixed target, etc. In general, the selected motion model should roughly correspond to the observed motion in the scene. If not, then the model fitting procedure described previously is not likely to succeed.

The examples in this thesis use two different models: (1) the linear motion with constant velocity model and (2) the Hitchcock "Vertigo effect" model. These models are described in the following two sections.

Linear Motion with Constant Velocity Model

In this model, the stabilized camera should move at a constant velocity along a straight line while looking in the same direction. With linear motion, it is well-known that all image features move radially to (or from) a point called the focus of expansion [Horn 1986].

To use a linear motion model, it is first necessary to estimate (or specify by hand) the focus of expansion, which may or may not be in the field of view, and to fit radial lines to the initial feature tracks.

Next, the unstabilized features are mapped to points on the radial lines. These points on the radial lines become the target locations of the unstabilized features. Often, the original features do not map well onto the lines (see Figure 6-4a), but in any case the original features are mapped to the closest point on the line.

Finally, the target locations are redistributed along the length of the line according to the constant velocity assumption: as time progresses, points should move monotonically toward (or away from) the focus of expansion. More specifically, the projection of a point moving with constant velocity should fit a log function of the form $a \log |t - t_0| + b$, where t is a time index and a, b, and t_0 are unknown parameters. These parameters are estimated for each target location, and they depend in some way on the relative depths of each structure point. The actual depths are unimportant, although proper distribution of the target locations helps ensure a physically plausible camera motion.

Hitchcock "Vertigo Effect" Model

In this model, the camera simultaneously moves forward (backward) and zooms out (in) to keep the size of a foreground object constant. This motion imitates a cinematographic effect made popular by Alfred Hitchcock. Note that this motion model incorporates changes in both the intrinsic and extrinsic camera parameters. As a result, the stabilized video has smooth motion and zooming.

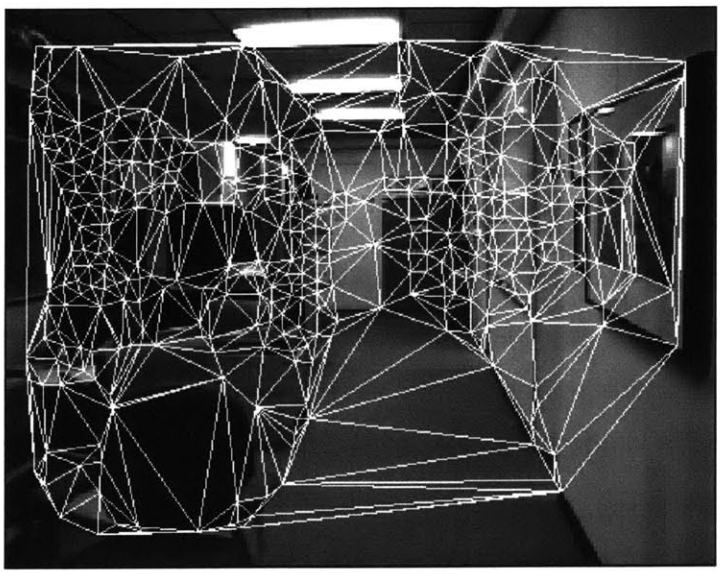


Figure 6-5: An example triangulation that is used for determining the proxy.

In the Hitchcock zoom model, there are two classes of features: foreground features and background features. Foreground features are constrained to remain stationary, while background features move radially much like in the linear motion model. The focus of expansion is taken to be the center of the foreground object. The background target locations are redistributed uniformly along the radial lines.

In this implementation, the foreground features are selected by hand.

6.4.3 Rendering

As the final step, the stabilized image sequence is rendered using non-metric unstructured lumigraph rendering. The execution of the rendering algorithm proceeds as in previous chapters, except that the geometry proxy is created dynamically for each frame. For each desired view, the renderer first computes the triangular tessellation of the visible structure points. A structure point is deemed visible in a desired view if it is visible in the corresponding original view. An example proxy is shown in Figure 6-5.

Briefly, for each pixel in the desired view, the renderer computes the corresponding pixels in the reference views using the planar homographies $\mathbf{H}_{ijk}$. Each reference pixel is

assigned a weight according to the vanishing point distances, which are computed using the homographies $\mathbf{H}_{ij\infty}$. Field-of-view and resolution can also be taken into account. Typically $k = 4$ reference images are used at each pixel.

One additional modification to the original ULR algorithm can also be used. Because arbitrary video sequences can contain moving objects (i.e., they violate the static scene assumption), blending multiple images together from different points in time can lead to temporal artifacts. To alleviate this problem, the images used to generate each output image can be restricted to a subset that falls within a temporal window of the desired image time. In the examples used in this thesis, a temporal window of 11 frames is used. That is, each output image contains contributions from at most 11 images. At 30 frames-per-second, 11 frames corresponds to a temporal window of $\pm.183$ seconds on either side of the desired frame. The temporal window can be made bigger (smaller) if the video contains slower (faster) motions.

6.5 Examples

This section presents three examples of the video stabilization technique in action. The first two examples use the linear motion with constant velocity model, while the third example uses the Hitchcock Vertigo effect motion model. All of the video sequences are acquired with a hand-held video camera that has had its own video stabilization features disabled. All other automatic features of the camera have been enabled.

6.5.1 Example #1

Figure 6-6 demonstrates the feature filtering results from the first linear motion model example. In this example, the hand-held camera is moved forward down a hallway. The figure plots the motion of the feature points over time. The tracks drawn with solid lines are the original features that were tracked with automatic feature tracking software [Shi and Tomasi 1994]. The tracks drawn with red dotted lines are the features as seen from the stabilized camera trajectory. A small circle marks the focus of expansion that is used for the linear motion model.

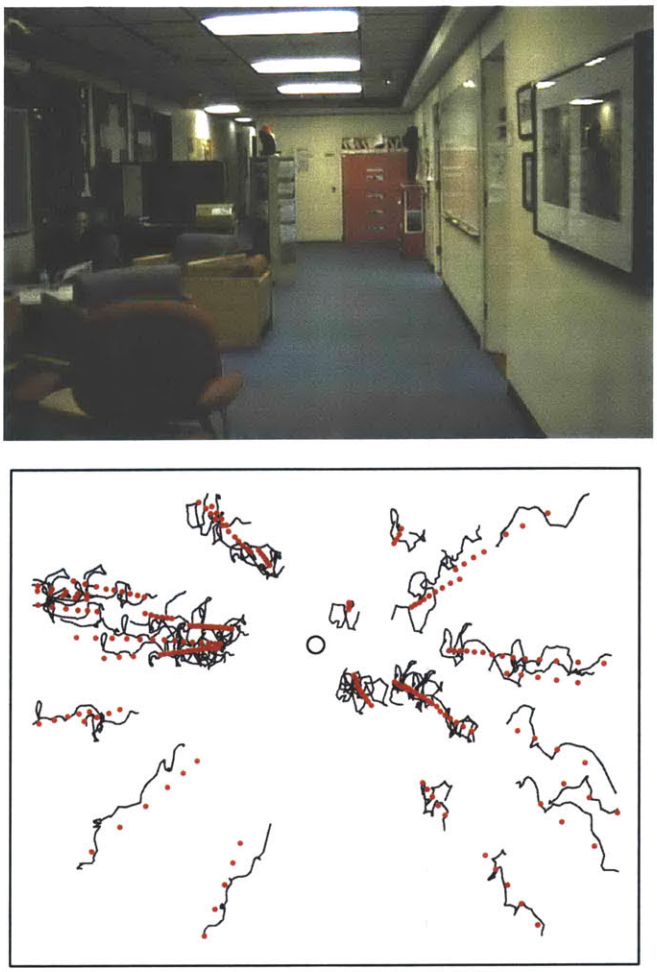


Figure 6-6: Feature tracks for a video sequence with forward camera motion. The original features are solid black, and stabilized features are red dots. The features are stabilized using the linear motion with constant velocity model, whose focus of expansion is shown with a circle. An original frame from the sequence is shown in the top image.

Figure 6-7 shows example renderings from the stabilized video sequence. The first column of images shows split-screen comparisons of the original videos (left half) and the stabilized videos (right half). The stabilization is most clearly demonstrated by the position of the red filing cabinet (right-center of the images). Note how the red filing cabinet remains nearly stationary in the stabilized portion of the images, as the direction of motion is constrained to be toward the top of the cabinet. In the unstabilized image portions, the cabinet moves up-and-down and left-to-right in an uncontrolled way.

The second column of images in Figure 6-7 shows the associated false-color visualiza-

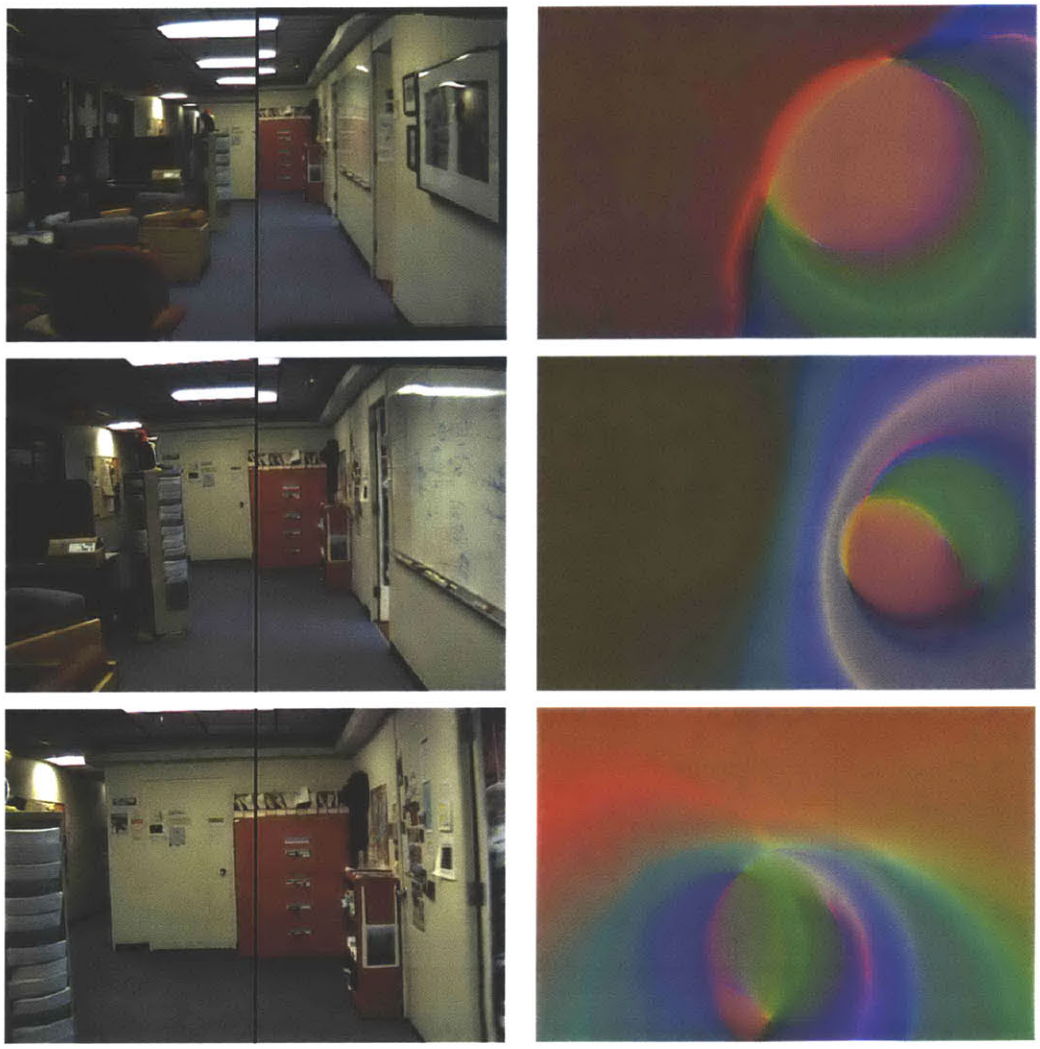


Figure 6-7: Renderings from the first video stabilization example. Stabilized frames (right half) are compared to original frames (left half) in split-screen images (first column). The associated false-color visualizations are shown in the second column.

tions for each of the three rendered frames. The visualizations have circular shapes similar to those seen in the example of Section 5.3.3, which suggests that the quasi-affine angle measure is reasonable. These visualizations do look slightly different because of the limitation to only 11 images in each rendering. This limitation causes large portions of the image to to be taken from a single image in the video sequence, which shows up as large constant-color areas in the false-color visualization.

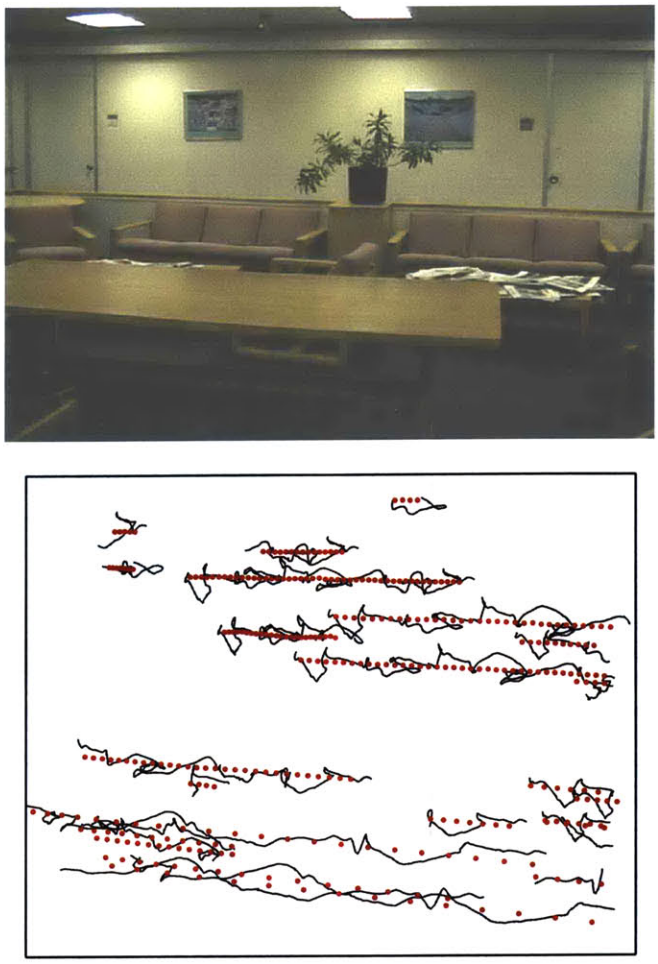


Figure 6-8: Feature tracks for a video sequence with sideways camera motion. The original features are solid black, and stabilized features are red dots. The features are stabilized using the linear motion with constant velocity model, whose focus of expansion (not shown) is to the left-hand side of the image. An original frame from the sequence is shown in the top image.

6.5.2 Example #2

Figure 6-8 demonstrates the feature filtering results from the second linear motion model example. In this example, the hand-held camera is moved sideways in a furniture-filled room. As before, the tracks drawn with solid lines are the original features, and the tracks drawn with red dotted lines are the features as seen from the stabilized camera trajectory. The focus of expansion, although not shown in this figure, is off to the left-hand side.

Figure 6-9 shows split-screen comparisons of the rendered images (right half) to the

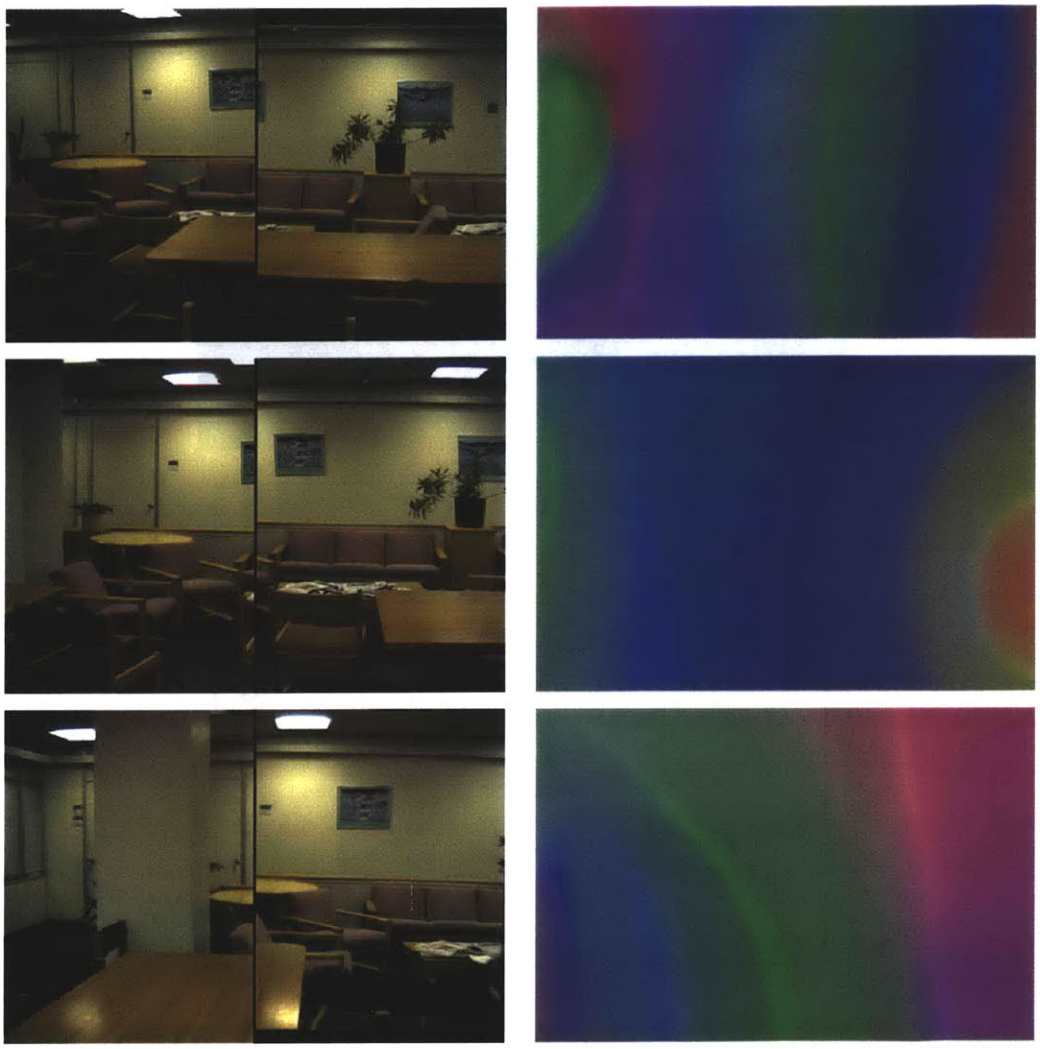


Figure 6-9: Renderings from the second video stabilization example. Stabilized frames (right half) are compared to original frames (left half) in split-screen images (first column). The associated false-color visualizations are shown in the second column.

original images (left half). Note how the tables, railings, and other horizontal features remain at the same height throughout the stabilized halves of the image. In the unstabilized portions, the horizontal features show visible up-and-down deviations from the stabilized trajectory.

The second column of images in Figure 6-9 shows the associated false-color visualizations for each of the three rendered frames. The visualizations have shapes similar to those seen in the example of Section 5.3.2, which was made from a video sequence that

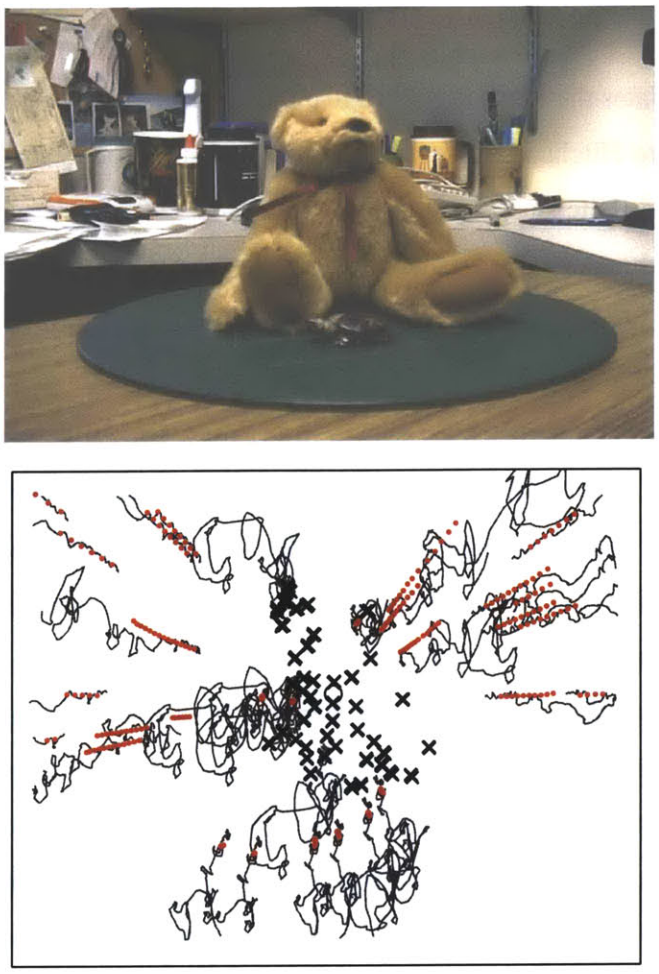


Figure 6-10: Feature tracks for a video sequence with motion that approximates Hitchcock's Vertigo effect. The original features are black lines. Stationary features (on the bear) are marked with bold X's. The remaining features are constrained to move radially from the center of the fixed features. These features are shown with red dots. An original frame from the sequence is shown in the top image.

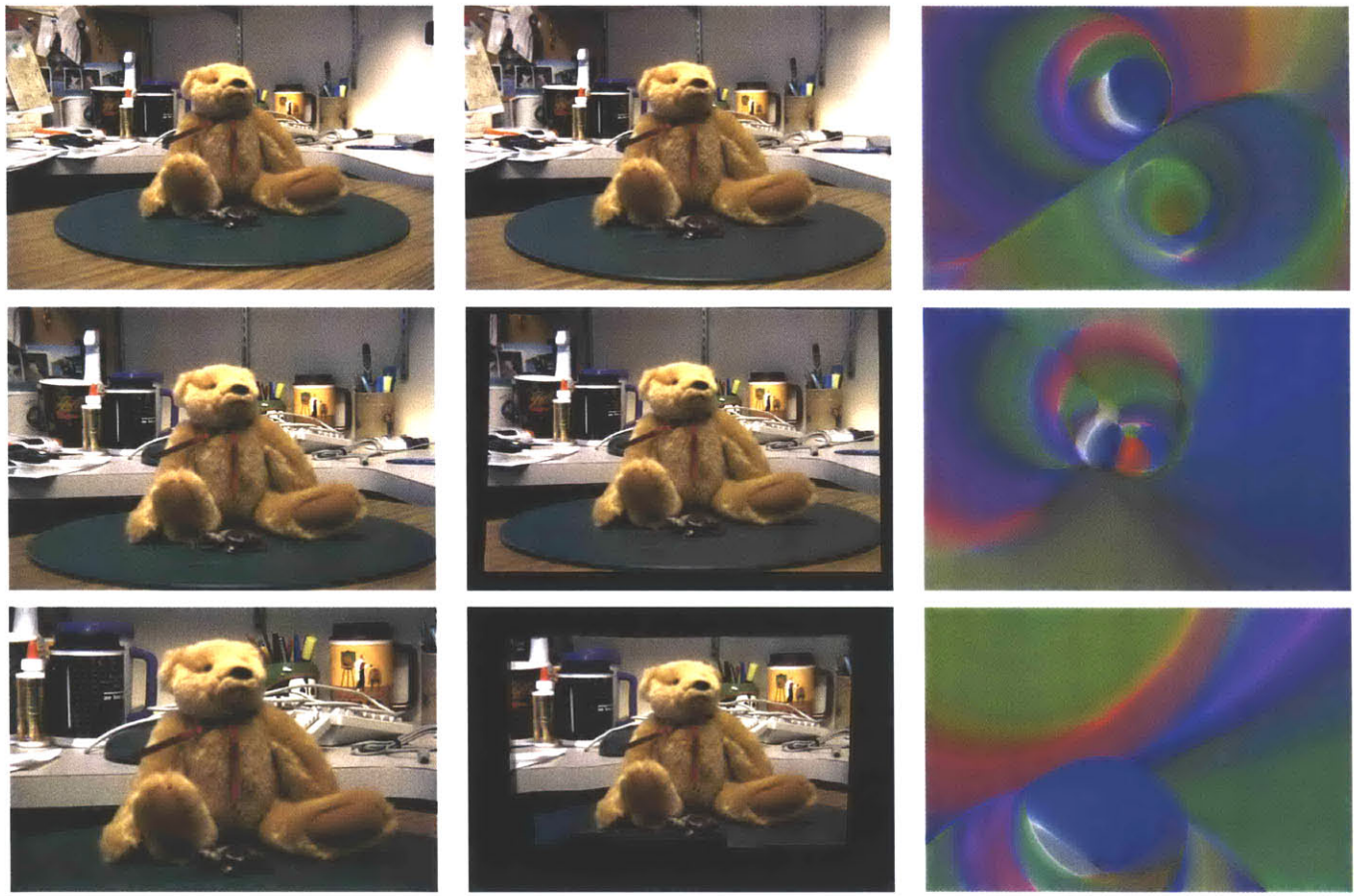
also exhibits sideways motion. The 11 camera limitation is not as apparent in this example, because the sideways motion prevents large numbers of cameras from being visible in any single view.

6.5.3 Example #3

This example simulates the Hitchcock Vertigo effect motion model. The unstabilized video attempts to replicate Hitchcock's Vertigo effect by simultaneously moving and zooming the camera by hand (Hitchcock used a mechanical apparatus in his movies). Even after repeated attempts, the manual results are mediocre at best. However, the stabilized results are effective. The algorithm is able to stabilize both the motion of the camera and the variation of the focal length.

The feature track comparisons are shown in Figure 6-10. As before, the original features are solid black. The Hitchcock Vertigo effect motion model is specified by fixing the locations of a set of features, whose positions are marked with bold X's. The remaining features are constrained to move radially from the center of the fixed features. The radial features are shown with red dots, as in previous examples.

Figure 6-11 shows the rendering results from the Hitchcock Vertigo effect example. In this example, the teddy bear is the focal object whose position is constrained to remain fixed in the video frame. The background features are constrained to move radially. In this example, the stabilization is dramatic enough that a split-screen comparison is unnecessary. It is clear from these images that the bear stays in the same position in the stabilized images, while it increases in size in the unstabilized video. Note that background objects that are occluded by the bear in some frames become visible in other frames, indicating that the background is indeed moving relative to the foreground.



(a) (b) (c)

Figure 6-11: Rendered frames from the Hitchcock Vertigo effect motion model: (a) the original frames (b) the stabilized frames (c) the associated false-color visualizations. Note how the size of the bear, which is the focus of the effect, remains constant in the stabilized images.

6.6 Summary

This chapter has presented modifications to the unstructured lumigraph rendering algorithm to make it suitable for use with non-metric image calibration data. Four aspects of the original ULR algorithm need to be modified: the angle measure, the simplified resolution measure, the proxy construction, and the virtual view specification. The first three aspects can be accommodated with little trouble. However, specifying the virtual view in a non-metric space is difficult. The proposed solution is to specify the desired locations of image features using a predefined motion model.

While the proposed navigation mode is non-intuitive, it is well-suited to the task of video stabilization, which is considered in the second half of this chapter. It is shown how simple image motion models can be used to filter the motion in an unstabilized video sequence. The result is a sequence of stabilized camera projection matrices, which can then be used with the non-metric ULR algorithm to render a stable video sequence. The success of this technique is demonstrated with three examples.

CHAPTER 7

Time-Dependent Unstructured Lumigraph Rendering

One of the main drawbacks to light field or lumigraph techniques is the static scene assumption: that the scene and lighting remain constant within the collection of images. This chapter considers a relaxation of the static scene assumption: allowing a scene to vary with time.

Adding time-dependence to lumigraphs is conceptually simple. The primary difficulty lies in data size and acquisition. First, time-dependence adds an additional dimension to lumigraph data, which requires much more memory resources. Second, acquiring time-dependent lumigraph data is non-trivial. One needs to capture multiple images of a dynamic scene from dozens of different viewpoints, a daunting task.

However, with the ever-increasing power of computers, time-dependent lumigraphs are becoming practical. It is not uncommon to see desktop computers with multiple gigabytes of memory and even larger hard disks. Thus, storage requirements are becoming less of an issue. Further, in recent years multi-camera systems have been developed that can capture arrays of video data necessary for a fully general time-dependent lumigraph [Ooi et al. 2001; Naemura et al. 2002; Wilburn et al. 2002]. MIT has its own 64-camera array of inexpensive 1394 video cameras (see Figure 7-1).

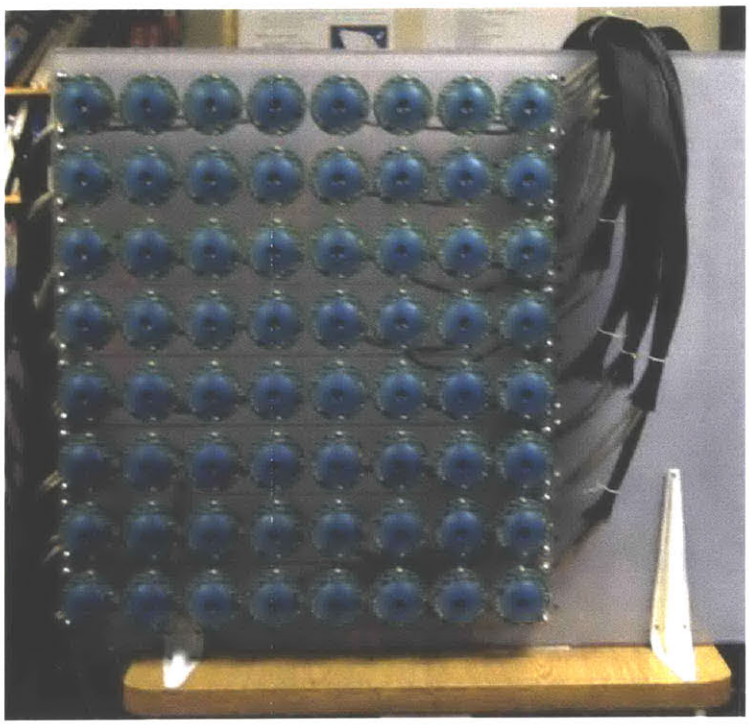


Figure 7-1: MIT's 8×8 array of video cameras. The array delivers a 64-image light field that users can interact with in real-time.

In light of these developments, there is a growing need for rendering algorithms that can handle time-dependent data. This chapter describes how to modify unstructured lumigraph rendering for this purpose. Special emphasis is placed on unstructured time-dependent lumigraphs, that is, lumigraphs that are sampled at irregular locations in space and time.

7.0.1 Overview

The chapter begins by describing a simple time-dependent representation of lumigraphs. Next, the unstructured lumigraph rendering algorithm is modified to accommodate time-dependent data.

The second half of the chapter describes a special type of time-dependent lumigraph: the time-periodic lumigraph. A time-periodic lumigraph captures a scene that repeats itself indefinitely. It is shown how time-periodic lumigraphs can be cheaply and easily acquired using continuous video from a single video camera. The resulting time-periodic lumigraphs are irregularly sampled in both space and time, making them perfect candidates for the

modified unstructured lumigraph algorithm.

7.1 Time-Dependent Lumigraphs

In previous chapters, a lumigraph consisted of a geometry proxy, a collection of images, and the projection matrices $\mathbf{P}_i$ associated with those images. A time-dependent lumigraph is defined to be a geometry proxy, a collection of images, the projection matrices $\mathbf{P}_i$, and a timestamp t_i associated with each image. Note that this is not the only way to define a time-dependent lumigraph (e.g., the proxy could also be time-dependent), but this is the time-dependent lumigraph that is considered in this thesis.

The time-dependent image-based rendering problem is the same as that defined in Chapter 3, if "calibrated images" is taken to mean calibrated in both space and time and if the specification of the desired view includes a desired time t_{des} as well as a desired projection matrix $\mathbf{P}_{des}$. A similar change of interpretation applies for the definition of the time-dependent radiance reconstruction problem.

7.1.1 Time-Dependent Extensions to ULR

The modifications to ULR to support time-dependent rendering are similar to those used for handling field-of-view and resolution issues. A time-dependent correlation function $R_I(\theta, f, r, \tau)$ is defined, where τ is a time dissimilarity measure defined much like the field-of-view and resolution dissimilarity measures. Given the time dissimilarity τ, the modified correlation function is

$$R_{I,gen}(\theta, f, r, \tau) = R_I(h(\theta, f, r, \tau)),$$

where $h(\theta, f, r, \tau)$ is the generalized angle measurement (analogous to Equation 4.7 in Section 4.2), given by a linear combination of the dissimilarity measures

$$h(\theta, f, r, \tau) = \alpha\theta + \gamma f + \beta r + \varepsilon\tau.$$

The constants α, β, γ, and ε control the relative importance of each type of dissimilarity measure.

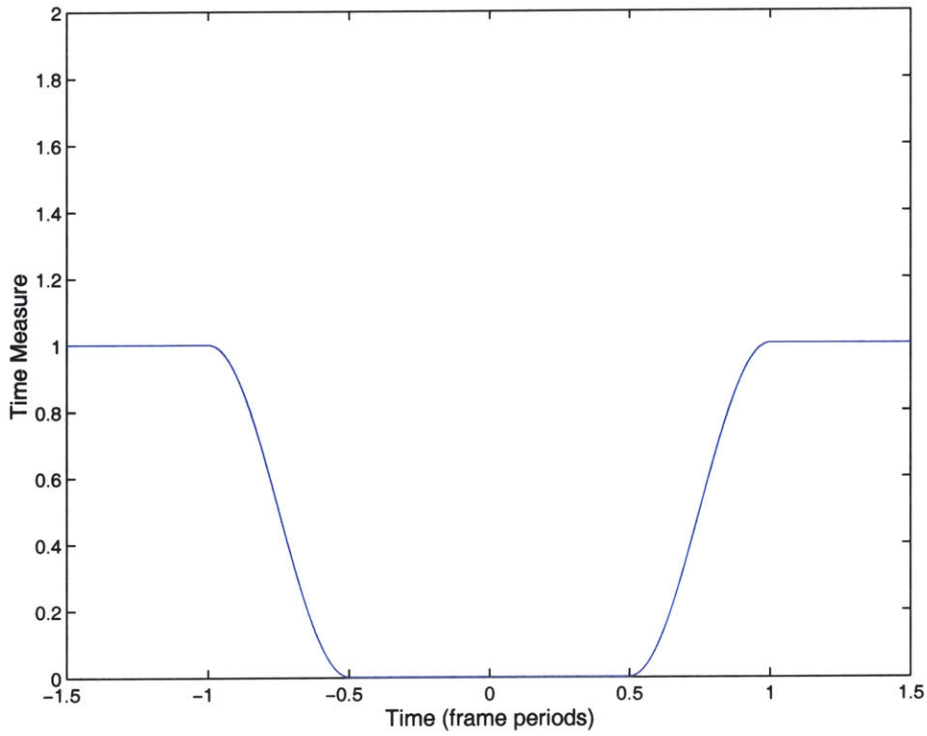


Figure 7-2: A function that modifies the time difference between two images. Images within 1 frame period of one another are considered equivalent, while differences greater than 1 frame period are penalized.

Time Dissimilarity Measure

Given two radiance observations with timestamps t_1 and t_2, a simple time dissimilarity measure is their difference,

$$\tau = |t_1 - t_2|,$$

which is zero for identical times and increases unbounded for different times. For greater control over the time dependent aspects of rendering, it is useful to transform the time measure using a function $f(\tau)$ similar to those used in Sections 4.2.1 and 4.2.2 for the field-of-view and resolution dissimilarity measures. For example, the function shown in Figure 7-2 does not penalize a difference in time until a certain threshold is passed. When one of the cameras corresponds to the desired view, then this measure strongly penalizes cameras when their time stamp differs by the threshold amount.

The operation of this time dissimilarity measure has a simple interpretation. It essen-

tially selects cameras that are within a window of time centered about the desired time t_{des}. Of course, the time window should be "soft" so that the influence of a camera falls off continuously as the camera leaves the time window. The width of the time window is a parameter that determines a tradeoff between spatial and temporal resolution. A wide window admits more radiance observations into the lumigraph and yields higher spatial resolution at the cost of temporal blurring. Non–moving features are well–defined, but moving ones are blurred. A narrow window gives good temporal locality, but may result in sparsely populated spatial dimensions. The time window width is somewhat analogous to the exposure time in traditional cameras.

Given this time dissimilarity measurement, it is simple to apply the unstructured lumigraph rendering algorithm to time-dependent data. The modified correlation function is used in place of the normal one and rendering proceeds as usual.

7.2 Time-Periodic Lumigraph Rendering

In this section, simple techniques are presented for acquiring and rendering a restricted class of time-dependent lumigraphs: time-periodic lumigraphs. A time-dependent lumigraph is time-periodic if the lumigraph at time t is identical to the lumigraph at time $t+T$ for some T, which is the period of the lumigraph. If the scene of interest exhibits periodicity, then it can be represented with a time-periodic lumigraph. While this restriction may seem extreme, time-periodic lumigraphs can represent useful scenes (and objects) such as people walking or running, turning wheels and gears, or flashing lights.

The technique described here uses a single hand-held video camera for time-periodic lumigraph acquisition. Because the scene is assumed periodic, a single moving video camera can sufficiently sample the lumigraph in space and time. However, the resulting lumigraph is irregularly sampled, so unstructured lumigraph rendering is required to render novel views from any point in space-time.

In order to use the time-dependent ULR extensions, the positions of the input cameras must be known in both space and time. In Section 7.2.1, a simple space-time calibration technique is described. Then, in Section 7.2.2 some examples of time-periodic lumigraphs

are demonstrated.

7.2.1 Time-Periodic Lumigraph Acquisition

Time–periodic lumigraph acquisition is essentially equivalent to calibrating a camera (i.e., determining its position) in both time and space.

Calibration in Time

To calibrate the camera in time, the period of the scene is sought in terms of the period of the camera. The scene is assumed to exhibit periodic motion with fixed period T_{scene}. The video camera has a fixed sampling period, which is taken to be $T_{cam} = 1$. With the camera's position fixed, the scene is recorded for enough time to capture multiple (10 or 20) periods of motion (the images in this video are not used in the final lumigraph). The video is manually inspected to determine two widely spaced (in time) frames that appear identical. The number of scene periods and the number of frames between those two key frames are counted. Then, the period of the scene can be expressed as

$$T_{scene} = \frac{\#frames}{\#periods}.$$

Now, given T_{scene} and the frame number of any video frame, the time in the scene's period at which the video frame was taken is

$$t_{periodic} = frame\,\# \mod T_{scene}.$$

Calibration in Space

Next, the scene is recorded with a moving video camera. The images from this video form the basis of the lumigraph. Care must be taken that the camera is not moved in a periodic motion that has a period proportional to T_{scene}, or else the sampling will be inadequate. In addition, the camera should be moved in a region of space that coincides with the desired viewing region of the lumigraph. The length of the video is determined by the desired number of spatial samples in a time window of the lumigraph. On average, the number of

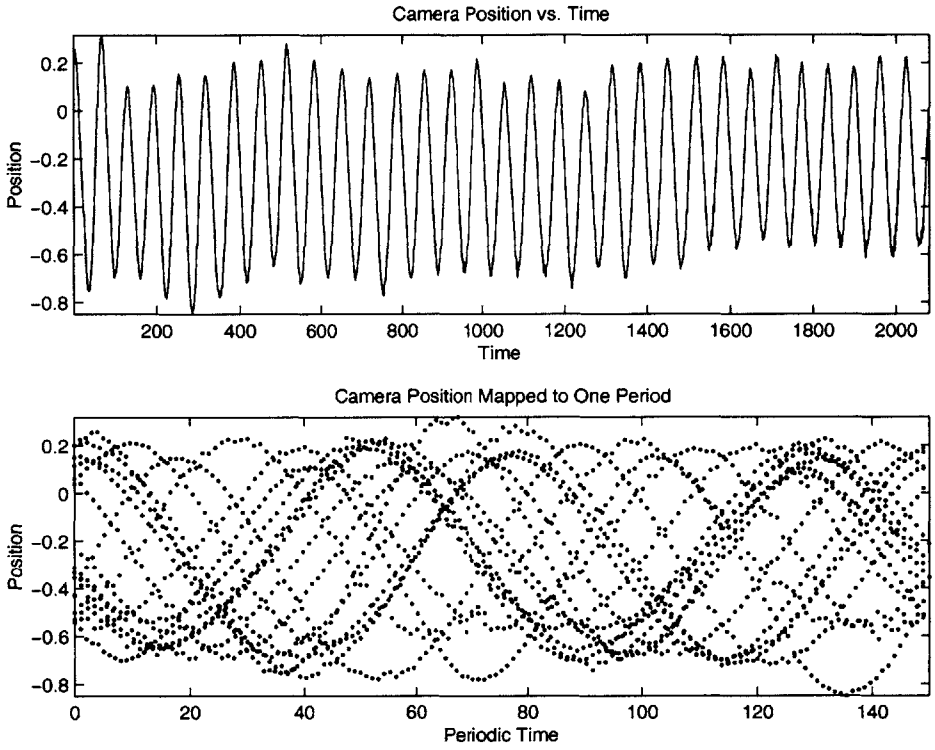


Figure 7-3: The results of space-time calibration. The top plot shows the motion of the camera (in x), and the bottom plot shows the camera positions remapped into one period of the lumigraph.

spatial samples in any time window of width 1 is given by

$$\#samples = \frac{\#frames}{T_{scene}}.$$

The required length of the video sequence can be determined from this equation.

To calibrate the camara in space, the time–varying elements of the scene are ignored, and standard computer vision techniques for static scenes are applied (see Chapter 5 for some example techniques). In this implementation, the time–varying elements are manually segmented from the static elements. Since the time–varying elements tend to be localized, this segmentation is trivial. This approach has the limitation that at least some portion of the scene must be static for traditional computer vision algorithms to work. If one can determine the position of the camera by other means (such as with a motion control rig or motion sensors), then this limitation can be removed.

Some results of the space-time calibration are shown in Figure 7-3. The top plot shows the position of the camera against absolute time. Only the x coordinate is shown, as the

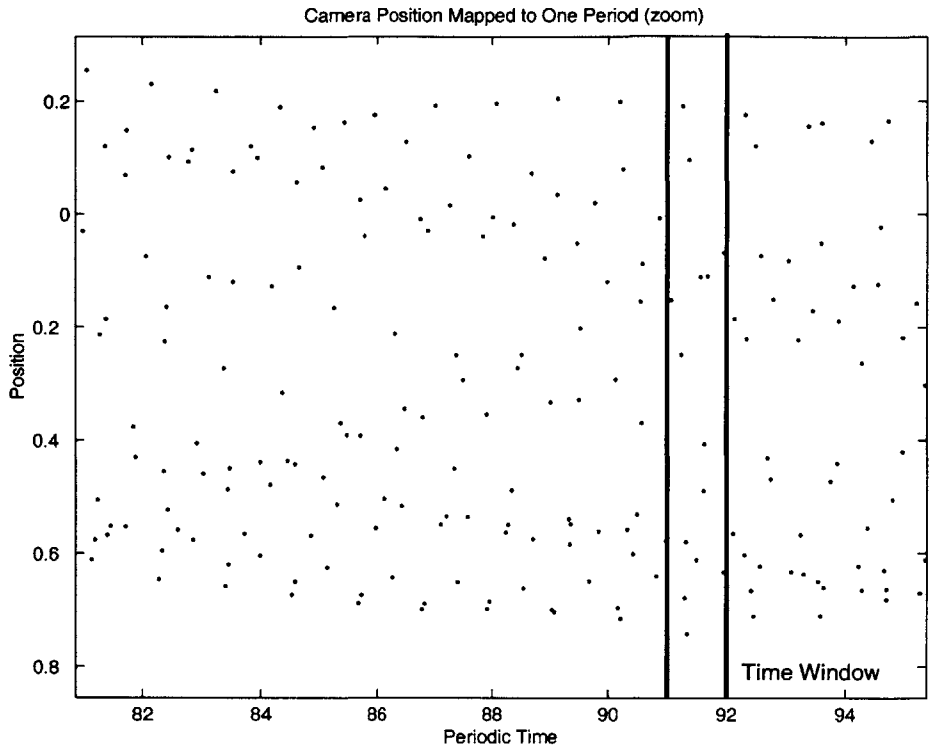


Figure 7-4: A zoomed view of a unit width time window superimposed on top of the space-time plot. The window contains 14 cameras.

camera was moved (mostly) horizontally in an oscillating pattern. In the bottom plot, the positions are remapped into one period of the lumigraph (T_{scene} is about 150 frames). This second plot can be used to judge the uniformity of the sampling. Ideally, the space-time plane would be uniformly sampled. The case in Figure 7-3 is far from ideal, although it is good enough to make convincing renderings.

7.2.2 Examples

Two time-periodic lumigraphs captured using the technique described in Section 7.2.1 are shown in Figures 7-5 and 7-6. The first is a scene containing a rotating lamp. It was captured by waving a hand-held video camera in front of the lamp for a minute or so. The period of the lamp is about 150 video frames, and about 2000 video frames are used in the lumigraph. This leaves, on average, about 13 spatial cameras per unit time window. Figure 7-4 illustrates the action of the time window. The unit time window shown has 14

temporally close images that (primarily) contribute to the desired image.

Three images from the lamp lumigraph are shown in Figure 7-5. The images show the camera moving left and right with the lamp turning. Note that the virtual camera moves horizontally because the lumigraphs only exhibit parallax in that dimension.

While these images demonstrate the viability of the approach, they also exhibit a number of artifacts. First of all, the spatial resolution is very low (only 13 cameras per unit time window), so there is some image doubling and blurring in the images. These "out-of-focus" effects are largely due to using a single plane as the geometry proxy. These artifacts are especially noticeable in moving lumigraphs, and even more so in unstructured ones, as it appears that the out-of-focus regions are "dancing" relative to the focused regions. These problems are new to lumigraph rendering since unstructured time-dependent lumigraph data has never before been available.

The second time-periodic lumigraph depicts a helicopter with rotating blades. The scene period is only about 15 frames. About 900 video frames are used, achieving on average 60 spatial samples per unit time window. The results are much better for the helicopter (less image dancing) because of the higher spatial resolution, although the faster motion produces more noticeable temporal blurring (which is also present as motion blur in the original video). Figure 7-6 shows three virtual images of the helicopter lumigraph.

7.3 Summary

This chapter has presented simple techniques for extending unstructured lumigraph rendering to time-dependent scenes. These techniques are demonstrated on a specific type of time-dependent lumigraph, the time-periodic lumigraph.

Time-periodic lumigraphs represent scenes containing periodic motion. A method for acquiring time-periodic lumigraphs is outlined. The distinguishing feature of this method is that it uses a single continuous video sequence from one video camera to construct the lumigraph. The resulting time-periodic lumigraphs are irregularly sampled in both space and time, which is well-suited to the time-dependent ULR algorithm.

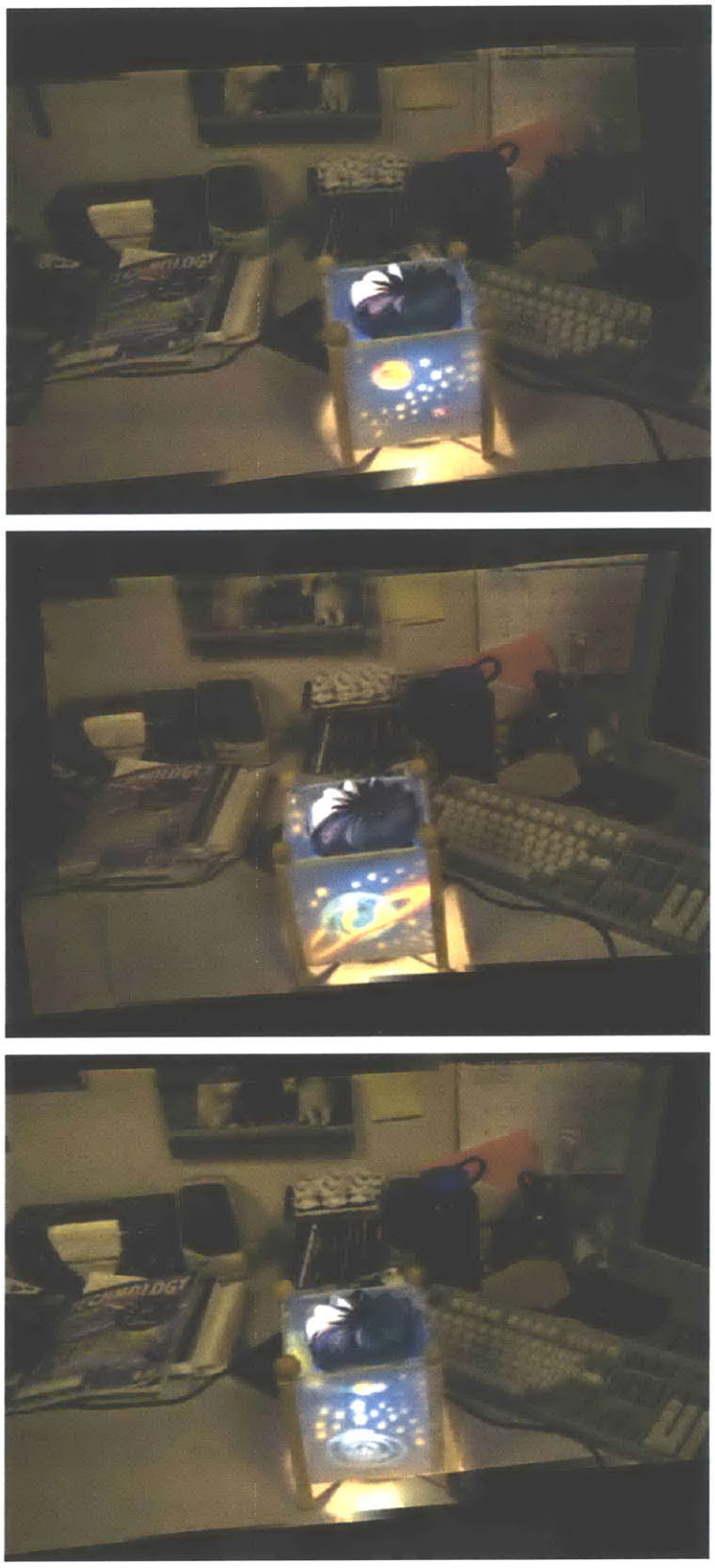


Figure 7-5: Three virtual views of a rotating lamp.

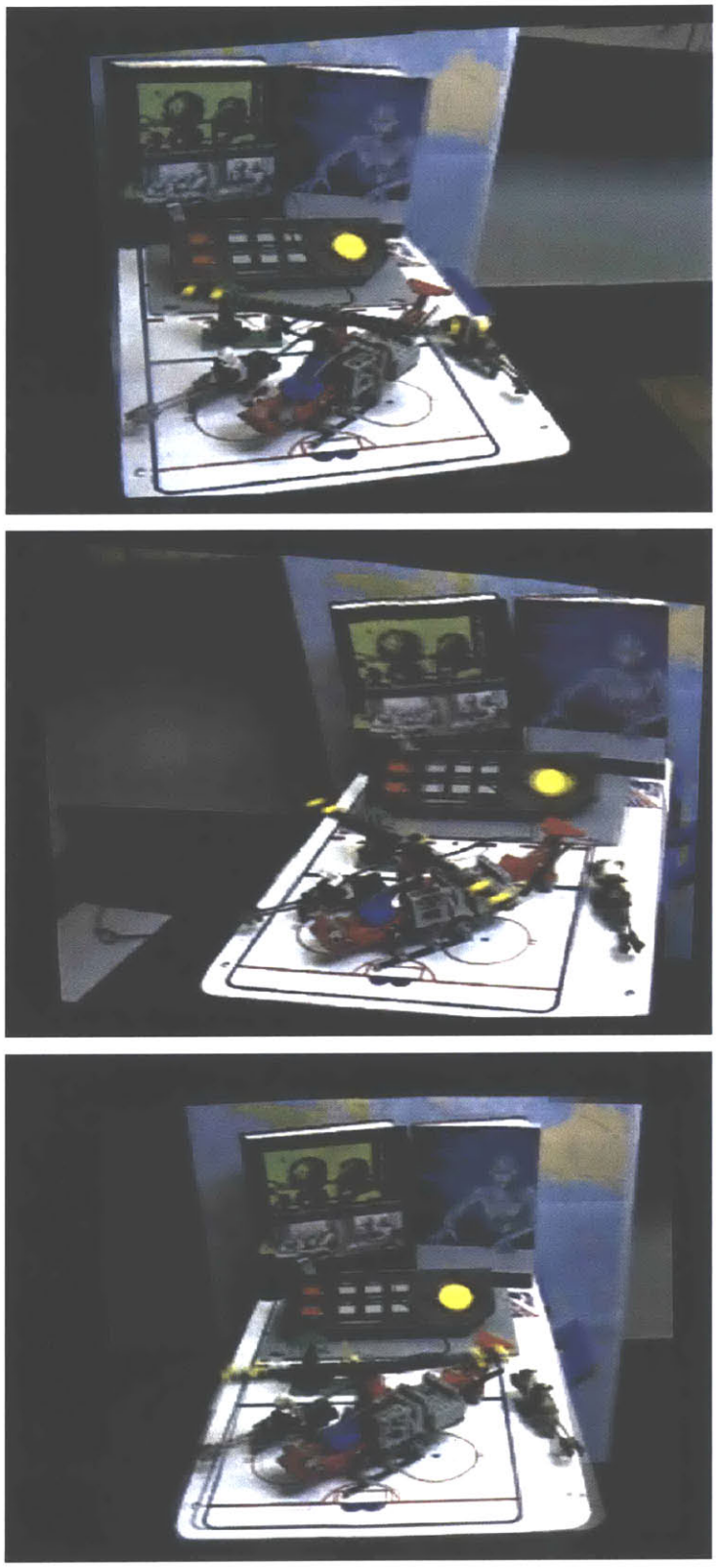


Figure 7-6: Three virtual views of a helicopter with spinning rotors.

CHAPTER 8

Conclusions and Future Work

Image-based rendering has become a popular alternative to traditional three-dimensional computer graphics because of its promise to deliver photorealistic images at real-time rates. This popularity is demonstrated by the recent development of a wide variety of rendering algorithms and by the success of commercial systems using image-based representations.

This thesis introduces a new image-based rendering algorithm called unstructured lumigraph rendering. ULR is designed to produce high-quality images given any available inputs (inputs consist of images and scene geometry). In doing so, it subsumes most earlier algorithms, which can be largely distinguished by limitations in the types of inputs they accept or do not accept. This flexibility allows ULR to generate views from input configurations that are unsuitable for any previous algorithm, such as hand-held video camera footage with looming motions or images of a time-dependent scene that are irregularly sampled in space-time.

8.1 Future Areas of Research

While unstructured lumigraph rendering solves many image-based rendering problems, there are still many difficult problems that remain. The following sections detail weaknesses in the current ULR approach and suggests potential solutions that deserve more investigation.

8.1.1 Better Image Correlation Functions

The image correlation function used in Chapter 4 provides a reasonable image model for addressing image reconstruction, but it has some drawbacks. One problem is the assumption that the same image correlation function is appropriate for each pixel in the desired view. This assumption is often made because it simplifies the calculations and makes determining the correlation function easier. However, in many cases, it would be better to have a spatially-varying image correlation function.

As an example, consider the sequence of images that results from moving a video camera toward an object. When the camera is far from the object, the image probably contains the object of interest plus many others. As the camera moves toward the object, the object becomes prominent in the images. When the camera is very close to the object, the object generally fills the field-of-view of the camera. In this example, it is clear that $P_{same}(\theta)$, the probability that two pixels lie on the same object, varies as the camera moves toward the object.

Spatial variations in the image correlation function would be expected in an image-based rendering application with an approximate scene geometry. In areas where the geometry approximation is good, the image correlation function should have the "close to object" form. In areas where the geometry proxy is poor, the correlation function should reflect "far from object" behavior. Such spatial variation would provide a more "optimal" solution to the radiance reconstruction problem.

Such a variation could be accommodated by manipulating the parameters of the analytical image correlation function given in Equations 4.4 and 4.5. Of course, some measure of the fidelity of the geometry proxy is also required.

8.1.2 Non-static Video

One of the most compelling applications of ULR is rendering from simple video sequences. This ability is extremely powerful and greatly simplifies the acquisition of image-based scenes. However, the ULR algorithm does not work for every video sequence. It is restricted to video sequences of static scenes: scenes in which objects do not move and the lighting does not change. Equivalently, ULR is restricted to videos in which all image variation is due to motion of the camera.

It would be a great advance to extend rendering techniques to handle video sequences with moving objects, variable lighting, or both. In such an algorithm, a virtual view would be specified not only by the projection matrix of a camera, but also by the desired lighting of the scene and the desired positions of objects in the scene. Interactive techniques for doing these operations on a single image have been developed [Oh et al. 2001], but these techniques do not generalize well to video.

One way to handle moving objects is to segment the video frames into different regions, where one region represents a static background and the other regions correspond to different moving objects in the scene. Then, new views of the static background can be generated using standard unstructured lumigraph rendering techniques. Holes in the background (from removed foreground objects) can be filled in using a modified version of the field-of-view measure.

The moving objects can be recomposited into the scene using, for example, textured billboards. A textured billboard is simply a two-dimensional planar surface with an image mapped on it. The image can be moved slightly by changing the position of the billboard. In this case, the textured billboard is essentially a time-dependent unstructured lumigraph with only one image per time instant and a planar geometry proxy.

8.1.3 Multi-Image Editing

Manipulating objects within an unstructured lumigraph is useful even for static scenes. A similar segmentation approach can be used for this task. This segmentation effectively partitions the unstructured lumigraph into multiple unstructured lumigraphs, one for each

region.

By manipulating each lumigraph separately, individual objects can be moved relative to one another and composited into new images. One drawback is that the lighting of the static scene moves with the objects, so this approach may only be suitable for small motions or for scenes in which the lighting is predominantly ambient.

Each of the sub-lumigraphs contains only a subset of the pixels from the original images. It is interesting to consider what to do with the "holes" in the images. It seems necessary to distinguish two types of holes: background holes and foreground holes.

A background hole is a hole in a lumigraph that should remain a hole (i.e., the background should show through). When the lumigraph is composited on top of other lumigraphs, the other lumigraphs should be seen through the hole. These holes are easily represented using transparency in the lumigraph images.

A foreground hole is caused by an object that is in front of another. When rendering a lumigraph with foreground holes, these holes should be filled in using nearby images in the lumigraph. The rationale is that the foreground hole might contain partially occluded objects that are visible in other views. These views can be used to fill in the hole. A good example of this is a background lumigraph, which contains foreground holes where objects have been segmented out.

8.2 Conclusion

Unstructured lumigraph rendering possesses a number of properties that make it well-suited to many rendering tasks. For maximum flexibility, ULR admits unstructured inputs, both in terms of placement and number of images as well as in the pixel correspondence (e.g., scene geometry) specification. It also allows simple and intuitive navigation so that users have complete control over the virtual view. After applying number of optimizations, the algorithm is efficient enough to run in real-time, making it suitable for interactive applications.

Unstructured lumigraph rendering takes advantage of pixel correspondence information, when available, to generate quality renderings with sparse image information. It also

combines images in a view-dependent way to ensure quality rendering when geometric information is approximate or poor. Finally, ULR can take into account the limited field-of-view and resolution of real-world cameras. It automatically fills in invisible image regions with information from other images, and it disregards input images whose apparent resolution differ significantly from that of the desired view.

Unstructured lumigraph rendering is also very easy to extend to other modalities. Its simple notion of a viewing ray distance can be easily extended to include other factors, as demonstrated with time-periodic lumigraphs. Unstructured lumigraph rendering can also be modified to work with non-metric scene representations, further extending its usefulness. Although virtual navigation becomes much more difficult, a non-metric representation is still useful, as shown by applying it to the practical problem of video stabilization.

Bibliography

AVIDAN, S., AND SHASHUA, A. 1997. Novel view synthesis in tensor space. In *Proceedings of CVPR 1997*, 1034–1040.

CAMAHORT, E., LERIOS, A., AND FUSSELL, D. 1998. Uniformly sampled light fields. In *Eurographics Rendering Workshop 1998*, Springer Wein / Eurographics, Vienna, Austria, 117–130. ISBN 3-211-83213-0.

CHAI, J.-X., TONG, X., CHAN, S.-C., AND SHUM, H.-Y. 2000. Plenoptic sampling. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, 307–318. ISBN 1-58113-208-5.

CHANG, N. L., AND ZAKHOR, A. 1997. View generation for three-dimensional scenes from video sequences. *IEEE Trans. Image Process 6*, 4 (April), 584–598.

CHEN, S. E., AND WILLIAMS, L. 1993. View interpolation for image synthesis. In *Proceedings of SIGGRAPH 93*, Computer Graphics Proceedings, Annual Conference Series, 279–288. ISBN 0-201-58889-7.

CHEN, S. E. 1995. Quicktime VR - an image-based approach to virtual environment navigation. In *Proceedings of SIGGRAPH 95*, ACM SIGGRAPH / Addison Wesley, Los

Angeles, California, Computer Graphics Proceedings, Annual Conference Series, 29–38. ISBN 0-201-84776-0.

COWAN, R., AND TSANG, A. 1994. The falling-leaves mosaic and its equilibrium properties. *Adv. Appl. Prob. 26*, 54–62.

DARSA, L., SILVA, B. C., AND VARSHNEY, A. 1997. Navigating static environments using image-space simplification and morphing. In *1997 Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, 25–34. ISBN 0-89791-884-3.

DEBEVEC, P. E., AND MALIK, J. 1997. Recovering high dynamic range radiance maps from photographs. In *Proceedings of SIGGRAPH 97*, ACM SIGGRAPH / Addison Wesley, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, 369–378. ISBN 0-89791-896-7.

DEBEVEC, P. E., TAYLOR, C. J., AND MALIK, J. 1996. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proceedings of SIGGRAPH 96*, ACM SIGGRAPH / Addison Wesley, New Orleans, Louisiana, Computer Graphics Proceedings, Annual Conference Series, 11–20. ISBN 0-201-94800-1.

DEBEVEC, P. E., YU, Y., AND BORSHUKOV, G. D. 1998. Efficient view-dependent image-based rendering with projective texture-mapping. In *Eurographics Rendering Workshop 1998*, Springer Wein / Eurographics, Vienna, Austria, 105–116. ISBN 3-211-83213-0.

FAUGERAS, O., AND LAVEAU, S. 1994. Representing three-dimensional data as a collection of images and fundamental matrices for image synthesis. In *Proceedings of the International Conference on Pattern Recognition*, Computer Society Press, 689–691.

FAUGERAS, O. D., LUONG, Q.-T., AND MAYBANK, S. J. 1992. Camera self-calibration: Theory and experiments. In *Proceedings of European Conference on Computer Vision*, 321–334.

GORTLER, S. J., GRZESZCZUK, R., SZELISKI, R., AND COHEN, M. F. 1996. The lumigraph. In *Proceedings of SIGGRAPH 96*, ACM SIGGRAPH / Addison Wesley, New

Orleans, Louisiana, Computer Graphics Proceedings, Annual Conference Series, 43–54. ISBN 0-201-94800-1.

HARTLEY, R. I., DE AGAPITO, L., REID, I. D., AND HAYMAN, E. 1999. Camera calibration and the search for infinity. In *Proceedings of ICCV '99*, 510–517.

HARTLEY, R. 1993. Cheirality invariants. In *Proc. DARPA Image Understanding Workshop*, 745–753.

HEIGL, B., KOCH, R., POLLEFEYS, M., DENZLER, J., AND VAN GOOL, L. 1999. Plenoptic modeling and rendering from image sequences taken by hand-held camera. *Proc. DAGM '99*, 94–101.

HORN, B. K. P. 1986. *Robot Vision*. MIT Press.

HORRY, Y., ICHI ANJYO, K., AND ARAI, K. 1997. Tour into the picture: Using a spidery mesh interface to make animation from a single image. In *Proceedings of SIGGRAPH 97*, ACM SIGGRAPH / Addison Wesley, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, 225–232. ISBN 0-89791-896-7.

IRANI, M., ROUSSO, B., AND PELEG, S. 1994. Recovery of ego-motion using image stabilization. In *Proceedings of ICVPR '94*, 454–460.

ISAKSEN, A., MCMILLAN, L., AND GORTLER, S. J. 2000. Dynamically reparameterized light fields. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, 297–306. ISBN 1-58113-208-5.

LEE, A. B., HUANG, J., AND MUMFORD, D. 1999. Random-collage model for natural images. *(Submitted to) International Journal of Computer Vision.*

LEVOY, M., AND HANRAHAN, P. M. 1996. Light field rendering. In *Proceedings of SIGGRAPH 96*, ACM SIGGRAPH / Addison Wesley, New Orleans, Louisiana, Computer Graphics Proceedings, Annual Conference Series, 31–42. ISBN 0-201-94800-1.

LEVOY, M., PULLI, K., CURLESS, B., RUSINKIEWICZ, S., KOLLER, D., PEREIRA, L., GINZTON, M., ANDERSON, S., DAVIS, J., GINSBERG, J., SHADE, J., AND FULK, D. 2000. The digital Michelangelo project: 3D scanning of large statues. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, 131–144. ISBN 1-58113-208-5.

LHUILLIER, M., AND QUAN, L. 1999. Image interpolation by joint view triangulation. In *Proceedings of CVPR '99.*

MATUSIK, W., BUEHLER, C., AND MCMILLAN, L. 2001. Polyhedral visual hulls for real-time rendering. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, Eurographics, 115–126. ISBN 3-211-83709-4.

MCMILLAN, L., AND BISHOP, G. 1995. Plenoptic modeling: An image-based rendering system. In *Proceedings of SIGGRAPH 95*, ACM SIGGRAPH / Addison Wesley, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, 39–46. ISBN 0-201-84776-0.

MCMILLAN, L. 1996. *An Image-Based Approach to Three-Dimensional Computer Graphics*. PhD thesis, Dept. of Computer Science, University of North Carolina at Chapel Hill.

NAEMURA, T., TAGO, J., AND HARASHIMA, H. 2002. Real-time video-based modeling and rendering of 3d scenes. *IEEE Computer Graphics and Applications*, 66–73.

NAYAR, S. K. 1997. Catadioptric omnidirectional camera. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 1997.*

OH, B. M., CHEN, M., DORSEY, J., AND DURAND, F. 2001. Image-based modeling and photo editing. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press / ACM SIGGRAPH, Computer Graphics Proceedings, Annual Conference Series, 433–442. ISBN 1-58113-292-1.

OOI, R., HAMAMOTO, T., NAEMURA, T., AND AIZAWA, K. 2001. Pixel independent random access image sensor for real time image-based rendering. *IEEE Intern. Conf. Image Process. (ICIP2001) 20*, 3, 193–196.

PIGHIN, F., HECKER, J., LISCHINSKI, D., SZELISKI, R., AND SALESIN, D. H. 1998. Synthesizing realistic facial expressions from photographs. In *Proceedings of SIGGRAPH 98*, ACM SIGGRAPH / Addison Wesley, Orlando, Florida, Computer Graphics Proceedings, Annual Conference Series, 75–84. ISBN 0-89791-999-8.

POLLEFEYS, M., KOCH, R., AND VAN GOOL, L. 1999. Self-calibration and metric reconstruction in spite of varying and unknown internal camera parameters. *International Journal of Computer Vision 32*, 1, 7–25.

PULLI, K., COHEN, M. F., DUCHAMP, T., HOPPE, H., SHAPIRO, L., AND STUETZLE, W. 1997. View-based rendering: Visualizing real objects from scanned range and color data. In *Eurographics Rendering Workshop 1997*, Eurographics / Springer Wien, St. Etienne, France, 23–34. ISBN 3-211-83001-4.

RADEMACHER, P., AND BISHOP, G. 1998. Multiple-center-of-projection images. In *Proceedings of SIGGRAPH 98*, ACM SIGGRAPH / Addison Wesley, Orlando, Florida, Computer Graphics Proceedings, Annual Conference Series, 199–206. ISBN 0-89791-999-8.

RASKAR, R., BROWN, M. S., YANG, R., CHEN, W.-C., WELCH, G., TOWLES, H., SEALES, B., AND FUCHS, H. 1999. Multi-projector displays using camera-based registration. In *IEEE Visualization '99*, IEEE, San Francisco, California, 161–168. ISBN 0-7803-5897-X.

RUDERMAN, D. L. 1997. Origins of scaling in natural images. *Vision Research 37*, 23, 3385–3398.

SANDER, P. V., GU, X., GORTLER, S. J., HOPPE, H., AND SNYDER, J. 2000. Silhouette clipping. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH

/ Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, 327–334. ISBN 1-58113-208-5.

SCHARSTEIN, D. 1996. Stereo vision for view synthesis. In *Proceedings of CVPR '96*, 852–858.

SEITZ, S. M., AND DYER, C. R. 1996. View morphing: Synthesizing 3d metamorphoses using image transforms. In *Proceedings of SIGGRAPH 96*, ACM SIGGRAPH / Addison Wesley, New Orleans, Louisiana, Computer Graphics Proceedings, Annual Conference Series, 21–30. ISBN 0-201-94800-1.

SHADE, J., GORTLER, S. J., WEI HE, L., AND SZELISKI, R. 1998. Layered depth images. In *Proceedings of SIGGRAPH 98*, ACM SIGGRAPH / Addison Wesley, Orlando, Florida, Computer Graphics Proceedings, Annual Conference Series, 231–242. ISBN 0-89791-999-8.

SHEWCHUK, J. R. 1996. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. *First Workshop on Applied Computational Geometry*, 124–133.

SHI, J., AND TOMASI, C. 1994. Good features to track. In *1994 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'94)*, 593 – 600.

SHUM, H.-Y., AND HE, L.-W. 1999. Rendering with concentric mosaics. In *Proceedings of SIGGRAPH 99*, ACM SIGGRAPH / Addison Wesley Longman, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, 299–306. ISBN 0-20148-560-5.

SLOAN, P.-P., COHEN, M. F., AND GORTLER, S. J. 1997. Time critical lumigraph rendering. In *1997 Symposium on Interactive 3D Graphics*, ACM SIGGRAPH, 17–24. ISBN 0-89791-884-3.

SZELISKI, R., AND SHUM, H.-Y. 1997. Creating full view panoramic mosaics and environment maps. In *Proceedings of SIGGRAPH 97*, ACM SIGGRAPH / Addison Wesley, Los Angeles, California, Computer Graphics Proceedings, Annual Conference Series, 251–258. ISBN 0-89791-896-7.

SZELISKI, R. 1996. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications* (March), 22–30.

TRIGGS, B., MCLAUCHLAN, P., HARTLEY, R., AND FITZGIBBON, A. 2000. Bundle adjustment – A modern synthesis. In *Vision Algorithms: Theory and Practice*, W. Triggs, A. Zisserman, and R. Szeliski, Eds., LNCS. Springer Verlag, 298–375.

TRIGGS, B. 1996. Factorization methods for projective structure and motion. In *Proceedings of CVPR '96*, IEEE Computer Society Press, 845–851.

WILBURN, B., SMULSKI, M., LEE, K., AND HOROWITZ, M. A. 2002. The light field video camera. *Proceedings of Media Processors 2002, SPIE Electronic Imaging 2002.*

WOOD, D. N., AZUMA, D. I., ALDINGER, K., CURLESS, B., DUCHAMP, T., SALESIN, D. H., AND STUETZLE, W. 2000. Surface light fields for 3d photography. In *Proceedings of ACM SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, Computer Graphics Proceedings, Annual Conference Series, 287–296. ISBN 1-58113-208-5.

ZHANG, Z. 1998. A flexible new technique for camera calibration. *Microsoft Research Technical Report: MSR-TR-98-71* (December).