

SIGGRAPH 98
1998

15

Image-Based Modeling and Rendering



25th International Conference on Computer Graphics and Interactive Techniques
Exhibition **21-23 July** 1998 Conference **19-24 July** 1998
Orlando, Florida USA

course notes



15

Image-Based Modeling and Rendering

Organizers

Paul Debevec

University of California, Berkeley

Steven Gortler

Harvard University

Lecturers

Chris Bregler

Paul Debevec

University of California, Berkeley

Steven Gortler

Harvard University

Leonard McMillan

Massachusetts Institute of Technology

25th International Conference on Computer Graphics and Interactive Techniques

Exhibition **21-23 July 1998** Conference **19-24 July 1998**

Orlando, Florida USA

course notes

Image-Based Modeling and Rendering

Paul Debevec (co-organizer)
University of California at Berkeley

Steven Gortler (co-organizer)
Harvard University

Leonard McMillan
Massachusetts Institute of Technology

Richard Szeliski
Microsoft Corporation

Christoph Bregler
University of California at Berkeley

SIGGRAPH 98 Course 15 (Full Day)
Monday, July 20, 1998

Course Abstract

Image-based modeling and rendering differs from traditional graphics in that both the geometry and appearance of the scene are derived from real photographs. Within certain limitations, the technique allows for shorter modeling times, faster rendering speeds, and unprecedented levels of photorealism. In this course we will explain and demonstrate a variety of ways of turning photographs into models and then back into renderings, including movie maps, panoramas, image warping, photogrammetry, light fields, and 3D scanning. The course will overview the relevant topics in computer vision, and show how these methods relate to image-based rendering techniques. We will show ways of applying the techniques to animation as well as to 3D navigation. An underlying theme will be how the various modeling techniques make tradeoffs between navigability, geometric accuracy, manipulability, ease of acquisition, and level of photorealism. We will illustrate the described techniques with results from recent research as well as from creative applications.

Course Schedule and Syllabus

1. 08:30 - 09:00, 30 minutes (Debevec)

Introduction and Overview

1. What is Image-based modeling and rendering (IBMR)
2. The differences between Image-based modeling and rendering and traditional 3D graphics
3. Why this is a promising area
4. Some Examples
5. The spectrum of IBMR - from image indexing to 3D scanning
6. Advantages and disadvantages

2. 09:00 - 10:00, 60 minutes (Gortler)

Projective image warping

1. Simple projective geometry, and the pin-hole camera model
2. Image mosaicing and cylindrical panoramic viewing
3. Talisman affine sprite warping
4. Image caching techniques

Break

3. 10:15 - 11:20, 65 Minutes (McMillan)

Warping images with depth

1. Explanation of a depth map
2. Ways to warp an image based on depth
3. Panoramic image warping
4. Turning images and depth into a navigable environment
5. Advanced warping techniques and uncalibrated approaches

4. 11:20 - 12:00, 40 Minutes (Szeliski)

Recovering Geometry I

1. Why geometry is useful
2. Computer Vision as Inverse Computer Graphics
3. Notes camera calibration
4. Computing depth maps with stereo and multi-baseline stereo
5. Image correspondence techniques
6. Structure from Motion

Lunch

5. 01:30 - 02:00, 30 Minutes (Debevec)

Recovering Geometry II

1. Interactive photogrammetric methods
2. Structured Light and Laser Scanners

6. 02:00 - 03:00, 60 Minutes (Gortler)

Lightfield representations

1. The plenoptic function
2. Reduction to 4D

3. Holograms and the ambiguity between geometry and photometry
4. Light field rendering and the Lumigraph
 - methods for input
 - methods for rendering
5. Combining light fields with geometry
 - silhouette models (Lumigraph)
 - view-dependent texture-mapping (Facade)

Break

7. 03:15 - 03:50, 35 Minutes (Debevec)

Applications of IBMR in Art and Cinema

1. Matte paintings vs. 3D Models in Movies
2. The Aspen and San Francisco Movie Map projects (Lippman)
3. "Displacements" - physically projecting images onto geometry
4. Rouen Revisited (SIGGRAPH'96 art show), Like a Rolling Stone, The Campanile Movie, Tour into the Picture (SIGGRAPH '96, '97, '97 Electronic Theater)

8. 03:50 - 04:40, 50 Minutes (Bregler)

Applications of IBMR in human animation

1. How IBMR generalizes from 3D navigation to kinematic domains
2. Facial animation with image-based rendering
3. Human figure animation with image-based modeling

9. 04:40 - 05:00, 20 Minutes (Everyone)

Questions and Dialog

Presenters

Paul Debevec
Research Scientist
University of California at Berkeley
387 Soda Hall #1776
Computer Science Division, UC Berkeley
Berkeley, CA 94720-1776
(510) 642-9940
(510) 642-5775 Fax
debevec@cs.berkeley.edu
<http://www.cs.berkeley.edu/~debevec/>

Paul Debevec earned degrees in Math and Computer Engineering at the University of Michigan in 1992 and completed his Ph.D. at the University of California at Berkeley in 1996, where he is now a full-time researcher. His work in image-based modeling and rendering began in the summer of 1991 when he derived a realistic model of a 1980 Chevette from photographs for an animation project. Paul has collaborated on projects at Interval Research Corporation in Palo Alto that used a variety of image-based techniques for interactive applications; the piece "Rouen Revisited" showed at the SIGGRAPH'96 art show. His Ph.D. thesis under Professor Jitendra Malik and done with the collaboration of C.J. Taylor presented an interactive method of modeling architectural scenes from sparse sets of photographs and for rendering these scenes realistically. He has co-authored papers in both computer vision and computer graphics, including several SIGGRAPH papers relating to the course topic. Paul has assisted in teaching Berkeley's introductory computer graphics class and has spoken at a variety of venues on image-based topics. Recently, Paul helped create an image-based model of the Berkeley campus for a short film shown at the SIGGRAPH'97 Electronic Theater.

Steven J. Gortler
Assistant Professor
Harvard University
Eng. Sci. Lab 410
(617) 495-3751
(617) 496-1066
sjg@cs.harvard.edu
<http://hillbilly.eas.harvard.edu/~sjg/>

Steven J. Gortler received a B.A. in 1989 in Computer Science and Applied Mathematics from Queen's College, CUNY in 1989 and his Ph.D. in 1995 from Princeton University. His thesis, "Wavelet Methods for Computer Graphics", explored both geometric modeling and realistic rendering applications of the wavelet transform. In this work he also developed a framework for understanding matrix solution methods used for radiosity. As a Postdoctoral Researcher in the Microsoft Graphics group, he performed research in representing and rendering scenes from dense photographic data and performing efficient image-based rendering with layered depth images. Steven is now an Assistant Professor at Harvard University, where he has taught several courses in computer graphics, including a course devoted to image-based rendering.

Leonard McMillan
Assistant Professor
Massachusetts Institute of Technology
545 Technology Square
Cambridge, MA 02139
Office phone: (617) 258-0381
Fax: (617) 253-6652
email: mcmillan@graphics.lcs.mit.edu
<http://graphics.lcs.mit.edu/~mcmillan/>

Leonard McMillan is an assistant professor of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. He received B.S. and M.S. degrees in Electrical Engineering from the Georgia Institute of Technology in 1983 and 1984, and his Ph.D. in computer science in 1997 from the University of North Carolina at Chapel Hill. His experiences designing digital signal processing hardware have fueled his interest in making image-based rendering run at interactive speeds. His plenoptic modeling work from SIGGRAPH'95 demonstrated how the optical flow information derived from panoramic images could be used to simulate a three-dimensional immersive environments. Leonard is currently exploring new algorithms and hardware designs for the accelerating image-based rendering methods. He currently teaches introductory computer graphics and computer architecture and lectures on a wide range of issues related to image-based rendering.

Richard Szeliski
Senior Researcher
Microsoft Corporation, Vision Technology Group
One Microsoft Way
Redmond, WA 98052-6399
Office phone: (425) 936-4774
Fax: (425) 936-0502
email: szeliski@microsoft.com
<http://www.research.microsoft.com/research/vision/szeliski/>

Richard Szeliski is a Senior Researcher in the Vision Technology Group at Microsoft Research, where he is pursuing research in 3-D computer vision, video scene analysis, and image-based rendering. His current focus is on constructing photorealistic 3D scene models from multiple images and video, and on automatically parsing video for editing and retrieval applications. Dr. Szeliski received a B. Eng. degree in Honours Electrical Engineering from McGill University, Montreal, in 1979, a M. Appl. Sc. degree in Electrical Engineering from the University of British Columbia, Vancouver, in 1981, and a Ph. D. degree in Computer Science from Carnegie Mellon University, Pittsburgh, in 1988. He joined Microsoft Research in 1995. Prior to Microsoft, he worked at Bell-Northern Research, Montreal, at Schlumberger Palo Alto Research, Palo Alto, at the Artificial Intelligence Center of SRI International, Menlo Park, and at the Cambridge Research Lab of Digital Equipment Corporation, Cambridge. Dr. Szeliski has published over 60 research papers in computer vision, computer graphics, medical imaging, neural nets, and parallel numerical algorithms, as well as the book Bayesian Modeling of Uncertainty in Low-Level Vision. He is a member of the Association of Computing Machinery, the Institute of Electrical and Electronic Engineers, and Sigma Xi. He was an organizer of the first Workshop on Image-Based Modeling and Rendering, and is currently an Associate Editor of the IEEE Transactions on Pattern Analysis and Machine Intelligence.

Christoph Bregler

University of California at Berkeley

387 Soda Hall #1776

Computer Science Division, UC Berkeley

Berkeley, CA 94720-1776

(510) 642-9940

(510) 642-5775 Fax

bregler@cs.berkeley.edu

<http://www.cs.berkeley.edu/~bregler/>

Chris Bregler received his undergraduate degree from University of Karlsruhe in Germany in 1993, and is now completing his Ph.D. in computer science at the University of California at Berkeley. Before coming to Berkeley he started the computer lipreading project in Professor Alex Waibel's group (University of Karlsruhe / Carnegie Mellon University), and continued working on machine learning, lip-tracking, and speechreading systems with Steve Omohundro and Yochai Konig at ICSI in Berkeley. At Interval Research Corporation, together with Michele Covell and Malcolm Slaney, he created the Video Rewrite system that uses such analysis techniques and graphics techniques for image-based facial animation. As a completion to his thesis at U.C. Berkeley with advisors Jitendra Malik and Jerry Feldman he is working on applying new visual motion tracking techniques to image-based animation of entire bodies. His publications range from papers, book chapters, and patents in the field of Computer Vision, Computer Graphics, Speech Recognition, Statistical Learning, and Neural Networks.

Table of Contents

1. Introduction and Overview

Notes: Introduction to Image-Based Modeling and Rendering

Slides: What is Image-based Modeling and Rendering?

2. Projective Image Warping

Notes: Projective Image Warping

Paper: *Video Mosaics for Virtual Environments*
Richard Szeliski

Paper: *Plenoptic Modeling*
Leonard McMillan and Gary Bishop, Proc. SIGGRAPH 95

3. Warping Images with Depth

Notes: *Image-Based Rendering using Image Warping*

Notes: *Computing Visibility without Depth*

Slides: Image-Based Rendering using Image Warping

4. Recovering Geometry I

Slides: Camera Calibration

Slides: Passive Vision

Paper: *Non-Parametric Local Transforms for Computing Visual Correspondence*
Ramin Zabih and John Woodfill

Paper: *Stereo Matching with Transparency and Matting*
Richard Szeliski and Pollina Golland

5. Recovering Geometry II

Presentation: Modeling and Rendering Architecture from Photographs

Presentation: Active Shape Acquisition

Paper: *Modeling and Rendering Architecture from Photographs*
Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik, Proc. SIGGRAPH 96

Paper: *A Volumetric Method for Building Complex Models from Range Images*
Brian Curless and Marc Levoy, Proc. SIGGRAPH 96

6. Lightfield Representations

Slides: Polyhedral Geometry and the Two-Plane Parameterization

Slides: The Lumigraph

Paper: *Polyhedral Geometry and the Two-Plane Parameterization*
Xianfeng Gu, S. J. Gortler, and M. F. Cohen

Paper: *Light Field Rendering*
Marc Levoy and Pat Hanrahan, Proc. SIGGRAPH 96

Paper: *The Lumigraph*
S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, Proc. SIGGRAPH 96

7. Applications of IBMR in Art and Cinema

Slides: Applications of IBMR in Art and Cinema
Paper: *Elements of Realspace Imaging: A Proposed Taxonomy*
Michael Naimark
Notes: Rouen Revisited

8. Applications of IBMR in Human Animation

Notes: Video Based Animation Techniques for Human Motion
Slides: IBR Techniques for Non-Rigid Domains: -
Video-Based Animation of Human Motion
Paper: *Video Rewrite: Driving Visual Speech with Audio*
Christoph Bregler, Michele Covell, and Malcolm Slaney, Proc. SIGGRAPH 96
Paper: *Video Motion Capture*
Christoph Bregler Jitendra Malik

Introduction

What is Image-Based Modeling and Rendering?

Paul Debevec
University of California at Berkeley

A principal endeavor of computer graphics research has been the pursuit of photorealism. Early two-dimensional computer graphics gained a sense of depth by combining the simple algorithms for drawing lines with the mathematics of perspective projection. The wireframe look of such drawings fed a desire for a more solid appearance, which inspired the development of hidden surface removal algorithms. Shading algorithms allowed rendering surfaces with varying brightness levels as if they were being illuminated by sources of light, and shadow calculation techniques allowed objects to realistically cast shadows on each other. Techniques for representing and displaying curved surfaces expanded the variety of shapes that could be rendered, and we created modeling tools to help us generate complex models. Renderings that look as realistic as photographs have finally been achieved by using ray tracing and radiosity to simulate the myriad complex paths that light can take as it travels from its sources to the viewer.

The evolution of tools for modeling and rendering scenes with photorealistic fidelity - much of it represented in the twenty-five years of the SIGGRAPH conference - is a monumental achievement that has had an inestimable influence on the visual medium. Nonetheless, the tools for creating complex models require a great deal of effort and skill to use, and the algorithms for rendering such images with accurate illumination remain computationally intensive and are still somewhat experimental. To wit: modeling is hard, and rendering is slow. As a result, achieving truly compelling photorealism is extremely difficult.

Suppose, for example, that we wanted to generate a photorealistic image of the cathedral of Notre Dame in Paris. We could start by figuring out the dimensions of the cathedral, perhaps by borrowing the architectural plans from the most recent restoration project, or by conducting our own surveying. We would then build up the towers and the rose window, brick by brick and pane by pane, and assign appropriate reflectance properties to each surface. We could use L-systems to generate synthetic trees in the adjacent garden, and we could specify an appropriate distribution of incident light from the sky. We could then use a global illumination algorithm which, with a great deal of computation, would simulate how light would bounce around the scene to generate a rendered image of the cathedral.

Alternately, we could simply visit the cathedral and take a picture of it. Taking the picture would not only require far less effort, but the picture would almost certainly be a far more convincing rendition of the scene - it is, by definition, photorealistic. But while a single photograph gives us an amazing amount of information about the scene's structure and appearance, it is a static frozen image. What we have lost is the ability to look in different directions, to move about in the scene, to collide with its surfaces, to change the light, to add objects, and to modify the scene itself. If we had constructed the computer model, all of this would have been possible.

Image-based modeling and rendering is about leveraging the ease with which photographs can be taken, the speed at which they can be displayed, and their amazing power to communicate, while at the same time transcending their limitations. The various forms of IBMR transcend the limitations by deriving some sort of representation of the scene from the photographs, and then using this representation to create renderings. The principal reason that image-based modeling and rendering is interesting is that these representations do not need to be as complete as traditional computer graphics models in order to transcend many of the limitations of photographs. To remove the restriction that it is impossible to look in different directions, we can take photographs of the scene looking in all directions, assemble the photographs into a panorama, and then allow the user to look around by displaying different sections of the panorama. To remove the restriction that one can't move about the scene, we can take many images of the scene from different locations, and then display the various images depending on where the user wants to go. To reduce the number of images necessary, we can derive geometric representations of the scene through image correspondence, interactive photogrammetry, or active sensing, and then render this geometry from the

desired viewpoint texture-mapped with the original photographs. As the techniques for deriving representations become more sophisticated, the fewer limitations there are.

Image-Based Modeling and Rendering is a new field, but it has already produced degrees of interactivity and levels of photorealism previously thought impossible. With its current level of interest, it promises to continue to amaze us in the years to come. Furthermore, IBMR has the potential to fundamentally change the way we understand computer graphics. By starting with the answer - photorealistic renderings in the form of photographs and video - and discovering what it takes to transform them into models and then back into renderings, we have no choice but to gain an understanding of every perceptually relevant aspect of image synthesis.

A central goal of this course is to give a basic understanding of the variety of techniques that have been developed in image-based modeling and rendering. But the more important goal is to present the larger picture in which this variety of work can best be understood. To achieve this, an effort has been made to cover not just core material such as image warping and light fields, but to also present what lies near the frontier, such as movie maps, morphing, image-based human figure animation, and artistic applications. The result, I hope, will be a learning experience for all of us.

Paul Debevec
April 1998

Image-Based Modeling and Rendering

SIGGRAPH 98 Course 15
Monday, July 20, 1998

Paul Debevec
UC Berkeley

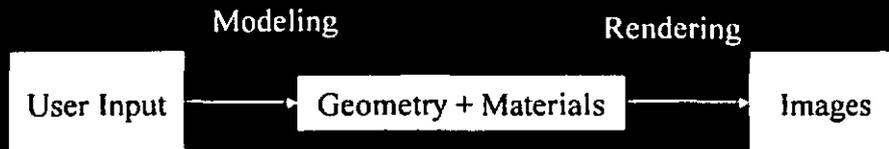
Steven Gortler
Harvard

Leonard McMillan
MIT

Chris Bregler
UC Berkeley

Richard Szeliski
Microsoft

Traditional Modeling and Rendering



- For photorealism,
 - Modeling is hard
 - Rendering is slow

Image-Based Modeling

- Images are used to determine
 - Scene Geometry
 - Scene Appearance
 - Kinematic Properties
 - Reflectance Characteristics
- Modeling scenes photorealistically is easier

Image-Based Rendering

- Appearance in available views is used to determine appearance in novel views =>
- Rendering is faster

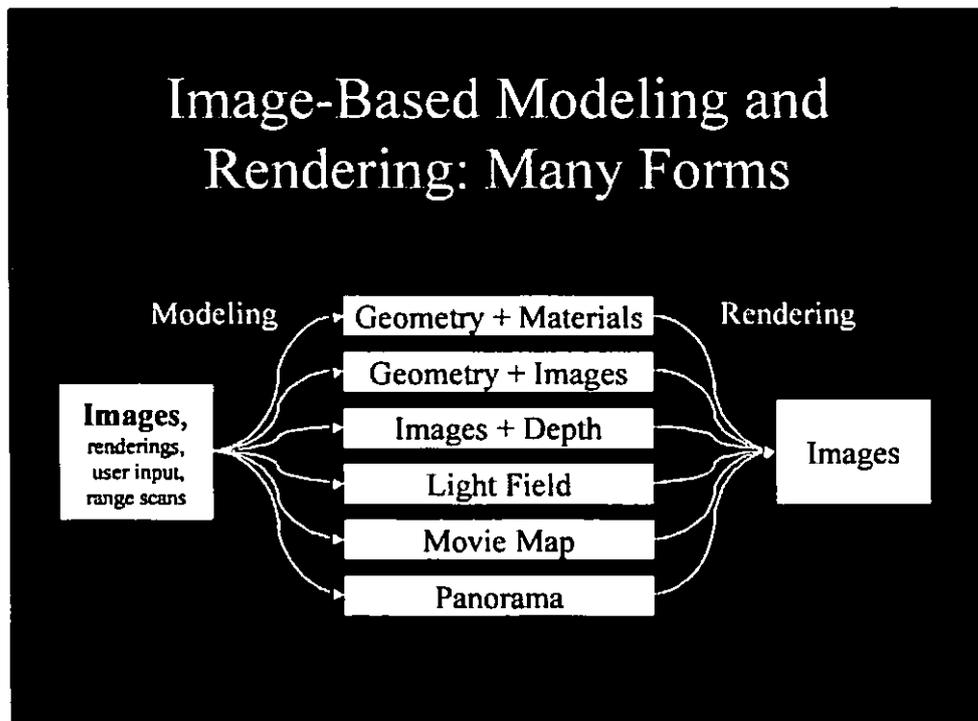
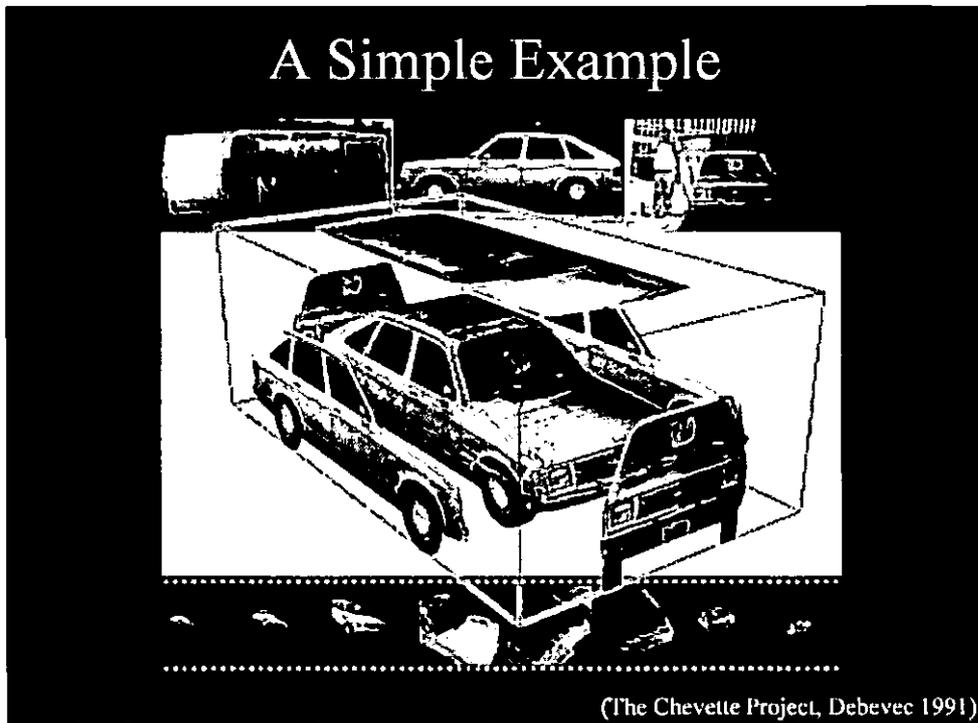


Image-Based Representations: Comparison

Representation	Movement	Geometry	Lighting
Geometry + Materials	Continuous	Global	Dynamic
Geometry + Images	Continuous	Global	Fixed
Images + Depth	Continuous	Local	Fixed
Light Field	Continuous	None	Fixed
Movie Map	Discrete	None	Fixed
Panorama	None	None	Fixed

Projective Image Warping

As suggested by its name, image based rendering is concerned with creating new images from old ones. Recently there have been lots of descriptions of novel representations, such as light-fields, and the warping of depth images to produce 3d like parallax effects. But the basic idea of using old images to make new ones is quite old. In particular the use of texture mapping takes input image data (the texture), places it in some virtual 3d world, and views it (texture mapping) on the image plane. The original texture image is warped onto the new image plane using what is called a projective warp.

In this section we will discuss a simple family of warps called projective warps. These warps do not use any sort of depth or optical flow information. Warps that use depth and flow will be discussed in detail later in the course. We will demonstrate two powerful ideas. First that if one views a planar object using two (or more) pin hole cameras, then those two images are related by a projective warp; ie. if we apply the warp to the first image, we obtain the second image. This result is at the core of texture mapping. In texture mapping, we use one image, the texture, to represent the photometric appearance of a (nearly) planar object, and this is viewed from an arbitrary viewpoint to create a second image.

The second idea about projective warps we will demonstrate is that if one creates two (or more) images from a single center of projection (the camera only rotates, and changes its intrinsic properties, such as its zoom), then those two images are related by a projective warp. This result is at the core of environment mapping and panoramic images. If one takes a number of pictures from a single point of view, one can stitch this panorama of images together using a projective warp. From this data one can simulate a new view from the same center of projection by applying the projective warp.

In this section we will discuss two types of applications and benefits of simple warps. These two types applications run through the core of most IBR work. First image warping lets us use photographs of real world scenes in an interactive graphics setting. Second, image warping is fast, and can allow us to render imagery faster than other techniques.

First, simple image warping allows us to easily model complicated real world appearances that may be difficult to model manually. By using a photograph as a texture, one effectively captures real world photometric information, and places it in a virtual environment. Projective warps know nothing about the depths of objects in the world, and so they can be applied to real photographs without any sort of active scanning techniques or vision-type analysis. We discuss texture mapping and how it is a simple example of a projective image transform. We discuss environment maps and image panoramas which give a simple description of the world as seen looking out from a single point. These panoramas offer a simple way to photographically capture interesting real world environments in which to place a virtual user.

The second benefit of simple warps is fast rendering. Image warping involves simple and regular computation. It can therefore be computed very efficiently, especially with the use of hardware. Texture mapping is useful, even if the texture is generated by a computer, and not from a photograph. This efficiency of image warping is used to render complicated appearance, (even synthetically modeled appearance) more quickly than if we were to use simpler rendering techniques, such as smoothly shaded polygon rendering.

These simple types of warps can also be used to speed up the rendering of purely synthetic environments. In typical virtual settings, as the user or objects move around, many parts of the image

stay the same, or perhaps warp in simple ways. Various rendering systems have been designed to make use of this temporal coherence by essentially “reusing” various image regions. These systems rely on similar types of image warps seen in texture mapping and environment mapping. In this section we will discuss these systems as well.

Finally, to draw the connection between projective warping, and image based methods that use depth information, we will discuss the relationship between these warping methods and the warping methods that have motion or depth information associated with the pixels.

Basics: Projective Maps

A projective map is a mapping that maps from one image coordinate system (x_{i1}, y_{i1}) to another image coordinate system (x_{i2}, y_{i2}) . It is computed multiplying the first coordinates by a 3 by 3 matrix, and dividing out by r .

$$\begin{bmatrix} x_{i2}r \\ y_{i2}r \\ r \end{bmatrix} = \begin{bmatrix} m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 \\ m_7 & m_8 & m_9 \end{bmatrix} \cdot \begin{bmatrix} x_{i1} \\ y_{i1} \\ 1 \end{bmatrix}$$

Explicitly this can be written out as

$$\begin{aligned} x_{i2} &= \frac{m_1x_{i1} + m_2y_{i1} + m_3}{m_7x_{i1} + m_8y_{i1} + m_9} \\ y_{i2} &= \frac{m_4x_{i1} + m_5y_{i1} + m_6}{m_7x_{i1} + m_8y_{i1} + m_9} \end{aligned}$$

Note that if we multiply the whole matrix by a scale factor, we will get the same transformation, and so there are really eight degrees of freedom specifying the map. In general during texture mapping, the mapping from texture coordinates to the new image is a projective transformation.

There is a special simple subfamily of projective maps that come from matrices with a bottom row of 0, 0, 1 called affine maps.

$$\begin{bmatrix} x_{i2} \\ y_{i2} \\ 1 \end{bmatrix} = \begin{bmatrix} m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_{i1} \\ y_{i1} \\ 1 \end{bmatrix}$$

Affine maps can translate, scale, skew, and rotate 2d images. A full projective map can do fancier things, like make parallel lines converge to a point. This is necessary to get effects like foreshortening. Affine maps can be computed a bit quicker than projective maps, because no division is required.

Basics: Pin hole cameras

In graphics we take the three dimensional world, and project it down to a two dimensional image using a pin-hole camera model. This is described using matrix operations.

Computer graphics practitioners like to have an image space z value after projection, in order to compute the visible surface, so they tend to use four by four matrices. During rendering, some

point in space (x_g, y_g, z_g) , perhaps some polygon vertex is mapped to “image coordinates” (x_i, y_i, z_i) using the four by four matrix C .

$$\begin{bmatrix} x_i w_i \\ y_i w_i \\ z_i w_i \\ w_i \end{bmatrix} = C \cdot \begin{bmatrix} x_g \\ y_g \\ z_g \\ 1 \end{bmatrix}$$

The four by four matrix $C = V \cdot F \cdot E$ can be considered a composition of three matrices. E is a Euclidian transform expressing a rotation and a translation

$$E = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_1 \\ r_{31} & r_{32} & r_{33} & t_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

F expresses the projection using a camera frustum projection. And V , a viewport transformation maps us to pixel coordinates. In order to extract the actual image coordinates, one needs to divide out the w_i coordinate.

Computer vision practitioners tend to describe real cameras, that don't keep around any z values, and as such they tend to describe the imaging process as

$$\begin{bmatrix} x_i w_i \\ y_i w_i \\ w_i \end{bmatrix} = P \cdot \begin{bmatrix} x_g \\ y_g \\ z_g \\ 1 \end{bmatrix}$$

$P = A \cdot N \cdot E$ is a 3 by 4 matrix, composed of the following parts. The position and orientation of the camera is determined by E . The canonical camera projection is described by

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

And the intrinsic camera parameters are defined by the three by three upper diagonal matrix matrix

$$A = \begin{bmatrix} -f s_x & \theta & t_x \\ 0 & -f s_y & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

Where f is the focal length, s_x, s_y are horizontal and vertical pixel scales, t_x, t_y positions the focal center, and θ allows for skewed, non-rectangular pixels.

Texture Mapping

In this section we briefly visit the topic of texture mapping and describe how it is an application of a projective mapping operation.

Texture mapping is a process where we place some interesting looking pattern onto a polygon in space. This pattern may be computer generated, hand drawn, or be a photograph. The pattern is represented as an image made up of pixels called texels. During the rendering process, these texels are conceptually mapped onto the polygon and then viewed through the pin hole camera and viewed on the screen. This can be expressed in matrices as [1]

$$\begin{bmatrix} x_i w_i \\ y_i w_i \\ w_i \end{bmatrix} = P_{3 \times 4} \cdot G_{4 \times 3} \cdot T_{3 \times 3} \cdot \begin{bmatrix} x_t \\ y_t \\ 1 \end{bmatrix}$$

The 3 by 3 T matrix scales and translates and rotates the texture to put it on the two dimensional polygon; this is an affine map. The 4 by 3 matrix G takes the two dimensional polygon and puts it into 3d space. This is also an affine map. The 3 by 4 matrix P is the pin hole projection that maps the 3d geometry onto the image plane. The concatenation of these three matrices is a three by three matrix $M = P \cdot G \cdot T$. This matrix represents the transformation from texture space to image space. Thus texture mapping is a projective map from texture space to image space using a 3 by 3 matrix M .

In a real graphics language, such as OpenGL, one does not specify M directly. P is specified by the camera commands. M is specified by the placement of the polygon in the world. T is specified by specifying texture coordinates (x_t, y_t) at the three vertices of a triangle.

The above mapping describes the map from texture to image. Typically, in a texture mapping implementation, we in fact use the inverse map, that maps from image domain to texture domain. For each image pixel location (x_i, y_i) , we compute the mapped texture location, and then the proper color value is interpolated.

Texture mapping can be implemented quite efficiently. During the rendering, as the renderer moves across a scan line, the matrix multiply can be evaluated incrementally with a few add operations, and the division by the third coordinate.

Two views of a planar object

In this section we show how two views of a planar object, such as the facade of a building, are related by a projective map. If I take a picture of a facade, and wish to view it from a different position in space, I can create the new view by applying a projective map to the first image.

Suppose one takes a picture of a planar object, and without loss of generality lets assume that the plane is defined by $z_g = 0$, then the image of the planar points in camera 1 are

$$\begin{bmatrix} x_{i1} w_{i1} \\ y_{i1} w_{i1} \\ w_{i1} \end{bmatrix} = P_1 \cdot \begin{bmatrix} x_g \\ y_g \\ 0 \\ 1 \end{bmatrix}$$

By dropping the third column of P_1 , we can obtain a three by three matrix M_1 that maps points on the planar object to the image plane.

$$\begin{bmatrix} x_{i1} w_{i1} \\ y_{i1} w_{i1} \\ w_{i1} \end{bmatrix} = M_1 \cdot \begin{bmatrix} x_g \\ y_g \\ 1 \end{bmatrix}$$

Suppose the same plane is viewed from a second camera in a different location, apply the same reasoning to obtain

$$\begin{bmatrix} x_{i2} w_{i2} \\ y_{i2} w_{i2} \\ w_{i2} \end{bmatrix} = M_2 \cdot \begin{bmatrix} x_g \\ y_g \\ 1 \end{bmatrix}$$

Assuming the M matrices are invertible, we obtain

$$\begin{bmatrix} x_{i1} w_{i1} \\ y_{i1} w_{i1} \\ w_{i1} \end{bmatrix} = M_1 M_2^{-1} \cdot \begin{bmatrix} x_{i2} w_{i2} \\ y_{i2} w_{i2} \\ w_{i2} \end{bmatrix}$$

Dividing both sides by w_{i2} gives us

$$\begin{bmatrix} x_{i1} r \\ y_{i1} r \\ r \end{bmatrix} = M_1 M_2^{-1} \cdot \begin{bmatrix} x_{i2} \\ y_{i2} \\ 1 \end{bmatrix}$$

Hence we see that one can map the view of a planar object from one camera to another camera, simply by multiplying the first image coordinates by a three by three matrix, and dividing through by r .

Returning to our simple example, If I take a picture of a facade, and wish to view it from a different position in space, I can create the new view by applying a projective map to the first image. This can be thought of as essentially a texture mapping operation. One models the 3d geometry of the planar facade, one specifies that the texture should be mapped onto that plane, and then one views it from a some virtual camera. But the situation is slightly trickier. In simple texture mapping $M = P \cdot G \cdot T$, the T matrix representing the mapping of the texture onto the geometry, was assumed to be an affine map. This mapping was specified by specifying the (x_t, y_t) texture coordinates at 3 vertices of a triangle. This affine does not allow for perspective effects in the original texture. In our case here, the texture itself is a photograph with pin hole perspective. We really need the ability to specify a general projective transform for T . OpenGL allows us to do this in two different ways. The first way is to directly manipulate the so called texture matrix stack. The other way is to pass three texture coordinates $(x_t \cdot q, y_t \cdot q, q)$ per triangle vertex instead of two. This q essentially accounts for the divide of the mapping M_2 , and so it should be set to w_{i2}

Environment Mapping and Panoramas

Another quite similar technique to texture mapping is environment mapping. The basic idea of environment mapping is that as I look out into my environment from some single center of projection point, I can measure what is seen in each direction, and represent that in some appropriate data structure. This data structure organizes the data sampled at the various directions. Examples for the organizing data structures include cubes, spheres, and cylinders.

Graphics systems have used environments maps as ways of simply representing the distant part of the world, either for direct viewing, or for approximate reflection computation. More recently, people have been interested in capturing environment maps photographically, and then using this data to place a user in a virtual version of a real world environment.

The types of projections that let us do texture mapping, also allow us to do environment mapping. If I take a camera and move it back and forth, parallax occurs and certain objects can occlude one another. But when I take a camera and rotate it around its center of projection, this cannot occur. All I can see in these cameras is what can be seen by the pencil of rays that meet at the center of projection. This suggests that there should be a way to map the pixels in one view to pixels in the other. This is in fact true, and can be achieved by a projective transformation.

We can see this with an argument similar to the one used for planar objects

Suppose one has a single camera, and without loss of generality, lets us place it at the origin, then the image of the scene points in camera 1 are

$$\begin{bmatrix} x_{i1} w_{i1} \\ y_{i1} w_{i1} \\ w_{i1} \end{bmatrix} = A_1 \cdot N \cdot \begin{bmatrix} R_1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_g \\ y_g \\ z_g \\ 1 \end{bmatrix} = A_1 \cdot R_1 \cdot \begin{bmatrix} x_g \\ y_g \\ z_g \end{bmatrix}$$

where R_1 is the 3 by 3 matrix describing the rotation of the first camera. Hence the mapping from world coordinates to camera 1 coordinates is described by a simple 3 by 3 matrix $A_1 \cdot R_1$

If we have a second camera with the same center of projection, namely the origin, then the mapping from world to camera 2 coordinates is also described by a 3by3 matrix $A_2 \cdot R_2$. As a result, one can map from image 2 coordinates to image 1 coordinates by multiplying by the 3by3 matrix $A_1 R_1 R_2^{-1} A_2^{-1}$, and dividing by the homogeneous coordinate. Hence we conclude *two views of a scene from a single center of projection are related by a projective transformation.*

Image Mosaics

Image mosaics and panoramic images are based on the idea that all images that share a center of projection can be easily warped to each other's coordinate systems. A user takes multiple pictures from a common point. These images are then warped into a common coordinate system. This represents a wide field of view, or perhaps a full surround view from this point. New planar images can be constructed from this representation.

Here we sketch the basic steps in creating a planar image mosaic.

Step1: take multiple pictures from a single center of projection. If one wants to be very careful about this, one should use a tripod. As one spins the camera, one should check to see that there is no parallax (occlusion changes) occurring. If there is parallax, one must adjust the camera on the tripod so that it rotates about a different point.

Step2: Pick a single image to be the reference view. All of the other images will be mapped to this view.

Step3: For each other image, I , determine the projective transform that maps I to the reference view coordinate system. This transform will make the overlapping parts of the two images align perfectly. There are a few ways this transform could be computed. If the user marks four corresponding points in the reference image, and in I , then one can directly solve for the appropriate matrix. Another method is to use numerical optimization to find the correct matrix [9]. Essentially one solves for the matrix such that after warping, the region of the I , that overlaps with the reference image is as similar as possible.

If an image does not contain significant overlapping data with the reference view, then we cannot directly compute the required projective transform. But this is not a problem. As long as I

has overlap with an image J , and we know the correct transformation from J to the reference view, then we can compute the transform from I 's coordinate system to J 's, and compose that with the transform from J to the reference view.

step4: Warp all images to the reference view's coordinate system. Resampling will be necessary since the warped sample locations will not generally lie on integer values.

Panoramas

There are a few limitations to the image mosaicing approach outlined above. First it artificially picks a single view to be the reference coordinate system. Secondly a single image coordinate system can at best represent a 180 degree field of view. Thirdly, in a typical example, the images are gathered by rotating a camera around a single axis, and so there is really only one unknown degree of freedom specifying the relationship between any two images, not eight.

To solve for the limited field of view, global data structures, such as a cylinder, sphere, or cube can be used to represent the view of the world from a single point. If one knows the intrinsic and extrinsic camera parameters for each image (ie. the P matrix), then one knows the mapping between pixels in the image, and rays in the world, and can easily map them onto the global data structure. If one does not know this information, then one must solve this information from the given views. Details of how to do this are described for example in [5] and [8]. In these algorithms, assumptions are made on the real degrees of freedom, such as that the camera does not change its intrinsic parameters as the pictures are taken, and that it may only be rotating about the y axis.

Image reuse for fast rendering

Image warping techniques are used not only to capture "real" world environments, but also to speed up the rendering of synthetic scenes. During many rendering sequences, subsequent frames can be quite similar. As a user translates, distant objects don't appear to move at all. As a user rotates, distant objects move simply across the screen with almost no parallax effects. A number of rendering algorithms have been developed that take advantage of this coherence by reusing image regions over a series of frames. When a frame is generated, some parts of the scene are rendered simply by warping the associated pixels from the previous frame. Parts of the image where the visual changes are dramatic are re-rendered directly from the geometry. There are a variety of issues in determining which scene elements to image warp, and which ones to redraw from geometry. There are also issues determining what is the best warp to use for the reused regions.

Shade et al. [6] describe a system for accelerating walkthroughs of complex environments. In their system, the objects in the environment are organized into a spatial BSP tree. Each BSP tree leaf node contains an object, and the internal nodes hierarchically contain collections of objects. When the first frame of a sequence is rendered all of the objects are rendered from their geometry. Each of these images is stored as an "image cache". They also store image caches for the hierarchy of object collections corresponding to the internal BSP tree nodes.

When the user moves, the BSP tree of the scene is traversed to re-render the scene. For each node, an error criteria is checked to see if the corresponding image cache can be reused. To reuse the image cache, it is used as a texture that is mapped on to a planar billboard in space. Their error metric approximates the discrepancy between the correct re-rendered object and the appearance of

the texture mapped billboard. If the error of using the image cache is too high, then the object is rendered by recursively calling the algorithm on the BSP children of this node. When a BSP leaf fails the error measure, then the appropriate geometry is rerendered. In effect what happens is that nearby objects get rerendered from the geometry almost at every frame. As we consider objects further away from the viewer, they get rerendered less often, and their billboards are just rendered as texture maps. At very far distances, large clusters of objects are rerendered as a single textured billboard.

The Talisman rendering architecture [10] is a hardware rendering architecture based on the concept of image reuse. The hardware has a standard polygon renderer that renders each object into its own sprite. Another component of the hardware takes sprites, applies an arbitrary affine warp to it, and composites it onto the image screen. As a geometric object moves about a scene, its sprite can be affinely warped (ex. scaled and translated) to approximate the appearance of the proper motion. If the affine transform is a poor approximation to the proper appearance of the object, its sprite can be updated, by rerendering it in the rendering hardware component.

In Talisman, these two hardware components, the renderer and the warper, are decoupled. The warper runs at frame rate, and must warp and composite each sprite per frame. The renderer updates sprites when requested, and does so as fast as it can. A controlling program decides which sprites are invalid in each frame must be rendered, typically, only a fraction of the sprites are updated in a frame. The controlling program also decides what is the best affine warp to use, and passes that information onto the warping engine. Lengyel and Snyder [3] describe the systematic way in which such a program can decide on the best affine warp. They also perform a comparison between using affine warps, and the full power of a projective transform. Their experiments find that the full projective transform does not add a significant advantage over affine transforms. Affine transforms are much cheaper to implement, since they require no divide operations.

Warping with depth

In subsequent sections of this course, we will see uses of warping that use depth and flow type information. These warps are more general than the projective transforms discussed so far. Projective warps can in effect only simulate the motion of planar geometry. This is a very poor approximation for nearby non planar objects. In this section we bridge the gap by briefly discussing these types of warps.

In a real scene as a user's view translates, different objects appear to move at different speeds. Closeby objects move quickly, while distant objects appear not to move at all. To achieve this type of effect by image warping, one must use a more complicated warp that has can achieve the proper optical flow. If one knows the intrinsic parameters of some virtual or real camera, and one knows the actual depth (z) values at each pixel, then warping to some new known view is straightforward. If the four by four matrix C_1 describes the first camera, and C_2 , the desired view, we have

$$\begin{bmatrix} x_{i1} w_{i1} \\ y_{i1} w_{i1} \\ z_{i1} w_{i1} \\ w_{i1} \end{bmatrix} = C_1 \cdot \begin{bmatrix} x_g \\ y_g \\ z_g \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x_{i2}w_{i2} \\ y_{i2}w_{i2} \\ z_{i2}w_{i2} \\ w_{i2} \end{bmatrix} = C_2 \cdot \begin{bmatrix} x_g \\ y_g \\ z_g \\ 1 \end{bmatrix}$$

and so

$$\begin{bmatrix} x_{i2}r \\ y_{i2}r \\ z_{i2}r \\ r \end{bmatrix} = C_2C_1^{-1} \cdot \begin{bmatrix} x_{i1} \\ y_{i1} \\ z_{i1} \\ 1 \end{bmatrix}$$

Like texture mapping, much of the computation of this warp can be done incrementally. As one moves across across a scanline of image 1 and warps the pixels, x_{i1} is simply incremented and y_{i1} remains the constant, and so most of the multiplies can be avoided. In contrast, z_{i1} can change arbitrarily and so these multiplies must be performed per pixel. And like texture mapping, one must perform the divide per pixel.

Computationally, there a big difference between these warps and projective warps. Projective warps are easily invertible, simply by using the inverse of the 3 by 3 matrix. Inverse warping is very useful. It allows us to warp each output pixel location to some position in the input image. One can then interpolate the color at that position. In the depth warp describe here, z_{i1} can change arbitrarily. As a result, it is difficult to compute an inverse map that goes from $i2$ coordinates to $i1$ coordinates. This means that we can't simply interpolate image 1 pixel colors for each output pixel as done in texture mapping. There have been a variety of approaches taken to this problem include forward splatting type methods [7], micropolygon scan conversion methods[4], and more complicated inverse warping methods [2].

Wrap Up

Image based rendering is concerned with the use of using input images to create output images. The images are typically reused using some type of warp. Many image based method use simple warps that do not need any depth for flow per pixel information. We have focused on one such family of warps, the projective warp. Projective warps are simple warps computed by multiplying with a 3 by 3 matrix and dividing by a third coordinate.

Projective warps are fundamental to computer graphics, and are at the root of texture mapping and environment mapping. More recently, These techniques have been used to reuse photographs of real world scenery. In particular, simple warps are used to create and render from image mosaics and panoramas.

Projective warps have been used recently in new rendering systems that try to reuse much of the imagery from frame to frame. These systems attempt to approximate the view of some of the objects by warping those pixels to the new view. Where it is determined that the warping approximation is too inaccurate, traditional rendering is used. These kinds of systems are important methods for enabling the interactive rendering of complicated environments.

Given a z value stored at each pixel, one can compute where that point would be viewed in a second image. This can be used to drive a more complicated kind of image warping. People

are currently understanding how these kinds of warps can be used to allow us to virtually interact with real world captured images. These kinds of warps are also being investigated for image reuse rendering systems.

References

- [1] P. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, The University of California at Berkeley, June 1989.
- [2] S. Laveau and O. Faugeras. 3-D scene representation as a collection of images and fundamental matrices. Technical Report 2205, INRIA-Sophia Antipolis, February 1994.
- [3] Jed Lengyel and John Snyder. Rendering with coherent layers. *SIGGRAPH 1997*, pages 233–242.
- [4] William R Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3d warping. *Proceedings 1997 Symposium on Interactive 3D Graphics*, pages 7–16.
- [5] Leonard Mcmillan and Gary Bishop. Plenoptic modeling: an image based rendering system. *SIGGRAPH 1996*, pages 39–46.
- [6] Johnathan Shade, Dani Lischinski, David Salesin, Toney DeRose, and John Snyder. hierarchical image caching for accelerated walkthroughs of complex environments. *SIGGRAPH 1996*, pages 75–82.
- [7] Jonathan Shade, Steven Gortler, Li wi He, and Richard Szeliski. Layered depth images. *SIGGRAPH 1998*.
- [8] Richard Szeliski and H. Shum. Creating full view panoramic image mosaics and environment maps. *SIGGRAPH 1997*, pages 251–258.
- [9] Rick Szeliski. Video mosaics for virtual environments. *IEEE CG&A*, pages 22–30, march 1996.
- [10] Jay torborg and James Kajiya. Talisman: Commodity realtime 3d graphics for the pc. *SIGGRAPH 1996*, pages 353–364.

Video Mosaics for Virtual Environments

Richard Szeliski, *Microsoft Corporation*

By panning a camera over a scene and automatically compositing the video frames, this system creates large panoramic images of arbitrary shape and detail. Depth recovery from motion parallax also enables limited 3D rendering.

The use of photographic imagery as part of the computer graphics creation process is a well established and popular technique. Still imagery can be used in a variety of ways, including the manipulation and compositing of photographs inside video paint systems, and the texture mapping of still photographs onto 3D graphical models to achieve photorealism. Although laborious, it is also possible to merge 3D computer graphics seamlessly with video imagery to produce dramatic special effects. As computer-based video becomes ubiquitous with the expansion of transmission, storage, and manipulation capabilities, it will offer a rich source of imagery for computer graphics applications.

This article looks at one way to use video as a new source of high-resolution, photorealistic imagery for these applications. In its current broadcast-standard forms, video is a low-resolution medium that compares poorly with computer displays and scanned imagery. It also suffers, as do all input imaging devices, from a limited field of view. However, if you walked through an environment, such as a building interior, and filmed a video sequence of what you saw, you could subsequently register and composite the video images together into large mosaics of the scene. In this way, you can achieve an essentially unlimited resolution. Furthermore, since you can acquire the images using any optical technology (from microscopy to hand-held videocams to satellite photography), you can reconstruct any scene regardless of its range or scale.

Video mosaics can be used in many different applications, including the creation of virtual reality environments, computer-game settings, and movie special effects. Such applications commonly use an *environment map*—that is, a 360-degree spherical image of the environment—both to serve as a backdrop and to correctly generate reflections from shiny objects.¹

In this article, I present algorithms that align images and composite scenes of increasing complexity—beginning with simple planar scenes and progressing to panoramic scenes and, finally, to scenes with depth variation. I begin with a review of basic imaging equations and conclude with some novel applications of the virtual environments created using the algorithms presented.

Basic imaging equations

The techniques developed here are all based on the ability to align different pieces of a scene (tiles) into a larger picture of the scene (mosaic) and then to seamlessly blend the images together. In many ways, this resembles current image morphing techniques,² which use a combination of image warping³ and image blending.⁴ To automatically construct virtual environments, however, we must automatically derive the alignment (warping) transformations directly from the images, rather than relying on manual intervention.

Before proceeding, we need to consider the geometric transformations that relate the images to the mosaic. To do this, we use *homogeneous coordinates* to represent points, that is, we denote 2D points in the image plane as (x, y, w) . The corresponding Cartesian coordinates are $(x/w, y/w)$.⁴ Similarly, 3D points with homogeneous coordinates (x, y, z, w) have Cartesian coordinates $(x/w, y/w, z/w)$.

Using homogeneous coordinates, we can describe the class of 2D planar projective transformations using matrix multiplication:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ m_6 & m_7 & m_8 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix} \quad \text{or} \quad \mathbf{u}' = \mathbf{M}\mathbf{u} \quad (1)$$

The simplest transformations in this general class are pure translations, followed by translations and rotations (rigid transformations), plus scaling (similarity transformations), affine transformations, and full projective transformations. Figure 1 shows a square and possible rigid, affine, and projective deformations. Forms for the rigid and affine transformation matrix \mathbf{M} are

$$\mathbf{M}_{\text{rigid-2D}} = \begin{bmatrix} \cos \theta & -\sin \theta & t_x \\ \sin \theta & \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix},$$
$$\mathbf{M}_{\text{affine-2D}} = \begin{bmatrix} m_0 & m_1 & m_2 \\ m_3 & m_4 & m_5 \\ 0 & 0 & 1 \end{bmatrix}$$

with 3 and 6 degrees of freedom, respectively, while projective transformations have a general \mathbf{M} matrix with 8 degrees of freedom. (Note that two \mathbf{M} matrices are equivalent if they are scalar multiples of each other. We remove this redundancy by setting $m_8 = 1$.)

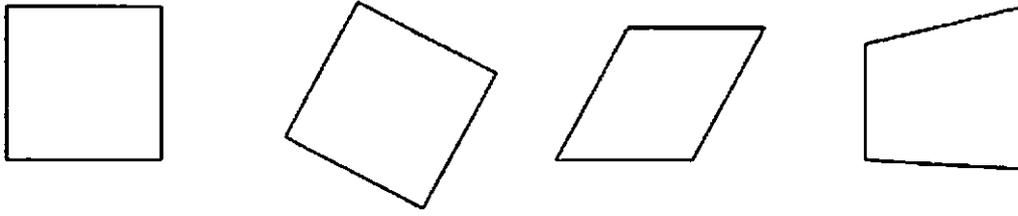


Figure 1. Square and rigid, affine, and projective transformations.

The same hierarchy of transformations exists in 3D, with rigid, similarity, affine, and full projective transformations having 6, 7, 12, and 15 degrees of freedom, respectively. The \mathbf{M} matrices in this case are 4×4 . Of particular interest are the rigid (Euclidean) transformation

$$\mathbf{E} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \quad (2)$$

where \mathbf{R} is a 3×3 orthonormal rotation matrix and \mathbf{t} is a 3D translation vector, and the 3×4 viewing matrix

$$\hat{\mathbf{V}} = [\mathbf{V} \ \mathbf{0}] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \quad (3)$$

which projects 3D points through the origin onto a 2D projection plane a distance f along the z axis.⁴

(Note that a more general camera model, where \mathbf{V} is an upper triangular matrix, can also account for aspect ratio, an offset optical center, and skew. A real camera might also have optical distortions that do not follow the pinhole model.)

The combined equations projecting a 3D world coordinate $\mathbf{p} = (x, y, z, w)$ onto a 2D screen location $\mathbf{u} = (x', y', w')$ can thus be written as

$$\mathbf{u} = \hat{\mathbf{V}}\mathbf{E}\mathbf{p} = \mathbf{P}\mathbf{p}$$

where \mathbf{P} is a 3×4 camera matrix. This equation is valid even if the camera calibration parameters and/or the camera orientation are unknown.

Planar image mosaics

The simplest possible set of images to mosaic are views of a planar scene such as a document, whiteboard, or flat desktop. Imagine a camera fixed directly over a desk. As you slide a document under the camera, different portions of the document become visible. Any two such pieces are related to each other by a translation and a rotation (that is, a 2D rigid transformation).

Now imagine scanning a whiteboard with a hand-held video camera that you can move to any position. The class of transformations relating two pieces of the board, in this case, is the full family of 2D projective transformations. (Just imagine how a square or grid in one image can appear in another.) These transformations can be computed without any knowledge of the internal camera calibration parameters, such as focal length and optical center, or of the relative camera motion between frames. The fact that 2D projective transformations capture all such possible mappings (at least for an ideal pinhole camera) is a basic result of projective geometry (see sidebar).

Given this knowledge, how do we compute the transformations relating the various scene pieces so that we can paste them together? A variety of techniques are possible, some more automated than others. For example, we could manually identify four or more corresponding points between the two views, which is enough information to solve for the eight unknowns in the 2D projective transformation. We could also iteratively adjust the relative positions of input images using either a blink comparator (alternating between the two images at a high rate) or transparency. Unfortunately, these kinds of manual approaches are too tedious to be useful for large compositing applications.

Local image registration

The approach used here directly minimizes the discrepancy in intensities between pairs of images after applying the recovered transformation. This has the advantages of not requiring any easily identifiable feature points and of being statistically optimal, that is, giving the maximum likelihood estimate once we are in the vicinity of the true solution. Let's rewrite our 2D transformations as

$$x' = \frac{m_0x + m_1y + m_2}{m_6x + m_7y + 1},$$

$$y' = \frac{m_3x + m_4y + m_5}{m_6x + m_7y + 1} \quad (5)$$

Our technique minimizes the sum of the squared intensity errors

$$E = \sum [I'(x', y') - I(x, y)]^2 = \sum e^2 \quad (6)$$

over all corresponding pairs of pixels i inside both images $I(x, y)$ and $I'(x', y')$. (Pixels that are mapped outside image boundaries do not contribute.) Since (x', y') generally do not fall on integer pixel coordinates, we use bilinear interpolation of the intensities in I' to perform the resampling.

To perform the minimization, we use the Levenberg-Marquardt iterative nonlinear minimization algorithm.⁵ This algorithm requires computation of the partial derivatives of e_i with respect to the unknown motion parameters $\{m_0 \dots m_7\}$. These are straightforward to compute. For example,

$$\frac{\partial e}{\partial m_0} = \frac{x}{D} \frac{\partial I'}{\partial x'}, \dots, \frac{\partial e}{\partial m_7} = -\frac{y}{D} \left(x' \frac{\partial I'}{\partial x'} + y' \frac{\partial I'}{\partial y'} \right) \quad (7)$$

where D_i is the denominator in Equation 5 and $(\partial I' / \partial x', \partial I' / \partial y')$ is the image intensity gradient of I' at (x', y') . From these partial derivatives, the Levenberg-Marquardt algorithm computes an approximate Hessian matrix \mathbf{A} and the weighted gradient vector \mathbf{b} with components

$$a_{kl} = \sum \frac{\partial e}{\partial m_k} \frac{\partial e}{\partial m_l}, \quad b_k = -\sum e \frac{\partial e}{\partial m_k} \quad (8)$$

and then updates the motion parameter estimate \mathbf{m} by an amount $\Delta\mathbf{m} = (\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{b}$, where λ is a time-varying stabilization parameter.⁵ The advantage of using Levenberg-Marquardt over straightforward gradient descent is that it converges in fewer iterations.

The complete registration algorithm thus consists of the following steps:

1. For each pixel i at location (x_i, y_i) ,

(a) compute its corresponding position in the other image (x', y') using Equation 5;

(b) compute the error in intensity between the corresponding pixels $e = I'(x', y') - I(x, y)$ (Equation 6) and the intensity gradient $(\partial I' / \partial x', \partial I' / \partial y')$ using bilinear intensity interpolation on I' ;

(c) compute the partial derivative of e_i with respect to the m_k using

$$\frac{\partial e}{\partial m_k} = \frac{\partial I'}{\partial x'} \frac{\partial x'}{\partial m_k} + \frac{\partial I'}{\partial y'} \frac{\partial y'}{\partial m_k}$$

as in Equation 7;

(d) add the pixel's contribution to \mathbf{A} and \mathbf{b} as in Equation 8.

2. Solve the system of equations $(\mathbf{A} + \lambda\mathbf{I})\Delta\mathbf{m} = \mathbf{b}$ and update the motion estimate $\mathbf{m}^{(t+1)} = \mathbf{m}^{(t)} + \Delta\mathbf{m}$.

3. Check that the error in Equation 6 has decreased; if not, increment λ (as described in Press et al.⁵) and compute a new $\Delta\mathbf{m}$.

4. Continue iterating until the error is below a threshold or a fixed number of steps has been completed.

The steps in this algorithm are similar to the operations performed when warping images,^{2,3} with additional operations for correcting the current warping parameters based on local intensity error and its gradients. For more details on the exact implementation, see Szeliski and Coughlan.⁶

Once we have found the best transformation \mathbf{M} , we can blend the resampled image $I'(x', y')$ together with the reference image $I(x_i, y_i)$. To reduce visible artifacts—that is, to hide the edges of the component images—we use a weighted average with pixels near the center of each image contributing more to the final composite. The weighting function is a simple bilinear function:

$$w(x', y') = w_1(x')w_2(y')$$

(9)

where w_i is a triangle (hat) function that goes to zero at both edges of the image. In practice, this approach completely eliminates edge artifacts (see Figure 2), although a low-frequency "mottling" might still remain if the individual tiles have different exposures.

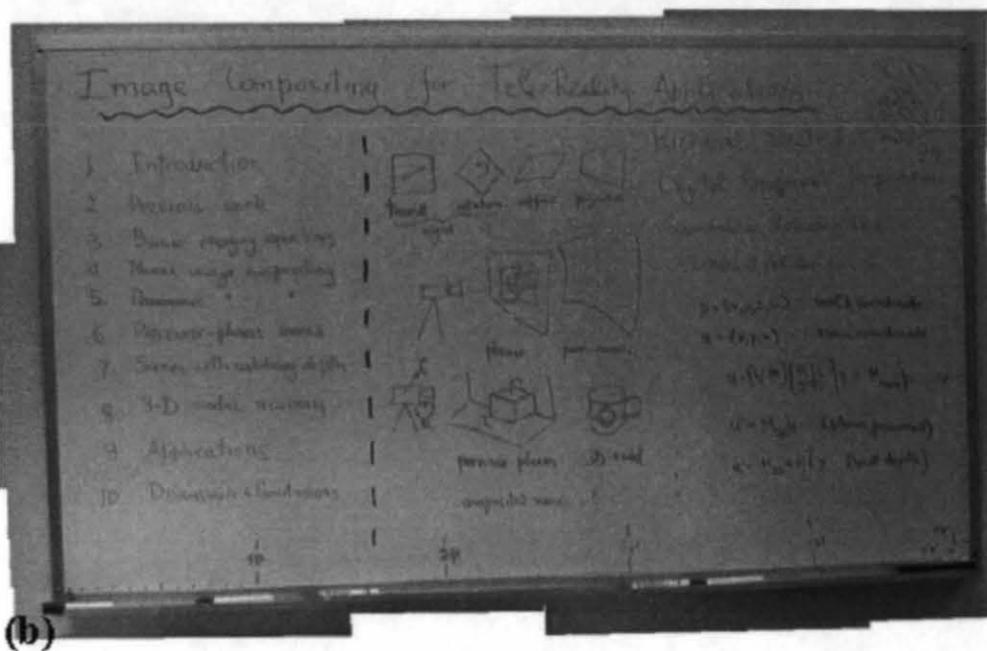
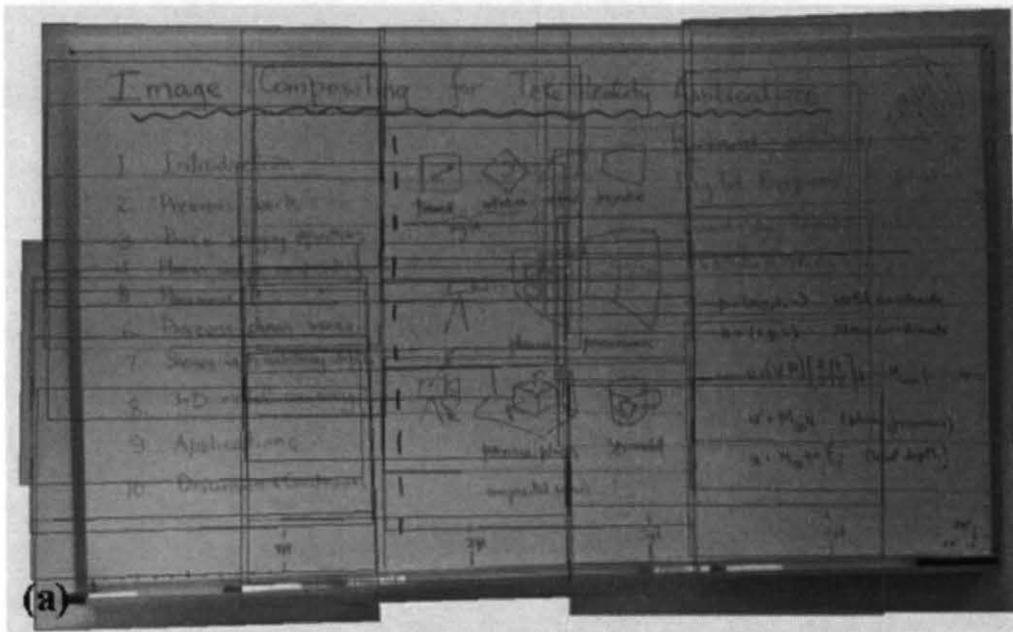


Figure 2. Whiteboard image mosaic example: (a) mosaic with component locations shown as colored outlines, (b) complete color mosaic (the central square shows the size of one input tile).

Global image registration

Unfortunately, both gradient descent and Levenberg-Marquardt only find locally optimal solutions. If the motion between successive frames is large, we must use a different strategy to find the best registration. Two different techniques can be used to handle this problem. The first technique, which is commonly used in computer vision, is *hierarchical matching*, which first registers smaller, subsampled versions of the images where the apparent motion is smaller. Motion estimates from these smaller, coarser levels are then used to initialize motion estimates at finer levels, thereby avoiding the local minimum problem (see Szeliski and Coughlan⁶ for details). While this technique is not guaranteed to find the correct registration, it has proved empirically to work well when the initial misregistration is only a few pixels (the exact domain of convergence depends on the intensity pattern in the image).

For larger displacements, you can use *phase correlation*.⁷ This technique estimates the 2D translation between a pair of images by taking 2D Fourier transforms of each image, computing the phase difference at each frequency, performing an inverse Fourier transform, and searching for a peak in the magnitude image. I have found this technique to work remarkably well in experiments, providing good initial guesses for image pairs that overlap by as little as 50 percent, even when there are moderate projective distortions (such as those that occur when using wide-angle lenses). The technique will not work if the interframe motion has large rotations or zooms, but this does not often occur in practice.

Results

To demonstrate the performance of the algorithm developed above, I digitized an image sequence with a camera panning over a whiteboard. Figure 2a shows the final mosaic of the whiteboard with the constituent images outlined in color. Figure 2b shows the final mosaic with the location of a single image shown as a white outline. This mosaic is $1,300 \times 2,046$ pixels, based on compositing 39 NTSC (640×480) resolution images.

To compute this mosaic, I developed an interactive image-manipulation tool that lets the user coarsely position successive frames relative to each other. The tool includes an automatic registration option that uses phase correlation to compute the initial rough placement of each image with respect to the previous one. The algorithm then refines the location of each image by minimizing Equation 6 using the current mosaic as $I(x, y)$ and the input frame being adjusted as $I(x', y')$. The images in Figure 2 were automatically composited without user intervention by employing the middle frame (center of the image) as the *base* image (no deformation). As you can see, the technique works well on this example.

Panoramic image mosaics

To build a panoramic image mosaic or *environment map*,¹ you can rotate a camera around its optical center. This resembles the action of panoramic still photographic cameras where the rotation of a camera on a tripod is mechanically coordinated with the film transport.⁸ In our case, however, we can mosaic multiple 2D images of arbitrary detail and resolution, and we need not know the camera motion. Examples of applications include constructing true scenic panoramas (say, of the view at the rim of Bryce Canyon) or limited virtual environments (a recreated meeting room or office as seen from one location).

Images taken from the same viewpoint with a stationary optical center are related by 2D projective transformations, just as in the planar scene case. (The sidebar presents a quick proof.) Because there is no motion parallax, you cannot see the relative depth of points in the scene as you rotate, so the images might as well be located on any plane.

More formally, the 2D transformation denoted by \mathbf{M} is related to the viewing matrices \mathbf{V} and \mathbf{V}' and the inter-view rotation \mathbf{R} by

$$\mathbf{M} = \mathbf{V}'\mathbf{R}\mathbf{V}^{-1} \tag{10}$$

(see sidebar). For a calibrated camera, we only have to recover the three independent rotation parameters (or five parameters if the focal length f values are unknown) instead of the usual eight.

How do we represent a panoramic scene composited using these techniques? One approach is to divide the viewing sphere into several large, potentially overlapping regions and to represent each region with a plane onto which we paste the images.¹ Figure 3 shows a mosaic of a bookshelf and cluttered desk composited onto a single plane (the highlighted central square forms the base relative to which all other images are registered). The images were obtained by tilting and panning a video camera mounted on a tripod, without taking any special steps to ensure that the rotation was around the true center of projection. As you can see, the complete scene is registered quite well.

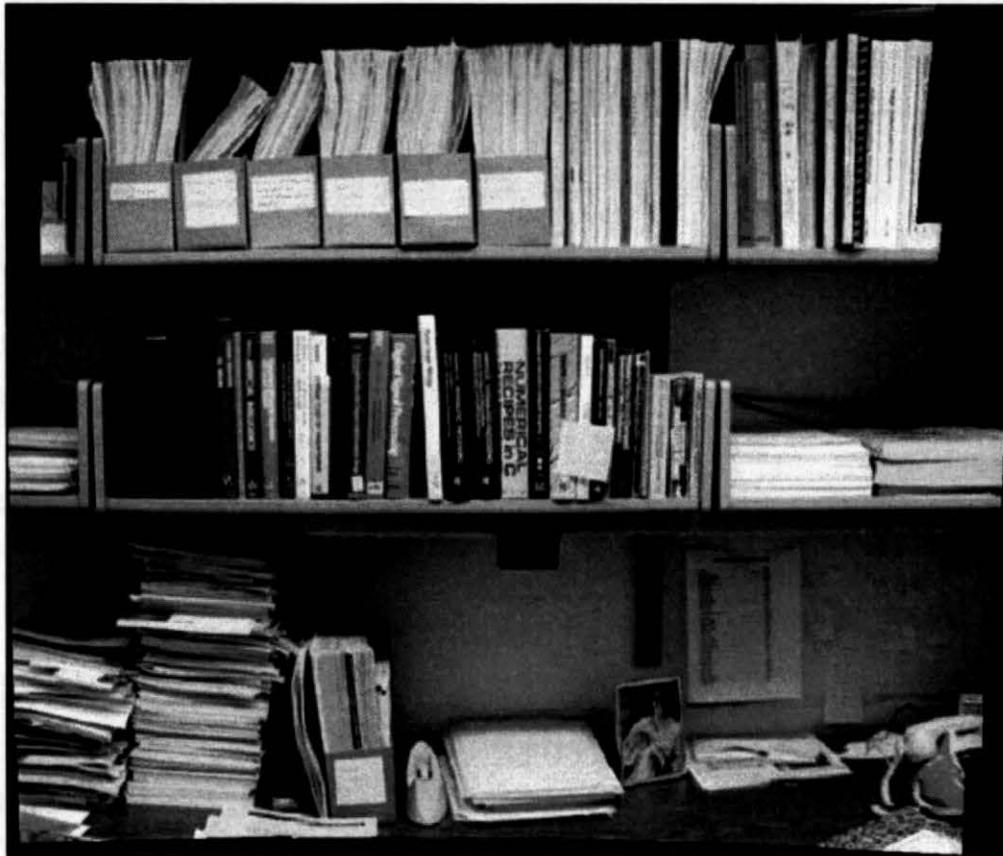


Figure 3. Panoramic image mosaic example (bookshelf and cluttered desk). These images were pasted onto a planar viewing surface.

Another approach is to compute the relative position of each frame relative to some base frame and to periodically choose a new base frame for doing the alignment. (Note that the algebraic properties of the 2D projective transformation group—that is, the associativity of matrix multiplication—make it possible to always compute the transformation between any two frames. However, to represent arbitrary views (including 90-degree rotations) requires replacing the condition $m_8 = 1$ in Equation 1 with $m_6^2 + m_7^2 + m_8^2 = 1$.) We can then recompute an arbitrary view on the fly from all visible pieces, given a particular view direction \mathbf{R} and zoom factor f . This is the approach used to composite the large wide-angle mosaic of Bryce Canyon shown in Figure 4.

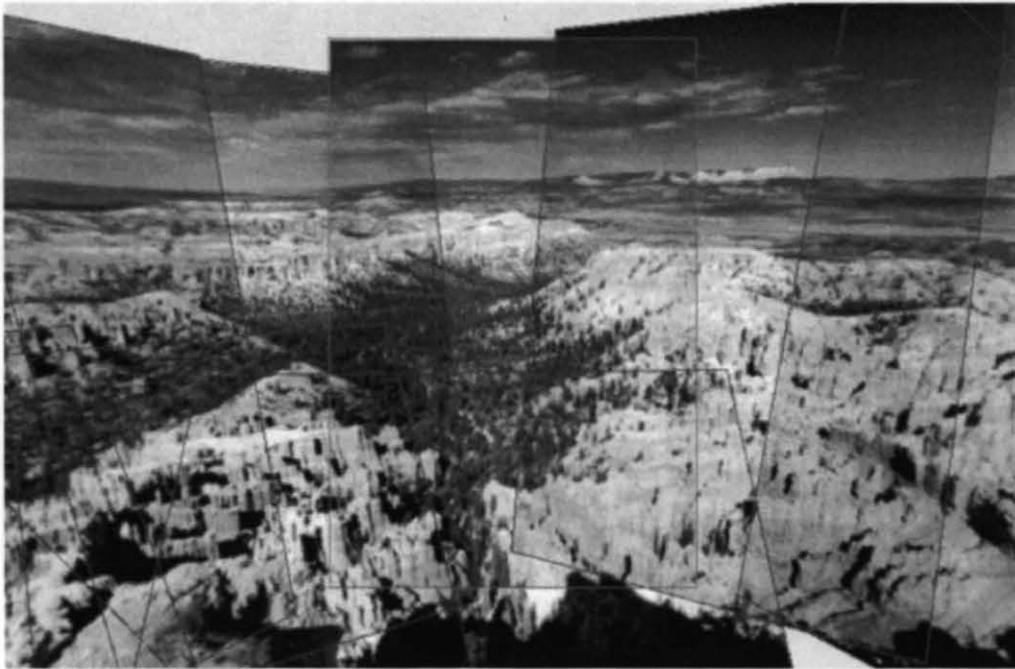


Figure 4. A portion of the Bryce Canyon mosaic. Because of the large motions involved, a single plane cannot represent the whole mosaic. Instead, different tiles are selected as base images.

A third approach is to use a cylindrical viewing surface to represent the image mosaic.⁹⁻¹² In this approach, we map world coordinates $\mathbf{p} = (x, y, z, w)$ onto 2D cylindrical screen locations $\mathbf{u} = (\theta, v)$, $\theta \in (-\pi, \pi]$ using

$$\theta = \tan^{-1}(x/z), \quad v = y/\sqrt{x^2 + z^2} \tag{11}$$

Figure 5 shows a complete circular panorama of an office unrolled onto a cylindrical surface. To build this panorama, each image is first mapped into cylindrical coordinates (using a known focal length and assuming the camera was horizontal). Then, the complete sequence is registered and composited using pure translations. The focal length of the camera can, if necessary, be recovered from images registered on a planar viewing surface. Figure 6 shows a similar panorama taken on the banks of the Charles River in Cambridge.



Figure 5. Circular panoramic image mosaic example (office interior). A total of 36 images

are pasted onto a cylindrical viewing surface.

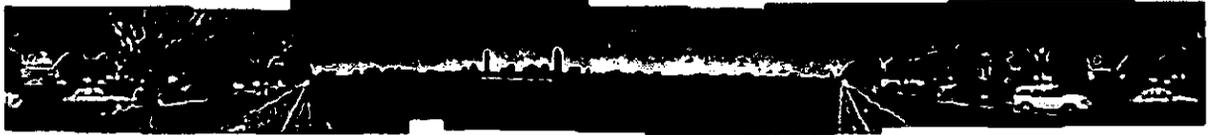


Figure 6. Circular panoramic image mosaic example (exterior scene). A total of 29 images are pasted onto a cylindrical viewing surface.

In addition to constructing large, single-resolution mosaics, we can also build mosaics with spatially varying amounts of resolution, for example, to zoom in on areas of interest. The modifications to the algorithm described so far are relatively straightforward and affect only the image-blending portion of it. As more images are added at varying resolutions, we can use the last image already registered as the new base image (since it is likely to be close in size to the new image). To create the new composite mosaic, we can use a generalization of the pyramidal parametrics used in texture mapping.¹³

Projective depth recovery

While mosaics of flat or panoramic scenes are useful for many virtual reality and office applications, such as scanning whiteboards or viewing outdoor panoramas, some applications need the depth associated with the scene to give the illusion of 3D. Once the depth has been recovered, nearby views can be generated using view interpolation.¹⁴ Two possible approaches are to model the scene as piecewise-planar or to recover dense 3D depth maps.

The first approach assumes that the scene is piecewise-planar, as is the case with many constructed environments such as building exteriors and office interiors. The mosaicing technique developed above for planar images can then be applied to each of the planar regions in the image. The segmentation of each image into its planar components can be done either interactively (for example, by drawing the polygonal outline of each region to be registered) or automatically by associating each pixel with one of several global motion hypotheses. Once the independent planar pieces have been composited, we could, in principle, recover the relative geometry of the various planes and the camera motion. However, rather than pursuing this approach here, we will develop the second, more general solution, which is to recover a full depth map. That is, we will infer the missing z component associated with each pixel in a given image sequence.

When the camera motion is known, the problem of depth map recovery is called *stereo reconstruction* (or *multiframe stereo* if more than two views are used). This problem has been extensively studied in photogrammetry and computer vision.¹⁵ When the camera

motion is unknown, we have the more difficult *structure-from-motion* problem.¹⁵ This section presents a solution to the latter problem based on recovering *projective depth*. The solution is simple and robust, and fits in well with the methods already developed in this article.

Formulation

To formulate the projective structure-from-motion recovery problem, note that the coordinates corresponding to a pixel \mathbf{u} with projective depth w in some other frame can be written as

$$\mathbf{u}' = \hat{\mathbf{V}}' \mathbf{E} \mathbf{p} = \mathbf{V}' \mathbf{R}^{-1} \mathbf{u} + w \mathbf{V}' \mathbf{t} = \mathbf{M} \mathbf{u} + w \tilde{\mathbf{t}} \quad (12)$$

where $\hat{\mathbf{V}}$, \mathbf{E} , \mathbf{R} , and \mathbf{t} are defined in Equations 2 and 3, and \mathbf{M} and $\tilde{\mathbf{t}}$ are the computed planar projective motion matrix and the epipole, that is, where the center of projection appears in the other camera (see Equation 24 in the sidebar). To recover the parameters in \mathbf{M} and $\tilde{\mathbf{t}}$ for each frame together with the depth values w (which are the same for all frames), we can use the same Levenberg-Marquardt algorithm as before.⁵ Once the projective depth values are recovered, they can be used directly in viewpoint interpolation (using new \mathbf{M} and $\tilde{\mathbf{t}}$ matrices), or they can be converted to true Euclidean depth using at least four known depth measurements.¹⁵

In more detail, we write the projection equation as

$$\begin{aligned} x' &= \frac{m_0 x + m_1 y + t_0 w + m_2}{m_6 x + m_7 y + t_2 w + 1}, \\ y' &= \frac{m_3 x + m_4 y + t_1 w + m_5}{m_6 x + m_7 y + t_2 w + 1} \end{aligned} \quad (13)$$

with $\mathbf{t} = (t_0, t_1, t_2)$. We compute the partial derivatives of x' , y' , and e_i with respect to the m_{κ} and t_{κ} (which we concatenate into the motion vector \mathbf{m}) as before in Equation 7. Similarly, we compute the partials of x' and y' with respect to w_i . That is,

$$\frac{\partial x'}{\partial w} = \frac{t_0 - x' t_2}{D}, \quad \frac{\partial y'}{\partial w} = \frac{t_1 - y' t_2}{D}$$

where D_i is the denominator in Equation 13.

To estimate the unknown parameters, we alternate iterations of the Levenberg-Marquardt algorithm over the motion parameters $\{m_0, \dots, t_2\}$ and the depth parameters $\{w_i\}$, using the partial derivatives defined above to compute the approximate Hessian matrices \mathbf{A} and the weighted error vectors \mathbf{b} as in Equation 8.

In the current implementation of this technique, the total number of parameters being estimated is decreased by using a tensor-product spline to represent the depth map and only recovering the depth estimates at the spline control vertices.⁶ (The complete depth map is computed by interpolation.)

Results

Figure 7 shows an example of using the projective depth recovery algorithm. The image sequence was taken by moving the camera up and over a table with stacks of papers (Figure 7a). The resulting depth map is shown in Figure 7b as intensity-coded range values. Figure 7c shows the original color image texture mapped onto the surface seen from a side viewpoint that is not part of the original sequence (an example of view extrapolation). Figure 7d shows a set of grid lines overlaid on the recovered surface to better judge its shape. The shape is recovered reasonably well in areas where there is sufficient texture to give depth cues (uniform intensity areas give none), and the extrapolated views look reasonable.

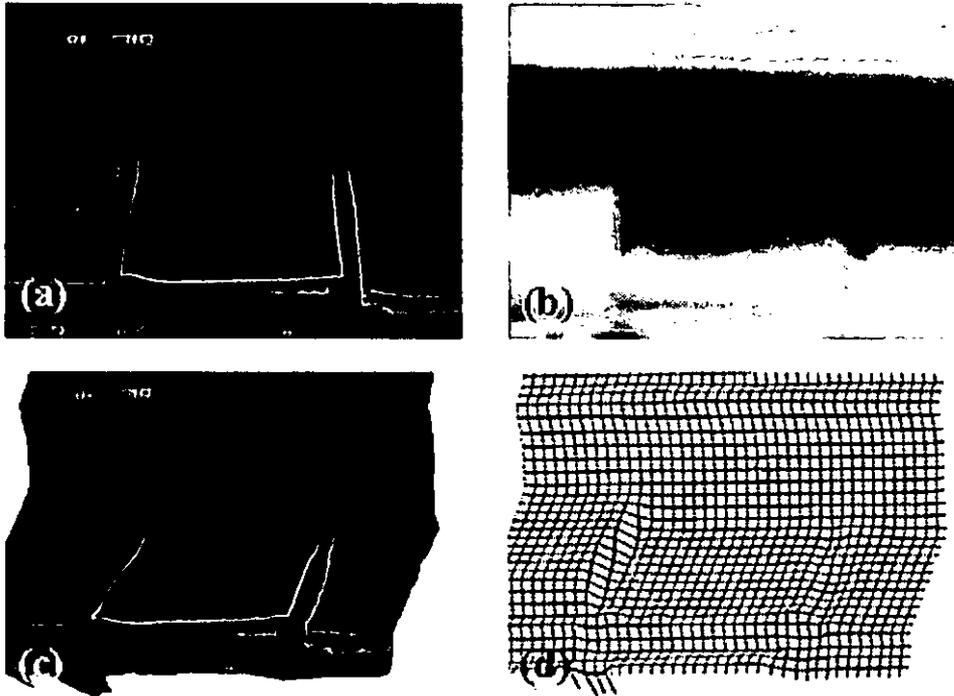


Figure 7. Depth recovery example—table with stacks of papers: (a) input image, (b) intensity-coded depth map (dark is farther back), (c) texture-mapped surface seen from a novel viewpoint, and (d) gridded surface.

Figure 8a shows results from another sequence—an outdoor scene of some trees. Again, the geometry of the scene is recovered reasonably well, as indicated in the depth map in Figure 8b.

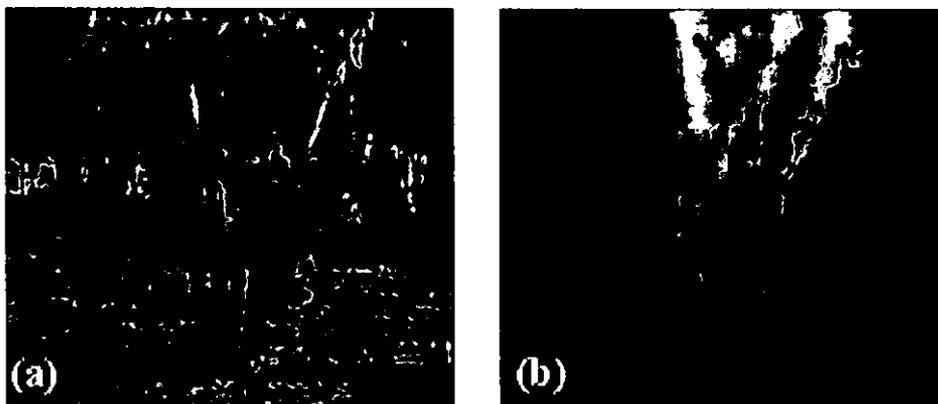


Figure 8. Depth recovery example—outdoor scene with trees: (a) input image, (b) intensity-coded depth map (dark is farther back).

Applications

Given automated techniques for building 2D and 3D scenes from video sequences, what can we do with these models? This section describes several potential applications of video mosaics, including whiteboard and document scanning, environment and backdrop acquisition for special effects and video games, supermarket shopping at home, interactive walkthroughs of historical buildings, and live telepresence applications.

The most straightforward application of image mosaics is scanning whiteboards or blackboards as an aid to videoconferencing or as an easy way to capture ideas. Scanning can produce images of much greater resolution than single wide-angle lens shots. The techniques developed in this article can be used with any video camera attached to a computer. In specialized situations, for example, in classrooms, a computer-controlled camera could do the scanning, removing the need for automatic registration of the images. In general, combinations of image mosaicing and superresolution can be used to produce photographic stills of extremely high resolution.^{9,16}

Document scanning is another obvious application of this technology. Hand-held scanners currently perform this function quite well. However, because they are based on linear CCDs, they are subject to "skewing" problems in each strip of the scanned image, which must be corrected manually. Using 2D video images as input removes some of these problems. A final global skew may still be needed to make the document square, but this can be performed automatically by detecting document edges or internal horizontal and vertical edges (for example, column borders).

The ability to mosaic video frames in real time opens up additional possibilities, such as using the camera as a "paintbrush" to generate large composite scenes interactively. The combination of such mosaics with live video streams produces interesting effects, where the newest frames appear to be "live," while older frames (presumably in other areas of the composite image) appear to be "frozen" in time.

A potential mass-market application is home shopping access to complete stores, such as your local supermarket. This has the advantage of a familiar look and organization, and lets you plan your next shopping trip. The images of the aisles—with their current contents—can be digitized by rolling a video camera through the store. More detailed geometric models of individual items can be acquired by a video-based 3D model-building process.¹⁰ In the future, these models will be available directly from the manufacturer. The shopper can then stroll down the aisles, pick out individual items, and look at their ingredients and prices.

Panoramic mosaics (environment maps) can enhance special effects used in movies, for example, computing illumination maps and reflections or filling in backgrounds. Such panoramas could also serve as backdrops for video games. A collection of such panoramas could be used in a limited form of virtual reality, such as showing the views inside different

rooms in a museum or historic building.¹¹ A museum scenario might include the ability to look at individual 3D objects such as sculptures and to bring up related information in a hypertext system.

Building true 3D models of buildings, perhaps using piecewise-planar representations, opens up many more possibilities. For example, interactive 3D walkthroughs of your home, built by walking a video camera through the rooms and processing the image sequences, could be used for selling your house (an extension of existing still-image based systems) or for remodeling or renovations. However, building complete models of room interiors, including furniture, requires true 3D model reconstruction techniques (see Szeliski¹⁰ for some recent results).

You can also create walk- or fly-throughs of general outdoor 3D scenes. Example applications include flight and driving simulators, and virtual travel (*teletourism*). Such 3D scene models can have extremely high complexity and require solutions to problems in representing the images, employing partial 3D models, and switching between different levels of resolution.

The ultimate in virtual reality systems is dynamic virtual reality (sometimes called *telepresence*), which composites video from multiple sources in real time to create the illusion of being in a dynamic (and perhaps reactive) 3D environment. An example application might be to view a 3D version of a concert or sporting event with control over the camera shots, even seeing the event from a player's point of view. Other examples might be to participate or consult in a surgery from a remote location (*telemedicine*) or to participate remotely in a virtual classroom. Building such dynamic 3D models at frame rates is beyond the processing power of today's high-performance superscalar workstations, but it could be achieved using a collection of such machines or special-purpose stereo hardware.

Discussion

Video mosaics provide a powerful new way of creating the detailed environments needed for virtual reality applications. By quickly registering multiple images together, it is possible to create scenes of extremely high resolution and simultaneously recover partial 3D geometric information. The approach used here—namely, direct minimization of intensity differences between warped images—has a number of advantages over more traditional techniques, which are based on tracking features from frame to frame.¹⁵ The techniques here produce dense estimates of shape. They work in highly textured areas where features may not be reliably observed, and they make statistically optimal use of all the information. In addition, the depth map recovery algorithm, described in the section "Projective depth recovery," makes it possible to perform view interpolation directly on real-world scenes, rather than just on synthetically generated graphics.

While the techniques described in this article have worked well in the scenes in which I have tried them, I remain cautious about their general applicability. Intensity-based

techniques are sensitive to image intensity variation, such as those caused by video camera gain control and vignetting (darkening of the corners at wide lens apertures). Working with band-pass filtered images and proper image blending can remove many of these problems. Registration techniques are also sensitive to geometric distortions (deviations from the pinhole model) in the optics, so careful calibration is necessary for optimal accuracy (the results reported here were obtained with uncalibrated cameras). Other potential sources of error include limited depth of field and imperfect alignment of rotational axes in panoramic scene compositing.

The depth extraction techniques rely on the presence of texture in the image. Even in areas of sufficient texture, the registration/matching algorithm can still compute erroneous depth estimates, for example, due to repetitive texture patterns or occlusions. Where texture is absent, interpolation must be used, and this can also lead to erroneous depth estimates.

Conclusion

The techniques presented here automatically register video frames into 2D and partial 3D scene models. Truly realistic virtual environments will require 3D object models as well as environment maps. The automatic construction of such models directly from video is the subject of ongoing investigations.¹⁰

The creation of realistic high-resolution scenes from video imagery opens up many new applications. Ultimately, as processing speeds and reconstruction algorithms improve further, video mosaics and related techniques will enable an even more exciting range of interactive computer graphics, telepresence, and virtual reality applications.

References

1. N. Greene, "Environment Mapping and Other Applications of World Projections," *IEEE CG&A*, Vol. 6, No. 11, Nov. 1986, pp. 21-29.
2. T. Beier and S. Neely, "Feature-Based Image Metamorphosis," *Computer Graphics* (Proc. Siggraph), Vol. 26, No. 2, July 1992, pp. 35-42.
3. G. Wolberg, *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, Calif., 1990.
4. J.D. Foley et al., *Computer Graphics: Principles and Practice*, 2nd edition, Addison-Wesley, Reading, Mass., 1990.
5. W.H. Press et al., *Numerical Recipes in C: The Art of Scientific Computing*, 2nd edition, Cambridge Univ. Press, Cambridge, England, 1992.
6. R. Szeliski and J. Coughlan, "Hierarchical Spline-Based Image Registration," *Proc. IEEE Computer Soc. Conf. Computer Vision and Pattern Recognition (CVPR 94)*, IEEE CS Press, Los Alamitos, Calif., 1994, pp. 194-201.
7. C.D. Kuglin and D.C. Hines, "The Phase Correlation Image Alignment Method,"

IEEE Conf. on Cybernetics and Society, IEEE, New York, 1975, pp. 163-165.

8. H.E. Malde, "Panoramic Photographs," *American Scientist*, Vol. 71, No. 2, Mar.-Apr. 1983, pp. 132-140.

9. S. Mann and R.W. Picard, "Virtual Bellows: Constructing High-Quality Images from Video," *Proc. First Int'l Conf. Image Processing*, Vol. I, IEEE CS Press, Los Alamitos, Calif., 1994, pp. 363-367.

10. R. Szeliski, "Image Mosaicing for Tele-Reality Applications," *Proc. IEEE Workshop on Applications of Computer Vision*, IEEE CS Press, Los Alamitos, Calif., 1994, pp. 44-53.

11. S.E. Chen, "QuickTime VR_An Image-Based Approach to Virtual Environment Navigation," *Proc. Siggraph 95*, ACM, New York, 1995, pp. 29-38.

12. L. McMillan and G. Bishop, "Plenoptic Modeling: An Image-Based Rendering System," *Proc. Siggraph 95*, ACM, New York, 1995, pp. 39-46.

13. L. Williams, "Pyramidal Parametrics," *Computer Graphics (Proc. Siggraph)*, Vol. 17, No. 3, July 1983, pp. 1-11.

14. S. Chen and L. Williams, "View Interpolation for Image Synthesis," *Proc. Siggraph 93*, ACM, New York, 1993, pp. 279-288.

15. O. Faugeras, *Three-Dimensional Computer Vision: A Geometric Viewpoint*, MIT Press, Cambridge, Mass., 1993.

16. L. Teodosio and W. Bender, "Panoramic Overview for Navigating Real-World Scenes," *Proc. ACM Multimedia 93*, ACM Press, New York, 1993, pp. 359-364.



Richard Szeliski is a senior researcher in the Advanced Technology and Research division of Microsoft Corporation. His research interests include 3D computer vision, motion estimation, and virtual environments. He received a BEng in electrical engineering from McGill University, Montreal, in 1979, an MEng in electrical engineering from the University of British Columbia, Vancouver, in 1981, and a PhD in computer science from Carnegie Mellon University, Pittsburgh, in 1988. Szeliski is the author of *Bayesian Modeling of Uncertainty in Low-Level Vision* (Kluwer). He is a member of ACM, IEEE, and Sigma Xi and associate editor of the *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

2D projective transformations

Planar scene views are related by projective transformations. This is true in the most general case—that is, for any 3D-to-2D mapping

$$\mathbf{u} = \mathbf{P}\mathbf{p} \tag{i}$$

where $\mathbf{p} = (x, y, z, w)$ is a 3D world coordinate, $\mathbf{u} = (x', y', w')$ is the 2D screen location, and \mathbf{P} is the 3×4 camera matrix defined in Equation 4.

Since \mathbf{P} is of rank 3, we have

$$\mathbf{p} = \mathbf{M} * \mathbf{u} + s\mathbf{m} \tag{ii}$$

where $\mathbf{M}^* = \mathbf{P}^T(\mathbf{P}\mathbf{P}^T)^{-1}$ is the pseudoinverse of \mathbf{P} and \mathbf{m} is in the null space of \mathbf{P} , that is, $\mathbf{P}\mathbf{m} = 0$. The equation of a plane in world coordinates can be written as

$$ax + by + cz = d \quad \text{or} \quad \mathbf{n} \cdot \mathbf{p} = 0 \tag{iii}$$

from which we can conclude that

$$\mathbf{n}^T \mathbf{M} * \mathbf{u} + s\mathbf{n} \cdot \mathbf{m} = 0 \quad \text{or} \quad s = -(\mathbf{n}^T \mathbf{M} * \mathbf{u}) / (\mathbf{n} \cdot \mathbf{m}) \tag{iv}$$

and hence

$$\mathbf{p} = (\mathbf{I} - (\mathbf{n} \cdot \mathbf{m})^{-1} \mathbf{m}\mathbf{n}^T) \mathbf{M} * \mathbf{u} = \tilde{\mathbf{M}}\mathbf{u} \tag{v}$$

From any other viewpoint, we have

$$\mathbf{u}' = \mathbf{P}'\tilde{\mathbf{M}}\mathbf{u} = \mathbf{M}\mathbf{u} \quad (\text{vi})$$

which is a 2D planar projective transformation.

In the absence of prior information about the camera location, we can assume that the world coordinate system coincides with the first camera position, that is, $\mathbf{E} = \mathbf{I}$ and $\mathbf{P} = \hat{\mathbf{V}} = [\mathbf{V} \ \mathbf{0}]$, and therefore $\mathbf{M}^* = \mathbf{V}^{-1}$ and $\mathbf{m} = (0, 0, 0, 1)^T$ in Equations ii through v. An image point \mathbf{u} then corresponds to a world coordinate \mathbf{p} of the form

$$\mathbf{p} = \begin{bmatrix} \mathbf{V}^{-1}\mathbf{u} \\ w \end{bmatrix} \quad (\text{vii})$$

where w is the unknown fourth coordinate of \mathbf{p} (called *projective depth*), which determines how far the point is from the origin.

For panoramic image mosaics, we can rotate the point \mathbf{p} around the camera optical center to obtain

$$\mathbf{p}' = \begin{bmatrix} \mathbf{R} & \mathbf{0} \\ \mathbf{0}^T & 1 \end{bmatrix} \mathbf{p} = \begin{bmatrix} \mathbf{R}\mathbf{V}^{-1}\mathbf{u} \\ w \end{bmatrix} \quad (\text{viii})$$

with a corresponding screen coordinate

$$\mathbf{u}' = \mathbf{V}'\mathbf{R}\mathbf{V}^{-1}\mathbf{u} = \mathbf{M}\mathbf{u} \quad (\text{ix})$$

Thus, the mapping between the two screen coordinate systems can be described by a 2D projective transformation.

For projective depth recovery, each image point \mathbf{u} corresponds to a 3D point \mathbf{p} with an unknown projective depth w as given in Equation vii. In some other frame, whose relative orientation to the first frame is denoted by \mathbf{E} , we have

$$\mathbf{u}' = \mathbf{V}'\mathbf{E}\mathbf{p} = \mathbf{V}'\mathbf{R}\mathbf{V}^{-1}\mathbf{u} + w\mathbf{V}'\mathbf{t} = \mathbf{M}\mathbf{u} + w\tilde{\mathbf{t}}$$

(x)

Thus, the motion between the two frames can be described by our familiar 2D planar projective motion, plus an amount of motion proportional to the projective depth along the direction \mathbf{t} (which is called the *epipole*).

Plenoptic Modeling: An Image-Based Rendering System

Leonard McMillan[†] and Gary Bishop[‡]

Department of Computer Science
University of North Carolina at Chapel Hill

ABSTRACT

Image-based rendering is a powerful new approach for generating real-time photorealistic computer graphics. It can provide convincing animations without an explicit geometric representation. We use the "plenoptic function" of Adelson and Bergen to provide a concise problem statement for image-based rendering paradigms, such as morphing and view interpolation. The plenoptic function is a parameterized function for describing everything that is visible from a given point in space. We present an image-based rendering system based on sampling, reconstructing, and resampling the plenoptic function. In addition, we introduce a novel visible surface algorithm and a geometric invariant for cylindrical projections that is equivalent to the epipolar constraint defined for planar projections.

CR Descriptors: 1.3.3 [Computer Graphics]: Picture/Image Generation—*display algorithms, viewing algorithms*; 1.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—*hidden line/surface removal*; 1.4.3 [Image Processing]: Enhancement—*registration*; 1.4.7 [Image Processing]: Feature Measurement—*projections*; 1.4.8 [Image Processing]: Scene Analysis.

1. INTRODUCTION

In recent years there has been increased interest, within the computer graphics community, in image-based rendering systems. These systems are fundamentally different from traditional geometry-based rendering systems. In image-based systems the underlying data representation (i.e. model) is composed of a set of photometric observations, whereas geometry-based systems use either mathematical descriptions of the boundary regions separating scene elements (B-rep) or discretely sampled space functions (volumetric).

The evolution of image-based rendering systems can be traced through at least three different research fields. In photogrammetry the initial problems of camera calibration, two-dimensional image registration, and photometrics have progressed toward the determination of three-dimensional models. Likewise, in computer vision, problems such as robot navigation, image discrimination, and image understanding have naturally led in the same direction. In computer graphics, the progression toward image-based rendering systems

was initially motivated by the desire to increase the visual realism of the approximate geometric descriptions by mapping images onto their surface (texture mapping) [7], [12]. Next, images were used to approximate global illumination effects (environment mapping) [5], and, most recently, we have seen systems where the images themselves constitute the significant aspects of the scene's description [8].

Another reason for considering image-based rendering systems in computer graphics is that acquisition of realistic surface models is a difficult problem. While geometry-based rendering technology has made significant strides towards achieving photorealism, creating accurate models is still nearly as difficult as it was ten years ago. Technological advances in three-dimensional scanning provide some promise in model building. However, they also verify our worst suspicions—the geometry of the real-world is exceedingly complex. Ironically, the primary subjective measure of image quality used by proponents of geometric rendering systems is the degree with which the resulting images are indistinguishable from photographs.

One liability of image-based rendering systems is the lack of a consistent framework within which to judge the validity of the results. Fundamentally, this arises from the absence of a clear problem definition. Geometry-based rendering, on the other hand, has a solid foundation; it uses analytic and projective geometry to describe the world's shape and physics to describe the world's surface properties and the light's interaction with those surfaces.

This paper presents a consistent framework for the evaluation of image-based rendering systems, and gives a concise problem definition. We then evaluate previous image-based rendering methods within this new framework. Finally, we present our own image-based rendering methodology and results from our prototype implementation.

2. THE PLENOPTIC FUNCTION

Adelson and Bergen [1] assigned the name *plenoptic* function (from the latin root *plenus*, meaning complete or full, and *optic* pertaining to vision) to the pencil of rays visible from any point in space, at any time, and over any range of wavelengths. They used this function to develop a taxonomy for evaluating models of low-level vision. The plenoptic function describes all of the radiant energy that can be perceived from the point of view of the observer rather than the point of view of the source. They postulate

"... all the basic visual measurements can be considered to characterize local change along one or two dimensions of a single function that describes the structure of the information in the light impinging on an observer."

Adelson and Bergen further formalized this functional description by providing a parameter space over which the plenoptic function is valid, as shown in Figure 1. Imagine an idealized eye which we are free to place at any point in space (V_x, V_y, V_z). From there we can select any of the viewable rays by choosing an azimuth and elevation angle

^{††}CB 3175 Sitterson Hall, Chapel Hill, NC 27599

[†](919) 962-1797 mcmillan@cs.unc.edu <http://www.cs.unc.edu/~mcmillan>

[‡](919) 962-1886 gb@cs.unc.edu <http://www.cs.unc.edu/~gb>

(θ, ϕ) as well as a band of wavelengths, λ , which we wish to consider.

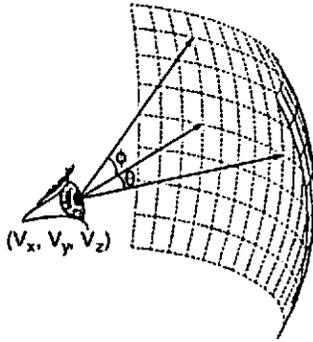


FIGURE 1. The plenoptic function describes all of the image information visible from a particular viewing position.

In the case of a dynamic scene, we can additionally choose the time, t , at which we wish to evaluate the function. This results in the following form for the plenoptic function:

$$p = P(\theta, \phi, \lambda, V_x, V_y, V_z, t) \quad (1)$$

In computer graphics terminology, the plenoptic function describes the set of all possible environment maps for a given scene. For the purposes of visualization, one can consider the plenoptic function as a scene representation. In order to generate a view from a given point in a particular direction we would need to merely plug in appropriate values for (V_x, V_y, V_z) and select from a range of (θ, ϕ) for some constant t .

We define a complete sample of the plenoptic function as a full spherical map for a given viewpoint and time value, and an incomplete sample as some solid angle subset of this spherical map.

Within this framework we can state the following problem definition for image-based rendering. *Given a set of discrete samples (complete or incomplete) from the plenoptic function, the goal of image-based rendering is to generate a continuous representation of that function.* This problem statement provides for many avenues of exploration, such as how to optimally select sample points and how to best reconstruct a continuous function from these samples.

3. PREVIOUS WORK

3.1 Movie-Maps

The Movie-Map system by Lippman [17] is one of the earliest attempts at constructing an image-based rendering system. In Movie-Maps, incomplete plenoptic samples are stored on interactive video laser disks. They are accessed randomly, primarily by a change in viewpoint; however, the system can also accommodate panning, tilting, or zooming about a fixed viewing position. We can characterize Lippman's plenoptic reconstruction technique as a nearest-neighbor interpolation because, when given a set of input parameters $(V_x, V_y, V_z, \theta, \phi, t)$, the Movie-Map system can select the nearest partial sample. The Movie-Map form of image-based rendering can also be interpreted as a table-based evaluation of the plenoptic function. This interpretation reflects the database structure common to most image-based systems.

3.2 Image Morphing

Image morphing is a very popular image-based rendering technique [4], [28]. Generally, morphing is considered to occur between two images. We can think of these images as endpoints along some path through time and/or space. In this interpretation, morphing becomes a method for reconstructing partial samples of the continuous plenoptic function along this path. In addition to photometric data, morphing uses additional information describing the image flow field. This information is usually hand crafted by an animator. At first

glance, this type of augmentation might seem to place it outside of the plenoptic function's domain. However, several authors in the field of computer vision have shown that this type of image flow information is equivalent to changes in the local intensity due to infinitesimal perturbations of the plenoptic function's independent variables [20], [13]. This local derivative behavior can be related to the intensity gradient via applications of the chain rule. In fact, morphing makes an even stronger assumption that the flow information is constant along the entire path, thus amounting to a locally linear approximation. Also, a blending function is often used to combine both reference images after being partially flowed from their initial configurations to a given point on the path. This blending function is usually some linear combination of the two images based on what percentage of the path's length has been traversed. Thus, morphing is a plenoptic reconstruction method which interpolates between samples and uses local derivative information to construct approximations.

3.3 View Interpolation

Chen's and Williams' [8] view interpolation employs incomplete plenoptic samples and image flow fields to reconstruct arbitrary viewpoints with some constraints on gaze angle. The reconstruction process uses information about the local neighborhood of a sample. Chen and Williams point out and suggest a solution for one of the key problems of image-based rendering—determining the visible surfaces. Chen and Williams chose to presort the quadtree compressed flow-field in a back-to-front order according to its (geometric) z -value. This approach works well when all of the partial sample images share a common gaze direction, and the synthesized viewpoints are restricted to stay within 90 degrees of this gaze angle.

An image flow field alone allows for many ambiguous visibility solutions, unless we restrict ourselves to flow fields that do not fold, such as rubber-sheet local spline warps or thin-plate global spline warps. This problem must be considered in any general-purpose image-based rendering system, and ideally, it should be done without transporting the image into the geometric-rendering domain.

Establishing flow fields for a view interpolation system can also be problematic. Chen and Williams used pre-rendered synthetic images to determine flow fields from the z -values. In general, accurate flow field information between two samples can only be established for points that are mutually visible to both samples. This points out a shortcoming in the use of partial samples, because reference images seldom have a 100% overlap.

Like morphing, view interpolation uses photometric information as well as local derivative information in its reconstruction process. This locally linear approximation is nicely exploited to approximate perspective depth effects, and Chen and Williams show it to be correct for lateral motions relative to the gaze direction. View interpolation, however, adds a nonlinearity by allowing the visibility process to determine the blending function between reference frames in a closest-take-all (a.k.a. winner-take-all) fashion.

3.4 Laveau and Faugeras

Laveau and Faugeras [15] have taken advantage of the fact that the epipolar geometries between images restrict the image flow field in such a way that it can be parameterized by a single disparity value and a fundamental matrix which represents the epipolar relationship. They also provide a two-dimensional raytracing-like solution to the visibility problem which does not require an underlying geometric description. Their method does, however, require establishing correspondences for each image point along the ray's path. The Laveau and Faugeras system also uses partial plenoptic samples, and results are shown only for overlapping regions between views.

Laveau and Faugeras also discuss the combination of information from several views but primarily in terms of resolving visibility. By relating the reference views and the desired views by the homogenous transformations between their projections, Laveau and Faugeras can compute exact perspective depth solutions. The recon-

struction process again takes advantage of both image data and local derivative information to reconstruct the plenoptic function.

3.5 Regan and Pose

Regan and Pose [23] describe a hybrid system in which plenoptic samples are generated on the fly by a geometry-based rendering system at available rendering rates, while interactive rendering is provided by the image-based subsystem. At any instant, a user interacts with a single plenoptic sample. This allows the user to make unconstrained changes in the gaze angle about the sample point. Regan and Pose also discuss local reconstruction approximations due to changes in the viewing position. These approximations amount to treating the objects in the scene as being placed at infinity, resulting in a loss of the kinetic depth effect. These partial updates can be combined with the approximation values.

4. PLENOPTIC MODELING

We claim that all image-based rendering approaches can be cast as attempts to reconstruct the plenoptic function from a sample set of that function. We believe that there are significant insights to be gleaned from this characterization. In this section, we propose our prototype system in light of this plenoptic function framework.

We call our image-based rendering approach Plenoptic Modeling. Like other image-based rendering systems, the scene description is given by a series of reference images. These reference images are subsequently warped and combined to form representations of the scene from arbitrary viewpoints. The warping function is defined by image flow field information that can either be supplied as an input or derived from the reference images.

Our discussion of the plenoptic modeling image-based rendering system is broken down into four sections. First, we discuss the representation of the plenoptic samples. Next, we discuss their acquisition. The third section covers the determination of image flow fields, if required. And, finally, we describe how to reconstruct the plenoptic function from these sample images.

4.1 Plenoptic Sample Representation

The most natural surface for projecting a complete plenoptic sample is a unit sphere centered about the viewing position. One difficulty of spherical projections, however, is the lack of a representation that is suitable for storage on a computer. This is particularly difficult if a uniform (i.e. equal area) discrete sampling is required. This difficulty is reflected in the various distortions which arise in planar projections of world maps in cartography. Those uniform mappings which do exist are generally ill-suited for systematic access as a data structure. Furthermore, those which do map to a plane with consistent neighborhood relationships are generally quite distorted and, therefore, non-uniform.

A set of six planar projections in the form of a cube has been suggested by Greene [10] as an efficient representation for environment maps. While this representation can be easily stored and accessed by a computer, it provides significant problems relating to acquisition, alignment, and registration when used with real, non-computer-generated images. The orthogonal orientation of the cube faces requires precise camera positioning. The wide, 90 degree field-of-view of each face requires expensive lens systems to avoid optical distortion. Also, the planar mapping does not represent a uniform sampling, but instead, is considerably oversampled in the edges and corners. However, the greatest difficulty of a cube-oriented planar projection set is describing the behavior of the image flow fields across the boundaries between faces and at corners. This is not an issue when the six planar projections are used solely as an environment map, but it adds a considerable overhead when it is used for image analysis.

We have chosen to use a cylindrical projection as the plenoptic sample representation. One advantage of a cylinder is that it can be easily unrolled into a simple planar map. The surface is without boundaries in the azimuth direction, which simplifies correspondence searches required to establish image flow fields. One short-

coming of a projection on a finite cylindrical surface is the boundary conditions introduced at the top and bottom. We have chosen not to employ end caps on our projections, which has the problem of limiting the vertical field of view within the environment.

4.2 Acquiring Cylindrical Projections

A significant advantage of a cylindrical projection is the simplicity of acquisition. The only acquisition equipment required is a video camera and a tripod capable of continuous panning. Ideally, the camera's panning motion would be around the exact optical center of the camera. In practice, in a scene where all objects are relatively far from the tripod's rotational center, a slight misalignment offset can be tolerated.

Any two planar perspective projections of a scene which share a common viewpoint are related by a two-dimensional homogenous transform:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad (2)$$

$$x' = \frac{u}{w} \quad y' = \frac{v}{w}$$

where x and y represent the pixel coordinates of an image I , and x' and y' are their corresponding coordinates in a second image I' . This well known result has been reported by several authors [12], [28], [22]. The images resulting from typical camera motions, such as pan, tilt, roll, and zoom, can all be related in this fashion. When creating a cylindrical projection, we will only need to consider panning camera motions. For convenience we define the camera's local coordinate system such that the panning takes place entirely in the x - z plane.

In order to reproject an individual image into a cylindrical projection, we must first determine a model for the camera's projection or, equivalently, the appropriate homogenous transforms. Many different techniques have been developed for inferring the homogenous transformation between images sharing common centers of projection. The most common technique [12] involves establishing four corresponding points across each image pair. The resulting transforms provide a mapping of pixels from the planar projection of the first image to the planar projection of the second. Several images could be composited in this fashion by first determining the transform which maps the N th image to image $N-1$. These transforms can be catenated to form a mapping of each image to the plane of the first. This approach, in effect, avoids direct determination of an entire camera model by performing all mappings between different instances of the same camera. Other techniques for deriving these homogeneous transformations without specific point correspondences have also been described [22], [25].

The set of homogenous transforms, H_i , can be decomposed into two parts which will allow for arbitrary reprojections in a manner similar to [11]. These two parts include an intrinsic transform, S , which is determined entirely by camera properties, and an extrinsic transform, R_i , which is determined by the rotation around the camera's center of projection:

$$D = H_i X = S^{-1} R_i S X \quad (3)$$

This decomposition decouples the projection and rotational components of the homogeneous transform. By an appropriate choice of coordinate systems and by limiting the camera's motion to panning, the extrinsic transform component is constrained to a function of a single parameter rotation matrix describing the pan.

$$R_y = \begin{bmatrix} \cos \theta & 0 & \sin \theta \\ 0 & 1 & 0 \\ -\sin \theta & 0 & \cos \theta \end{bmatrix} \quad (4)$$

Since the intrinsic component's properties are invariant over all of the images, the decomposition problem can be broken into two parts: the determination of the extrinsic rotation component, R_x , followed by the determination of an intrinsic projection component, S . The first step in our method determines estimates for the extrinsic panning angle between each image pair of the panning sequence. This is accomplished by using a linear approximation to an infinitesimal rotation by the angle θ . This linear approximation results from substituting $1 + O(\theta^2)$ for the cosine terms and $\theta + O(\theta^3)$ for the sine terms of the rotation matrix. This infinitesimal perturbation has been shown by [14] to reduce to the following approximate equations:

$$\begin{aligned} x' &= x - f\theta - \frac{\theta(x - C_x)^2}{f} + O(\theta^2) \\ y' &= y - \frac{\theta(x - C_x)(y - C_y)}{f} + O(\theta^2) \end{aligned} \quad (5)$$

where f is the apparent focal length of the camera measured in pixels, and (C_x, C_y) is the pixel coordinate of the intersection of the optical axis with the image plane. (C_x, C_y) is initially estimated to be at the center pixel of the image plane. A better estimate for (C_x, C_y) is found during the intrinsic matrix solution.

These equations show that small panning rotations can be approximated by translations for pixels near the image's center. We require that some part of each image in the sequence must be visible in the successive image, and that some part of the final image must be visible in the first image of the sequence. The first stage of the cylindrical registration process attempts to register the image set by computing the optimal translation in x which maximizes the normalized correlation within a region about the center third of the screen. This is first computed at a pixel resolution, then refined on a 0.1 sub-pixel grid, using a Catmull-Rom interpolation spline to compute sub-pixel intensities. Once these translations, t_i , are computed, Newton's method is used to convert them to estimates of rotation angles and the focal length, using the following equation:

$$2\pi - \sum_{i=1}^N \text{atan}\left(\frac{t_i}{f}\right) = 0 \quad (6)$$

where N is the number of images comprising the sequence. This usually converges in as few as five iterations, depending on the original estimate for f . This first phase determines an estimate for the relative rotational angles between each of the images (our extrinsic parameters) and the initial focal length estimate measured in pixels (one of the intrinsic parameters).

The second stage of the registration process determines the S , or structural matrix, which describes various camera properties such as the tilt and roll angles which are assumed to remain constant over the group of images. The following model is used:

$$S = \Omega_x \Omega_z P \quad (7)$$

where P is the projection matrix:

$$P = \begin{bmatrix} 1 & \sigma & -C_x \\ 0 & \rho & -C_y \\ 0 & 0 & f \end{bmatrix} \quad (8)$$

and (C_x, C_y) is the estimated center of the viewplane as described previously, σ is a skew parameter representing the deviation of the sampling grid from a rectilinear grid, ρ determines the sampling grid's aspect ratio, and f is the focal length in pixels as determined from the first alignment stage.

The remaining terms, Ω_x and Ω_z , describe the combined effects of camera orientation and deviations of the viewplane's orientation from perpendicular to the optical axis. Ideally, the viewplane would be normal to the optical axis, but manufacturing tolerances allow these numbers to vary slightly [27].

$$\Omega_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \omega_x & -\sin \omega_x \\ 0 & \sin \omega_x & \cos \omega_x \end{bmatrix} \quad (9)$$

$$\Omega_z = \begin{bmatrix} \cos \omega_z & -\sin \omega_z & 0 \\ \sin \omega_z & \cos \omega_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (10)$$

In addition, the ω_z term is indistinguishable from the camera's roll angle and, thus, represents both the image sensor's and the camera's rotation. Likewise, ω_x is combined with an implicit parameter, ϕ , that represents the relative tilt of the camera's optical axis out of the panning plane. If ϕ is zero, the images are all tangent to a cylinder and for a nonzero ϕ the projections are tangent to a cone.

This gives six unknown parameters, $(C_x, C_y, \sigma, \rho, \omega_x, \omega_z)$, to be determined in the second stage of the registration process. Notice that, when combined with the θ_i and f parameters determined in the first stage, we have a total of eight parameters for each image, which is consistent with the number of free parameters in a general homogeneous matrix.

The structural matrix, S , is determined by minimizing the following error function:

$$\text{error}(C_x, C_y, \sigma, \rho, \omega_x, \omega_z) = \sum_{i=1}^n |1 - \text{Correlation}(I_{i-1}, S^{-1}R_y S I_i)| \quad (11)$$

where I_{i-1} and I_i represent the center third of the pixels from images $i-1$ and i respectively. Using Powell's multivariable minimization method [23] with the following initial values for our six parameters,

$$\begin{aligned} C_x &= \frac{\text{image width}}{2} & C_y &= \frac{\text{image height}}{2} \\ \sigma &= 0 & \rho &= 1 & \omega_x &= 0 & \omega_z &= 0 \end{aligned} \quad (12)$$

the solution typically converges in about six iterations. At this point we will have a new estimate for (C_x, C_y) which can be fed back into stage one, and the entire process can be repeated.

The registration process results in a single camera model, $S(C_x, C_y, \sigma, \rho, \omega_x, \omega_z, f)$, and a set of the relative rotations, θ_i , between each of the sampled images. Using these parameters, we can compose mapping functions from any image in the sequence to any other image as follows:

$$I_i = S^{-1}R_{y_{i-1}}R_{y_{i-2}}R_{y_{i-3}}\dots R_{y_j}S I_j \quad (13)$$

We can also reproject images onto arbitrary surfaces by modifying S . Since each image pixel determines the equation of a ray from the center-of-projection, the reprojection process merely involves intersecting these rays with the projection manifold.

4.3 Determining Image Flow Fields

Given two or more cylindrical projections from different positions within a static scene, we can determine the relative positions of centers-of-projection and establish geometric constraints across all potential reprojections. These positions can only be computed to a scale factor. An intuitive argument for this is that from a set of images alone, one cannot determine if the observer is looking at a model or a full-sized scene. This implies that at least one measurement is required to establish a scale factor. The measurement may be taken either between features that are mutually visible within images, or the distance between the acquired image's camera positions can be used. Both techniques have been used with little difference in results.

To establish the relative relationships between any pair of cylindrical projections, the user specifies a set of corresponding points that are visible from both views. These points can be treated as rays in space with the following form:

$$x_a(\theta, v) = C_a + tD_a(\theta, v) \quad D_a(\theta, v) = \begin{bmatrix} \cos(\phi_a - \theta) \\ \sin(\phi_a - \theta) \\ k_a(C_{v_a} - v) \end{bmatrix} \quad (14)$$

where $C_a = (A_x, A_y, A_z)$ is the unknown position of the cylinder's center of projection, ϕ_a is the rotational offset which aligns the angular orientation of the cylinders to a common frame, k_a is a scale factor which determines the vertical field-of-view, and C_{v_a} is the scanline where the center of projection would project onto the scene (i.e. the line of zero elevation, like the equator of a spherical map).

A pair of tiepoints, one from each image, establishes a pair of rays which ideally intersect at the point in space identified by the tiepoint. In general, however, these rays are skewed. Therefore, we use the point that is simultaneously closest to both rays as an estimate of the point's position, p , as determined by the following derivation.

$$p(\theta_a, v_a, \theta_b, v_b) = \frac{x_a - x_b}{2} \quad (15)$$

where (θ_a, v_a) and (θ_b, v_b) are the tiepoint coordinates on cylinders A and B respectively. The two points, x_a and x_b , are given by

$$\begin{aligned} x_a &= C_a + tD_a(\theta_a, v_a) \\ x_b &= C_b + sD_b(\theta_b, v_b) \end{aligned} \quad (16)$$

where

$$\begin{aligned} t &= \frac{\text{Det}[C_a - C_b, D_b(\theta_b, v_b), D_a(\theta_a, v_a) \times D_b(\theta_b, v_b)]}{|D_a(\theta_a, v_a) \times D_b(\theta_b, v_b)|^2} \\ s &= \frac{\text{Det}[C_b - C_a, D_a(\theta_a, v_a), D_b(\theta_b, v_b) \times D_a(\theta_a, v_a)]}{|D_b(\theta_b, v_b) \times D_a(\theta_a, v_a)|^2} \end{aligned} \quad (17)$$

This allows us to pose the problem of finding a cylinder's position as a minimization problem. For each pair of cylinders we have two sets of six unknowns, $(A_x, A_y, A_z, \phi_a, k_a, C_{v_a}), (B_x, B_y, B_z, \phi_b, k_b, C_{v_b})$. In general, we have good estimates for the k and C_v terms, since these values are found by the registration phase. The position of the cylinders is determined by minimizing the distance between these skewed rays. We also choose to assign a penalty for shrinking the vertical height of the cylinder in order to bring points closer together. This penalty could be eliminated by accepting either the k or C_v values given by the registration.

We have tested this approach using from 12 to 500 tiepoints, and have found that it converges to a solution in as few as ten iterations of Powell's method. Since no correlation step is required, this process is considerably faster than the minimization step required to determine the structural matrix, S .

The use of a cylindrical projection introduces significant geometric constraints on where a point viewed in one projection might appear in a second. We can capitalize on these restrictions when we wish to automatically identify corresponding points across cylinders. While an initial set of 100 to 500 tiepoints might be established by hand, this process is far too tedious to establish a mapping for the entire cylinder. Next, we present a geometric constraint for cylindrical projections that determines the possible positions of a point given its position in some other cylinder. This constraint plays the same role that the epipolar geometries [18], [9], used in the computer vision community for depth-from-stereo computations, play for planar projections.

First, we will present an intuitive argument for the existence of such an invariant. Consider yourself at the center of a cylindrical projection. Every point on the cylinder around you corresponds to a ray in space as given by the cylindrical epipolar geometry equation. When one of the rays is observed from a second cylinder, its path projects to a curve which appears to begin at the point corresponding to the origin of the first cylinder, and it is constrained to pass through

the point's image on the second cylinder.

This same argument could obviously have been made for a planar projection. And, since two points are identified (the virtual image of the camera in the second projection along with the corresponding point) and, because a planar projection preserve lines, a unique, so called epipolar line is defined. This is the basis for an epipolar geometry, which identifies pairs of lines in two planar projections such that if a point falls upon one line in the first image, it is constrained to fall on the corresponding line in the second image. The existence of this invariant reduces the search for corresponding points from an $O(N^2)$ problem to $O(N)$.

Cylindrical projections, however, do not preserve lines. In general, lines map to quadratic parametric curves on the surface of a cylinder. Surprisingly, we can completely specify the form of the curve with no more information than was needed in the planar case.

The paths of these curves are uniquely determined sinusoids. This *cylindrical epipolar geometry* is established by the following equation.

$$v(\theta) = \frac{N_x \cos(\phi_a - \theta) + N_y \sin(\phi_a - \theta)}{N_z k_a} + C_{v_a} \quad (18)$$

where

$$N = (C_b - C_a) \times D_a(\theta_a, v_a) \quad (19)$$

This formula gives a concise expression for the curve formed by the projection of a ray across the surface of a cylinder, where the ray is specified by its position on some other cylinder.

This cylindrical epipolar relationship can be used to establish image flow fields using standard computer vision methods. We have used correlation methods [9], a simulated annealing-like relaxation method [3], and the method of differences [20] to compute stereo disparities between cylinder pairs. Each method has its strengths and weaknesses. We refer the reader to the references for further details.

4.4 Plenoptic Function Reconstruction

Our image-based rendering system takes as input cylindrically projected panoramic reference images along with scalar disparity images relating each cylinder pair. This information is used to automatically generate image warps that map reference images to arbitrary cylindrical or planar views that are capable of describing both occlusion and perspective effects.

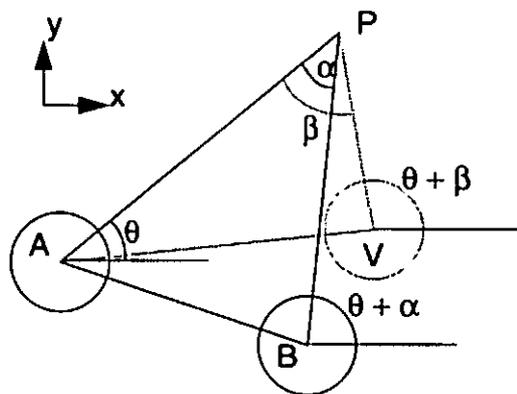


FIGURE 2. Diagram showing the transfer of the known disparity values between cylinders A and B to a new viewing position V.

We begin with a description of cylindrical-to-cylindrical mappings. Each angular disparity value, α , of the disparity images, can be readily converted into an image flow vector field, $(\theta + \alpha, v(\theta + \alpha))$ using the epipolar relation given by Equation 18 for each position on the cylinder, (θ, v) . We can transfer disparity values from the known cylindrical pair to a new cylindrical projection

in an arbitrary position, as in Figure 2, using the following equations.

$$\begin{aligned}
 a &= (B_x - V_x) \cos(\phi_A - \theta) + (B_y - V_y) \sin(\phi_A - \theta) \\
 b &= (B_y - A_y) \cos(\phi_A - \theta) - (B_x - A_x) \sin(\phi_A - \theta) \\
 c &= (V_y - A_y) \cos(\phi_A - \theta) - (V_x - A_x) \sin(\phi_A - \theta) \quad (20) \\
 \cot(\beta(\theta, v)) &= \frac{a + b \cot(\alpha(\theta, v))}{c}
 \end{aligned}$$

By precomputing $[\cos(\phi_i - \theta), \sin(\phi_i - \theta)]$ for each column of the cylindrical reference image and storing $\cot(\alpha)$ in place of the disparity image, this transfer operation can be computed at interactive speeds.

Typically, once the disparity images have been transferred to their target, the cylindrical projection would be reprojected as a planar image for viewing. This reprojection can be combined with the disparity transfer to give a single image warp that performs both operations. To accomplish this, a new intermediate quantity, δ , called the *generalized angular disparity* is defined as follows:

$$\begin{aligned}
 d &= (B_x - A_x) \cos(\phi_A - \theta) + (B_y - A_y) \sin(\phi_A - \theta) \\
 \delta(\theta, v) &= \frac{1}{d + b \cot(\alpha(\theta, v))} \quad (21)
 \end{aligned}$$

This scalar function is the cylindrical equivalent to the classical stereo disparity. Finally, a composite image warp from a given reference image to any arbitrary planar projection can be defined as

$$\begin{aligned}
 x(\theta, v) &= \frac{r \cdot D_A(\theta, v) + k_r \delta(\theta, v)}{n \cdot D_A(\theta, v) + k_n \delta(\theta, v)} \\
 y(\theta, v) &= \frac{s \cdot D_A(\theta, v) + k_s \delta(\theta, v)}{n \cdot D_A(\theta, v) + k_n \delta(\theta, v)} \quad (22)
 \end{aligned}$$

where

$$\begin{aligned}
 r &= \mathbf{v} \times \mathbf{o} & k_r &= r \cdot (\mathbf{C}_a - \mathbf{V}) \\
 s &= \mathbf{o} \times \mathbf{v} & k_s &= s \cdot (\mathbf{C}_a - \mathbf{V}) \\
 n &= \mathbf{o} \times \mathbf{v} & k_n &= n \cdot (\mathbf{C}_a - \mathbf{V}) \quad (23)
 \end{aligned}$$

and the vectors \mathbf{p} , \mathbf{o} , \mathbf{v} and \mathbf{V} are defined by the desired view as shown in Figure 3.

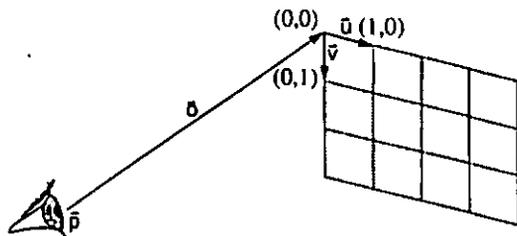


FIGURE 3. The center-of-projection, \mathbf{p} , a vector to the origin, \mathbf{o} , and two spanning vectors (\mathbf{u} and \mathbf{v}) uniquely determine the planar projection.

In the case where $\delta(\theta, v) = \text{constant}$, the image warp defined by Equation 22, reduces to a simple reprojection of the cylindrical image to a desired planar view. The perturbation introduced by allowing $\delta(\theta, v)$ to vary over the image allows arbitrary shape and occlusions to be represented.

Potentially, both the cylinder transfer and image warping approaches are many-to-one mappings. For this reason we must consider visibility. The following simple algorithm can be used to determine an enumeration of the cylindrical mesh which guarantees a proper back-to-front ordering, (See Appendix). We project the desired viewing position onto the reference cylinder being warped and partition the cylinder into four toroidal sheets. The sheet boundaries are defined by the θ and v coordinates of two points, as shown in Figure 4. One point is defined by the intersection of the cylinder

with the vector from the origin through the eye's position. The other is the intersection with the vector from the eye through the origin.

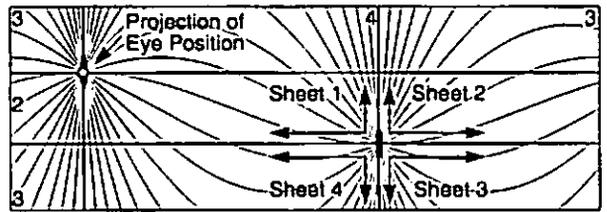


FIGURE 4. A back-to-front ordering of the image flow field can be established by projecting the eye's position onto the cylinder's surface and dividing it into four toroidal sheets.

Next, we enumerate each sheet such that the projected image of the desired viewpoint is the last point drawn. This simple partitioning and enumeration provides a back-to-front ordering for use by a painter's style rendering algorithm. This hidden-surface algorithm is a generalization of Anderson's [2] visible line algorithm to arbitrary projected grid surfaces. Additional details can be found in [21].

At this point, the plenoptic samples can be warped to their new position according to the image flow field. In general, these new pixel positions lie on an irregular grid, thus requiring some sort of reconstruction and resampling. We use a forward-mapping [28] reconstruction approach in the spirit of [27] in our prototype. This involves computing the projected kernel's size based on the current disparity value and the derivatives along the epipolar curves.

While the visibility method properly handles mesh folds, we still must consider the tears (or excessive stretching) produced by the exposure of previously occluded image regions. In view interpolation [8] a simple "distinguished color" heuristic is used based on the screen space projection of the neighboring pixel on the same scanline. This approach approximates stretching for small regions of occlusion, where the occluder still abuts the occluded region. And, for large exposed occluded regions, it tends to interpolate between the boundaries of the occluded region. These exposure events can be handled more robustly by combining, on a pixel-by-pixel basis, the results of multiple image warps according to the smallest-sized reconstruction kernel.

5. RESULTS

We collected a series of images using a video camcorder on a leveled tripod in the front yard of one of the author's home. Accurate leveling is not strictly necessary for the method to work. When the data were collected, no attempt was made to pan the camera at a uniform angular velocity. The autofocus and autoiris features of the camera were disabled, in order to maintain a constant focal length during the collection process. The frames were then digitized at a rate of approximately 5 frames per second to a resolution of 320 by 240 pixels. An example of three sequential frames are shown below.



Immediately after the collection of the first data set, the process was repeated at a second point approximately 60 inches from the first. The two image sequences were then separately registered using the methods described in Section 4.2. The images were reprojected onto the surface of a cylinder with a resolution of 3600 by 300 pixels. The results are shown in Figures 5a and 5b. The operating room scene, in Figure 5c, was also constructed using these same methods.

Next, the epipolar geometry was computed by specifying 12 tie-points on the front of the house. Additional tiepoints were gradually added to establish an initial disparity image for use by the simulated

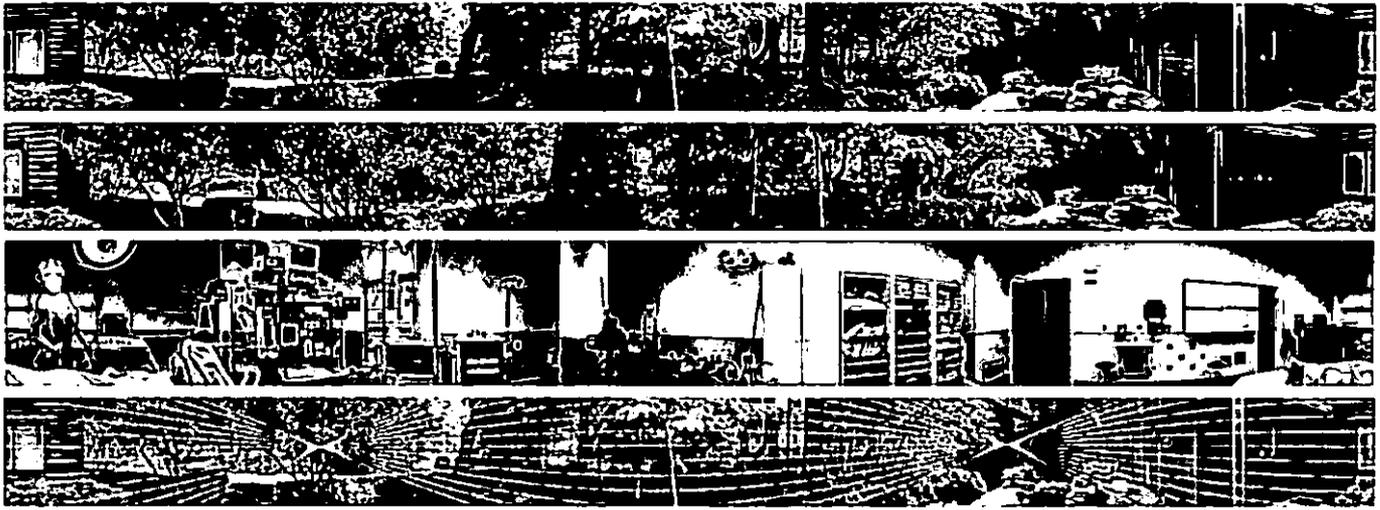


FIGURE 5. Cylindrical images a and b are panoramic views separated by approximately 60 inches. Image c is a panoramic view of an operating room. In image d, several epipolar curves are superimposed onto cylindrical image a.

annealing and method of differences stereo-correspondence routines. As these tiepoints were added, we also refined the epipolar geometry and cylinder position estimates. The change in cylinder position, however, was very slight. In Figure 5d, we show a cylindrical image with several epipolar curves superimposed. Notice how the curves all intersect at the alternate camera's virtual image and vanishing point.

After the disparity images are computed, they can be interactively warped to new viewing positions. The following four images show various reconstructions. When used interactively, the warped images provide a convincing kinetic depth effect.



6. CONCLUSIONS

The plenoptic function provides a consistent framework for image-based rendering systems. The various image-based methods, such as morphing and view interpolation, are characterized by the different ways they implement the three key steps of sampling, reconstructing, and resampling the plenoptic function.

We have described our approach to each of these steps. Our method for sampling the plenoptic function can be done with equipment that is commonly available, and it results in cylindrical samples about a point. All the necessary parameters are automatically estimated from a sequence of images resulting from panning a video camera through a full circle.

Reconstructing the function from these samples requires estimating the optic flow of points when the view point is translated. Though this problem can be very difficult, as evidenced by thirty years of computer vision and photogrammetry research, it is greatly simplified when the samples are relatively close together. This is because there is little change in the image between samples (which makes the estimation easier), and because the viewer is never far from

a sample (which makes accurate estimation less important).

Resampling the plenoptic function and reconstructing a planar projection are the key steps for display of images from arbitrary viewpoints. Our methods allow efficient determination of visibility and real-time display of visually rich environments on conventional workstations without special purpose graphics acceleration.

The plenoptic approach to modeling and display will provide robust and high-fidelity models of environments based entirely on a set of reference projections. The degree of realism will be determined by the resolution of the reference images rather than the number of primitives used in describing the scene. Finally, the difficulty of producing realistic models of real environments will be greatly reduced by replacing geometry with images.

ACKNOWLEDGMENTS

We are indebted to the following individuals for their contributions and suggestions on this work: Henry Fuchs, Andrei State, Kevin Arthur, Donna McMillan, and all the members of the UNC/UPenn collaborative telepresence-research group.

This research is supported in part by Advanced Research Projects Agency contract no. DABT63-93-C-0048, NSF Cooperative Agreement no. ASC-8920219, Advanced Research Projects Agency order no. A410, and National Science Foundation grant no. MIP-9306208. Approved by ARPA for public release - distribution unlimited.

REFERENCES

- [1] Adelson, E. H., and J. R. Bergen, "The Plenoptic Function and the Elements of Early Vision," *Computational Models of Visual Processing*, Chapter 1, Edited by Michael Landy and J. Anthony Movshon. The MIT Press, Cambridge, Mass. 1991.
- [2] Anderson, D., "Hidden Line Elimination in Projected Grid Surfaces," *ACM Transactions on Graphics*, October 1982.
- [3] Barnard, S.T. "A Stochastic Approach to Stereo Vision," *SRI International, Technical Note 373*, April 4, 1986.
- [4] Beier, T. and S. Neely, "Feature-Based Image Metamorphosis," *Computer Graphics (SIGGRAPH'92 Proceedings)*, Vol. 26, No. 2, pp. 35-42, July 1992.
- [5] Blinn, J. F. and M. E. Newell, "Texture and Reflection in Computer Generated Images," *Communications of the ACM*, vol. 19, no. 10, pp. 542-547, October 1976.
- [6] Bolles, R. C., H. H. Baker, and D. H. Marimont, "Epipolar-Plane Image Analysis: An Approach to Determining Structure from Motion," *International Journal of Computer Vision*, Vol. 1, 1987.
- [7] Catmull, E., "A Subdivision Algorithm for Computer Display of Curved Surfaces" (Ph. D. Thesis), Department of Computer Sci-

ence, University of Utah, Tech. Report UTEC-CSc-74-133, December 1974.

[8] Chen, S. E. and L. Williams. "View Interpolation for Image Synthesis," *Computer Graphics (SIGGRAPH'93 Proceedings)*, pp. 279-288, July 1993.

[9] Faugeras, O., *Three-dimensional Computer Vision: A Geometric Viewpoint*, The MIT Press, Cambridge, Massachusetts, 1993.

[10] Greene, N., "Environment Mapping and Other Applications of World Projections," *IEEE Computer Graphics and Applications*, November 1986.

[11] Hartley, R.I., "Self-Calibration from Multiple Views with a Rotating Camera," *Proceedings of the European Conference on Computer Vision*, May 1994.

[12] Heckbert, P. S., "Fundamentals of Texture Mapping and Image Warping," Masters Thesis, Dept. of EECS, UCB, Technical Report No. UCB/CSD 89/516, June 1989.

[13] Horn, B., and B.G. Schunck, "Determining Optical Flow," *Artificial Intelligence*, Vol. 17, 1981.

[14] Kanatani, K., "Transformation of Optical Flow by Camera Rotation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 10, No. 2, March 1988.

[15] Laveau, S. and O. Faugeras, "3-D Scene Representation as a Collection of Images and Fundamental Matrices," INRIA, Technical Report No. 2205, February, 1994.

[16] Lenz, R. K. and R. Y. Tsai, "Techniques for Calibration of the Scale Factor and Image Center for High Accuracy 3D Machine Vision Metrology," *Proceedings of IEEE International Conference on Robotics and Automation*, March 31 - April 3, 1987.

[17] Lippman, A., "Movie-Maps: An Application of the Optical Video-disc to Computer Graphics," *SIGGRAPH '80 Proceedings*, 1980.

[18] Longuet-Higgins, H. C., "A Computer Algorithm for Reconstructing a Scene from Two Projections," *Nature*, Vol. 293, September 1981.

[19] Longuet-Higgins, H. C., "The Reconstruction of a Scene From Two Projections - Configurations That Defeat the 8-Point Algorithm," *Proceedings of the First IEEE Conference on Artificial Intelligence Applications*, Dec 1984.

[20] Lucas, B., and T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision," *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, Vancouver, 1981.

[21] McMillan, Leonard, "A List-Priority Rendering Algorithm for Redisplaying Projected Surfaces," Department of Computer Science, UNC, Technical Report TR95-005, 1995.

[22] Mann, S. and R. W. Picard, "Virtual Bellows: Constructing High Quality Stills from Video," *Proceedings of the First IEEE International Conference on Image Processing*, November 1994.

[23] Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C*, Cambridge University Press, Cambridge, Massachusetts, pp. 309-317, 1988.

[24] Regan, M., and R. Pose, "Priority Rendering with a Virtual Reality Address Recalculation Pipeline," *SIGGRAPH'94 Proceedings*, 1994.

[25] Szeliski, R., "Image Mosaicing for Tele-Reality Applications," DEC and Cambridge Research Lab Technical Report, CRL 94/2, May 1994.

[26] Tomasi, C., and T. Kanade, "Shape and Motion from Image Streams: a Factorization Method; Full Report on the Orthographic Case," Technical Report, CMU-CS-92-104, Carnegie Mellon University, March 1992.

[27] Tsai, R. Y., "A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-Shelf TV Cameras and Lenses," *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 4, August 1987.

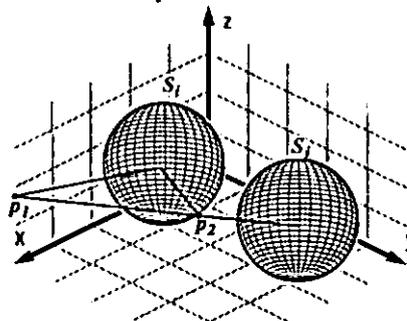
[28] Westover, L. A., "Footprint Evaluation for Volume Rendering," *SIGGRAPH'90 Proceedings*, August 1990.

[29] Wolberg, G., *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, CA, 1990.

APPENDIX

We will show how occlusion compatible mappings can be determined on local spherical frames embedded within a global cartesian frame, W . The projected visibility algorithm for cylindrical surfaces given in the paper can be derived by reducing it to this spherical case.

First, consider an isolated topological multiplicity on the projective mapping from S_i to S_j , as shown below



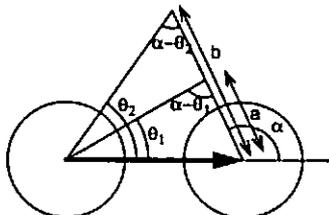
Theorem 1: In the generic case, the points of a topological multiplicity induced by a mapping from S_i to S_j and the two frame origins are coplanar.

Proof: The points of the topological multiplicity are colinear with the origin of S_j since they share angular coordinates. A second line segment connects the local frame origins, S_i and S_j . In general, these two lines are distinct and thus they define a plane in three space.

Thus, a single affine transformation, A , of W can accomplish the following results.

- Translate S_i to the origin
- Rotate S_j to lie on the x -axis
- Rotate the line along the multiplicity into the xy -plane
- Scale the system so that S_j has the coordinate $(1,0,0)$.

With this transformation we can consider the multiplicity entirely within the xy -plane, as shown in the following figure.



Theorem 2: If $\cos\theta_1 > \cos\theta_2$ and $(\theta_1, \theta_2, \alpha) \in [0, \pi]$ then $a < b$.

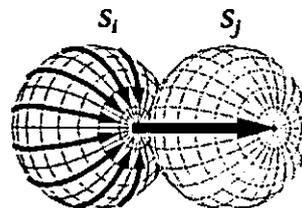
Proof: The length of sides a and b can be computed in terms of the angles θ_1, θ_2 and α using the law of sines as follows.

$$\frac{a}{\sin\theta_1} = \frac{1}{\sin(\alpha - \theta_1)} \quad \frac{b}{\sin\theta_2} = \frac{1}{\sin(\alpha - \theta_2)}$$

$$\frac{a}{b} = \frac{\sin\alpha \cot\theta_2 - \cos\alpha}{\sin\alpha \cot\theta_1 - \cos\alpha}$$

if $\cos\theta_1 > \cos\theta_2$, then $\cot\theta_1 > \cot\theta_2$, thus $a < b$

Thus, an occlusion compatible mapping, can be determined by enumerating the topological mesh defined on AS_i in an order of increasing $\cos\theta$, while allowing later mesh facets to overwrite previous ones. This mapping is occlusion compatible since, by Theorem 2, greater range values will always proceed lesser values at all multiplicities. Notice, that this mapping procedure only considers the changes in the local frame's world coordinates, and makes no use of the range information itself.



Rendering With Coherent Layers

Jed Lengyel and John Snyder
Microsoft Research

Abstract

For decades, animated cartoons and movie special effects have factored the rendering of a scene into layers that are updated independently and composed in the final display. We apply layer factorization to real-time computer graphics. The layers allow targeting of resources, whether the ink and paint artists of cartoons or the graphics pipeline as described here, to those parts of the scene that are most important.

To take advantage of frame-to-frame coherence, we generalize layer factorization to apply to both dynamic geometric objects and terms of the shading model, introduce new ways to trade off fidelity for resource use in individual layers, and show how to compute warps that reuse renderings for multiple frames. We describe quantities, called *fiducials*, that measure the fidelity of approximations to the original image. Layer update rates, spatial resolution, and other quality parameters are determined by geometric, photometric, visibility, and sampling fiducials weighted by the content author's preferences. We also compare the fidelity of various types of reuse warps and demonstrate the suitability of the affine warp.

Using Talisman, a hardware architecture with an efficient layer primitive, the work presented here dramatically improves the geometric complexity and shading quality of scenes rendered in real-time.

CR Categories and Subject Descriptors: I.3.3 [Computer Graphics]: Picture/Image Generation; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism.

Additional Keywords: sprite, affine transformation, image compositing, image-based rendering, Talisman

1 Introduction

The layered pipeline separates or *factors* the scene into layers that represent the appearance of an object (e.g., a space ship separate from the star field background) or a special lighting effect (e.g., a shadow, reflection, highlight, explosion, or lens flare.) Each layer produces a 2D-image stream as well as a stream of 2D transformations that place the image on the display. We use *sprite* to refer to a layer's image (with alpha channel) and transformation together.

The layered pipeline decouples rendering of layers from their display. Specifically, the sprite transformation may be updated more frequently than the sprite image. Rendering (using 3D CG) updates the sprite image only when needed. Sprite transforming and compositing [Porter84] occur at display rates. The sprite transformation scales low-resolution sprites up to the display resolution, and transforms sprites rendered earlier to approximate their later appearance. In other words, the sprite transformation interpolates rendered image streams to display resolution in both space and time.

Layered rendering has several advantages for real-time CG. First, layered rendering better exploits coherence by separating fast-moving foreground objects from slowly changing background layers. Second, layered rendering more optimally targets rendering resources by allowing less important layers to be degraded to conserve resources for more important layers. Finally, layered rendering naturally integrates 2D elements such as overlaid video, offline rendered sprites, or hand-animated characters into 3D scenes.

As an architectural feature, decoupling rendering from compositing is advantageous. Compositing is 2D rather than 3D, requires no

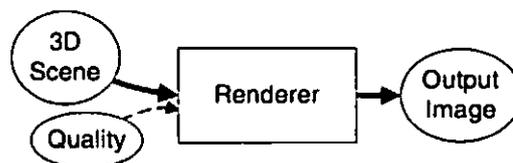


Figure 1: **TRADITIONAL PIPELINE** processes the entire scene database to produce each output image. The quality parameters for texture and geometry (such as level-of-detail) may be set independently for each object in the scene. However, the sampling resolutions in time (frame rate) and space (image resolution and compression) are the same for all objects in the scene.

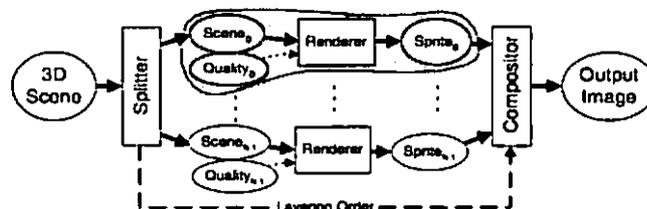


Figure 2: **LAYERED PIPELINE** partitions the scene into independent layers. A single layer's pipeline (highlighted at top) is similar to the traditional pipeline. By adjusting each layer's quality controls, the content author targets rendering resources to perceptually important parts of the scene. Slowly changing or unimportant layers are updated at lower frame rates, at lower resolution, and with higher compression.

z-buffer, no lighting computations, no polygon edge antialiasing, and must handle few sprites (which are analogous to texture-mapped polygons) relative to the number of polygons in the rendered geometry. This simplicity allows compositing hardware to be made with pixel fill rates much higher than 3D rendering hardware. Our investigation demonstrates that the saving in 3D rendering justifies the extra hardware expense of a compositor.

The layered pipeline augments the set of traditional rendering quality parameters such as geometric level-of-detail and shading model (e.g., flat-, Gouraud-, or Phong-shaded), with the temporal and spatial resolution parameters of each layer. The *regulator* adjusts the quality parameters in order to achieve optimal quality within fixed rendering resources. The regulator dynamically measures both the costs of changing the quality parameters – how much more or less of the rendering budget they will consume – and the benefits – how much improvement or loss in fidelity will occur.

The specific contributions of this paper include extending the generality of factoring. While some authors have considered factoring over static geometric objects [Regan94, Maciel95, Shade96, Schaufler96ab], we consider dynamic situations and factoring over shading expressions (Section 2). We describe how to render using the layered pipeline (Section 3). We investigate different types of sprite transformations and show why an affine transformation is a good choice (Section 4). We discuss low-computation measures of image fidelity, which we call *fiducials*, and identify several classes of fiducials (Section 5). We add the spatial and temporal resolution of layers as regulation parameters and propose a simple regulator that balances them to optimize image fidelity (Section 6). Finally, we demonstrate that the ideas presented here enhance performance of the Talisman architecture by factors of 3-10, by using interpolated triple-framing or by regulating the heterogeneous update of sprite images (Section 7).

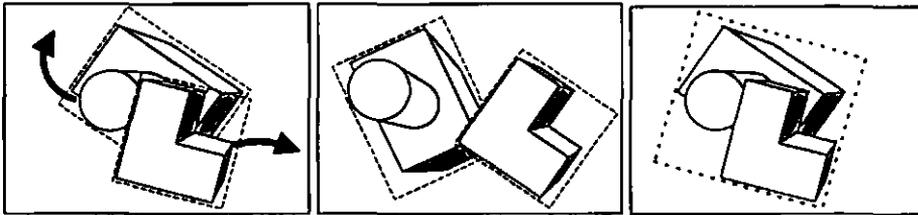


Figure 3: INDEPENDENT UPDATE depends on choice of factoring. The left and middle figures show how factoring the geometry into two separate layers allows each layer to be reused. The right figure shows a less effective partition.

1.1 The Layered Pipeline and Talisman

The Talisman reference hardware architecture was designed to support the layered pipeline (see [Torborg96] for details.) Rendering occurs within one 32×32 chunk at a time, so that z-buffer and fragment information for antialiasing can be stored on-chip. The resulting sprites with alpha channel are compressed and written to sprite memory. In parallel with 3D rendering, for every frame, the compositor applies an affine warp to each of an ordered set of sprites uncompressed from sprite memory and composites the sprites just ahead of the video refresh, eliminating the need for a frame buffer. Sprite composition is limited to the "over" operator [Porter84].

Although our experiments assume the Talisman reference architecture, the ideas can be usefully applied to traditional architectures. Layer composition can be emulated with rendering hardware that supports texture mapping with transparency by dedicating some of the pixel-fill rate of the renderer for sprite composition. This may be a good sacrifice if sprite rendering is polygon limited rather than pixel fill limited. Clearly though, Talisman is a superior layered pipeline in that sprite composition is "for free" (i.e., sacrifices few rendering resources) and very high speed (because of sprite compression and the simplicity of sprite composition in relation to rendering).¹

1.2 Previous Work

To avoid visibility sorting of layers, alternative architectures use what are essentially sprites with z information per pixel [Molnar92, Regan94, Mark97]. Such systems are more costly in computation, bandwidth, and storage requirements since z must be stored and transmitted to a more complicated compositor. Z information is also difficult to interpolate and compress. We observe that z information per pixel is greatly redundant when used solely to determine a layering order. But such an ordering is necessary to ensure an antialiased result.² Our approach of factoring into layers allows warping per coherent object. It also avoids problems with uncovering of depth-shadowed information. Of course, sprites could store multiple z layers per pixel, a prohibitively costly approach for hardware, but one near to ours in spirit. Such a scheme stores all the layers within each pixel, rather than all the pixels for each layer.³

[Funkhouser93] adjusts rendering parameters to extract the best quality. We add sprite resolution and update rate to the set of regulated parameters and make photometric measurements rather than relying on a *a priori* assignment of benefit to sampling rate. [Maciel95] takes a similar approach but use fiducials and impostor representations optimized for walkthroughs of static scenes.

Taking advantage of temporal coherence has been an ongoing theme in computer graphics [Hubschman81, Shelley82]. [Hofmann89] presents techniques for measuring how much camera

¹ In a prototype implementation of Talisman, the compositor is planned to run at 320M pixels/second compared to 40Mps for the renderer.

² Penetrating z-sprites will have a point-sampled and thus aliased boundary where visibility switches.

³ Post-warping of unfactored z-images also fails to address the case of independently moving objects.

movement is allowed before changes in the projected geometry exceed a given tolerance. [Chen93, Chen95] show how to take advantage of coherence between viewpoints to produce nearly constant cost per frame walkthroughs of static environments. [McMillan95] re-projects images to produce an arbitrary view.

Our use of temporal coherence is most similar to [Regan94], who observed that not all objects need updating at the display rate. Rather than factoring globally across

object sets requiring a common update rate, our scheme factors over geometric objects and shading model terms, and accounts for relative motion between dynamic objects.

[Shade96] and [Schaufler96ab] use image caches and texture-mapped quadrilaterals to warp the image. This is conceptually similar to our work, but does not include the factoring across shading or the idea of real-time regulation of quality parameters. We harness simpler image transformations (affine rather than perspective) to achieve greater fidelity (Section 4.2). Our work also treats dynamic geometry.

Shading expressions [Cook84, Hanrahan90] have been studied extensively. [Dorsey95] factors shading expressions by light source and linearly combines the resulting images in the final display. [Gunter95] caches intermediate results. [Meier96] uses image processing techniques to factor shadow and highlight regions into separate layers which are then re-rendered using painterly techniques and finally composited. The novel aspects of our technique are the independent quality parameters for each layer and warp of cached terms.

2 Factoring

The guiding principle of our approach is to factor into separate layers elements that require different spatial or temporal sampling rates. This section discusses guidelines for manually factoring across geometry and shading, visibility sorting of layers, and annotating models with layer information.

2.1 Factoring Geometry

Geometry factoring should consider the following properties of objects and their motions:

1. *Relative velocity* – A sprite that contains two objects moving away from each other must be updated more frequently than two sprites each containing a single object (Figure 3). Relative velocity also applies to shading.
2. *Perceptual distinctness* – Background elements require fewer samples in space and time than foreground elements, and so must be separated into layers to allow independent control of the quality parameters.
3. *Ratio of clear to "touched" pixels* – Aggregating many objects into a single layer typically wastes sprite area where no geometry projects. Finer decompositions are often tighter. Reducing wasted sprite space saves rendering resources especially in a chunked architecture where some chunks can be eliminated, and makes better use of the compositor, whose maximum speed limits the average depth complexity of sprites over the display.

2.2 Visibility Sorting

Visibility sorting of dynamic layer geometry can be automated. We have implemented a preliminary algorithm for which we provide a sketch here. A full discussion along with experimental results is in progress [Snyder97].

To determine visibility order for layers containing moving geometry, we construct an incrementally changing kd-tree based on a set of constant directions. A convex polyhedron bounds each layer's geometry, for which we can incrementally compute bounding extents

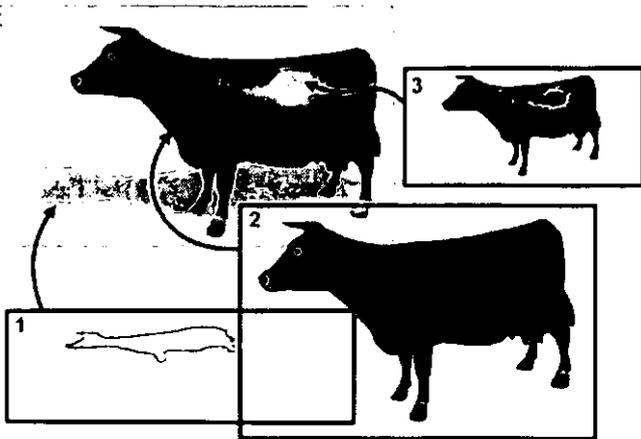


Figure 4: **FACTORED SHADING EXPRESSION** separates shadow, diffuse, and specular terms. In this example, the shadow and specular sprites are both computed at 25% (50% in x and y) of the display resolution. The shadow sprite modulates the color. The specular sprite adds to the output without changing alpha.

in each direction. The kd-tree quickly determines the set of objects that can possibly occlude a given object, based on these extents. Using this query, the visibility sort computes an incremental topological sort on the strongly connected components (which represent occlusion cycles) of the occlusion graph. Strongly connected components must be temporarily aggregated in the same layer, since a priority ordering does not exist.⁴

2.3 Factoring Shading

Shading may also be factored into separate layers. Figure 4 shows a typical multipass example, in which a shadow layer modulates the fully illuminated scene, and a reflection layer adds a reflection (in this case the reflection is the specular reflection from a light.) Figure 5 shows a schematic view of the steps needed to create the multipass image. The shadow layer is generated from a depth map rendered from the point of view of a light. The reflection layer is generated from a texture map produced by a separate rendering with a reflected camera. The layers shown in the figure represent post-modulation images using the same camera. With traditional architectures, the three layers are combined in the frame buffer using pixel blend operations supported by the 3D hardware, as described in [Segal92].

Shadows and reflections may instead be separated into layers as shown in Figure 6, so that the blend takes place in the compositor rather than the renderer. We call these *shade sprites* in reference to shade trees [Cook84]. To take advantage of temporal coherence, highlights from fast moving lights, reflections of fast moving reflected geometry, and animated texture maps should be in separate layers and rendered at higher frame rates than the receiving geometry. To take advantage of spatial coherence, blurry highlights, reflections, or shadows should be in separate layers and given fewer pixel samples.

For reflections, the correctness of using the compositor is evident because the reflection term is simply added to the rest of the shading. More generally, any terms of the shading expression that are combined with '+' or 'over' may be split into separate layers. 'A + B' can be computed using 'A over B' and setting A's alpha channel to zero.

The separation of shadows is slightly more difficult. The shadowing term multiplies each part of the shading expression that depends on a given light source. Many such terms can be added for

⁴ At least, an ordering does not exist with respect to hulls formed by the set of bounding directions, which is a more conservative test than with the original bounding polyhedra. Note that visibility order for aggregated layers is computed simply by rendering into the same hardware z-buffer.

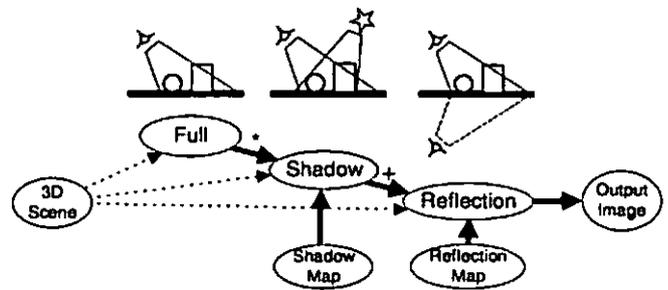


Figure 5: **MULTIPASS RENDERING** combines the results of several rendering passes to produce effects such as shadows and reflections. With a traditional architecture, the rendering passes are combined using blending operations in the 3D renderer (multiplication for shadow modulation and addition for adding reflections.)

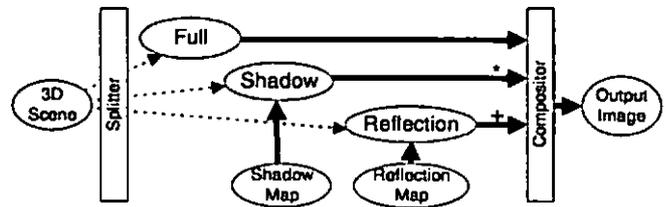


Figure 6: **SHADE SPRITES** are combined in the final composition phase to produce the multipass rendering. Each shading term may have different resolutions in space and time.

multiple shadowing light sources. We describe an approximation to multiplicative blending using the 'over' composition operator in an appendix.

Consider a simple example of a shading model with two textures and a shadow, $S(N \cdot L)(T_1 + T_2)$, where S is the shadowing term, N is the normal to the light, L is the light direction, and T_1 and T_2 are texture lookups. This shading model can be factored into three layers: S , $(N \cdot L)T_1$, and $(N \cdot L)T_2$, which are composited to produce the final image. The fact that this expression can be reordered and partial results cached is well known [Gunter95]. What we observe here is that each of these factors may be given different sampling resolutions in space and time, and interpolated to display resolutions.

As an aside, we believe shade sprites will be useful in authoring. When modifying the geometry and animation of a single primitive, the artist would like to see the current object in the context of the fully rendered and animated scene. By pre-rendering the layers that are not currently being manipulated, the bulk of the rendering resources may be applied to the current layer. The layers in front of the current layer may be made partially transparent (using a per-sprite alpha multiplier) to allow better manipulation in occluded environments. By using separate layers for each texture shading term, the artist can manipulate the texture-blending factors interactively at the full frame rate.

2.4 Model Annotation

The first step of model annotation is to break the scene into "parts" such as the base level joints in a hierarchical animated figure. The parts are containers for all of the standard CG elements such as polygon meshes, textures, materials, etc., required to render an image of the part. A part is the smallest renderable unit.

The second step is to group the parts into layers according to the guidelines described above. The distinction is made between parts and layers to allow for reuse of the parts, for example in both a shadow map layer and a shadow receiver layer.

The final step is to tag the layers with resource-use preferences relative to other layers in the scene. The preferences are relative so that total resource consumption can change when, for example, other applications are started (as discussed in Section 6).

3 Image Rendering

This section discusses how a layer's sprite image is created (i.e., rendered). Once created, the image can be warped in subsequent frames to approximate its underlying motion, until the approximation error grows too large. Although the discussion refers to the Talisman reference architecture with its 2D affine image warp, the ideas work for other warps as well.

3.1 Characteristic Bounding Polyhedron

The motion of the original geometry is tracked using a *characteristic bounding polyhedron*, usually containing a small number of vertices (Figure 7). For rigidly moving objects, the vertices of the characteristic polyhedron, called *characteristic points*, are transformed using the original geometry's time-varying transform.

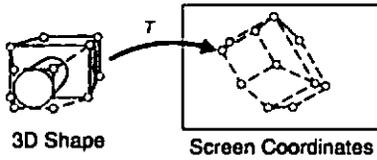


Figure 7: CHARACTERISTIC BOUNDING POLYHEDRON matches the shape of the geometry but has fewer vertices.

Nonrigidly deforming geometry can be tracked similarly by defining trajectories for each of the characteristic points. To group rigid bodies, we combine the characteristic bounding polyhedra, or calculate a single bounding polyhedron for the whole.

3.2 Sprite Extents

For a particular frame, there is no reason to render off-screen parts of the image. But in order to increase sprite reuse, it is often advantageous to expand the sprite image to include some off-screen area.

Figure 8a shows how clipping a sprite to the screen (solid box) prevents its later reuse because parts of the clipped image later become visible. In Figure 8b, the sprite extent (dashed box) has been enlarged to include regions that later become visible. The extra area to include depends on such factors as the screen velocity of the sprite (which suggests both where and how much the extents should be enlarged) and its expected duration of reuse.

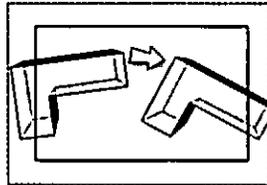
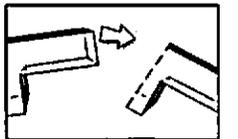


Figure 8: SPRITE EXTENTS enlarge the display extent to reuse sprites whose geometry lies partially off-screen.

3.3 Sprite Rendering Transformation

When creating a sprite image, we must consider a new transform in the pipeline in addition to the modeling, viewing, and projection transforms: a 2D affine transform that maps the sprite to the screen.

If T is the concatenation of the modeling, viewing, and projection matrices, a screen point p' is obtained from a modeling point p , by $p' = Tp$. For the sprite transformation, $p' = Aq$ where A is an affine transform and q is a point in sprite coordinates. To get the proper mapping of geometry to the display, the inverse 2D affine transform is appended to the projection matrix, so that $q = A^{-1}Tp$ results in the same screen point $p' = Aq = AA^{-1}Tp = Tp$ (Figure 9). The choice of matrix A determines how tightly the sprite fits the projected object. A tighter fit wastes fewer samples as discussed in Section 2.

To choose the affine transform that gives the tightest fit, we first project the vertices of the characteristic bounding polyhedron to the screen, clipping to the expanded sprite extent. Then, using discrete directions (from 2-30, depending on the desired tightness), we calculate 2D bounding slabs [Kay86]. Alternately, the slab directions may

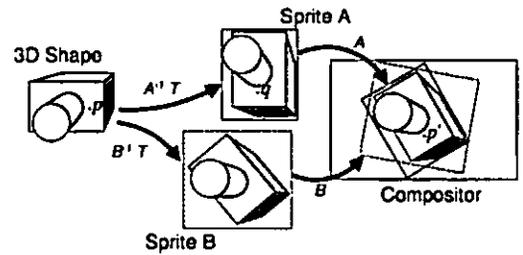


Figure 9: SPRITE RENDERING TRANSFORMATION maps the 3D shape into the sprite image. Affine transform B does not make the best use of image samples, while A fits the projected shape tightly.

be chosen by embedding preferred axes in the original model, and transforming the axes to screen space.

Using the bounding slabs, we find the bounding rectangle with the smallest area (Figure 10). The origin and edges of the rectangle determine the affine matrix. Initially, we searched for the smallest area parallelogram, but found the resulting affine transformation had too much anisotropy.

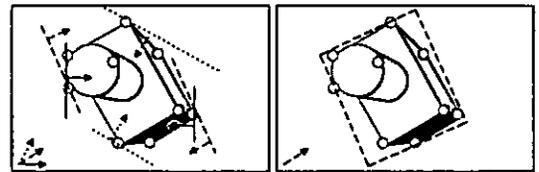


Figure 10: BOUNDING SLABS are obtained by taking the extremal values of the dot product of each slab direction with the characteristic points. A tight-fitting initial affine transform can be calculated by taking the minimum area rectangle or parallelogram that uses the slab directions.

3.4 Spatial Resolution

The choice of affine matrix A also determines how much the sprite is magnified on the display. Rendering using a sampling density less than the display's is useful for less important objects or for intentional blurring (Figure 11). The default is to use the same sampling density as the screen, by using the length in pixels of each side of the parallelogram from Section 3.3. See Figure 24 for an example of different sampling resolutions per sprite.

For a linear motion blur effect, the sprite sampling along one of the axes may be reduced to blur along that axis. The sprite rendering transformation should align one of the coordinate axes to the object's velocity vector by setting the bounding slab directions to the velocity vector and its perpendicular.

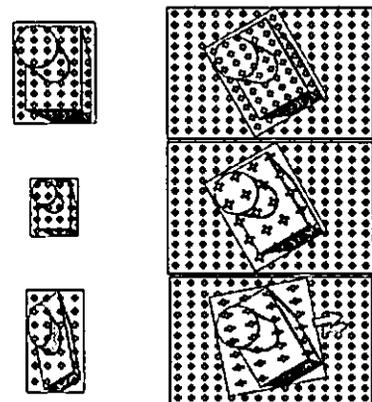


Figure 11: SPATIAL RESOLUTION is independent of the display resolution. The sampling density of the top sprite is the same as the screen. The middle sprite uses fewer samples than the screen, trading off pixel fill for blur. The bottom sprite aligns to the velocity vector and uses fewer samples along one dimension for a motion blur effect.

4 Image Warps

To reuse a rendered sprite image in subsequent frames, an image warp is used to approximate the actual motion of the object. We use the projected vertices of the bounding polyhedron (the characteristic points) to track the object's motion, as shown in Figure 12.

To reuse images where objects are in transition from off-screen to on-screen, and to prevent large distortions (i.e., ill-conditioning of the resulting systems of equations), the characteristic bounding polyhedron is clipped to the viewing frustum, which may be enlarged from the display's as discussed in Section 3.2. The clipped points are added to the set of characteristic points (Figure 13) and used to determine an approximating sprite transformation as described below.

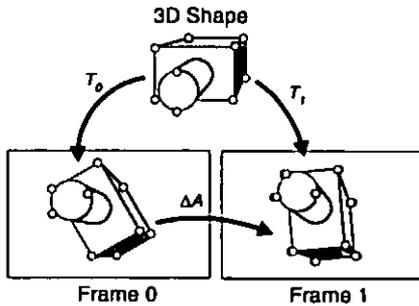


Figure 12: MATCHING CHARACTERISTIC POINTS on the 3D shape are projected to the screen to find a transform A that best matches the original points (white) to the points in the new frame (black).

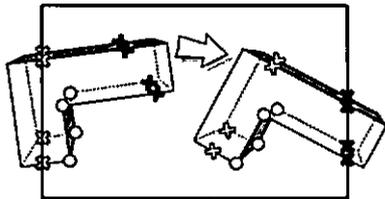


Figure 13: CLIPPED CHARACTERISTIC POLYHEDRON adds corresponding points introduced by clipping the characteristic polyhedron at the last-rendered and current frames.

hedron vertices, ignoring the z values and adding a row of 1's to account for the translation

$$P = \begin{bmatrix} x_0 & \dots & x_{n-1} \\ y_0 & \dots & y_{n-1} \\ 1 & \dots & 1 \end{bmatrix}$$

where n is the number of points (at least 3 for the affine transform). Let \hat{P} be the matrix of characteristic points at the initial time and P be the matrix at the desired time t . We solve for the best least-squares transform that matches the two sets of image-space points [Xie95].

In an affine transform, the x and y dimensions are decoupled and so may be solved independently. To solve $A\hat{P} = P$ at time t for the best A , in the least-squares sense, we use normal equations:

$$\begin{aligned} A\hat{P}\hat{P}^T &= P\hat{P}^T \\ A &= P\hat{P}^T(\hat{P}\hat{P}^T)^{-1} \end{aligned}$$

The normal-equations technique works well in practice, as long as the projected points are reasonably distributed. Adding the clipped characteristic points ensures that $\hat{P}\hat{P}^T$ is not rank deficient. Much of the right hand side may be collected into a single vector K that may be reused for subsequent frames.

$$\begin{aligned} K &= \hat{P}^T(P\hat{P}^T)^{-1} \\ A &= PK \end{aligned}$$

To calculate K requires the accumulation and inverse of a symmetric 3×3 matrix.

4.2 Comparison of Warps

Clearly, other types of image warps can be used in place of the affine described above. In order to compare alternative image warps, we ran a series of experiments to

1. measure update rate as a function of maximum geometric error for various warps, and
2. measure perceptual quality as a function of update rate for various warps.

Each series involved the animation of a moving rigid body and/or moving camera to see how well image warping approximates 3D motion. We tried several types of rigid bodies, including nearly planar and non-planar examples. We also tried many animated trajectories for each body including translations with fixed camera, translations accompanied by rotation of the body along various axes with various rotation rates, and head turning animations with fixed objects.

The types of 2D image warps considered were

1. pure translation,
2. translation with isotropic scale,
3. translation with independent scale in x and y ,
4. general affine, and
5. general perspective.

The fundamental simulation routine computes an animation given a geometric error threshold, attempting to minimize the number of renderings by approximating with an image warp of a particular type. A pseudo-code version is shown in Figure 14.

```

simulate(error-threshold, warp-type, animation)
{
  for each frame in animation
    compute screen position of characteristic points at current time
    compute transform (of warp-type) which best maps old
    cached positions to new positions
    compute maximum error for any characteristic point
    if error exceeds threshold
      re-render and cache current positions of characteristic points
    else
      display sprite with computed transformation
    endif
  endfor
  return total number of re-renderings
}

```

Figure 14: EXPERIMENT PSEUDOCODE shows steps used to compute update rates of various warps.

Ideally, we would like to compute approximations of each type that minimize the maximum error over all characteristic points, since this is the regulation metric. This is a difficult problem computationally, especially since the warping transformation happens at display rates for every layer in the animation. Minimizing the sum of squares of the error is much more tractable, yielding a simple linear system as we have already discussed. As a compromise, we simulated both kinds of error minimization: sum-of-squares and maximum error using an optimization method for L^∞ norms that iteratively applies the sum-of-square minimization, as described in [Gill81, pp. 96-98].

Further complicating matters, minimizing the error for perspective transformations is easier when done in homogeneous space rather than 2D space, again since the latter yields an 8×8 linear system rather than a difficult nonlinear optimization problem. We therefore included sum-of-square and maximum error methods for the first four (non-perspective) transformation types.

For perspective, we included sum-of-square minimization in homogeneous space (yielding a linear system as described above), maximum error in homogeneous space (using the technique of [Gill81]), and sum-of-square minimization in nonhomogeneous space (post-perspective divide), using gradient descent.⁵ The starting point

⁵ Minimization of the maximum nonhomogeneous error seemed wholly impractical for real-time implementation.

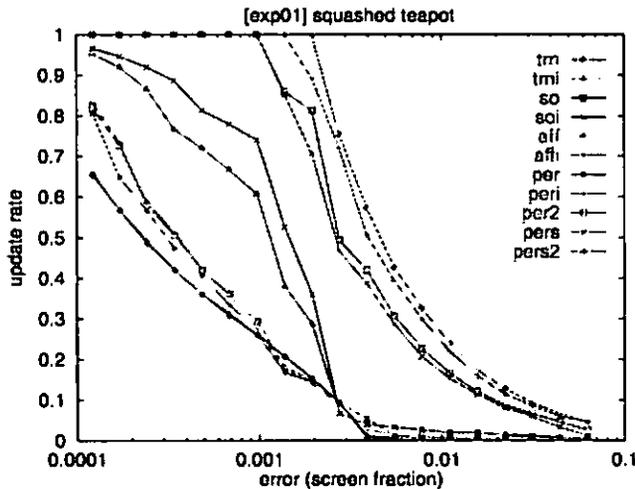


Figure 15: FLAT TEAPOT update-rate/error relations for the various warps show a surprisingly small difference between the affine and the perspective for a nearly flat object.

for the gradient descent was the sum-of-squares-error-minimizing affine transformation.

We also included a perspective transformation derived using the method of [Shade96], in which objects are replaced by a quadrilateral placed perpendicular to the view direction and through the center of the object's bounding box. Our derivation projects the characteristic points onto this quadrilateral, bounds the projected points with a rectangle, and projects the corners of the rectangle to the screen. The perspective transformation that maps the old corners of the rectangle to their current locations is selected as the approximation. In yet another version, Shade's method is used as a starting point and then refined using gradient descent in nonhomogeneous space with the sum-of-square error metric.

Representative results of the first series of experiments are shown in Figure 15 and Figure 16. In both figures, the experiment involved a rotating and translating teapot which is scaled nearly flat⁶ in Figure 15 and unscaled in Figure 16. Error thresholds ranging from 1/8 pixel to 64 pixels were used for each warp type/error minimization method, assuming an image size of 1024×1024, and the resulting rate of re-rendering measured via the simulation process described above. The meanings of the curve name keywords are as follows:

keyword	Warp type and minimization method
trn	translation, sum-of-square
trni	translation, max
so	translation with xy scale, sum-of-square
soi	translation with xy scale, max
aff	affine, sum-of-square
affi	affine, max
per	perspective, homogeneous, sum-of-square
peri	perspective, homogeneous, max
per2	perspective, nonhomogeneous, sum-of-square
pers	perspective, method of Shade
pers2	pers, followed by gradient descent

In the case of the flat teapot (Figure 15), note that the error/update curves cluster into groups – translation, translation with separate scale, Shade, affine, and perspective, in order of merit. Shade's method is significantly outperformed by affine. Note also that the

⁶ The teapot, a roughly spherical object, was scaled along its axis of bilateral symmetry to 5% of its previous size.

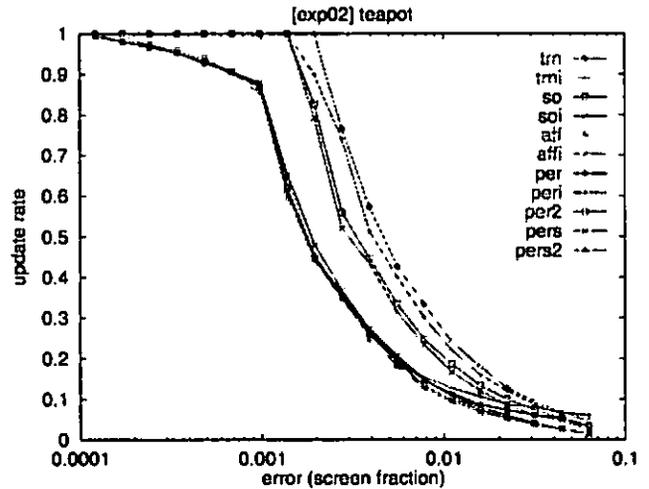


Figure 16: REGULAR TEAPOT update-rate/error relations show that affine and perspective are nearly indistinguishable.

sum-of-square error minimization is not much different than maximum error minimization for any of the warp types. The difference between perspective and affine is much less than one might expect in this case, given that perspective exactly matches motions of a perfectly flat object. Figure 16 (regular teapot) is similar, except that the clusters are translation, translation with separate scale, and all other warp types. In this case, perspective yields virtually no advantage over affine, and in fact is slightly worse towards the high-error/low update rate end of the curves for the homogeneous space metrics (per and peri).⁷ This is because the homogeneous metric weights the errors unevenly over the set of characteristic points. The method of Shade is slightly worse than affine in this case.

Since geometric error is a rather indirect measure of the perceptual quality of the warp types, the second series of experiments attempted to compare the perceptual quality of the set of warps given an update rate (i.e., an equal consumption of rendering resources). We used binary search to invert the relation between error threshold and update rate for each warp type, and then recorded the same animation, at the same update rate⁸, for various image warp approximations. Although subjective, the results confirm the merit of the affine transformation over less general transformations and the lack of improvement with the more general perspective transformation in typical scenarios.

4.3 Color Warp

Images can be "warped" to match photometry changes as well as geometry changes. For example, Talisman provides a per-sprite color multiplier that can be used to match photometry changes. To solve for this multiplier, we augment each characteristic point with a normal so that shading results can be computed (see Section 5.2). The color multiplier is selected using a simple least-squares technique that best matches the original color values of the shaded characteristic points to the new color values.

⁷ In the second series of experiments, the animations that used the homogeneous-weighted metric to determine an approximating perspective transformation looked visibly worse than those that used the simple affine transformation.

⁸ The update rate is the fraction of frames re-rendered; this balances the total consumption of rendering resources over the whole animation.

5 Fiducials

Fiducials measure the fidelity of the approximation techniques. Our fiducials are of four types. Geometric fiducials measure error in the screen-projected positions of the geometry. Photometric fiducials measure error in lighting and shading. Sampling fiducials measure the degree of distortion of the image samples. Visibility fiducials measure potential visibility artifacts.

We use conservative measurements where possible, but are willing to use heuristic measurements if efficient and effective. Any computation expended on warping or measuring approximation quality can always be redirected to improve 3D renderings, so the cost of computing warps and fiducials must be kept small relative to the cost of rendering.

5.1 Geometric Fiducials

Let \hat{P} be a set of characteristic points from an initial rendering, let P be the set of points at the current time, and let W be the warp computed to best match \hat{P} to P . The geometric fiducial is defined as

$$F_{geom} = \max_i \|P_i - W\hat{P}_i\|$$

5.2 Photometric Fiducials

We use two approaches to approximately measure photometric errors. The first uses characteristic points augmented with normals as described in Section 4.3 to point sample the lighting. Let \hat{C} be the colors that result from sampling the lighting at the characteristic points at the initial time, and C be the sampled colors at the current time. Let W_C be the color warp used to best match \hat{C} to C . Then the shading photometric fiducial is defined to be the maximum pointwise distance from the matched color to the current color.

$$F_{photo} = \max_i \|C_i - W_C \hat{C}_i\|$$

Another approach is to abandon color warping and simply measure the change in photometry from the initial time to the current. Many measures of photometric change can be devised. Ours measures the change in the apparent position of the light. Let \hat{L} be the position of the light at the initial time and L be its position at the current time (accounting for relative motion of the object and light). For light sources far away from the illuminated object, we can measure the angular change from \hat{L} to L with respect to the object, and the change in distance to a representative object "center". For diffuse shading, the angular change essentially measures how much the object's terminator moves around the object, and the change in distance measures the increase or decrease in brightness. Light sources close to the object are best handled with a simple Euclidean norm. For specular shading, changes in the eye point can also be measured.

⁹ Note that in architectures without color warping capability, W_C is the identity transform and we simply measure the maximum shading difference over all the characteristic points.

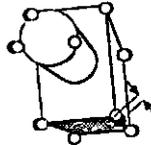


Figure 17: GEOMETRIC FIDUCIAL measures maximum pointwise distance between the warped original and current characteristic points.

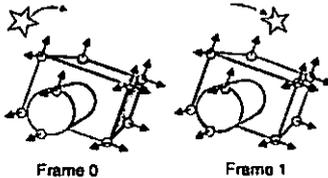


Figure 18: POINT-SAMPLED PHOTOMETRIC FIDUCIAL samples the shading at the initial and current characteristic points with normals.

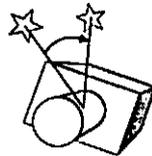


Figure 19: LIGHT SOURCE PHOTOMETRIC FIDUCIAL measures lighting change by the relative motion of light.

5.3 Sampling Fiducials

Sampling fiducials measure distortion of the samples in the image approximation. In Figure 20, both the geometric and photometric fiducials indicate high fidelity, but the image is blurry. The magnitudes of the singular values of the Jacobian of the image mapping function measure the greatest magnification and minification and the ratio measures the maximum anisotropy¹⁰. The affine warp has a spatially invariant Jacobian given by the left 2x2 part of the 2x3 matrix, for

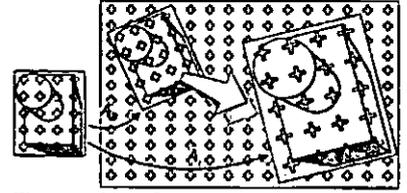


Figure 20: SAMPLING FIDUCIAL measures how the samples of a sprite are stretched or compressed. For which the two singular values are easily calculated [Blinn96]. For transforms with spatially varying Jacobians, such as the perspective warp, the singular values vary over the image. In this case, bounds on the singular values over the input domain can be computed.

5.4 Visibility Fiducials

Visibility fiducials measure potential visibility artifacts by tracking back-facing to front-facing transitions in the characteristic geometry (the simplified geometry makes these calculations tractable), and testing if the edges of clipped sprites become visible.

6 Regulation

A more complete treatment of regulation issues and directions may be found in [Horvitz96]. Our prototype regulator uses a simple cost-benefit scheduler and fiducial thresholds. The fiducial threshold provides a cutoff below which no attempt to re-render the layer is made (i.e., the image warp approximation is used). The regulator considers each frame separately, and performs the following steps:

1. Compute warp from previous rendering.
2. Use fiducials to estimate benefit of each warped layer.
3. Estimate rendering cost of each layer.
4. Sort layers according to benefit/cost.
5. Use fiducial thresholds to choose which layers to re-render.
6. Adjust parameters of chosen layers to fit within budget.
7. Render layers in order, stopping when all resources are used.

For a "budget-filling" regulator, the fiducial threshold is set to be small, on the order of a 1/1000 of the typical maximum error. All of the rendering resources are used in the attempt to make the scene as good as possible. For a "threshold" regulator, the threshold is raised to the maximum error that the user is willing to tolerate. This allows rendering resources to be used for other tasks.

Cost estimation [step 3] is based on a polygon budget, and measures the fraction of this budget consumed by the number of polygons in the layer's geometry. Parameter adjustments [step 6] are made to the sprite's spatial resolution using a budgeted total sprite size. This accounts for the rate at which the 3D rendering hardware can rasterize pixels.¹¹ Sprites that have been selected for re-rendering [step 5] are allocated part of this total budget in proportion to their desired area divided by the total desired area of the selected set. To dampen fluctuations in the regulation parameters which are perceptible when large, parameter changes are clamped to be no more than $\pm 10\%$ of their previous value at the time of last re-rendering. Note that factoring into many low-cost sprites allows smoother regulation.

¹⁰ The filtering capabilities of the hardware limit the amount of minification and anisotropy that can be handled before perceptible artifacts arise.

¹¹ A global average depth-complexity estimate is used to reduce the budget to account for rasterization of hidden geometry. Note that the depth complexity of factored geometry in a single layer is lower than would be the case for frame-buffer renderings of the entire scene.

7 Results

For the results presented here, we assume we have an engine that will composite all of the sprite layers with minimal impact on the rendering resources (as in the Talisman architecture.) We track the number of polygons and the amount of pixel fill used for rendering, but disregard compositing.

Both of the sequences described below are intended to represent typical content. For scenes with a moving camera, the typical speedup is 3-5 times what the standard graphics pipeline is capable of producing.

7.1 Canyon Flyby

This 250-frame sequence (Figure 25) used 10 sprite layers. We interpolated using affine warps and regulated the update rate with a geometric fiducial threshold of 4 pixels. Each sprite layer was re-rendered only when the geometric threshold was exceeded. The fiducial threshold of 4 pixels may seem large, but is acceptable for this sequence since the ships are well separated from the rest of the world.

The average update rate was 19%, and the cost-weighted average (based on polygon count) was 32%. About a third of the total polygons were rendered per frame.

In the canyon flyby scene, the entire background was placed in one sprite. Parallax effects from the rapidly moving camera make this a poor layering decision that yields a high update rate for the landscape sprite. In contrast, the sky is rendered just once

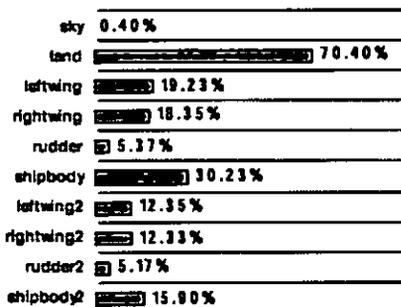


Figure 21: CANYON FLYBY AVERAGE UPDATE RATE for each sprite is the number of times each sprite was rendered divided by the number of frames in the sequence, 250

and then positioned with a new affine transformation each frame, and parts of the ships are updated relatively infrequently (5-30%).

7.2 Barnyard

The barnyard sequence (Figure 26) was chosen as an example in which a camera moves through a static scene¹². This is a difficult case, because the eye is sensitive to relative motion between static objects. Approximation errors in sequences in which objects already have relative motion are far less perceptible (e.g., the ships in the canyon flyby above.) Even with this difficult case, our interpolation technique is dramatically better than triple framing.

The scene is factored into 119 standard layers, 2 shadow map layers, and 2 shadow modulation layers. The contiguous landscape geometry was split into separate sprites. As an authoring pre-process, the geometry along the split boundaries was expanded to allow for small separation of the warped sprites (the "seam" artifact.) This is similar to the expansion of the geometry along the split used by [Shade96]. More work is needed for automatic determination of the geometric expansion and better blending along the split boundaries.

The resource-use graph in Figure 22 shows three types of regulation. Simple triple framing, in which a frame is rendered and then

¹² In the longer film from which the barnyard sequence is taken, many of the shots have fixed cameras. In these shots, only the main characters need to be updated, so the performance gain is extremely high. This is similar to the time-honored technique of saving the z-buffer for fixed camera shots.

held for three frames, requires the most resources. Interpolated triple-framing requires the same amount of rendering resources, but interpolates through time using the warp described in Section 4.1 – the sprites are still rendered in lock-step but track the underlying characteristic geometry between renderings. The rest of the curves show threshold regulation with increasing geometric error threshold, from 0.1-0.8 pixels – the sprites are updated heterogeneously when the geometric error threshold is exceeded. The graph is normalized to the resource use of triple-framing.

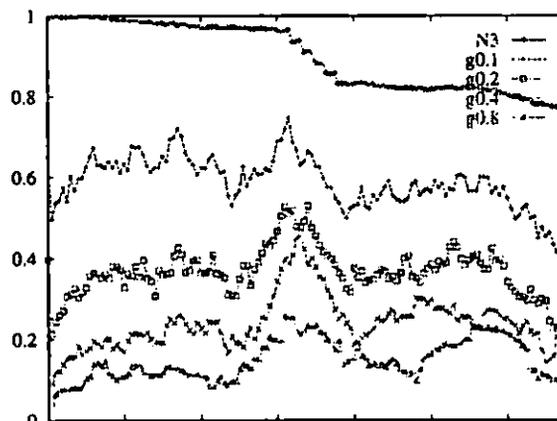


Figure 22: BARNYARD RESOURCE USE shows polygon counts as a 15-frame moving average. Pixel fill resource use is analogous. The top line is the triple-frame rendering. The lines below use threshold-regulation with increasing geometric threshold. As expected, as the pixel error tolerance goes up, the resource use goes down.

In the resulting animations, the most dramatic improvement in quality comes when the interpolation is turned on. The rest of the animations are nearly indistinguishable and use a fraction of the resources by rendering only those sprites whose geometric error exceeds the threshold.

Figure 23 shows the average pixel error for the same sequences shown in Figure 22. Each of the threshold-regulation sequences uses an error threshold smaller than the maximum error observed when triple-framing. Note that threshold-regulation is not the typical case and is shown here simply to demonstrate the performance advantage over the traditional graphics pipeline. Typically, all of the rendering resources are used to make the picture as good as possible.

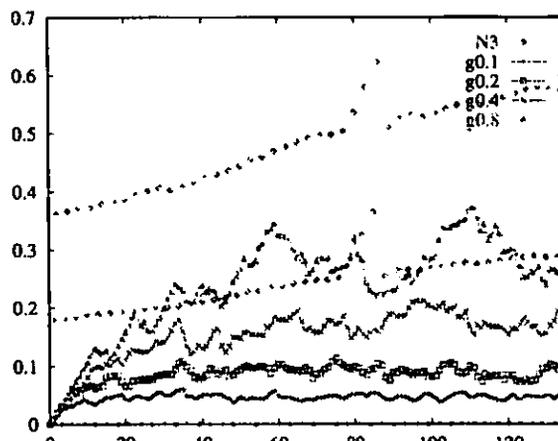


Figure 23: BARNYARD PIXEL ERROR shows average sprite pixel error per frame. Note that the triple-frame error is a saw-tooth that starts at 0, jumps to 1/2, and then jumps to full error value. The other curves oscillate below the given geometric threshold value.

8 Conclusions and Future Work

3D scenes should be factored into layers, with each layer having the proper sampling rates in both space and time to exploit the coherence of its underlying geometry and shading. By regulating rendering parameters using feedback from geometric, photometric, visibility, and sampling fiducials, rendering resources are applied where most beneficial. When not re-rendered, image warping suffices to approximate 3D motion of coherently factored layers. An affine warp provides a simple but effective interpolation primitive. These ideas yield 3-10 times improvement in rendering performance with the Talisman architecture with minor artifacts; greater performance can be controllably obtained with further sacrifices in fidelity.

Perceptual discontinuities may occur when a sprite's image is updated. Approximation with image warps captures the in-plane rotation, scale, and translation of an object, but not the out-of-plane rotation. The sprite image updates are sometimes perceived as a "clicking" or "popping" discontinuity. As the demand for higher quality 3D graphics increases display refresh rates, such artifacts will wane even at large factors of rendering amplification. More work is needed on the "seam" artifact (handling the boundaries of contiguous geometry placed in separate sprites.) Better modeling of the perceptual effects of regulation parameters is another area of future work [Horvitz97].

Factoring of shading terms is currently done using a fixed shading model that targets only the addition and over operations provided by hardware. Compilation of programmable shaders into layerable terms is an important extension. Many shading expressions, such as the shadow multiplication described in the appendix, can only be approximated with the over operator. We are interested in extending the system to target a fuller set of the image compositing operations.

Acknowledgements

The authors would like to thank the Microsoft Talisman Group and Microsoft Research, especially Jim Kajjya, Larry Ockenc, Mark Kenworthy, Mike Toelle, Kent Griffin, Conal Elliott, Brian Guenter, Hugues Hoppe, Eric Horvitz, David Jenner, Andrew Glassner, Bobby Bodenheimer, Joe Chauvin, Howard Good, Mikey Wetzel, Susan O'Donnell, Mike Anderson, Jim Blinn, Steve Gabriel, Dan Ling, and Jay Torborg. Thanks to Nathan Myhrvold for the research direction and core ideas, and to Russell Schick for initial work on error-based sprite re-rendering. Thanks also to Allison Hart Lengyel and Julia Yang-Snyder.

Appendix: Shadow Sprites

For a fast-moving shadow on a slow-moving receiver, we update only the fast-moving shadow and use the compositor to compute the shadow modulation. Since the compositor supports only 'over', we use the following approximation.

Let $B = [\beta B, \beta]$ be the receiver, where B is the color and β is the coverage. Let $A = [\alpha A, \alpha]$ be the desired shadow sprite, where A is the color and α is the coverage. The compositor computes

$$A \text{ over } B = [\alpha A + (1 - \alpha)\beta B, \alpha + (1 - \alpha)\beta].$$

Let s be the shadow modulation obtained by scan-converting the geometry of the background while looking up values in the shadow map of the fast moving object, where 0 means fully in shadow and 1 means fully illuminated.

The desired result is $C = [s\beta B, \beta]$. By letting $A = [0, (1-s)\beta]$, we get $C = A \text{ over } B$, or $C = [s\beta B + (1-s)(1-\beta)\beta B, \beta + (1-s)(1-\beta)\beta]$ which is close to the correct answer. Where there is no shadow, s is 1 and we get the correct answer of $[\beta B, \beta]$. Where coverage is complete, β is 1 and we get the correct answer of $[sB, 1]$. The problem lies in regions of shadow and partial coverage.

Ideally, the shadow modulation needs a color multiply operator '*' defined by $s * [\beta B, \beta] = [s\beta B, \beta]$. This is a per-pixel version of the Porter-Duff 'darken' operator. Note that this operator is not associative, and so requires the saving of partial results when used in a nested expression.

References

- [Blinn96] Consider the Lowly 2x2 Matrix, Jim Blinn, *IEEE Computer Graphics and Applications*, March 1996, pp. 82-88.
- [Chen93] View Interpolation for Image Synthesis, Shenchang Eric Chen and Lance Williams, *SIGGRAPH 93*, pp. 279-288.
- [Chen95] QuickTime VR - An Image-Based Approach to Virtual Environment Navigation, Shenchang Eric Chen, *SIGGRAPH 95*, pp. 29-38.
- [Cook84] Shade Trees, Robert L. Cook, *SIGGRAPH 94*, pp. 223-232.
- [Dorsey95] Interactive Design of Complex Time Dependent Lighting, Julie Dorsey, Jim Arvo, Donald P. Greenberg, *IEEE Computer Graphics and Applications*, March 1995, Volume 15, Number 2, pp. 26-36.
- [Funkhouser93] Adaptive Display Algorithm for Interactive Frame Rates During Visualization of Complex Virtual Environments, Thomas A. Funkhouser and Carlo H. Séquin, *SIGGRAPH 93*, pp. 247-254.
- [Gill81] Practical Optimization, Philip E. Gill, Walter Murray, and Margaret H. Wright, Academic Press, London, 1981.
- [Greene93] Hierarchical Z-Buffer Visibility, Ned Greene, Michael Kass, Gavin Miller, *SIGGRAPH 93*, pp. 231-238.
- [Greene94] Error-Bounded Antialiased Rendering of Complex Environments, Ned Greene and Michael Kass, *SIGGRAPH 94*, pp. 59-66.
- [Guenter95] Specializing Shaders, Brian Guenter, Todd B. Knoblock, and Erik Ruf, *SIGGRAPH 95*, pp. 343-350.
- [Hamrahan90] A Language for Shading and Lighting Calculations, Pat Hamrahan and Jim Lawson, *SIGGRAPH 90*, pp. 289-298.
- [Hofmann89] The Calculus of the Non-Exact Perspective Projection, Scene-Shifting for Computer Animation, Georg Rainer Hofmann, *Tutorial Notes for Computer Animation, Eurographics '89*.
- [Horvitz96] Flexible Rendering of 3D Graphics Under Varying Resources: Issues and Directions, Eric Horvitz and Jed Lengyel, In *Symposium on Flexible Computation*, AAAI Notes FS-96-06, pp. 81-88. Also available as Microsoft Technical Report MSR-TR-96-18.
- [Horvitz97] Decision-Theoretic Regulation of Graphics Rendering, Eric Horvitz and Jed Lengyel, In *Thirteenth Conference on Uncertainty in Artificial Intelligence*, D. Geiger and P. Shenoy, eds., August 1997.
- [Hubschman81] Frame to Frame Coherence and the Hidden Surface Computation: Constraints for a Convex World, Harold Hubschman, and Steven W. Zucker, *SIGGRAPH 81*, pp. 45-54.
- [Kay86] Ray Tracing Complex Scenes, Timothy L. Kay and James T. Kajiya, *SIGGRAPH 86*, pp. 269-278.
- [Maciel95] Visual Navigation of Large Environments Using Textured Clusters, Paulo W. C. Maciel and Peter Shirley, *Proceedings 1995 Symposium on Interactive 3D Graphics*, April 1995, pp. 95-102.
- [Mark97] Post-Rendering 3D Warping, William R. Mark, Leonard McMillan, and Gary Bishop, *Proceedings 1997 Symposium on Interactive 3D Graphics*, April 1997, pp. 7-16.
- [Meier96] Painterly Rendering for Animation, Barbara J. Meier, *SIGGRAPH 96*, pp. 477-484.
- [Molnar92] PixelFlow: High-Speed Rendering Using Image Composition, Steve Molnar, John Eyles, and John Poulton, *SIGGRAPH 92*, pp. 231-240.
- [McMillan95] Plenoptic Modeling: An Image-Based Rendering System, Leonard McMillan, Gary Bishop, *SIGGRAPH 95*, pp. 39-46.
- [Porter84] Compositing Digital Images, Thomas Porter and Tom Duff, *SIGGRAPH 84*, pp. 253-259.
- [Regan94] Priority Rendering with a Virtual Reality Address Recalculation Pipeline, Matthew Regan and Ronald Pose, *SIGGRAPH 94*, pp. 155-162.
- [Segal92] Fast Shadows and Lighting Effects Using Texture Mapping, Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, Paul Haeberli, *SIGGRAPH 92*, pp. 249-252.
- [Schaufler96a] Exploiting Frame to Frame Coherence in a Virtual Reality System, Gernot Schaufler, *Proceedings of VRAIS '96*, April 1996, pp. 95-102.
- [Schaufler96b] A Three Dimensional Image Cache for Virtual Reality, Gernot Schaufler and Wolfgang Stürzlinger, *Proceedings of Eurographics '96*, August 1996, pp. 227-235.
- [Shade96] Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments, Jonathan Shade, Dani Lischinski, David Salesin, Tony DeRose, John Snyder, *SIGGRAPH 96*, pp. 75-82.
- [Shelley82] Path Specification and Path Coherence, Kim L. Shelley and Donald P. Greenberg, *SIGGRAPH 82*, pp. 157-166.
- [Snyder97] Visibility Sorting for Dynamic, Aggregate Geometry, John Snyder, Microsoft Technical Report, MSR-TR-97-11.
- [Torborg96] Talisman: Commodity Real-time 3D Graphics for the PC, Jay Torborg, Jim Kajjya, *SIGGRAPH 96*, pp. 353-364.
- [Xie95] Feature Matching and Affine Transformation for 2D Cell Animation, Ming Xie, *Visual Computer*, Volume 11, 1995, pp. 419-428.

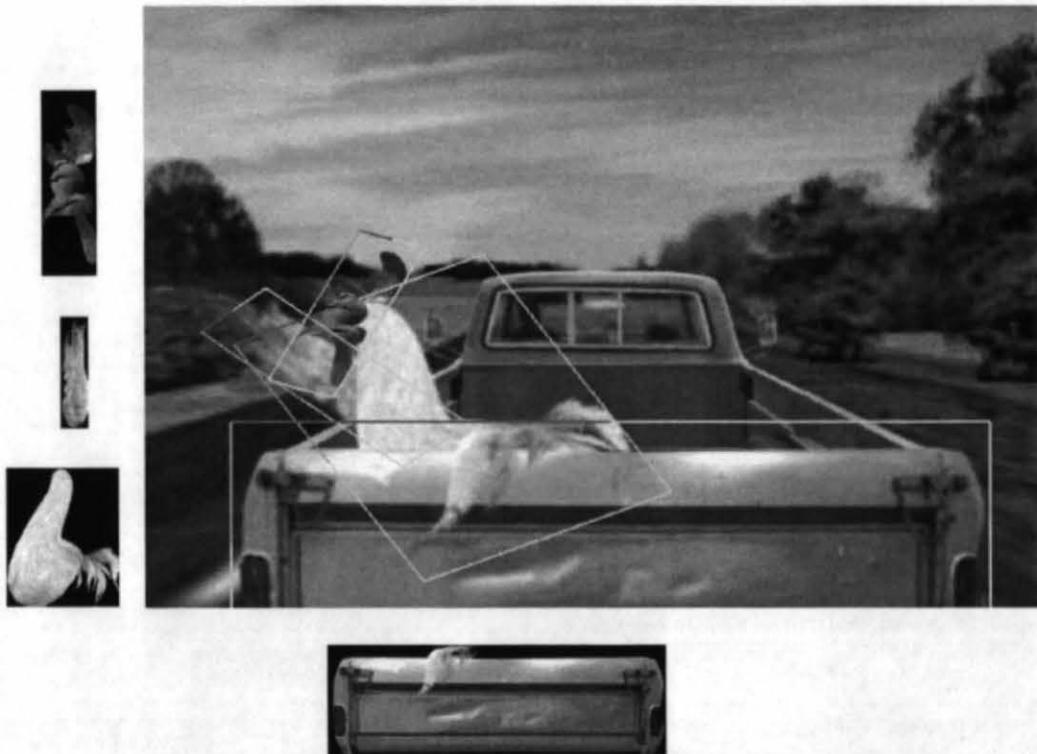


Figure 24: CHICKEN CROSSING sequence used 80 layers, some of which are shown separately (left and bottom) and displayed in the final frame with colored boundaries (middle). The sprite sizes reflect their actual rendered resolutions relative to the final frame. The rest of the sprites (not shown separately) were rendered at 40-50% of their display resolution. Since the chicken wing forms an occlusion cycle with the tailgate, the two were placed in a single sprite (bottom).



Figure 25: CANYON FLYBY used 10 layers with a geometric fiducial threshold of 4 pixels. The average sprite update rate was 19% with little loss of fidelity.



Figure 26: BARNYARD was factored into 119 geometry layers, 2 shadow map layers, and 2 shadow modulation layers. Threshold regulation for various geometric fiducial thresholds is compared in Figures 22 and 23.

Image-Based Rendering using Image-Warping – Motivation and Background

Leonard McMillan
LCS Computer Graphics Group
MIT

The field of three-dimensional computer graphics has long focused on the problem of synthesizing images from geometric models. These geometric models, in combination with surface descriptions characterizing the reflective properties of each element, represent the scene that is to be rendered. Computationally, the classical computer-graphics image synthesis process is a simulation problem in which light's interactions with the supplied scene description are computed.

Conventional computer vision considers the opposite problem of generating geometric models from images. In addition to images, computer-vision systems depend on accurate camera models and estimates of a camera's position and orientation in order to synthesize the desired geometric models. Often a simplified surface reflectance model is assumed as part of the computer vision algorithm.

The efforts of computer graphics and computer vision are generally perceived as complementary because the results of one field can frequently serve as an input to the other. Computer graphics often looks to the field of computer vision for the generation of complex geometric models, whereas computer vision relies on computer graphics for viewing results. Three-dimensional geometry has been the fundamental interface between the fields of computer vision and computer graphics since their inception. Only recently has this interface come under question. To a large extent the field of *image-based rendering* suggests an alternative interface between the image analysis of computer vision and the image synthesis of computer graphics. In this course I will describe one class of methods for synthesizing images, comparable to those produced by conventional three-dimensional computer graphics methods, directly from other images without an explicit three-dimensional geometric representation.

Another motivation for the development of image-based rendering techniques is that, while geometry-based rendering technology has made significant strides towards achieving photorealism, the process of creating accurate models is still nearly as difficult today as it was twenty-five years ago. Technological advances in three-dimensional scanning methods provide some promise for simplifying the process of model building. However, these automated model acquisition methods also verify our worst suspicions—the geometry of the real world is exceedingly complex. Ironically, one of the primary subjective measures of image quality used in geometry-based computer graphics is the degree to which a rendered image is indistinguishable from a photograph. Consider, though, the advantages of using photographs (images) as the underlying scene representation. Photographs, unlike geometric models, are both plentiful and easily acquired, and, needless to say photorealistic. Images are capable of representing both the geometric complexity and photometric realism of a scene in a way that is currently beyond our modeling capabilities.

Throughout the three-dimensional computer graphics community, researchers, users, and hardware developers alike, have realized the significant advantages of incorporating

images, in the form of texture maps, into traditional three-dimensional models. Texture maps are commonly used to add fine surface property variations as well as to substitute for small-scale geometric details. Texture mapping can rightfully be viewed as the precursor to image-based computer graphics methods. In fact, the image-based approach that I present can be viewed as an extension of texture-mapping algorithms commonly used today. However, unlike a purely image-based approach, an underlying three-dimensional model still plays a crucial role with traditional texture maps.

In order to define an image-based computer graphics method, we need a principled process for transforming a finite set of known images, which I will henceforth refer to as reference images, into new images as they would be seen from arbitrary viewpoints. I will call these synthesized images, desired images. Techniques for deriving new images based on a series of reference images or drawings are not new. A skilled architect, artist, draftsman, or illustrator can, with relative ease, generate accurate new renderings of an object based on surprisingly few reference images. These reference images are frequently illustrations made from certain cardinal views, but it is not uncommon for them to be actual photographs of the desired scene taken from a different viewpoint. One characterization of image-based rendering is to emulate the finely honed skills of these artisans by using computational powers.

While image-based computer graphics has come many centuries after the discovery of perspective illustration techniques by artists, its history is still nearly as long as that of geometry-based computer graphics. Progress in the field of image-based computer graphics can be traced through at least three different scientific disciplines. In photogrammetry the problems of distortion correction, image registration, and photometrics have progressed toward the synthesis of desired images through the composition of reference images. Likewise, in computer vision, problems such as navigation, discrimination, and image understanding have naturally led in the same direction. In computer graphics, as discussed previously, the progression toward image-based rendering systems was initially motivated by the desire to increase the visual realism of the approximate geometric descriptions. Most recently, methods have been introduced in which the images alone constitute the overall scene description. The remainder of this introduction discusses previous works in image-based computer graphics and their relationship to the image-warping approach that I will present.

In recent years, images have supplemented the image generation process in several different capacities. Images have been used to represent approximations of the geometric contents of a scene. Collections of images have been employed as databases from which views of a desired environment are queried. And, most recently, images have been employed as full-fledged scene models from which desired views are synthesized. In this section, I will give an overview of the previous work in image-based computer graphics partitioned along these three lines.

Images, mapped onto simplified geometry, are often used as an approximate representation of visual environments. Texture mapping is perhaps the most obvious example of this use. Another more subtle approximation involves the assumption that all, or most, of the geometric content of a scene is located so far away from the viewer that its actual shape is inconsequential. Much of the pioneering work in texture mapping is attributable to the classic work of Catmull, Blinn, and Newell. The flexibility of image textures as three-dimensional computer graphics primitives has since been extended to include small perturbations in surface orientation (bump maps) [Blinn76] and approximations to global illumination (environment and shadow mapping) [Blinn76] [Greene86] [Segal92]. Recent developments in texture mapping have concentrated on the use of visually rich textures mapped onto very approximate geometric descriptions [Shade96] [Aliaga96][Schaufler96].

Texture mapping techniques rely on mapping functions to specify the relationship of the texture's image-space coordinates to their corresponding position on a three-dimensional

model. A comprehensive discussion of these mapping techniques was undertaken in [Heckbert86]. In practice the specification of this mapping is both difficult and time consuming, and often requires considerable human intervention. As a result, the most commonly used mapping methods are restricted to very simple geometric descriptions, such as polygonal facets, spheres and cylinders.

During the rendering process, these texture-to-model mapping functions undergo another mapping associated with the perspective-projection process. This second mapping is from the three-dimensional space of the scene's representation to the coordinate space of the desired image. In actual rendering systems, one or both of these mapping processes occurs in the opposite or inverse order. For instance, when ray tracing, the mapping of the desired image's coordinates to the three-dimensional coordinates in the space of the visible object occurs first. Then, the mapping from the three-dimensional object's coordinates to the texture's image-space coordinates is found. Likewise, in z-buffering based methods, the mapping from the image-space coordinate to texture's image-space occurs during the rasterization process. These inverse methods are known to be subject to aliasing and reconstruction artifacts. Many techniques, including mip-maps [Williams83] and summed-area tables [Crow84] [Glassner86], have been suggested to address these problems.

Another fundamental limitation of texture maps is that they rely solely on the geometry of the underlying three-dimensional model to specify the object's shape. The precise representation of three-dimensional shape using primitives suitable for the traditional approach to computer graphics is, in itself, a difficult problem that has long been an active topic in computer graphics research. When the difficulties of representing three-dimensional shape are combined with the issues of associating a texture coordinate to each point on the surface (not to mention the difficulties of acquiring suitable textures in the first place), the problem becomes even more difficult. It is conceivable that, given a series of photographs, a three-dimensional computer model could be assembled. And, from those same photographs, various figures might be identified, cropped, and the perspective distortions removed so that a texture might be extracted. Then, using traditional three-dimensional computer graphics methods, renderings of any desired image could be computed. While the process outlined is credible, it is both tedious and prone to errors. The image-based approach to computer graphics described in this thesis attempts to sidestep many of these intermediate steps by defining mapping functions from the image-space of one or more reference images directly to the image-space of a desired image.

A new class of scene approximation results when an image is mapped onto the set of points at infinity. The mapping is accomplished in exactly the same way that texture maps are applied to spheres, since each point on a sphere can be directly associated with another point located an infinite distance from the sphere's center. This observation is also the basis of environment maps. However, environment maps are observed indirectly as either reflections within other objects or as representations of a scene's illumination environment. When such an image mapping is intended for direct viewing, a new type of scene representation results. An image-based computer graphics system of this type, called QuickTimeVR [Chen95], has been developed by Apple Computer. In QuickTimeVR, the underlying scene is represented by a set of cylindrical images. The system is able to synthesize new planar views in response to a user's input by warping one of these cylindrical images. This is accomplished at highly interactive rates (greater than 20 frames per second) and is done entirely in software. The system adapts both the resolution and reconstruction filter quality based on the rate of the interaction. QuickTimeVR must be credited with exposing to a wide audience the vast potential of image-based computer graphics. The QuickTimeVR system is a reprojection method. It is only capable of describing image variations due to changes in viewing orientation. Translations of the viewing position can only be approximated by selecting the cylindrical image whose center-of-projection is closest to the current viewing position.

The panoramic representation afforded by the cylindrical image description provides many practical advantages. It immerses the user within the visual environment, and it eliminates the need to consider the viewing angle when determining which reference image is closest to a desired view. However, several normal photographs are required to create a single cylindrical projection. These images must be properly registered and then reprojected to construct the cylindrical reference image. QuickTimeVR's image-based approach has significant similarity to the approach described here. Its rendering process is a special case of the cylinder-to-plane warping equation in the case where all image points are computed as if they were an infinite distance from the observer.

The movie-map system by Lippman [Lippman80] was one of the earliest attempts at constructing a purely image-based computer graphics system. In a movie-map, many thousands of reference images were stored on interactive video laser disks. These images could be accessed randomly, according to the current viewpoint of the user. The system could also accommodate simple panning, tilting, or zooming about these fixed viewing positions. The movie-map approach to image-based computer graphics can also be interpreted as a table-based approach, where the rendering process is replaced by a database query into a vast set of reference images. This database-like structure is common to other image-based computer graphics systems. Movie-maps were unable to reconstruct all possible desired views. Even with the vast storage capacities currently available on media such as laser disks, and the rapid development of even higher capacity storage media, the space of all possible desired images appears so large that any purely database-oriented approach will continue to be impractical in the near future. Also, the very subtle differences between images observed from nearby points under similar viewing conditions bring into question the overall efficiency of this approach. The image-based rendering approach described here could be viewed as a reasonable compression method for movie maps.

Levoy and Hanrahan [Levoy96] have developed another database approach to image-based computer graphics in which the underlying modeling primitives are rays rather than images. A key innovation of this technique, called light-field rendering, is the recognition that all of the rays that pass through a slab of empty space enclosed between two planes can be described using only four parameters rather than the five dimensions required for the typical specifications of a ray. They also describe an efficient technique for generating the ray parameters needed to construct any arbitrary view. The subset of rays originating from a single point on a light field's entrance plane can be considered as an image corresponding to what would have been seen at that point. The entire two-parameter family of images originating from points on the entrance plane can be considered as a set of reference images. During the rendering process, the three-dimensional entrance and exit planes are projected onto the desired viewing plane. The final image is constructed by determining the image-space coordinates of the two points visible at a specified pixel coordinate (one coordinate from the projected image of the entrance plane and the second from the image of the exiting plane). The desired ray can be looked up in the light field's database of rays using these four parameter values. Image generation using light fields is inherently a database query process, much like the movie map image-based process. The storage requirements for a lightfield's database of rays can be very large. Levoy and Hanrahan discuss a lossy method for compressing light fields that attempts to minimize some of the redundancy in the light-field representation.

The lumigraph [Gortler96] is another ray-database query algorithm closely related to the light-field. It also uses a four-dimensional parameterization of the rays passing through a pair of planes with fixed orientations. The primary differences in the two algorithms are the acquisition methods used and the final reconstruction process. The lumigraph, unlike the light field, considers the geometry of the underlying models when reconstructing desired views. This geometric information is derived from image segmentations based on the silhouettes of image features. The preparation of the ray database represented in a lumigraph requires considerable preprocessing when compared to the light field. This is a result of the arbitrary

camera poses that are used to construct the database of visible rays. In a lightfield, though, the reference images are acquired by scanning a camera along a plane using a motion platform. The lumigraph reconstruction process involves projecting each of the reference images as they would have appeared when mapped onto the exit plane. The exit plane is then viewed through an aperture on the entrance plane surrounding the center-of-projection of the reference image. When both the image of the aperture on the entrance plane and the reference image on the exit plane are projected as they would be seen from the desired view, the region of the reference image visible through the aperture can be drawn into the desired image. The process is repeated for each reference image. The lumigraph's approach to image-based three-dimensional graphics uses geometric information to control the blending of the image fragments visible through these apertures. Like the lightfield, the lumigraph is a data intensive rendering process.

The image-warping approach to IBR discussed here attempts to reconstruct desired views based on far less information. First, the reference image nearest the desired view is used to compute as much of the desired view as possible. Regions of the desired image that cannot be reconstructed based on the original reference image are subsequently filled in from other reference images. The warping approach to IBR can also be considered as a compression method for both light fields and lumigraphs. Considering the projective constraints induced by small variations in the viewing configuration reduces redundancy of the database representation. Thus, an image point, along with its associated mapping function, can be used to represent rays in many different images from which the same point is visible.

Many other computer graphics methods have been developed where images serve as the underlying representation. These methods handle the geometric relationships between image points very differently. In the case of image morphing, the appearance of a dynamic geometry is often a desired effect. Another method, known as *view interpolation* relies on an approximation to a true projective treatment in order to compute the mapping from reference images to desired images. Also, additional geometric information is required to determine correct visibility. A third method, proposed by Laveau and Faugeras, is based on an entirely projective approach to image synthesis. However, they have chosen to make a far more restrictive set of assumptions in their model, which allows for an ambiguous Euclidean interpretation.

Image morphing is a popular image-based computer graphics technique [Beier92], [Sietz96], [Wolberg90]. Generally, morphing describes a series of images representing a transition between two reference images. These reference images can be considered as endpoints along some path through time and/or space. An interpolation process is used to reconstruct intermediate images along the path's trajectory. Image morphing techniques have been used to approximate dynamic changes in camera pose [Sietz96], dynamic changes in scene geometry [Wolberg90], and combinations of these effects. In addition to reference images, the morphing process requires that some number of points in each reference be associated with corresponding points in the other. This association of points between images is called a *correspondence*. This extra information is usually hand crafted by an animator.

Most image-morphing techniques make the assumption that the transition between these corresponding points occurs at a constant rate along the entire path, thus amounting to a linear approximation. Also, a graduated blending function is often used to combine the reference images after they are mapped from their initial configuration to the desired point on the path. This blending function is usually some linear combination of the two images based on what percentage of the path's length has been traversed. The flexibility of image-morphing methods, combined with the fluidity and realism of the image transitions generated, have made a dramatic impact on the field of computer graphics, especially when considering how recently they have been developed. A subset of image morphing, called view morphing, is a special case of image-based computer graphics. In view morphing the scene geometry is fixed, and the pose of the desired views lies on a locus connecting the centers-of-projection of

the reference images. With the notable exception of the work done by Seitz, general image morphing makes no attempt to constrain the trajectory of this locus, the characteristics of the viewing configurations, or the shapes of the objects represented in the reference images. In this thesis, I will propose image-mapping functions that will allow desired images to be specified from any viewing point, including those off the locus. Furthermore, these mapping functions, like those of Seitz, are subject to constraints that are consistent with prescribed viewing conditions and the static Euclidean shape of the objects represented in the reference images.

Chen and Williams [Chen93] have presented a view interpolation method for three-dimensional computer graphics. It uses several reference images along with image correspondence information to reconstruct desired views. Dense correspondence between the pixels in reference images is established by a geometry-based rendering preprocess. During the reconstruction process, linear interpolation between corresponding points is used to map the reference images to the desired viewpoints, as in image morphing. In general, this interpolation scheme gives a reasonable approximation to an exact reprojection as long as the change in viewing position is slight. Indeed, as the authors point out, in some viewing configurations this interpolation is exact. Chen and Williams acknowledge, and provide a solution for, one of the key problems of image-based rendering—visible surface determination. Chen and Williams presort a quadtree-compressed flow-field in a back-to-front order according to the scene's depth values. This approach works only when all of the partial sample images share a common gaze direction and the synthesized viewpoints are restricted to stay within 90 degrees of this gaze angle. The underlying problem is that correspondence information alone (i.e., without depth values) still allows for many ambiguous visibility solutions unless we restrict ourselves to special flow fields that cannot fold (such as rubber-sheet local spline warps or thin-plate global spline warps).

Establishing the dense correspondence information required for a view interpolation system can also be problematic. Using pre-rendered synthetic images, Chen and Williams were able to determine the association of points by using the depth values stored in a z-buffer. In the absence of a geometric model, they suggest that approximate correspondence information can be established for all points using correlation methods. The image-based approach to three-dimensional computer graphics described in this research has a great deal in common with the view interpolation method. For instance, both methods require dense correspondence information in order to generate the desired image, and both methods define image-space to image-space mapping functions. In the case of view interpolation, the correspondence information is established on a pairwise basis between reference images. As a result the storage requirements for the correspondence data associating N reference images is $O(N)$. My approach is able to decouple the correspondence information from the difference in viewing geometries. This allows a single flow field to be associated with each image, requiring only $O(N)$ storage. Furthermore, the approach to visibility used in my method does not rely on any auxiliary geometric information, such as the presorted image regions based on the z -values, used in view interpolation.

Laveau's and Faugeras' [Laveau94] image-based computer-graphics system takes advantage of many recent results from computer vision. They consider how a particular projective geometric structure called epipolar geometry can be used to constrain the potential reprojections of a reference image. They explain how a fundamental matrix can describe the projective shape of a scene with scalar values defined at each image point. They also provide a two-dimensional ray-tracing-like solution to the visibility problem that does not require an underlying geometric description. Yet, it might require several images to assure an unambiguous visibility solution. Laveau and Faugeras also discuss combining information from several views, though primarily for the purpose of resolving visibility as mentioned before. By relating the reference views and the desired views by the homogenous transformations between their projections, Laveau and Faugeras can compute exact perspective depth solutions. However, the solutions generated using Laveau and Faugeras'

techniques do not reflect an unambiguous Euclidean environment. Their solution is consistent with an entire family of affine coordinate frames. They have pointed out elsewhere [Faugeras92b] that when additional constraints are applied, such as the addition of more reference images and the requirement that a fixed camera model be used for all reference images, then a unique Euclidean solution can be assured.

The methods described by Laveau and Faugeras are very similar to the image-based approach to computer graphics described here. The major difference in my approach is that I assume that more information is available than simply the epipolar geometries between reference images. Other significant differences are that the forward-mapping approach described here has fewer restrictions on the desired viewing position, and I provide a simpler solution to the visibility problem.

The delineation between computer graphics and computer vision has always been at the point of a geometric description. In IBR we tend to draw the lines somewhat differently. Rather than beginning the image synthesis task with a geometric description, we begin with images. In this overview I presented a summary of the various image-based rendering methods frequently used today.

Images have already begun to take on increasingly significant roles in the field of computer graphics. They have been successfully used to enhance the apparent visual complexity of relatively simple geometric screen descriptions. The discussion of previous work showed how the role of images has progressed from approximations, to databases, to actual scene descriptions.

In this course I will present an approach to three-dimensional computer graphics in which the underlying scene representation is composed of a set of reference images. I will present algorithms that describe the mapping of image-space points in these reference images to their image-space coordinates in any desired image. All information concerning the three-dimensional shape of the objects seen in the reference images will be represented implicitly using a scalar value that is defined for each point on the reference image plane. This value can be established using point correspondences between reference images. I will also present an approach for computing this mapping from image-space to image-space with correct visibility, independent of the scene's geometry.

Image-Based Rendering using Image Warping

Leonard McMillan
LCS Computer Graphics Group
Massachusetts Institute of Technology

In this report, I present a complete image-based rendering system. This includes the derivation of a mapping function from first principles, an algorithm for determining the visibility of these mapped points in the resulting image, and a method for reconstructing a continuous image from these mapped points. I refer to this type of mapping function as *image warping*, because it processes the elements of an image according to their image coordinates and produces outputs that are image coordinates in the resulting image.

In addition to the coordinates of the reference image additional information is required for each pixel. This information is related to the distance of the object seen at a particular pixel from the image plane. There are many different measures that can be used to describe this distance. Distance can be specified as *range values* describing the Euclidean distance from the visible object to image's center-of-projection. If the viewing or image plane is known and the coordinate system is chosen so that the normal of this plane lies a unit distance along the *z*-axis, then this distance information is called *depth* or the pixel's *z-value*. However, there are many other reasonable choices for representing this same distance. For instance distance values can be described indirectly by the relative motion of image points induced by a change in the camera's position, this distance representation is frequently called *optical flow*, and it is inversely related to the point's range. *Disparity* and *projective-depth* are two more representations of distance for which a warping equation can be developed. The choice of a distance metric often depends on knowing some additional information about how the image was formed (ex. knowledge of the image plane), but in some applications knowledge of this relationship will be unnecessary to perform the desired image warp. Rather than selecting a particular distance measure, and deriving the warping function based on it, I will instead develop a notion of depth that leads to the simplest expression for the desired warping function.

The warping function developed here will not be a one-to-one mapping. In those places where multiple points map to the same result a method to resolve which of the candidate points is visible is required. I will describe a simple method for determining these visible regions. This method will not rely on any geometric information from the scene, but only on the change in pose between the reference and viewing positions.

Finally, since an image will usually be represented as two-dimensional array of discrete samples, reconstruction methods are developed so that the transformed discrete points of the reference image can be used to estimate the appearance of the desired continuous image. I will suggest two methods for this reconstruction.

1. From Images to Rays

A perspective image describes a collection of rays from a given viewing position. Relative to this position any particular ray is uniquely determined by two angles. While the use of two angles sufficiently

describes the full range and dimension of the set of rays, it is not a very convenient representation for analysis or computation. Here I will consider an alternative parameterization of rays based on a general *planar-pinhole camera model*. This is the same planar-pinhole camera model that is commonly used in traditional three-dimensional computer graphics and computer vision.

The planar-pinhole camera is an idealized device for describing the rays that pass through a single point in space, called the *center-of-projection*, and are contained within some solid angle defined by a bounded planar section, called the *image plane*. This solid angle is well defined as long as the center-of-projection does not lie on the extended image plane. As the bounds of the image plane are extended indefinitely, the solid angle approaches 2π steradians, exactly half of the visual sphere.

Consider the rays emanating from the origin of a three-dimensional system with basis vectors $(\hat{i}, \hat{j}, \hat{k})$. Suppose also, that a second two-dimensional coordinate system is defined in the image plane, allowing each point on it to be identified by an image coordinate, (u, v) , where u and v are scalar multiples of the two basis vectors, (\hat{s}, \hat{t}) , defined in the image plane. Points on this image-plane can be given coordinates that are specified relative to this coordinate system. These points, along with their assigned coordinates, are referred to as *image-space points*. Without loss of generality, we can assume that the origin of image space lies at one of the corners of the bounded image plane. The following figure depicts these coordinate systems:

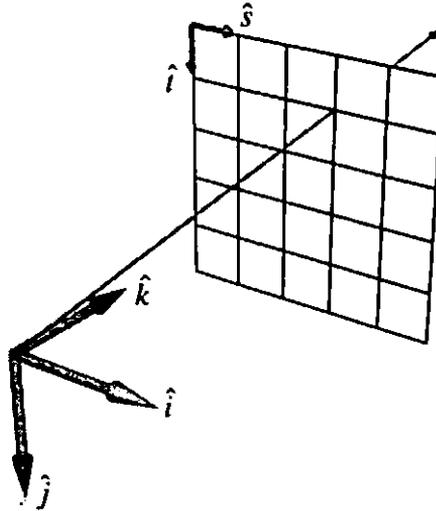


Figure 1: Mapping image-space point to rays

Each image-space point can be placed into one-to-one correspondence with a ray that originates from the Euclidean-space origin. This mapping function from image-space coordinates to rays can be described with a linear system:

$$\vec{d} = \begin{bmatrix} d_i \\ d_j \\ d_k \end{bmatrix} = \begin{bmatrix} a_i & b_i & c_i \\ a_j & b_j & c_j \\ a_k & b_k & c_k \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

Equation 1: Mapping from image coordinates to coordinates in three-space

where the coordinate of the image-space point is specified by the coordinate (u, v) , and the resulting vector \vec{d} represents the corresponding ray's direction. The entries of the mapping matrix, \mathbf{P} , can be easily understood by considering each column as a vector, \vec{a} , \vec{b} , and \vec{c} in the same coordinate system as \vec{d} . This relationship is shown in Figure 2, where \vec{a} and \vec{b} are images of the \hat{s} and \hat{t} basis vectors in the $(\hat{i}, \hat{j}, \hat{k})$ coordinate system, and \vec{c} is a vector from the ray origin to the origin of the image plane. Thus, while this parameterization is general, it still has a reasonable physical interpretation.

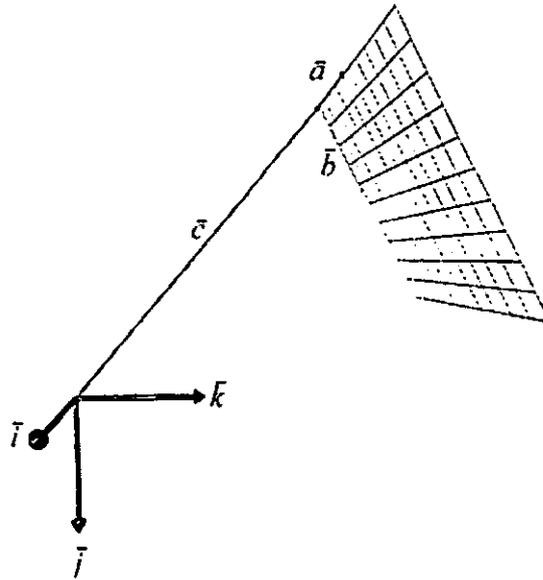


Figure 2: Relationship of the image-space basis vectors to the ray origin

The mapping function from image coordinates to rays is not uniquely defined. Any scalar multiple of the \mathbf{P} matrix will also yield an equivalent set of image-space point-to-ray correspondences. This independence of scale is a consequence of the fact that the space of possible directions from any point in a three-dimensional space can be described using only two parameters (i.e., two angles). Thus, any representation of rays that uses unconstrained three-dimensional vectors will allow for multiple representations of the same ray.

2. A Warping Equation for Synthesizing Projections of a Scene

Equipped with only the simple planar-pinhole-camera model described in the previous section, an image-warping equation, which remaps those rays visible from a given viewpoint to any arbitrary viewing position, can be derived. As before, I will refer to the source image, or domain, of the warp as the *reference image*, and the resulting image, after the mapping is applied, as the *desired image*. For the moment, I will assume that both the pinhole-camera parameters of the desired and reference views are known. In addition, I will also assume that a single scalar value, called *generalized disparity* or *projective depth*, is known for

all points of the reference image. The precise nature of this quantity will be discussed in more detail later in this section. For the moment it is sufficient to say that this quantity can be determined from a set of correspondences specified between images.

Consider the implications of the same point, \dot{X} , being sighted along rays from two different centers-of-projection, \dot{C}_1 and \dot{C}_2 , specified relative to their pinhole camera models. The following diagram illustrates this configuration.

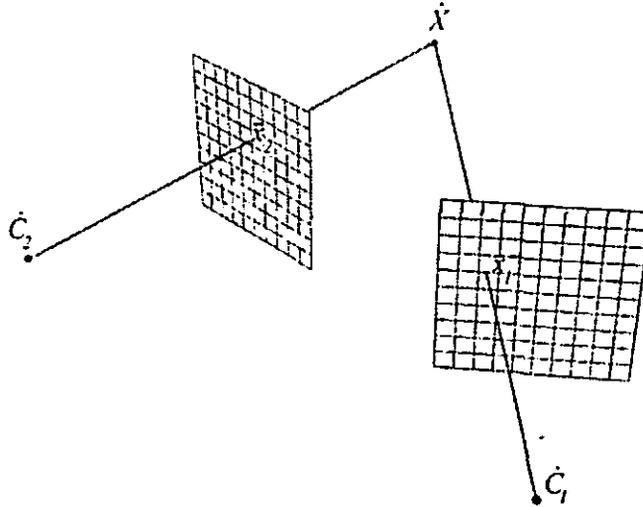


Figure 3: A point in three-dimensional space as seen from two pinhole cameras

The image coordinate, \bar{x}_1 , in the first image determines a ray via the pinhole camera mapping $\bar{d}_1 = P_1 \bar{x}_1$ with an origin of \dot{C}_1 . Likewise, the image coordinate, \bar{x}_2 , in the second image determines a ray, $\bar{d}_2 = P_2 \bar{x}_2$, with origin \dot{C}_2 . The coordinate of the point \dot{X} can, therefore, be expressed using either of the following:

$$\dot{X} = \dot{C}_1 + t_1 P_1 \bar{x}_1 = \dot{C}_2 + t_2 P_2 \bar{x}_2$$

Equation 2: Specification of a 3D point in terms of pinhole-camera parameters

where t_1 and t_2 are the unknown scaling factors for the vector from the origin to the viewing plane which make it coincident with the point \dot{X} . This expression can be reorganized to give

$$\frac{t_1}{t_2} P_2 \bar{x}_2 = \frac{1}{t_2} (\dot{C}_1 - \dot{C}_2) + P_1 \bar{x}_1$$

Equation 3: Transformation of a ray in one camera to its corresponding ray in another

The left-hand side of this expression is now a ray, as is the second term on the right hand side. If we relax our definition of equivalence to mean "equal down to some non-zero scale factor" (which is consistent with the notion that rays having the same direction are equivalent regardless of the length of the three-space vector specifying this direction), then the $\frac{1}{t_2}$ factor can be eliminated. I will use the symbol,

\doteq , to represent this equivalence relationship. Alternatively, we could take advantage of the property that both \mathbf{P}_1 and \mathbf{P}_2 are defined independent of scale, to absorb the scalar quantity, $\frac{1}{f_1}$, into the matrix \mathbf{P}_2 . Substituting the *generalized disparity* term $\delta(\bar{x})$ for $\frac{1}{f_1}$ gives

$$\mathbf{P}_2 \bar{x}_2 \doteq \delta(\bar{x}_1) (\hat{C}_1 - \hat{C}_2) + \mathbf{P}_1 \bar{x}_1$$

Equation 4: Simplified planar ray-to-ray mapping

The name, *generalized disparity*, comes from the notion of stereo disparity. In the normal depth-from-stereo case, the cameras are assumed to have a particular geometric configuration. Both image planes are required to have the same pinhole-camera model. The vector connecting the centers-of-projection must be parallel to both image planes. And, the coordinate system is selected so that the \hat{i} basis vector of the camera space is parallel to the \hat{s} basis vector of the image planes, as shown below.

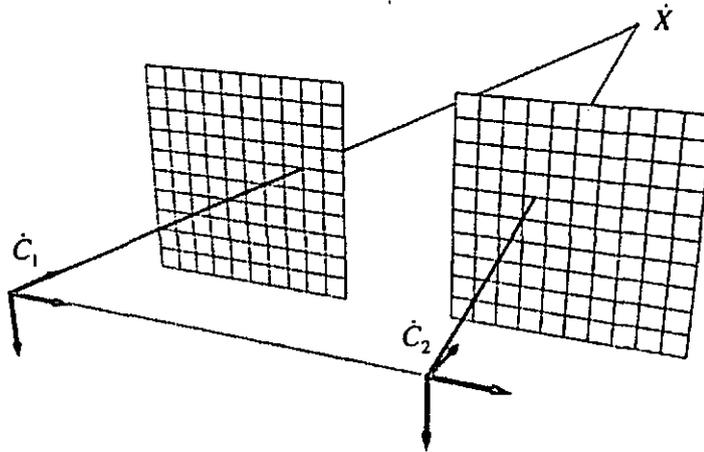


Figure 4: The depth-from-stereo camera configuration

This configuration can be accomplished either by accurate alignments of the cameras or by a post-processing rectification (using re-projection see Equation 13) of the acquired data. Under these conditions the planar ray-to-ray mapping equation simplifies to

$$\mathbf{P}_1 \bar{x}_2 \doteq \delta(\bar{x}_1) \left(\begin{bmatrix} C_{1x} \\ C_{1y} \\ C_{1z} \end{bmatrix} - \begin{bmatrix} C_{2x} \\ C_{2y} \\ C_{2z} \end{bmatrix} \right) + \mathbf{P}_1 \bar{x}_1 = \delta(\bar{x}_1) \begin{bmatrix} C_{1x} - C_{2x} \\ 0 \\ 0 \end{bmatrix} + \mathbf{P}_1 \bar{x}_1$$

Equation 5:

Multiplying both sides by the inverse of the pixel-to-ray mapping gives

$$\bar{x}_2 \doteq \delta(\bar{x}_1) \mathbf{P}_1^{-1} \begin{bmatrix} C_{1x} - C_{2x} \\ 0 \\ 0 \end{bmatrix} + \bar{x}_1$$

Equation 6:

The alignment constraint on the camera-space and image-space basis vectors (that the \hat{i} basis vector of the camera space is parallel to the \hat{s} basis vector) implies that a unit step in image space produces a unit step in camera space. This is equivalent to

$$\mathbf{P}_1 = \begin{bmatrix} 1 & p_{12} & p_{13} \\ 0 & p_{22} & p_{23} \\ 0 & p_{23} & p_{33} \end{bmatrix} \text{ so } \mathbf{P}_1^{-1} = \begin{bmatrix} 1 & q_{12} & q_{13} \\ 0 & q_{22} & q_{23} \\ 0 & q_{23} & q_{33} \end{bmatrix}$$

Equation 7:

After multiplying through and reorganizing terms we get the following:

$$\bar{x}_2 - \bar{x}_1 = \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} - \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix} = \delta(\bar{x}_1) \begin{bmatrix} C_{1x} - C_{2x} \\ 0 \\ 0 \end{bmatrix}$$

Equation 8:

Thus, when camera geometries satisfy the depth-from-stereo requirement, the image-space coordinates of corresponding points differ only along a single dimension in the images. The size of this change is proportional to both the distance between the centers-of-projection, which is often called the stereo baseline, and the generalized disparity quantity, which has units of pixels per unit length. Therefore, generalized disparity, $\delta(\bar{x})$, is equivalent to stereo disparity, $u_2 - u_1$, when normalized by dividing through by the length of the baseline.

$$\delta_{\text{stereo}}(\bar{x}) = \frac{u_2 - u_1}{C_{1x} - C_{2x}}$$

Equation 9:

The term, *projective depth*, is sometimes used instead of generalized disparity. This name is somewhat misleading since it increases in value for points closer to the center-of-projection. Generalized disparity, $\delta(\bar{x}_1)$, is inversely related to depth by a constant scale factor, and to range by a scale factor that varies from point to point on the viewing plane.

Let us consider further the generalized-disparity term, $\delta(\bar{x}_1)$. We can construct the following diagram of Equation 4 in the plane containing the three points \dot{C}_1 , \dot{C}_2 , and a visible point \dot{X} .

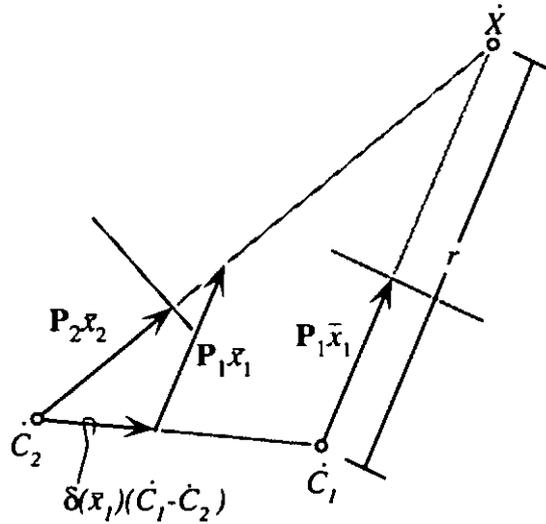


Figure 5: A point as seen from two images

Using similar triangles it can be shown that

$$\frac{r}{|C_1 - C_2|} = \frac{|P_1 \bar{x}_1|}{\delta(\bar{x}_1) |C_1 - C_2|}$$

Solving for generalized disparity gives the following expression:

$$\delta(\bar{x}_1) = \frac{|P_1 \bar{x}_1|}{r}$$

Thus, the generalized disparity depends only on the distance from the center-of-projection to the position on the image plane where the point is observed and the range value of the actual point. The image-space coordinate in the second image of the actual point can be found using the following transformation:

$$\bar{x}_2 = \delta(\bar{x}_1) P_2^{-1} (C_1 - C_2) + P_2^{-1} P_1 \bar{x}_1$$

Equation 10: Planar image-warping equation

Since the generalized-disparity term, $\delta(\bar{x}_1)$, is independent of both the desired center-of-projection, \dot{C}_2 , and the desired pinhole viewing parameters, \mathbf{P}_2 , Equation 10 can also be used to determine the image-space coordinate of the observed point on any other viewing plane. This is accomplished by simply substituting the desired center-of-projection and pinhole-camera model into Equation 10.

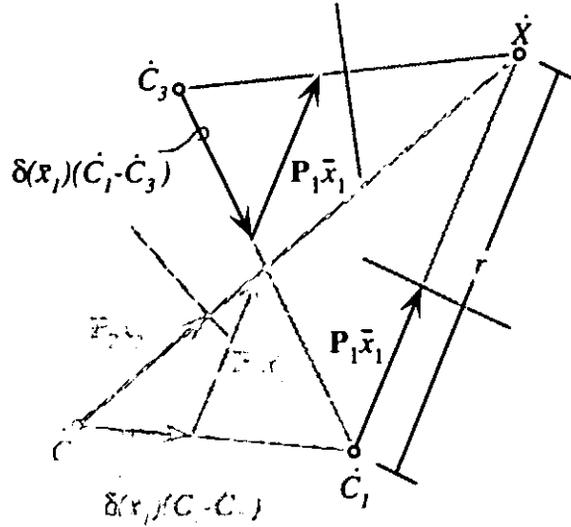


Figure 6: A third view of the point \dot{X}

Figure 6 illustrates how the planar warping equation (Equation 10) can be used to synthesize arbitrary views. The third viewing position, \dot{C}_3 , need not be in the same plane as \dot{C}_1 , \dot{C}_2 and \dot{X} . Figure 6 also depicts how generalized disparity is an invariant fraction of the baseline vector. This fraction of the baseline vector applies to all potential views, and therefore, it can be used to align corresponding rays of a given reference image to their direction in any desired image. In addition, the resulting projection will be consistent to a fixed three-dimensional point. Generalized disparity is a scalar quantity that determines how a translation of the center-of-projection affects the coordinates of points in image space. The remaining matrix quantity, $\mathbf{P}_2^{-1}\mathbf{P}_1$, determines how changes in the pinhole-camera model, independent of translation, affect the coordinates of points in image space. A further explanation of this last claim will be presented in the next section.

The warping equation can be used to synthesize arbitrary views of a given reference image via the following procedure. Given a reference image, the matrix describing its planar-pinhole projection model, \mathbf{P}_1 , its center-of-projection, \dot{C}_1 , a generalized-disparity value, $\delta(\bar{x}_1)$, for each pixel, and the center-of-projection of the desired image, \dot{C}_2 , and its projection model, the mapping of the reference image to a desired image can be computed. Using the vectors described in the generalized pinhole-camera model, the warping equation can be rewritten as

$$\alpha \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = [\bar{a}_2 \quad \bar{b}_2 \quad \bar{c}_2]^{-1} (\dot{C}_1 - \dot{C}_2) \delta(u_1, v_1) + [\bar{a}_2 \quad \bar{b}_2 \quad \bar{c}_2]^{-1} [\bar{a}_1 \quad \bar{b}_1 \quad \bar{c}_1] \begin{bmatrix} u_1 \\ v_1 \\ 1 \end{bmatrix}$$

where α is an arbitrary scale factor. This matrix sum can be rewritten as shown below:

$$\alpha \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \bar{a}_2 & \bar{b}_2 & \bar{c}_2 \end{bmatrix}^{-1} \begin{bmatrix} \bar{a}_1 & \bar{b}_1 & \bar{c}_1 & (\dot{C}_1 - \dot{C}_2) \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1 \\ \delta(u_1, v_1) \end{bmatrix}$$

Multiplying both sides by the determinant of \mathbf{P}_2 and substituting for its inverse gives the following 4×3 matrix equation:

$$\alpha (\bar{c}_2 \cdot (\bar{a}_2 \times \bar{b}_2)) \begin{bmatrix} u_2 \\ v_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \bar{a}_1 \cdot (\bar{b}_2 \times \bar{c}_2) & \bar{b}_1 \cdot (\bar{b}_2 \times \bar{c}_2) & \bar{c}_1 \cdot (\bar{b}_2 \times \bar{c}_2) & (\dot{C}_1 - \dot{C}_2) \cdot (\bar{b}_2 \times \bar{c}_2) \\ \bar{a}_1 \cdot (\bar{c}_2 \times \bar{a}_2) & \bar{b}_1 \cdot (\bar{c}_2 \times \bar{a}_2) & \bar{c}_1 \cdot (\bar{c}_2 \times \bar{a}_2) & (\dot{C}_1 - \dot{C}_2) \cdot (\bar{c}_2 \times \bar{a}_2) \\ \bar{a}_1 \cdot (\bar{a}_2 \times \bar{b}_2) & \bar{b}_1 \cdot (\bar{a}_2 \times \bar{b}_2) & \bar{c}_1 \cdot (\bar{a}_2 \times \bar{b}_2) & (\dot{C}_1 - \dot{C}_2) \cdot (\bar{a}_2 \times \bar{b}_2) \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \\ 1 \\ \delta(u_1, v_1) \end{bmatrix}$$

Equation 11: 4×3 matrix formulation of warping equation

This results in the following rational expressions for computing the re-projection of pixel coordinates from a reference image to the coordinates of a desired image:

$$u_2 = \frac{\bar{a}_1 \cdot (\bar{b}_2 \times \bar{c}_2) u_1 + \bar{b}_1 \cdot (\bar{b}_2 \times \bar{c}_2) v_1 + \bar{c}_1 \cdot (\bar{b}_2 \times \bar{c}_2) + (\dot{C}_1 - \dot{C}_2) \cdot (\bar{b}_2 \times \bar{c}_2) \delta(u_1, v_1)}{\bar{a}_1 \cdot (\bar{a}_2 \times \bar{b}_2) u_1 + \bar{b}_1 \cdot (\bar{a}_2 \times \bar{b}_2) v_1 + \bar{c}_1 \cdot (\bar{a}_2 \times \bar{b}_2) + (\dot{C}_1 - \dot{C}_2) \cdot (\bar{a}_2 \times \bar{b}_2) \delta(u_1, v_1)}$$

$$v_2 = \frac{\bar{a}_1 \cdot (\bar{c}_2 \times \bar{a}_2) u_1 + \bar{b}_1 \cdot (\bar{c}_2 \times \bar{a}_2) v_1 + \bar{c}_1 \cdot (\bar{c}_2 \times \bar{a}_2) + (\dot{C}_1 - \dot{C}_2) \cdot (\bar{c}_2 \times \bar{a}_2) \delta(u_1, v_1)}{\bar{a}_1 \cdot (\bar{a}_2 \times \bar{b}_2) u_1 + \bar{b}_1 \cdot (\bar{a}_2 \times \bar{b}_2) v_1 + \bar{c}_1 \cdot (\bar{a}_2 \times \bar{b}_2) + (\dot{C}_1 - \dot{C}_2) \cdot (\bar{a}_2 \times \bar{b}_2) \delta(u_1, v_1)}$$

Since the centers-of-projection and the planar-pinhole camera models for the reference and desired images are fixed for a given mapping, the warping equation simplifies to a pair of constant-coefficient linear rational expressions of the form

$$r(u_1, v_1, \delta(u_1, v_1)) = w_{11} u_1 + w_{12} v_1 + w_{13} + w_{14} \delta(u_1, v_1)$$

$$s(u_1, v_1, \delta(u_1, v_1)) = w_{21} u_1 + w_{22} v_1 + w_{23} + w_{24} \delta(u_1, v_1)$$

$$t(u_1, v_1, \delta(u_1, v_1)) = w_{31} u_1 + w_{32} v_1 + w_{33} + w_{34} \delta(u_1, v_1)$$

$$u_2 = \frac{r(u_1, v_1, \delta(u_1, v_1))}{t(u_1, v_1, \delta(u_1, v_1))}$$

$$v_2 = \frac{s(u_1, v_1, \delta(u_1, v_1))}{t(u_1, v_1, \delta(u_1, v_1))}$$

Equation 12: Warping equation as rational expressions

The mapping of a point in the reference image to its corresponding point in the desired image can be computed using nine adds, eleven multiplies, and one inverse calculation. When points of the reference image are processed sequentially, the number of adds is reduced to six, and the number of multiplies is reduced to five. Additional tests for a positive denominator, $t(u_i, v_i, \delta(u_i, v_i))$, and a valid range of the numerator can avoid two multiplies and the inverse calculation. This operation is the equivalent of screen-space clipping in traditional three-dimensional computer graphics.

3. Relation to Previous Results

Many other researchers [Szeliski96] [Faugeras92] have described similar warping equations. In most of these applications the image-space coordinates of the points \bar{x}_1 and \bar{x}_2 were given, and the projective depth, $\delta(\bar{x})$, was the quantity to be solved for. When this equation is used for image warping, coordinates of image points in the desired view, \bar{x}_2 , are computed from points in the reference image, \bar{x}_1 , and their projective depths.

This warping equation is also closely related to several other well-known results from computer graphics, image-warping, and projective geometry. Consider the situation where the reference image and the desired view share a common center-of-projection. In this case the planar-warping equation simplifies to

$$\bar{x}_2 \doteq P_2^{-1} P_1 \bar{x}_1$$

Equation 13: Image reprojection

This illustrates the well known result that images defined on planar viewing surfaces sharing a common center-of-projection are related by a *projective transformation* or *planar homography*.

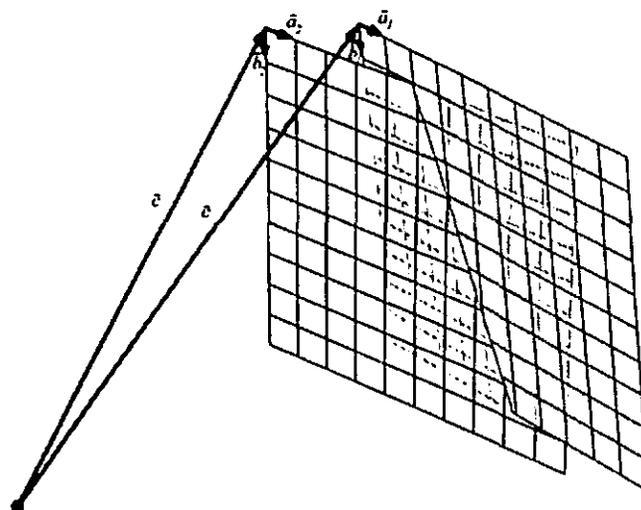


Figure 7: Reprojection of an image with the same center-of-projection

This projective transformation is merely the composition of the reference image's viewing matrix with the inverse of the desired image's viewing matrix, $\mathbf{H}_{reproject} = \mathbf{P}_2^{-1}\mathbf{P}_1$. This is indicative of the fact that, ignoring the clipping that occurs at the boundaries of the view plane, a change of viewing surface does not change the set of rays visible from the center-of-projection. It only changes their spatial distribution on the viewing surface. I will refer to mappings of this sort as reprojections.

A second well known result from the fields of computer graphics and projective geometry is that all images of a common planar surface seen in planar projection are also related by a projective transform as long as the plane does not project to a line. This result is the underlying basis for the texture mapping of images onto planar surfaces. It allows for the rasterization of textured planar primitives in screen space using a rational linear expression (an alternate formulation for a projective transform). The figure below illustrates the projection of a planar region onto several viewing planes and the resulting image.

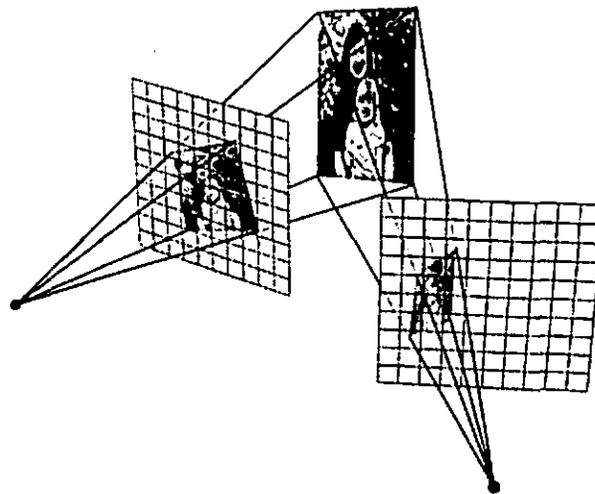


Figure 8: A planar region seen from multiple viewpoints

The mapping function that describes the possible views of a three-dimensional planar surface can be derived as a special case of Equation 10 by the following progression. The equation of a plane in the coordinate system having the reference image's center-of-projection as its origin is given by

$$\begin{bmatrix} A & B & C \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = D$$

Equation 14: Equation of a plane

where the scalars A , B , C , and D are four parameters defining the plane, and x , y , and z are the coordinates of a point in space. These three-space coordinates can be re-expressed in terms of image coordinates by using a planar-pinhole model image-to-ray mapping function as follows:

$$[A \quad B \quad C]t(u, v)\mathbf{P}_1 \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = D$$

where $t(u, v)$ is a multiple of the distance from the center-of-projection to the viewing plane for the ray. Dividing both sides by the scalar quantities appearing on opposite sides gives

$$\delta(u, v) = \frac{1}{t(u, v)} = \left[\frac{A}{D} \quad \frac{B}{D} \quad \frac{C}{D} \right] \mathbf{P}_1 \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

The inverse of $t(u, v)$ is equivalent to the generalized-disparity term discussed in Equation 3. When the generalized-disparity value from above is substituted into the warping equation (Equation 10), the following expression results:

$$\bar{x}_2 \doteq \mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2) \left[\frac{A}{D} \quad \frac{B}{D} \quad \frac{C}{D} \right] \mathbf{P}_1 \bar{x}_1 + \mathbf{P}_2^{-1} \mathbf{P}_1 \bar{x}_1 = \mathbf{P}_2^{-1} \left((\dot{C}_1 - \dot{C}_2) \left[\frac{A}{D} \quad \frac{B}{D} \quad \frac{C}{D} \right] + I \right) \mathbf{P}_1 \bar{x}_1$$

Equation 15: Mapping of a common plane seen at two centers-of-projection

The planar homography, $\mathbf{H}_{plane} = \mathbf{P}_2^{-1} \left((\dot{C}_1 - \dot{C}_2) \left[\frac{A}{D} \quad \frac{B}{D} \quad \frac{C}{D} \right] + I \right) \mathbf{P}_1$, is a projective mapping of the reference-image points on the plane to their coordinates in the desired image. When the reference and desired images share a common center-of-projection, the projective mapping reduces to the reprojection given in Equation 13 as expected.

The image plane of a reference image is but a plane in three-dimensional space. Therefore, its image in any reprojection is related by a projective mapping. The equation of the three-dimensional image plane of a given pinhole-camera model is given by

$$(\bar{a}_1 \times \bar{b}_1)^T \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \bar{c}_1 \cdot (\bar{a}_1 \times \bar{b}_1)$$

Substitution into Equation 15, gives

$$\bar{x}_2 \doteq \mathbf{P}_2^{-1} \left((\dot{C}_1 - \dot{C}_2) \frac{1}{\bar{c}_1 \cdot (\bar{a}_1 \times \bar{b}_1)} (\bar{a}_1 \times \bar{b}_1)^T + I \right) \mathbf{P}_1 \bar{x}_1 = \mathbf{P}_2^{-1} \left((\dot{C}_1 - \dot{C}_2) \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} + \mathbf{P}_1 \right) \bar{x}_1$$

which simplifies to

$$\bar{x}_2 \doteq \left(\begin{bmatrix} \bar{0} & \bar{0} & \mathbf{P}_2^{-1}(\dot{C}_1 - \dot{C}_2) \end{bmatrix} + \mathbf{P}_2^{-1} \mathbf{P}_1 \right) \bar{x}_1$$

Equation 16: Projection of the reference image plane in the desired image

Notice that the projective mapping, $H_{viewplane} = [0 \ 0 \ P_2^{-1}(C_1 - C_2)] + P_2^{-1}P_1$, describes the projection of the reference image's viewing plane in the desired image.

4. Resolving visibility

The mapping function described by the planar image warping equation (Equation 10) is not one-to-one. The locus of potential points in three-space, $\dot{X}(t)$, that project to the same image coordinate, $\bar{x} = (u, v)$, is described by the center-of-projection and a planar pinhole-camera model using the following equation:

$$\dot{X}(t) = \dot{C} + tP\bar{x}$$

Equation 17:

The parameter t identifies specific three-dimensional points that project to a given image-space coordinate. A policy of selecting the smallest of all positive t values for a given screen-space point can be used to determine visibility for opaque objects. This candidate t value is analogous to the z values stored in a z -buffer [Catmull74], or the ray-length maintained by a ray-casting algorithm [Appel68]. While the t parameter is an essential element of the reprojection process, (via its relationship to $\delta(\bar{x}_1)$) it is, surprisingly, not required to establish the visibility of a warped image.

In this section, an algorithm is presented for computing the visible surface at each image-space point of a desired image as it is being derived from a reference image via a warping equation. This is accomplished by establishing a warping order that guarantees a correct visibility solution. This ordering is established independently of the image contents. Only the centers-of-projection of the desired and reference images, as well as the pinhole-camera model for the reference image are needed.

In order to simplify the following discussion, the reference image is assumed to be stored as a two-dimensional array whose entries represent uniformly spaced image samples. This simplification is not strictly necessary for the algorithm to operate correctly, but it allows for a concise statement of the algorithm, and it is representative of many typical applications.

The approach of this visibility algorithm is to specify an ordering, or enumeration, of points from the reference image which guarantees that any scene element that is hidden by some other scene element in the desired image will always be drawn prior to its eventual occluder. This type of ordering is commonly called back-to-front. It is well known that a simple painter's algorithm [Rogers85] can be used to display any collection of scene elements with correct visibility when a back-to-front ordering can be established.

Given the reference image's center-of-projection, \dot{C}_1 , and ray-mapping function, P_1 , and the desired center-of-projection, \dot{C}_2 , the projection of \dot{C}_2 onto the reference image is first computed as follows:

$$\begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix} = \mathbf{P}_i^{-1}(\dot{C}_2 - \dot{C}_1)$$

Equation 18:

An example of this projection is illustrated in Figure 9.

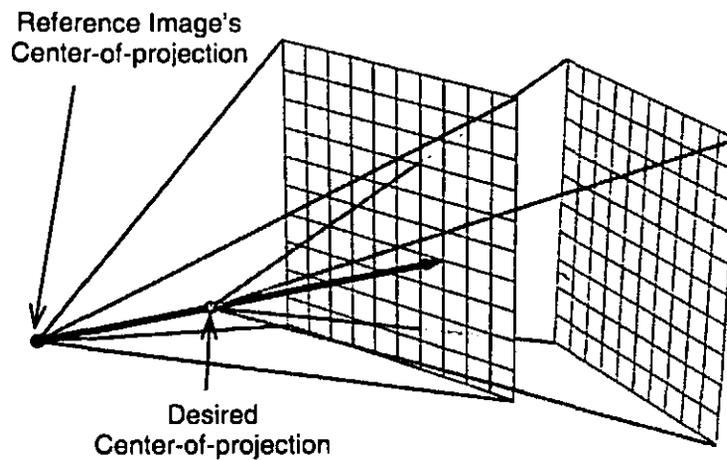


Figure 9: A desired center-of-projection projected onto the reference image

The image coordinate of \dot{C}_2 on the reference image is given by $\bar{e} = (e_x/e_t, e_y/e_t)$. It will fall into one of nine regions relative to the reference image shown below:

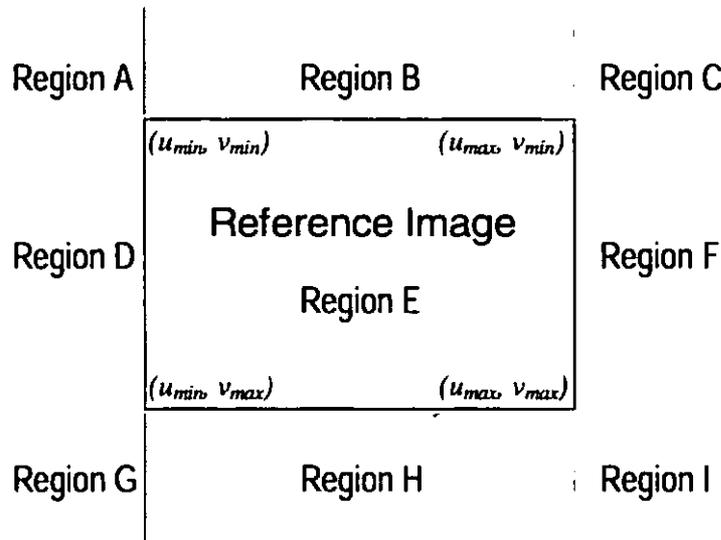


Figure 10: Figure of nine regions

Next, the reference image is subdivided into sheets that can be processed independently. Any set of orthogonal image-space basis vectors can be used to partition each sheet, but it is simplest to choose a coordinate basis aligned with the image's sampling grid. The reference image is partitioned into 1, 2, or 4 sheets depending on the image-space coordinates of the projected point, \bar{e} . When \bar{e} projects within the domain of the reference image, the image is divided into four sections separated along the row and column of \bar{e} .

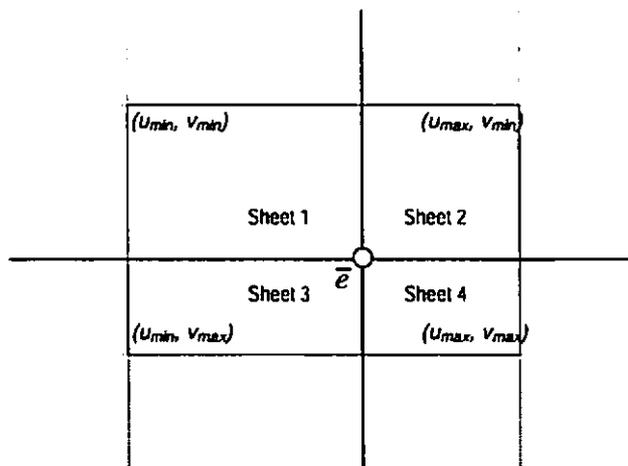


Figure 11: A desired center-of-projection that divides the reference image into 4 sheets

When only one coordinate of \bar{e} falls within the reference image, (i.e., when \bar{e} falls into regions B, D, F, and H) the image is subdivided into two sheets whose boundary is determined by either the row or column of the one coordinate that lies within the domain.

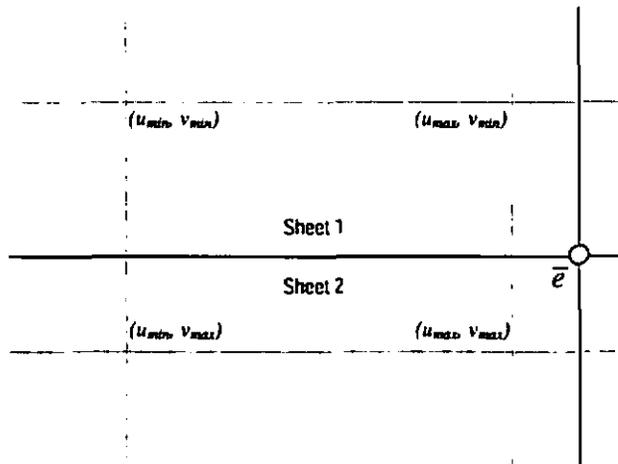


Figure 12: A desired center-of-projection that divides the reference image into 2 sheets

If neither component of \bar{e} falls within the reference image's domain, (when \bar{e} projects into regions A, C, H or J) then the entire image is treated as a single sheet.

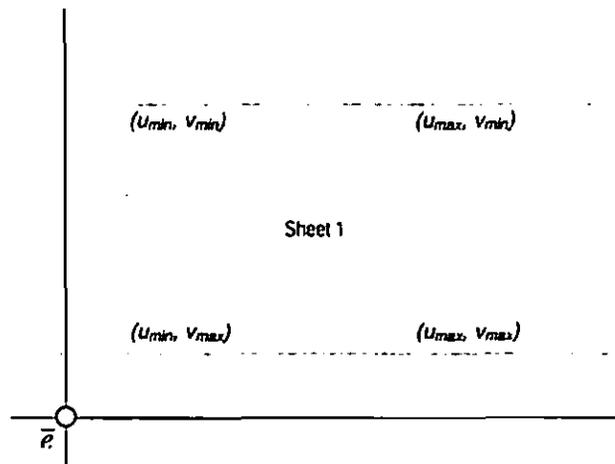


Figure 13: A desired center-of-projection that divides the reference image into 1 sheet

Once the reference image's domain is subdivided according to the projected position of the desired center-of-projection, the warping order for each sheet can be determined as follows. The sign of the e_z component from Equation 18 determines the enumeration direction. When e_z is non-negative the warping order of each sheet progresses toward the point \bar{e} , otherwise the warping progresses away from \bar{e} , as shown in the figure below. The case where e_z is zero indicates that the desired viewing position has no proper projection onto the reference image, because the vector $\dot{C}_1 - \dot{C}_2$ is parallel to the reference image's viewing plane. In this case, only one sheet will be enumerated, and the warping order progresses in the direction determined by the quadrant indicated by the signs of the remaining two vector components, e_x and e_y .

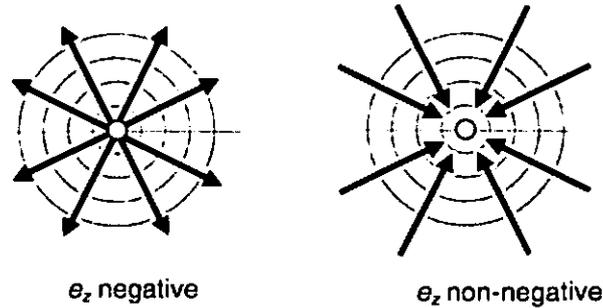


Figure 14: Enumeration direction

During the warp, each radial line originating from the projected image of the desired center-of-projection can be traversed independently. Alternatively, the warp can progress along either rows or columns of the sheets so long as the image of the desired center-of-projection, \bar{e} , is drawn at the appropriate time (i.e., p is drawn first when e_z is negative, and last otherwise), allowing either a row major or column major traversal. The advantage of the latter approach is that it allows the reference image's traversal to take maximum advantage of the access coherence of most memory systems.

The entire visibility algorithm involves three simple steps. First, the three-dimensional coordinate of the desired center-of-projection, \hat{C}_2 , is projected on the reference image's viewing plane. Second, the image-plane is divided into sheets determined by the image-space coordinate of the projected center-of-projection, \bar{e} , and whose boundaries are aligned with the image-space coordinate axes¹. Computing this coordinate involves a projective normalization. Finally, the sign of the planar normalizing element of the projective coordinate determines the traversal of the reference image².

The algorithm presented here is similar to Anderson's algorithm for bivariate functions [Anderson82]. The difference is that his visibility algorithm was defined for a different class of surfaces (i.e., a height field, or Monge patch, rather than a projective surface), and his algorithm enumerates the facets in a front-to-back occlusion order. Anderson's choice of a front-to-back order requires that some representation of the grid perimeter be maintained to aid in deciding what parts or edges of subsequent facets need to be rendered. Representing this perimeter requires auxiliary storage and extra clipping computations. This list must then be queried before each new facet's edge is displayed, and the display must be updated if any part of the facet is visible. In contrast, a back-to-front ordering requires no additional storage because the proper occlusion is handled by the drawing order using the painter's algorithm.

¹ Along the rows and columns of a discretely sampled image array

² This is often called the homogeneous element of the projective coordinate.

5. Reconstruction Issues

The planar-warping equation describes a mapping of the image-space points on a reference viewing plane to image-space points on a desired viewing plane. The underlying assumption is that both the reference and desired images are continuous over their domains. This is not generally the case for typical images. Usually, images are represented by a two-dimensional array of discrete samples. There are many subtle implications of warping sampled images rather than continuous ones. While the warping equation can easily be applied to the discrete coordinates of a sampled image, the likelihood that any sample will map exactly onto a sampling-grid point of the desired image is negligible. In addition to warping to locations off the sampling grid, the points of a reference image will also distribute unevenly over the desired image. The desired image must then be synthesized from this irregular distribution of sampled and reprojected image points.

The process of mapping a sampled image from one image-space to another is called *image resampling*. Conceptually, image resampling constructs a continuous representation of a reference image which is then mapped onto the desired viewing plane using the warping function and resampled to form the final discrete image. When the three-dimensional points represented in the reference image lie on a common plane, as shown in Figure 8, and the scene contents are appropriately band limited, reconstructing a continuous representation is a straightforward application of signal processing theory [Wolberg90]. The same situation occurs when three-dimensional points are constrained to lie along the same rays of different viewing planes, as when reprojecting an arbitrary set of points from the same center-of-projection as shown in Figure 7. The ideal continuous reconstruction of these surfaces is the summation of two-dimensional sinc functions centered at each sample-grid point and scaled by the sample's intensity. Since the sinc function has an infinite extent, local polynomial approximations are often used instead [Mitchell88]. However, when the three-dimensional points visible in an image are not constrained to lie in a plane or share a center-of-projection, the reconstruction of a continuous reference image representation is more involved.

Three-dimensional surfaces can be built up from discrete sub-surfaces, called *surface patches*. The composite of a group of surface patches can represent the reconstruction of a three-dimensional point set. One measure of the continuity of a composite set of surface patches, called *derivative continuity*, is the number of matching terms in the Taylor series expansions at abutting surface patches. When only the first terms of the composite surfaces' Taylor series expansions match along their common boundaries, the surface patches will have the same three-dimensional coordinate values along their abutting regions, and the overall surface has C^0 continuity. When both the first and second terms of the expansion agree, the tangent spaces of the abutting surfaces coincide, and the composite surface will have C^1 continuity.

One approach to reconstructing a continuous function from a sampled reference image is to consider each sample as specifying a surface patch. The basis for these surface patch definitions is typically polynomial. However, this is not a requirement for defining a continuous reconstruction. All that is

necessary is that the basis functions are sufficiently differentiable. For instance, a sinc or a Gaussian basis are both valid representations of surface patches³.

I will next describe two different methods for constructing a continuous representation of a reference image for use in image warping based on C^0 continuity models that use different surface patch bases. Recent work by Mark [Mark97] has extended these methods to include a model in which C^0 and C^1 continuity alternates throughout the domain. His approach requires a local estimate of the tangent space at each sample point. This requirement can be easily satisfied when the reference images are synthesized using traditional computer graphics techniques, but it is more difficult for acquired reference images. However, there is some promise in this area. It appears that many shape-from-shading [Horn89] and photometric-ratio-based correspondence methods [Wolff94] can be adapted to estimate a surface normal at each image-space point.

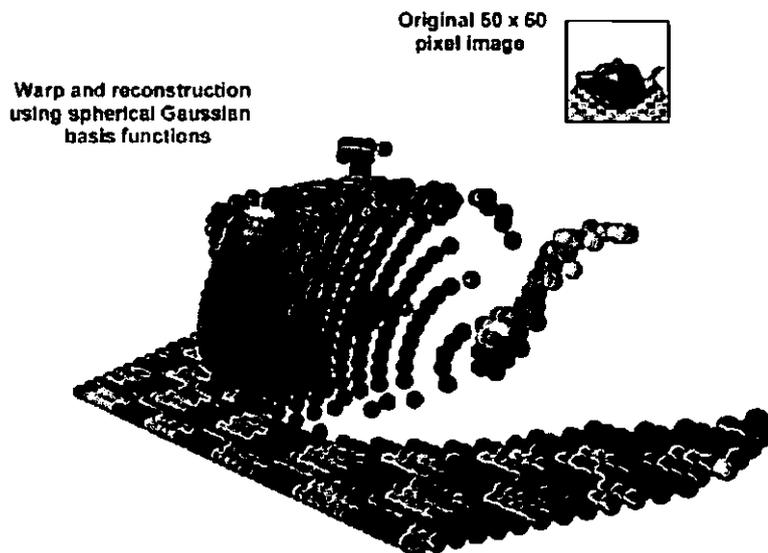


Figure 15: A Gaussian cloud representation of image-space points

The first reconstruction approach uses a spherical Gaussian basis function to represent each sample point. It assumes that every visible point in a reference image represents a three-dimensional spherical cloud density located somewhere along the ray determined by the image point. In three-dimensions this radius could be determined by considering the solid angle represented at each sample point and the distance of the cloud's center from the image's center-of-projection. A two-dimensional equivalent of this radius calculation can, however, be computed directly from the warping equation.

The image-space Gaussian reconstruction method described here is a straightforward adaptation of Heckbert's Elliptical Weighted Average (EWA) filter [Heckbert89], but it is used in a forward-mapping algorithm, similar in spirit to the Splatting algorithm described by Westover [Westover91]. The support of

³ The use of a sinc or Gaussian basis is somewhat complicated by their infinite extents.

the Gaussian reconstruction function is determined by computing the change in shape of differential circular regions surrounding each sample as they undergo the image warp. One useful measure of the change in shape is provided by the *Jacobian determinant* of the mapping function. The Jacobian determinant is a direct measure of the local change in area induced by a transformation. However, changes in differential area are not necessarily a good indication of the changes in differential shape⁴. The additional assumptions required to address this discrepancy will be discussed shortly. The Jacobian matrix of the planar-warping function given in Equation 12 is

$$\mathbf{J} = \begin{bmatrix} \frac{w_{11}(w_{22}v+w_{32}+w_{34}\delta(u,v))-w_{12}(w_{12}v+w_{13}+w_{14}\delta(u,v))}{(w_{31}u+w_{32}v+w_{33}+w_{34}\delta(u,v))^2} & \frac{w_{12}(w_{21}u+w_{22}+w_{24}\delta(u,v))-w_{11}(w_{11}u+w_{12}+w_{14}\delta(u,v))}{(w_{31}u+w_{32}v+w_{33}+w_{34}\delta(u,v))^2} \\ \frac{w_{21}(w_{22}v+w_{32}+w_{34}\delta(u,v))-w_{22}(w_{12}v+w_{13}+w_{14}\delta(u,v))}{(w_{31}u+w_{32}v+w_{33}+w_{34}\delta(u,v))^2} & \frac{w_{22}(w_{21}u+w_{22}+w_{24}\delta(u,v))-w_{21}(w_{11}u+w_{12}+w_{14}\delta(u,v))}{(w_{31}u+w_{32}v+w_{33}+w_{34}\delta(u,v))^2} \end{bmatrix}$$

Equation 19: Jacobian of the warping equation

The determinant of the Jacobian matrix simplifies to

$$\frac{\partial(u', v')}{\partial(u, v)} = \frac{\det(\mathbf{H}) + \delta(u, v) \det(\mathbf{G})}{t(u, v, \delta(u, v))^3}$$

Equation 20: Jacobian determinant of the warping equation

where

$$\mathbf{H} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \text{ and } \mathbf{G} = \begin{bmatrix} w_{11} & w_{12} & w_{14} \\ w_{21} & w_{22} & w_{24} \\ w_{31} & w_{32} & w_{34} \end{bmatrix}$$

Equation 21: Camera-model matrix, H, and the structure matrix, G

The \mathbf{H} component of the Jacobian determinant represents the change in projected area of an infinitesimal region of the reference image due entirely to the changes in the pinhole-camera model since $\mathbf{H} \doteq \mathbf{P}_2^{-1}\mathbf{P}_1$ and $\det(\text{Jacobian}(\mathbf{H}\bar{x})) = \det(\mathbf{H})/t(u, v, 0)^3$. The \mathbf{G} component represents the change in projected area due to the three-dimensional structure of the observed image point. This can be seen by letting $\mathbf{H} = \mathbf{I}$, which indicates no change in the pinhole camera model between the reference and desired image. This gives $\frac{\partial(u', v')}{\partial(u, v)} = \frac{t}{(1+\delta(u, v)w_{34})^2}$, indicating that the change in projected area is independent of the point's coordinate on the image plane, but instead it depends entirely on the generalized disparity value at that

⁴ For instance, consider the mapping $\phi: \{x = 10u, y = 0.1v\}$ with $\frac{\partial(x, y)}{\partial(u, v)} = \text{Determinant} \begin{bmatrix} 10 & 0 \\ 0 & 0.1 \end{bmatrix} = 1$. The

Jacobian determinant indicates that the differential area surrounding any sample remains constant; yet the mapping introduces considerable stretching along the u dimension and shrinking in the v dimension.

point. Since the matrices **H** and **G** are constant for a given pair of reference and desired views, they need only be calculated once per warp.

If we make the assumption that the warping function is well represented by a local linear approximation centered at each sample in the reference image with a constant generalized disparity value, then the Jacobian matrix can also be used to estimate the change in shape of a rotationally symmetric reconstruction kernel as follows. An infinitesimal circular region with a radius of dr is described by the expression

$$dr^2 = \begin{bmatrix} du & dv \end{bmatrix} \begin{bmatrix} du \\ dv \end{bmatrix}$$

Equation 22:

When that region undergoes an arbitrary mapping, $\phi: (u, v) \rightarrow (u', v')$, the best linear approximation to the differential mapping is given by the Jacobian matrix

$$\begin{bmatrix} du' \\ dv' \end{bmatrix} = \mathbf{J} \begin{bmatrix} du \\ dv \end{bmatrix}$$

Equation 23:

By substituting Equation 23 into Equation 22, the mapping of the differential circular region can be determined as follows:

$$dr^2 = \begin{bmatrix} du' & dv' \end{bmatrix} (\mathbf{J}^{-1})^T \mathbf{J}^{-1} \begin{bmatrix} du' \\ dv' \end{bmatrix}$$

Equation 24:

which expands to the following conic expression:

$$dr^2 = \frac{(j_{21}^2 + j_{22}^2)du'^2 - 2(j_{11}j_{21} + j_{12}j_{22})du'dv' + (j_{11}^2 + j_{12}^2)dv'^2}{(j_{11}j_{22} - j_{12}j_{21})^2}$$

Equation 25:

This equation describes an ellipse if the Jacobian determinant is positive. Therefore, any circular region centered about a reference image sample will project as an ellipse in the desired image. This property can be used to reconstruct a continuous representation of the reference image prior to resampling. The spatial domain response of any rotationally symmetric filter, such as a Gaussian, can be computed at a sample point in the desired image by evaluating Equation 25 at that point to determine the radius value in the undistorted filter's kernel. The resampling process can be optimized by first computing the extents of the ellipse in the desired image space. In terms of the Jacobian these extents are given by the following expressions:

$$\Delta u' = \pm \sqrt{\frac{dr^2(j_{11}j_{22} - j_{12}j_{21})^2}{(j_{21}^2 + j_{22}^2) - \frac{(j_{11}j_{12} + j_{21}j_{22})^2}{(j_{11}^2 + j_{12}^2)}}}$$

$$\Delta v' = \pm \sqrt{\frac{dr^2(j_{11}j_{22} - j_{12}j_{21})^2}{(j_{11}^2 + j_{12}^2) - \frac{(j_{11}j_{12} + j_{21}j_{22})^2}{(j_{21}^2 + j_{22}^2)}}}$$

Equation 26:

Another approach to reconstructing a continuous representation of the reference image attempts to fit a bilinear surface patch between any four neighboring grid samples. This representation also has C^0 continuity, but it uses a polynomial basis. First, the individual sample points of the reference image are mapped to the desire image's coordinate system. These warped points will generally not correspond to sample grid positions. The connectivity of the eight-connected neighborhood of each reference sample-point is maintained after the warp. This can be managed by maintaining a buffer of two scan lines while enumerating the reference image in a visibility compatible order. The warped sample points stored in these two buffers can be considered a single strip of patches at the completion of each scan line. A standard polygonal rasterizer can be used to scan convert each patch in the strip. Therefore, this technique can easily take advantage of specialized rasterization hardware if it is available. Once a strip is rasterized, one of the scanline buffers becomes available for storing the mapped values of the next scan line.

Shown below is an example of the same warp using each of the two reconstruction methods discussed.

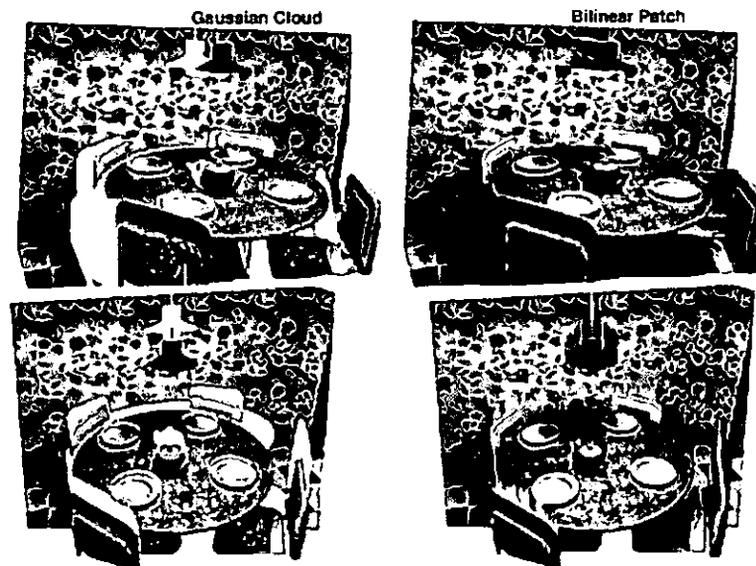


Figure 16: Example image warp using different reconstruction methods

6. Occlusion and Exposure Errors

The image warp can only correctly reproject those scene points visible in the reference image. In some cases, even the visibility of a point does not guarantee its proper reconstruction. These are not

weaknesses of either the image-warping or visibility methods described; they are, instead, inherent limitations of an image-based representation.

The major visual artifacts resulting from these limitations can be classified as one of two cases, *exposure errors* or *occlusion errors*. Exposure errors occur when a background region that should have been occluded is visible in a desired image because of the absence of some foreground element from the reference image. On the other hand, occlusion errors occur in a desired image when an interpolation error in the reconstruction process introduces a false foreground element that covers background regions visible in the actual scene. The choice of reconstruction methods plays a significant role in either amplifying or reducing these errors. However, a reduction in one class of artifact often causes an increase in the other.

Many exposures and occlusions are correct. For instance, when a viewpoint moves toward a foreground object the projection of the object will enlarge in the field-of-view such that it covers adjacent background points. As the viewpoint moves even closer to the foreground object, more of the background is occluded.

Exposure errors and occlusion errors take place either when the underlying assumptions of the reconstruction method are violated, or when there is insufficient information in the image to properly reconstruct the correct surface. These two error sources are closely related. The role of reconstruction kernel is to interpolate the missing gaps between samples. Often the information needed to correctly fill a missing image region is unavailable from the reference image.

Exposure errors are the subtler visual artifact. The region uncovered by a legitimate exposure lends itself to interpretation as a shadow produced by a light source placed at the reference image's center-of-projection. This is particularly noticeable when the exposure occurs along object boundaries. An exposure error occurs when a ray in the desired image passes through this shadow region, allowing some background element to be erroneously seen. Both exposure errors and actual exposures are illustrated below.

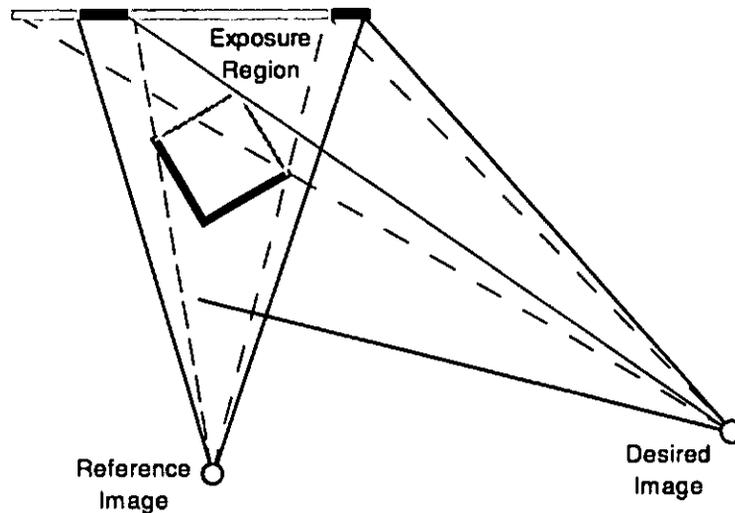


Figure 17: Exposures at occluding boundaries

The actual scene points that are visible from the reference image are shown darkened. The shaded region indicates where an exposure occurs in the desired image. The solid dividing line through the exposure region indicates the boundary between an actual exposure to the right and an exposure error to the left. However, without external information the difference between valid and invalid exposures cannot be resolved.

Exposure errors are most likely to occur at object silhouettes. They occur on smooth surfaces as well as along sharp depth discontinuities. This situation is depicted in Figure 18. Exposure errors occur immediately adjacent to those image points whose ray lies in the observed object's tangent plane. Therefore, as an observer moves to see around an object boundary, she should generally expect to see more of the object rather than any component of the background.

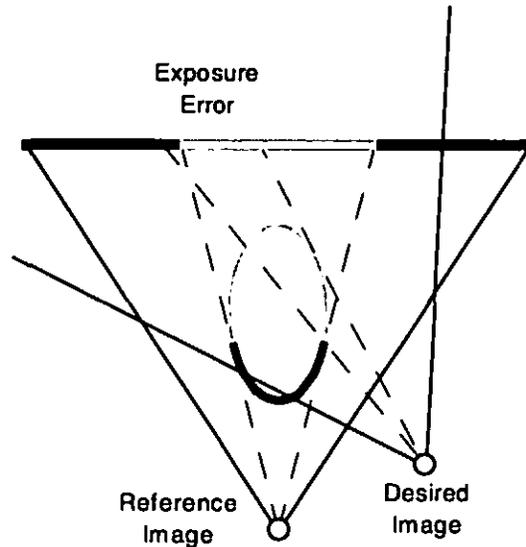


Figure 18: Exposure error on a smooth surface boundary

Merely changing the basis function used in the reconstruction of the reference image can eliminate exposure errors, but it introduces occlusion errors. Consider the example shown in Figure 19 when a polynomial basis, instead of the Gaussian cloud model, is used to approximate the underlying surface.

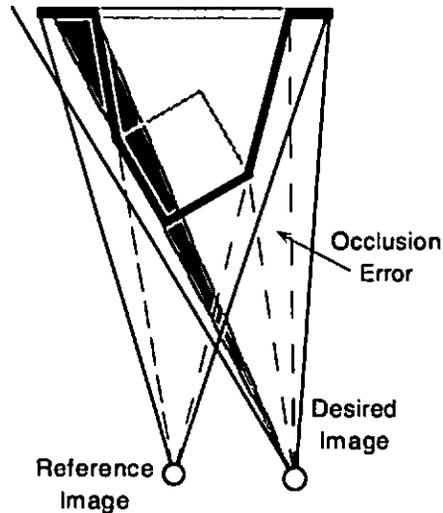


Figure 19: Occlusion errors introduced by polynomial reconstruction

The lighter shaded area indicates the extent of an occlusion error, whereas the darker shaded area represents an actual occlusion. The surface seen along the occlusion error corresponds to an extension of the foreground object's tangent plane. This surface will always enclose the actual surface. However, it is unlikely that any part of this extension will actually represent a point in the environment. The occlusion

error will usually hide valid exposures. Furthermore, since occlusion errors are not adjacent to an actual occlusion, they appear more unnatural than exposure errors.

The continuity of the polynomial interpolation basis causes an excessive estimation of the number of points in the scene. The polynomial interpolation model reconstructs images that are indistinguishable from a model that assumes that all of the points beyond each ray's observed point are occupied. Such a model will not miss any actual scene points, but it will erroneously include scene points that do not exist. The polynomial reconstruction method will, therefore, introduce a disproportionate number of occlusion errors. This can greatly hinder the usefulness of such a model.

In contrast, the Gaussian reconstruction method represents a vacuous estimate of a scene's contents. It assumes that the visible image point is the only scene point located along the ray's extent. Therefore, a Gaussian reconstruction basis will correctly represent all empty regions of an environment, while missing all scene points that are not visible in the reference image. It is, therefore, conservative in its estimate of the occupied volume of space, whereas the polynomial reconstruction makes a conservative estimate of the space's emptiness. Exposure errors should be expected when the Gaussian reconstruction model is used.

The visibility algorithm generally handles valid occlusions. One exception is the case when a scene point from outside the viewing frustum comes into view as a result of the change in the center-of-projection. This situation results in an *invisible occluder error*. A simple example of this case is shown in Figure 20. This problem is a direct result of the limited field-of-view available to a planar-pinhole camera. The use of panoramic pinhole-camera models can remedy this problem.

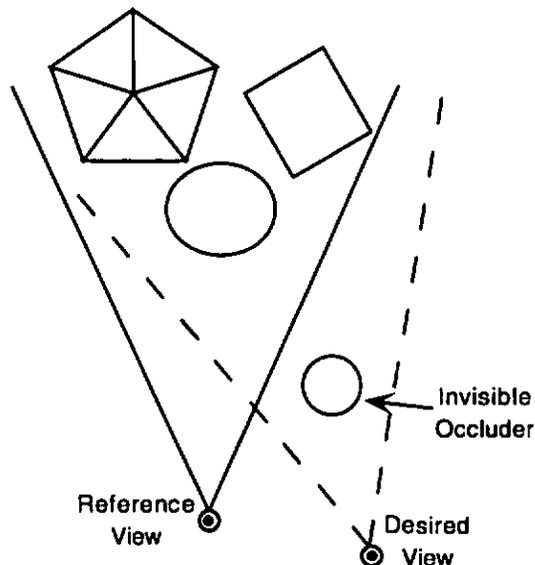


Figure 20: An external exposure error

The two reconstruction methods discussed previously represent two extreme assumptions concerning the structure of space. The use of either of these extreme positions introduces artifacts in the

final rendering. A more accurate reconstruction should combine elements of both methods. However, this might require additional information beyond that which is deducible from the reference image alone. This is an interesting area for future research.

7. Summary

This report has presented a complete example of an image-based method for synthesizing computer graphics. Special mapping functions, called image warps, were derived that enabled arbitrary views of a scene to be generated. This underlying scene was represented by a reference image that was augmented by a scalar value defined at each point, called generalized disparity. An algorithm was also presented to resolve the visibility of the mapped reference points at each coordinate in the desired image. Two methods were presented for reconstructing continuous representations of the warped reference image from these warped points.

8. References

- [Anderson82] Anderson, D., "*Hidden Line Elimination in Projected Grid Surfaces*," **ACM Transactions on Graphics**, October 1982.
- [Appel67] Appel, A., "*The notion of quantitative invisibility and the machine rendering of solids*," **Proceedings of the ACM National Conference**, ACM, New York, October, 1967, pp. 387-393.
- [Catmull74] Catmull, E., "*A Subdivision Algorithm for Computer Display of Curved Surfaces*," Ph.D. Thesis, Department of Computer Science, University of Utah, Tech. Report UTEC-CSc-74-133, December 1974.
- [Chen93] Chen, S. E. and L. Williams. "*View Interpolation for Image Synthesis*," **Computer Graphics (SIGGRAPH'93 Conference Proceedings)**, July 1993, pp. 279-288.
- [Faugeras92b] Faugeras, O. D., "*What can be seen in three dimensions with an uncalibrated stereo rig?*," **Proceedings of the 2nd European Conference on Computer Vision**, Springer-Verlag, 1992, pp. 563-578.
- [Heckbert89] Heckbert, P. S., "*Fundamentals of Texture Mapping and Image Warping*," Masters Thesis, Department of EECS, UCB, Technical Report No. UCB/CSD 89/516, June 1989.
- [Horn89] Horn, B.K.P., "*Obtaining Shape from Shading Information*," **Shape From Shading**, Chapter 6, Edited by Berthold K. Horn and Michael J. Brooks, The MIT Press, Cambridge, Massachusetts, 1989.
- [Mark97] Mark, W. R., L. McMillan, and G. Bishop, **Proceedings of 1997 Symposium on Interactive 3D Graphics** (Providence, Rhode Island, April 27-30, 1997).
- [Mitchell88] Mitchell, D.P. and A.N. Netravali, "*Reconstruction Filters in Computer Graphics*," **Computer Graphics (SIGGRAPH'88 Conference Proceedings)**, Vol. 22, No. 4, August 1988, pp. 221-228.

- [Rogers85] Rogers, D.F., **Procedural Elements for Computer Graphics**, McGraw-Hill, New York, NY, 1985.
- [Szcliski96] Szcliski, R., "*Video Mosaics for Virtual Environments*," **IEEE Computer Graphics and Applications**, March 1996, pp. 22-30.
- [Westover90] Westover, L. A., "*Footprint Evaluation for Volume Rendering*," **Computer Graphics (SIGGRAPH'90 Conference Proceedings)**, Vol. 24, No. 4, August 1990, pp. 367-376.
- [Wolberg90] Wolberg, G., **Digital Image Warping**, IEEE Computer Society Press, Los Alamitos, California, 1990.
- [Wolff94] Wolff, L. B., and E. Angelopoulou, "*3-D Stereo Using Photometric Ratios*," **Proceedings of the European Conference on Computer Vision**, Springer-Verlag, 1994, pp. 247-257.

Computing Visibility Without Depth

Leonard McMillan
Department of Computer Science
University of North Carolina,
Sitterson Hall, Chapel Hill, NC 27599
email: mcmillan@cs.unc.edu

1.0 Introduction

A fundamental problem in computer graphics is the determination of the visible subset of surface elements within a scene given the desired viewpoint. Often, the most appropriate method for computing visibility is dependent on the form and properties of the scene's description. For example, if the geometry of a scene is described by a collection of opaque convex planar regions (i.e. triangles), then a simple depth buffer can be used to determine the correct visibility. However, if the scene is described as a level-set of some implicit function (i.e. ellipsoid), then a ray-casting algorithm might be best.

This paper presents a visibility solution for a specific type of scene description, called a *perspective-projected surface*. Surfaces of this type are commonplace, since they describe nearly all physically-based image-formation processes. The visibility algorithm works by establishing a particular drawing order in which the last surface written at each pixel corresponds to the visible surface at that point. The drawing process can then proceed without intermediate tests or auxiliary storage. In this respect it is similar to the classic painter's visibility algorithm [Rogers85]. The unique aspect of this proposed technique is that the drawing order is established independent of any information relating to the scene's geometry, including the depth information. Thus, I claim that, for an important class of scene descriptions, there exists an algorithm for computing correct visibility solutions that requires no information about the geometric contents of the scene.

1.1 Background

Sutherland and Sproull [Sutherland74] first realized that the task of determining visibility can be reduced to a sorting problem. In their classic taxonomy of hidden-surface algorithms, they showed that all of the previously known visibility algorithms can be distinguished entirely by the permutations over which each of the three-dimensions are traversed while sorting. Moreover, if the scene primitives are placed into a common canonical frame (where x and y correspond to the coordinate system of the desired output image), then it can easily be shown that only one of the three sorts (the one in z) actually determines visibility, while the remaining two sorts merely provide algorithmic optimizations.

In essence, the visibility problem can be considered parallel sorting where separate lists

are maintained for various regions of the output image. The geometric properties of scene elements are used to both define these list regions, as well as to simplify their management. If one is willing to tolerate sampling artifacts, then these list regions can be defined to the smallest resolvable image element (a.k.a. pixel), making the region definition independent of the geometry of the scene's elements. You need only to add to Sutherland and Sproull's original analysis the distinction between forward-mapping (i.e. for each primitive, figure out which image regions are effected) and inverse-mapping (i.e. for each screen region, figure out what scene primitives belong in the list) to include all of the known visibility approaches to date¹.

1.2 Incremental Sorting

An obvious way of simplifying a sorting problem is to retain the list elements in some naturally ordered data structure. For example, many volumetric rendering techniques compute visibility by exploiting the ordering imposed by the underlying representation. This class of techniques, which I will call incremental sorting, begins with an initially sorted list that undergoes either a global or a local perturbation. In the case of a global perturbation, knowledge of the initial ordering is exploited to simplify the subsequent sorting process. For example, consider the initial ordered list shown in the following figure.

a. Original Ordered List before perturbation by $x^2 - 18x + 81$

3	5	8	11	12	14	17	21
---	---	---	----	----	----	----	----



Pivot position for merge stage as determined by the extrema of the global perturbation ($x = 9$)

b. List after global perturbation

36	16	1	4	9	25	64	144
----	----	---	---	---	----	----	-----

Once a global perturbation is applied, the resulting list can be partitioned into sub-lists that are separated by the extrema of the perturbation function. In the regions that lie between the extrema, the perturbing function is either monotonically increasing or decreasing. Since the initial sorted list is analogous to the number line, we can generate a

1. In the intervening time since Sutherland and Sproull's original insight, the theoretical algorithm community has more finely honed the distinctions between various sorting problems. In today's language, visibility would more accurately be considered as an order statistic calculation problem rather than a pure sorting problem (assuming that all surfaces considered are opaque; once multiple levels of transparency are allowed, then computing visibility reverts back to a pure sorting problem). The distinction between computing an order statistic problem and the pure sorting problem is that the final output is composed of only one list element. Typically, we are only interested in the single scene element that is visible for a given screen region, rather than a list of all the scene elements that project onto the region. The relevance of this distinction is that the order-statistic problem is known to require fewer operations, $O(n)$, than the general sorting problem, $O(n \log n)$. Thus, we can expect the cost of computing visibility to be linear in the number of scene primitives.

final sorted list by combining these sub-lists using a list merging algorithm similar to the one used by merge sort. This approach requires fewer operations than a general resorting after the perturbation. For example, consider the global perturbation defined by the mapping function $x^2 - 18x + 81$. This function has a single minimum at the point $x = 9$. For values of x less than 9 the function decreases monotonically as x increases, while for greater values it increases monotonically. The two sub-lists can be merged into a single sorted list in linear time as shown below. Likewise, the minimum value in the list can be determined in constant time by comparing the first values in each sub-list.

First Sub-list (Reversed)	1		16		36		
Second Sub-list		4	9		25		64, 144
Resulting Sorted List	1	4	9	16	25	36	64, 144

The visibility algorithm derived in this paper bears a close resemblance to the incremental sorting algorithm just described. It begins with a scene representation that can be considered as an initial sorting. Any subsequent reprojections of the scene act as global perturbations with easily established extrema. These extrema partition the representation into sub-parts whose relative ordering is unaffected by the perturbation. These sub-parts can then be merged into a final result exhibiting a correct visibility solution. In addition, special attributes of the representation, the global perturbation, and the re-projection process, allow the merging to be accomplished without referencing the actual list values.

This paper is organized as follows. First, an algorithm is given for computing the correct visibility of a perspective-projected image as it undergoes arbitrary reprojections. Both the original and its reprojections are assumed to be planar. Next, a proof is given for the correctness of this algorithm. Initially, the proof is shown for spherical projections. Finally, the proof for the spherical case is mapped to the planar case.

2.0 Visibility of Projected Surfaces

A perspective-projected surface is distinguished by an origin, a projection manifold, and a scalar range function. This origin, \hat{e} , which is often called the *center of projection*, can be considered as the origin of the set of all rays¹ that form the projection. It is convenient to consider the projection manifold, P , as a parameterized surface defined over a space with one less dimension than that of the manifold itself. For the sake of discussion, only three-dimensional manifolds with a parameterization over two-dimensions will be considered. Thus, every point of the manifold, $P(u, v)$, defines a specific ray whose origin is

1. More concisely a "pencil" of rays.

given by, \dot{e} , the center of projection¹. The range function, $R(u, v)$, specifies a length for each ray and is defined over the same parameter space as the projection manifold. Therefore, a perspective-projected surface is itself a parameterized manifold defined over u and v as shown below.

$$S(u, v) = \dot{e} + R(u, v) \frac{P(u, v) - \dot{e}}{|P(u, v) - \dot{e}|}$$

We will consider the problem of re-projecting a perspective-projected surface. This means that, given an initial perspective-projected surface, we would like to generate a consistent new surface with a different origin and/or projection manifold. One difficulty of the re-projection arises from the fact that only a single range value is defined for each ray. However, the re-projection process allows for any number of points from the initial surface to fall along a ray. We resolve this ambiguity by defining the notion of visibility. Visibility dictates that the appropriate choice for selecting one from a number of surface points that map to the same ray is to choose the closest one to the desired projection's origin. If we consider the surface to be composed of an opaque material, then visibility corresponds to the natural visual phenomenon of occlusion where far away surfaces along a given ray are hidden by closer surfaces.

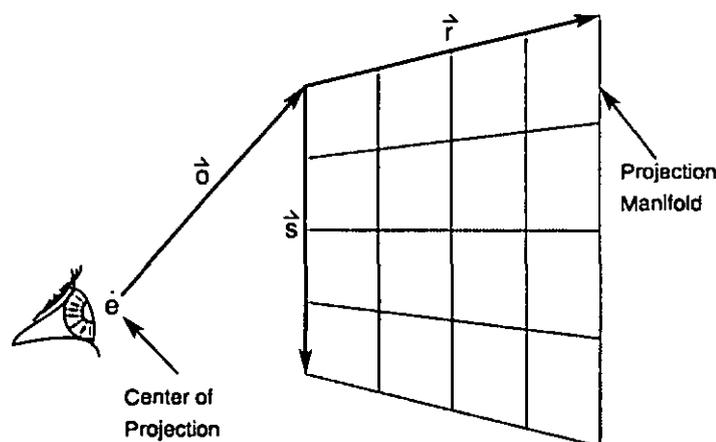
One might ask, why should we restrict ourselves to such limited surface representations in the first place? The primary motivation for this choice is that nearly all natural image formation processes directly record perspective-projected surfaces. This includes photographic images, video images, and the images processed by our visual systems. The goal of re-projecting images is to generate alternate views which correspond to different view positions.

2.1 Algorithm for determining correct visibility

Next, an algorithm for re-projecting planar perspective-projected surfaces is given [McMillan95b]. First, several quantities will be defined in order to describe the algorithm. The original perspective-projected surface is defined by an initial center of projection, \dot{e}_0 , and a planar projection matrix, P_0 . It is useful to consider the columns of the projection matrix as the independent vectors $P_0 = [\mathfrak{r}_0 \ \mathfrak{s}_0 \ \mathfrak{d}_0]$. We can then consider P_0 as a ray-generating function over the parametric domain, $\{(u, v) \mid u, v \in [0, 1]\}$, where the vector \mathfrak{d}_0 is from the center of projection to the parametric origin (i.e. $(u, v) = (0, 0)$) of the planar-projection manifold, and, the remaining two vectors, \mathfrak{r}_0 and \mathfrak{s}_0 , span the plane over the parametric domain. These relationships are depicted in

1. We disallow the case where the center of projection lies on the projection manifold.

the figure shown below.



We define the projection of a point, x , by two functions $\Phi_i(\dot{e}_p, P_p, x)$ and $\rho_i(\dot{e}_p, x)$, where the parametric mapping function Φ_i determines the parameter space coordinate of the ray passing from the center of projection to the point x , and the range function, ρ_i , gives the distance along this ray to x .

An algorithm for determining the correct visibility of a planar-projected surface when re-projected to an alternate viewpoint, \dot{e}_1 , is given as follows:

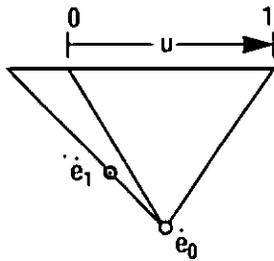
- 1) Find the projection of the desired viewpoint, \dot{e}_1 , on the surface to be reprojected $\Phi_0(\dot{e}_0, P_0, \dot{e}_1)$.
- 2) Partition the manifold into one, two, or four sheets within the parametric domain, according to the isoparametric lines given by $\Phi_0(\dot{e}_0, P_0, \dot{e}_1)$.
- 3) Enumerate each partition along sequential isoparametric lines toward the positive parametric image of the point, \dot{e}_1 , while re-projecting onto the new domain Φ_1 .

The visibility of the projected surface is determined by the combination of the partitioning and enumeration steps. Essentially these steps establish a visibility-compatible ordering. By this we mean any set of points that map to the same parametric coordinate in the desired view will be ordered such that the last point reprojected will be the one closest to the new center of projection. This ordering is established without any reference to the range values of the surface. Therefore, if the re-projection stage of the third step can also be accomplished without explicit reference to the range function, as in [McMillan95a] and [McMillan95c], the projected scene's visibility can be established without any depth information.

Next, several examples are given to illustrate the various enumeration directions used by the algorithm. In order to simplify the case analysis, the two parametric directions are treated independently. The following figure demonstrates the six different ways that the desired viewpoint can project onto the re-projected surface considering only the u

parameter.

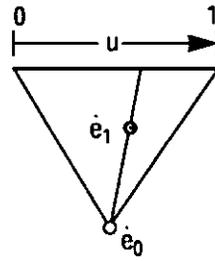
Case 1



$$\Phi_u(\dot{e}_0, P_0, \dot{e}_1) < 0$$

$$\text{and } R(\dot{e}_0, \dot{e}_1) > 0$$

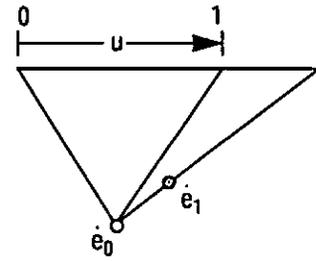
Case 2



$$0 \leq \Phi_u(\dot{e}_0, P_0, \dot{e}_1) \leq 1$$

$$\text{and } R(\dot{e}_0, \dot{e}_1) > 0$$

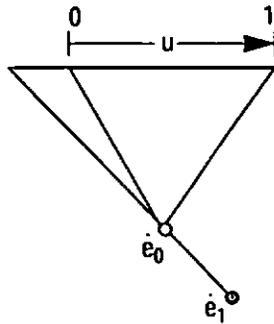
Case 3



$$\Phi_u(\dot{e}_0, P_0, \dot{e}_1) > 1$$

$$\text{and } R(\dot{e}_0, \dot{e}_1) > 0$$

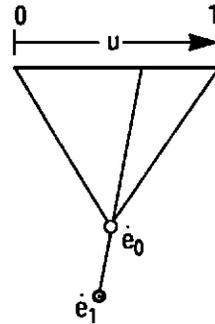
Case 4



$$\Phi_u(\dot{e}_0, P_0, \dot{e}_1) < 0$$

$$\text{and } R(\dot{e}_0, \dot{e}_1) < 0$$

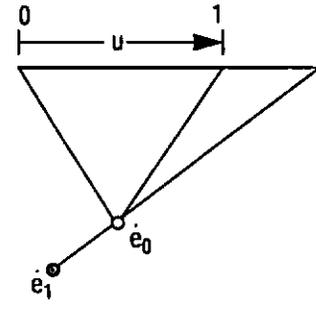
Case 5



$$0 \leq \Phi_u(\dot{e}_0, P_0, \dot{e}_1) \leq 1$$

$$\text{and } R(\dot{e}_0, \dot{e}_1) < 0$$

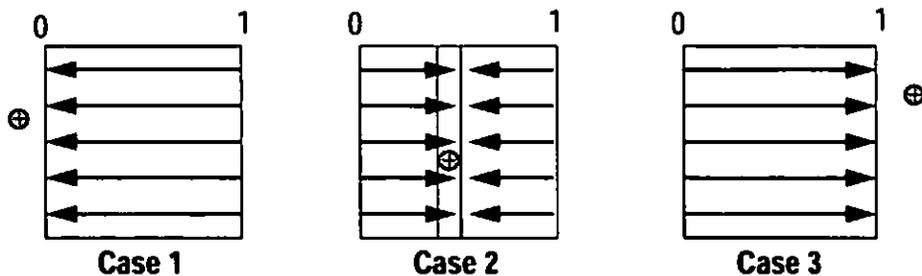
Case 6



$$\Phi_u(\dot{e}_0, P_0, \dot{e}_1) > 1$$

$$\text{and } R(\dot{e}_0, \dot{e}_1) < 0$$

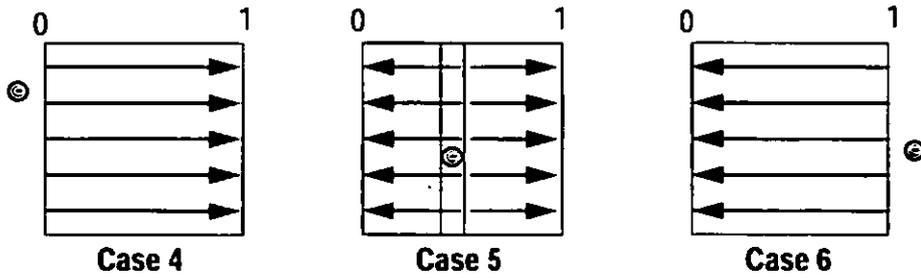
According to the algorithm's third step, the order of projection for the original surface is toward the positive parametric image of the desired center of projection. For the first three cases, this order is shown in the following figure, where the image of the desired center of projection is shown as a darkened circle.



Case 2 illustrates the case where the projection manifold is partitioned into multiple sheets. The only ordering constraint between sheets is that the sheet containing the image of the desired center-of-projection be rendered last. Since the center-of-projection's image can be made part of either of the two sheets, the sheet ordering is in fact

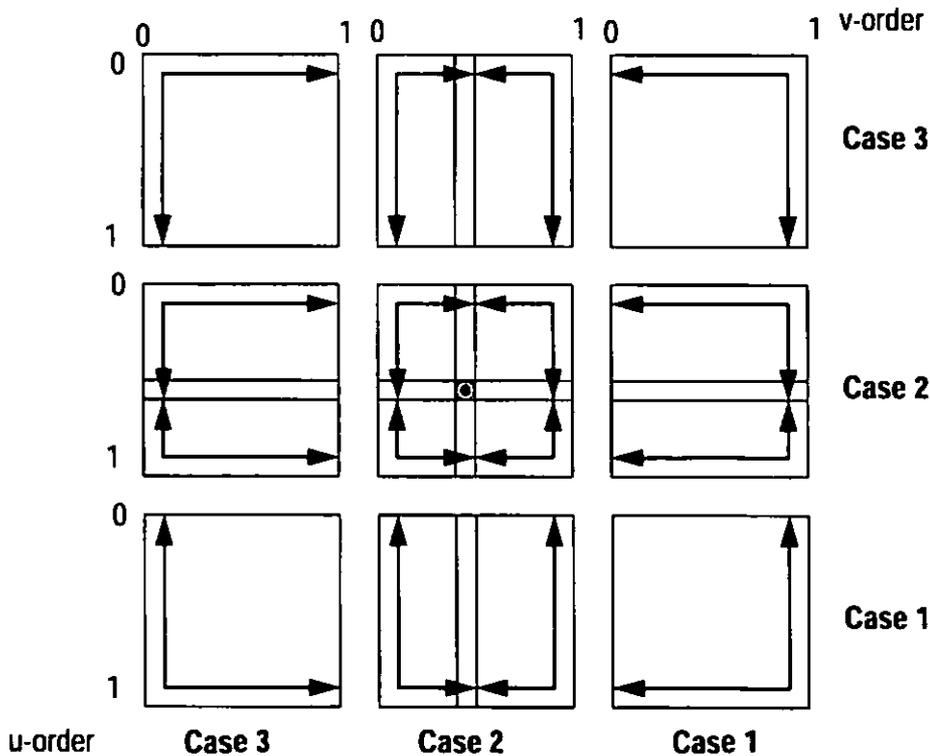
arbitrary.

In cases 4 through 6 the projected image of the desired center of projection is considered to be a negative image since the ray originating from the initial center of projection and passing through the desired one does not intersect the projection manifold. In these cases a point at infinity is treated as the desired center's positive image. The partitioning process will be the same as the positive image case; however, the enumeration order will be reversed as shown below.



Once again, Case 5 divides the surface into multiple sheets which can be projected in an arbitrary order as long as the sheet containing the negative image of the desired view is projected first.

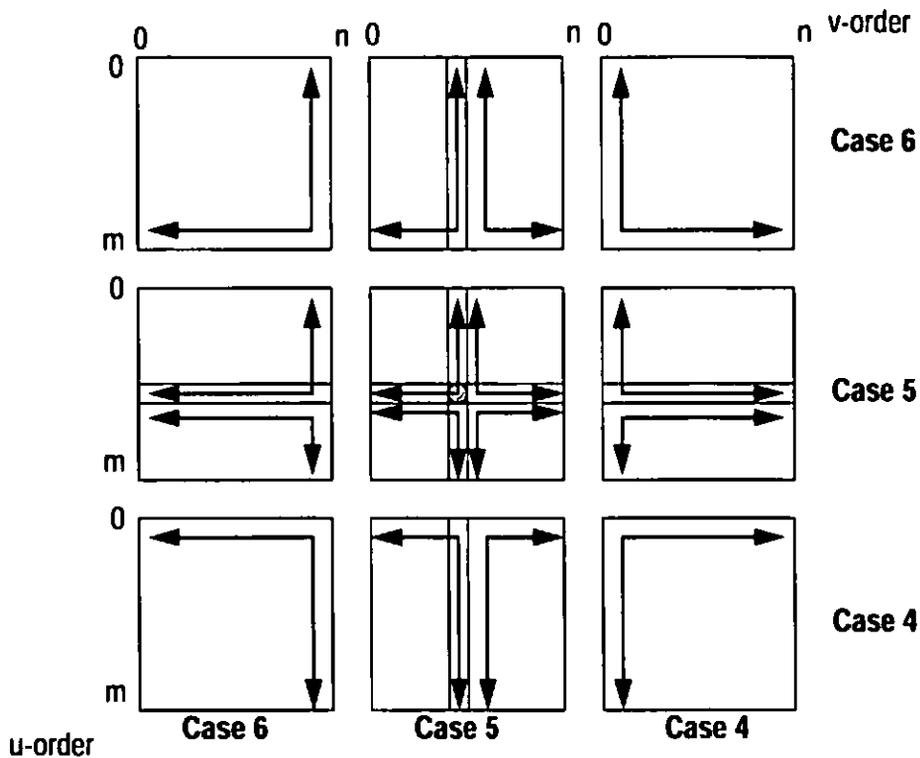
When both parameters of the projection manifold are considered, eighteen different enumeration cases arise. These cases can be considered in two groups.



The first group, shown above, gives the painting order for all the cases where the ray from the initial surface's center of projection to the desired center of projection intersects

the projection manifold with a positive range value. In these cases the sheets are enumerated toward the image of the desired center. Multiple sheets are again introduced when the desired point falls within the unit square of the surface's parameter space. We define the dominate parameter as the one that enumerates isoparametric lines of the other parameter. Since the enumeration of the re-projection can be considered as two nested loops over the points on the surface, the dominate parameter is the one enumerated in the outer loop. Both the sheet ordering and the choice of the dominate enumeration parameter are arbitrary, under the same conditions given for the one-parameter case.

The second group of enumeration cases arises when the desired center of projection projects as a negative image onto the surface's projection manifold. These nine cases are shown below.



The projected surface's points are mapped along isoparametric curves that are progressively farther away from the projected image of the desired viewpoint, toward the positive image at infinity. Multiple sheets arise as before, and their relative re-projection order is independent under the conditions given previously.

In review, the algorithm described claims to compute the correct visibility for the perspective-projected surface that arises from the re-projection of another perspective-projected surface. This is done by establishing an enumeration, or painting, order which is based solely on the relative positions of the two centers of projection and the projection manifold of the original surface, and this order is independent of the surface's range function.

3.0 Proof of Algorithm's Correctness

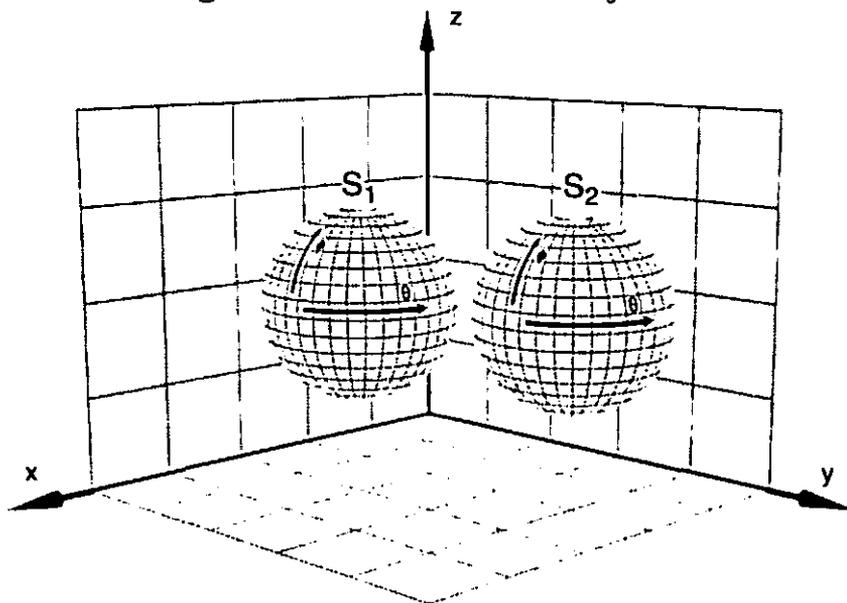
In order to prove the claims of the visibility algorithm discussed, we will generalize the planar-projection manifold to a spherical manifold. This allows for a single parameterization that encompasses all rays originating from the surface's center of projection. This generalization has two benefits. First, there will always be a positive image of the desired new center, thus halving the number of enumeration cases. Second, since the projected image of the desired center of projection will always fall within the parametric domain, the remaining nine cases are all collapsed into one.

Once the algorithm is shown to be correct for the spherical case, the result will be transferred back to the case of planar-projective surfaces. Then, an overview of how the algorithm can be extended to other projection manifolds is given.

3.1 Definitions

As stated previously, a perspective-projected surface is defined by a central point and a single-valued range function over some bundle of rays emanating from that point. The bundle of rays is defined over the parameter space of a projection manifold, and the central point is called the center of projection. For any perspective-projected surface it is useful to consider the equivalent surface that arises when the ray bundle is mapped onto a spherical manifold with the same center of projection. Within such a framework all the possible rays can be described by a single parameter space.

In this derivation we consider sets of local spherical coordinate frames, $\{S_1, S_2, \dots, S_n\}$ that are embedded within a global cartesian coordinate system, W .



This analysis assumes the existence of a set of points, called a scene, that are defined over the global coordinate frame, W , and independent of any local frame, S_i . We also assume

that all possible perspective projections are consistent with this scene in terms of both visibility and the positions of the surface points within W . For each spherical frame there exists a mapping of these scene points from W into the local coordinate system S_i ,

$$\tilde{P}_i: R_W^3 \rightarrow R_{S_i}^3 \quad \text{by} \quad \tilde{P}(x, y, z) = \begin{bmatrix} \rho_i(x, y, z) \\ \theta_i(x, y, z) \\ \phi_i(x, y, z) \end{bmatrix}$$

While the mapping of points from W to S_i is one-to-one, it is not generally a single-valued range-function when parameterized in θ and ϕ . Therefore, we are interested in a subset of this mapping where only a single value of ρ_i is defined for each unique pair of θ and ϕ . Such mappings are known as topological meshes. An example of one such unique mapping is defined by

$$\tilde{V}_i: R_W^3 \rightarrow R_{S_i}^3 \quad \text{by} \quad \tilde{V}_i(x, y, z) = \begin{bmatrix} r_i(x, y, z) \\ \theta_i(x, y, z) \\ \phi_i(x, y, z) \end{bmatrix}$$

and

$$r_i(x) = \min(\{\rho(y) \mid \theta_i(y) = \theta_i(x), \phi_i(y) = \phi_i(x)\}).$$

This particular mapping will be called the *occlusion compatible mapping*, and it is consistent with a proper visibility solution as defined in previous sections.

Next, we will consider both general mappings and occlusion compatible mappings¹ between pairs of spherical frames. For example,

$$\tilde{P}_{ji}: R_{S_i}^3 \rightarrow R_{S_j}^3 \quad \text{by} \quad \tilde{P}(\rho_p, \theta_p, \phi_p) = \begin{bmatrix} \rho_j(\rho_p, \theta_p, \phi_p) \\ \theta_j(\rho_p, \theta_p, \phi_p) \\ \phi_j(\rho_p, \theta_p, \phi_p) \end{bmatrix}$$

and

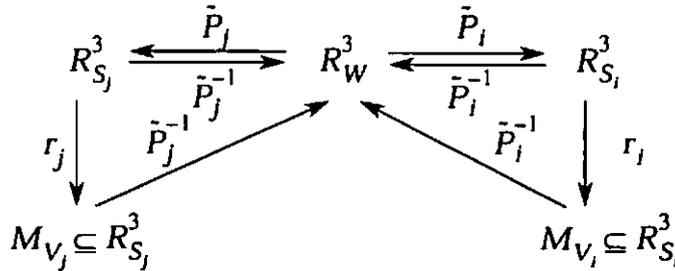
$$\tilde{V}_{ji}: R_{S_i}^3 \rightarrow R_{S_j}^3 \quad \text{by} \quad \tilde{V}_{ji}(\rho_p, \theta_p, \phi_p) = \begin{bmatrix} r_j(\rho_p, \theta_p, \phi_p) \\ \theta_j(\rho_p, \theta_p, \phi_p) \\ \phi_j(\rho_p, \theta_p, \phi_p) \end{bmatrix}$$

where

1. Two different types of occlusion compatible mappings between spherical frames could be defined: those where the domain, S_i , is a general mapping, or those where the domain is occlusion compatible. Since the latter is a proper subset of the former only the general case is considered in this definition. However, later we will deal primarily with the case where the domain is also an occlusion compatible mapping.

$$r_j(\tilde{x}_j) = \min(\{ \rho_j(\tilde{y}_j) \mid \theta_j(\tilde{y}_j) = \theta_j(\tilde{x}_j), \phi_j(\tilde{y}_j) = \phi_j(\tilde{x}_j) \}).$$

The following figure depicts the relationships between these mappings.



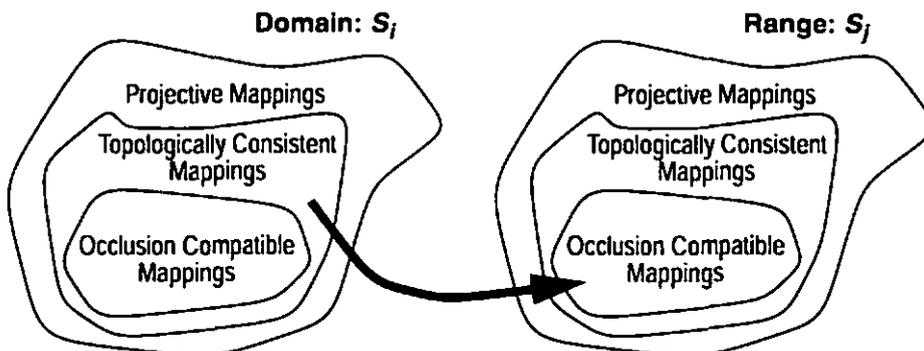
It has been claimed elsewhere [McMillan95c] that general two-dimensional mappings between spherical frames can be determined without an explicit representation of the range information, $\rho(\tilde{x})$ or $r(\tilde{x})$, as shown below:

$$\dot{P}_{ji}(\tilde{x}): R_{S_i}^2 \rightarrow R_{S_j}^2 = \begin{bmatrix} \theta_j(\theta_p \phi_p \delta_j(\theta_p \phi_i)) \\ \phi_j(\theta_p \phi_p \delta_j(\theta_p \phi_i)) \end{bmatrix} \quad \text{where} \quad \delta_j(\theta_p \phi_i) = -\delta_i(\theta_p \phi_j).$$

Unlike range information, the generalized disparity term, $\delta(\tilde{x})$, can be determined directly from observable image features, such as optical flow [Prazdny83] or point correspondences [Laveau94].

These two-dimensional mapping functions, $\dot{P}_{ji}(\tilde{x})$, may result in several points from the domain mapping to a single point in the range. Such range points will be referred to as *topological multiplicities* of the mapping. The specific claim of this report is that an occlusion compatible mapping can also be established independent of range information.

The conversion of a general projective mapping to an occlusion compatible one involves an inherently non-linear selection process. In our definitions, this selection was made on the basis of range information. I will show that the same result can be determined via an appropriately-ordered enumeration over the parameter space of the domain's spherical frame, assuming that it is topologically consistent. While this assumption limits the generality of this approach, it is valid for the important case of those occlusion compatible mappings within the domain.

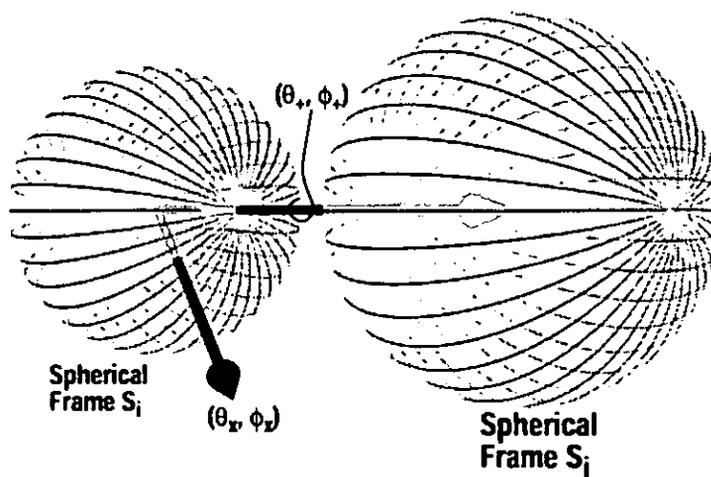


This class contains, as a subset, all perspective-projected surfaces. In a later section, I will discuss the potential for extending these results to a wider class of surface descriptions that allow for non-topologically consistent surfaces.

3.2 Derivation

Several important relationships exist between projective mappings. Embedded spherical frames, like those just defined, will be used to represent all aspects of a projective surface. The origin of the spherical frame will serve as the center of projection while the unit sphere will be used as the projection manifold. The initial projected surface is represented by the S_i frame while the desired re-projection is represented by S_j .

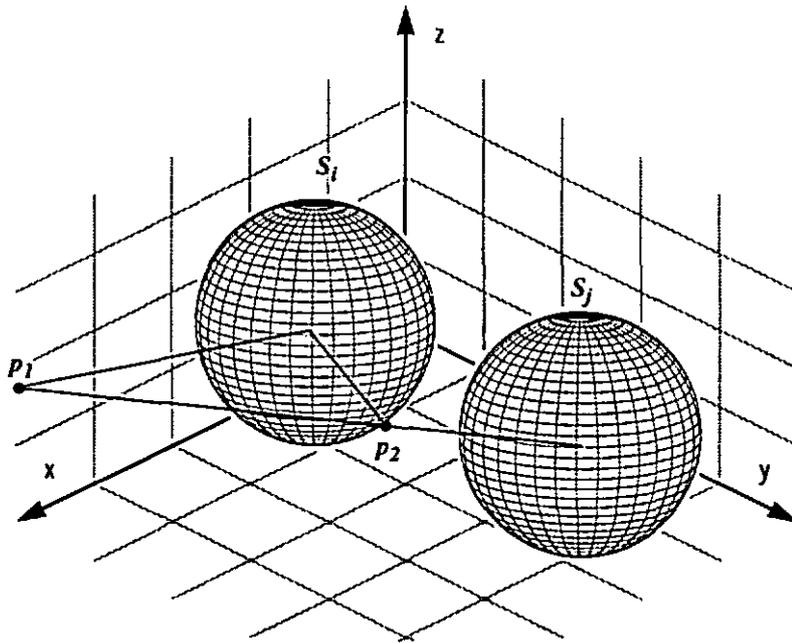
In general, all possible re-projections, \tilde{P}_{ji} , of points falling along a given ray, (θ_x, ϕ_x) , of S_i are constrained to lie within a plane that is defined by the line segment, l , connecting the initial and desired centers of projection, and the ray itself. There are two exceptions to this observation. On a spherical projection manifold there exists one ray in the direction of the line segment, l . This ray, (θ_+, ϕ_+) , which is called the *positive epipole*, is the virtual image of the desired center of projection on the original surface. A second ray, called the *negative epipole* (θ_-, ϕ_-) lies in the opposite direction. It corresponds to the vanishing point for all rays originating from S_j in the direction of S_i when seen in projection. Points along either of the epipolar rays are constrained to lie on a line which has l as a segment. This line is also common to all of those constraint planes induced by the re-projection of other rays. For this reason these constraint planes will henceforth be called *epipolar planes*¹. The images of these epipolar planes correspond to the lines of longitude for a sphere with the epipoles as poles. These relationships are shown in the following figure.



Next, consider an isolated topological multiplicity on the projective mapping from S_i to

1. It is common in the computer vision literature to call the intersections of the epipolar planes with the projection manifold *epipolar curves*. Likewise, the images of the epipolar rays on the projection manifold are often called an *epipolar points*.

S_j , as shown below:



Theorem 1. *Generically, the points of a topological multiplicity induced by a mapping from S_i to S_j , $\bar{P}_{ji}(\bar{x})$, and the two centers of projection lie in a plane.*

Proof: The points of the topological multiplicity are colinear with the origin of S_j since they share the same parametric coordinates. A second line segment connects the local frame origins, S_i and S_j . In general, these lines are distinct and, since they share a common point, they must be coplanar.

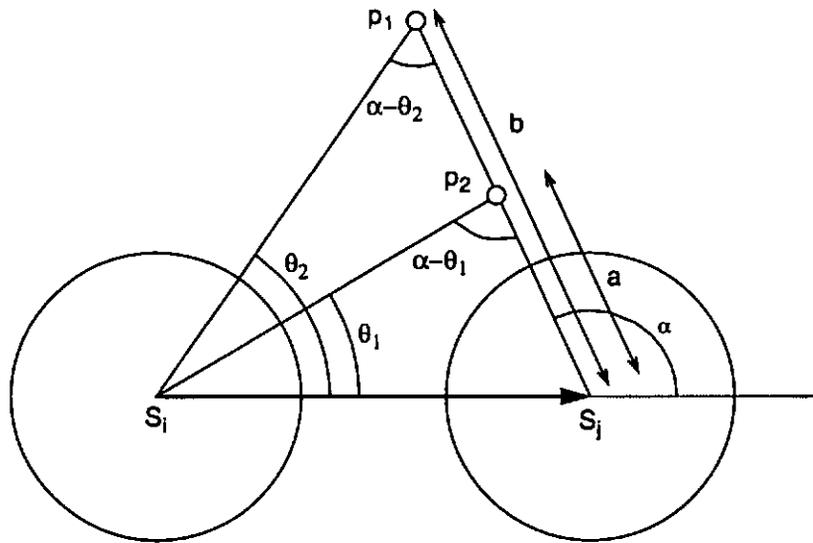
Notice that this plane of the multiplicity also corresponds to an epipolar plane. From Theorem 1 we can conclude that any multiplicity induced by a projective re-mapping takes place within an epipolar plane that is common to all of those rays whose surface positions take part in the multiple mapping. Thus, all interactions between surface points, when viewed from the desired center of projection, are constrained to occur within epipolar planes.

Having established the importance of these epipolar rays and planes, it is convenient to re-orient the local spherical frames so that the epipolar planes fall upon isoparametric curves. This can be accomplished with a global change of coordinate systems describable by a transformation, A , of W . The single affine transform, A , can be constructed to achieve all of the following results:

- Translate S_i to the origin
- Rotate S_j to lie on the x-axis
- Rotate the epipolar plane of the multiplicity into the xy-plane with a rotation about the x-axis
- Scale the system so that S_j has the coordinate (1,0,0).

With this transformation we can consider the multiplicity entirely within the xy-plane, as

shown in the following figure.



Theorem 2. If $\cos \theta_1 > \cos \theta_2$ and $0 < \theta_1, \theta_2, \alpha < \pi$ then $a < b$.

Proof: The length of sides a and b can be computed in terms of the angles $\theta_1, \theta_2,$ and α using the law of sines as follows.

$$\frac{a}{\sin \theta_1} = \frac{1}{\sin (\alpha - \theta_1)} \quad \frac{b}{\sin \theta_2} = \frac{1}{\sin (\alpha - \theta_2)}$$

Thus,

$$a = \frac{\sin \theta_1}{\sin (\alpha - \theta_1)} \quad \text{and} \quad b = \frac{\sin \theta_2}{\sin (\alpha - \theta_2)}$$

$$a = \frac{\sin \theta_1}{\sin \alpha \cos \theta_1 - \cos \alpha \sin \theta_1} \quad b = \frac{\sin \theta_2}{\sin \alpha \cos \theta_2 - \cos \alpha \sin \theta_2}$$

$$a = \frac{1}{\sin \alpha \cot \theta_1 - \cos \alpha} \quad b = \frac{1}{\sin \alpha \cot \theta_2 - \cos \alpha}$$

Note that the denominators in the expressions of a and b must be strictly positive, since their lengths are positive over the range of angles defined. Furthermore, the ratio of a to b can be expressed as:

$$\frac{a}{b} = \frac{\sin \alpha \cot \theta_2 - \cos \alpha}{\sin \alpha \cot \theta_1 - \cos \alpha}$$

The given relationship

$$\cos \theta_1 > \cos \theta_2,$$

implies

$$\cot \theta_1 > \cot \theta_2, \quad (\text{since } \cos \theta \text{ and } \cot \theta \text{ are monotonically decreasing})$$

$$\sin \alpha \cot \theta_1 > \sin \alpha \cot \theta_2, \quad (\sin \alpha \text{ is always positive over the interval})$$

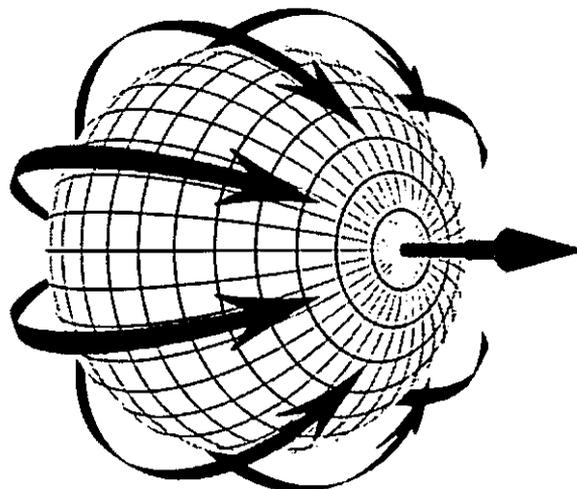
$$\sin \alpha \cot \theta_1 - \cos \alpha > \sin \alpha \cot \theta_2 - \cos \alpha, \quad (\text{subtraction from both sides})$$

$$1 > \frac{\sin \alpha \cot \theta_2 - \cos \alpha}{\sin \alpha \cot \theta_1 - \cos \alpha} \quad (\text{since both sides are strictly positive})$$

Therefore, $\frac{a}{b} < 1$ and $a < b$. For angles θ_1 and θ_2 within the interval $[0, \pi]$ the relationship $\cos \theta_1 > \cos \theta_2$ also implies that $\theta_1 > \theta_2$.

Next, consider all of the surface points of V_i whose rays fall along a common epipolar plane. If, during the re-projection from S_i to S_j , each point is processed in order of decreasing θ then, by Theorem 2, when a multiplicity occurs along a given ray defined on S_j , the most recently processed point will always have a closer range value than any point processed previously. We can, therefore, compute an occlusion compatible mapping, $V_{ji}(\tilde{x}_j)$, by mapping in order of decreasing θ and allowing later surface points to overwrite previous ones. Notice that this mapping procedure only considers the positions of the initial and desired centers of projection, and makes no use of the range information itself.

Since all surface interactions are constrained to take place within a single epipolar plane, we can compute all such planes independently. We then arrive at an algorithm for computing an entire occlusion-compatible surface from the re-projection of any other topologically consistent projective surface. The resulting surface will also be topologically equivalent to a sphere since only a single surface element is retained along each ray. The resulting re-projection order is depicted in the figure shown below.



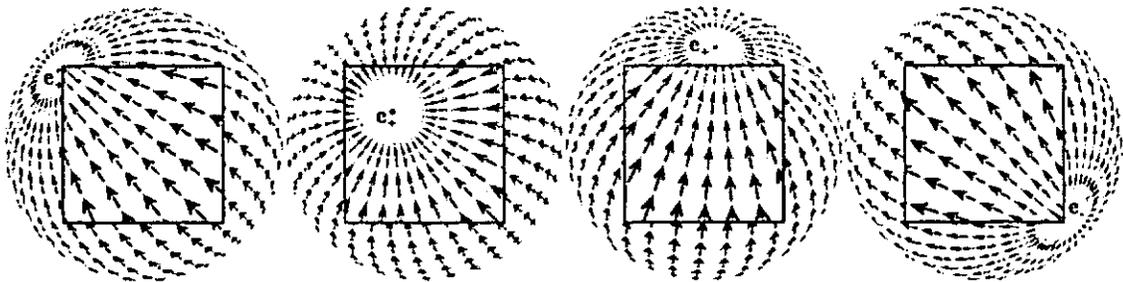
3.3 Mapping to a planar manifold

Next, consider the implications of this result to a planar-projected surface. The bundle of

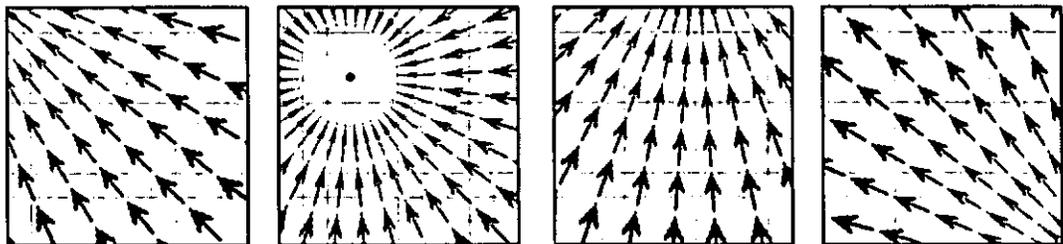
rays described by a center of projection and a planar projection manifold are a subset of those rays given by a spherical manifold with the same center.

The solid angle of this bundle can never be more than 2π , and this maximum occurs only when the projection plane is unbounded. We can, therefore, map any planar-projected surface onto a subset of a spherical-domain with a re-parameterization of the plane. Once the planar-projected surface is so mapped, the re-projection to a desired center of projection can proceed along epipolar planes in order of decreasing θ from the negative epipole towards the positive epipole as described previously. The resulting spherical mapping can then be re-parameterized to match the desired planar projection manifold.

We can also consider this series of steps in its entirety. In the figures shown below, the initial surface of projection is shown overlaying the equivalent spherical projection surface. The enumeration direction implied by the change in center of projection is also overlaid on both surfaces. The enumeration direction vector field shown on the planar surface is a projection of the field seen on the sphere. Notice that the field lines which correspond to epipolar planes map onto lines in their planar image. This is to be expected since the intersection of any two planes in the generic case is a line.

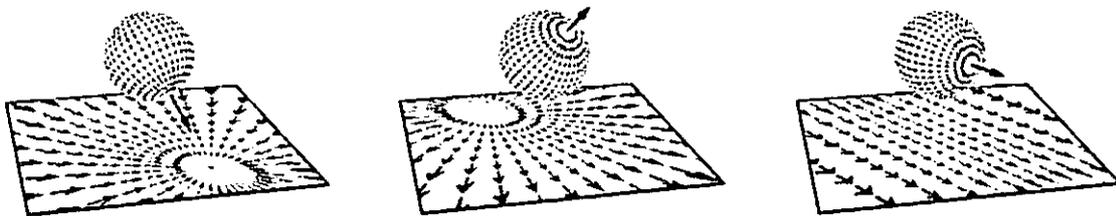


In the following images the enumeration direction fields are shown without the underlying spherical map. Isoparametric lines of the planar domain are also shown for each of the planar projection manifolds.



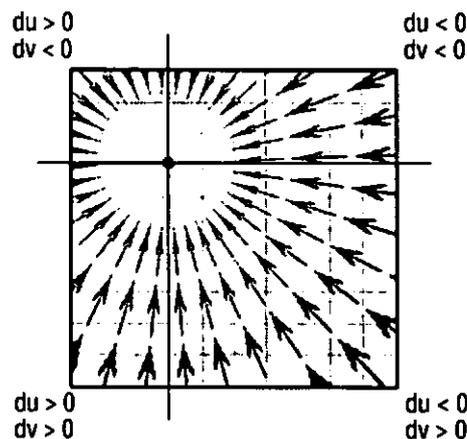
Consider the relationship of these images to the visibility algorithm given in Section 2. In the first step of the algorithm, the projection of the desired viewpoint onto the initial image is computed. By definition, this projection lies upon one of the two epipolar rays. Observe that these points are singularities of the vector field. In retrospect, the positive-

ness and negativeness of the epipolar rays are established by the index number of the singularity. Also, observe that the epipolar rays' images may or may not fall within the parametric domain of the planar image. Nonetheless, all images of the epipolar planes either converge to the image of the positive epipolar ray or diverge from the negative epipolar ray's image. It can easily be shown that there are only three possible configurations of the epipolar rays in relation to the planar projection manifold. The infinite plane of projection is intersected by either the positive or the negative epipolar ray, or it is parallel to both. Therefore, the set of epipolar planes, whose spherical images correspond to the longitudinal lines of the sphere, will, on a planar manifold, map to directed lines that either all converge, all diverge, or are parallel as determined by the projected image of the epipolar rays (see figure below).



Initially, we will consider only the two cases where one of the epipolar images intersects the projection manifold.

The second step of the visibility algorithm divides the manifold along isoparametric lines that are defined by the coordinates of the epipolar ray's image. If the projected epipolar ray's coordinates falls within the parametric domain of the planar image, this partitioning step divides the plane into four regions. When the epipolar planes' images converge or diverge, there exists exactly two epipolar planes that coincide with isoparametric lines. These planes define boundaries across which the vector field defined by the projected epipolar planes changes sign in slope.

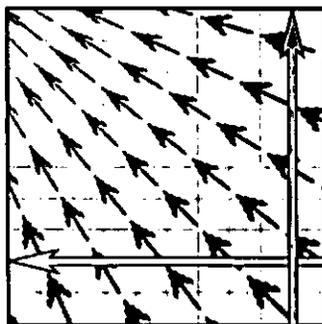


If we consider the subdivision of the manifold to be based on the change of sign in the slope of the epipolar plane images when mapped into the parametric coordinate system,

then the various cases of the visibility algorithm become obvious. If one coordinate of the epipolar ray's image onto the plane falls outside of the parametric domain, then the region is divided into two subregions. If both coordinates fall outside of the parametric domain, then the entire region has a single common slope direction. This last condition corresponds to the single sheet case of the algorithm.

The final step in mapping the spherical-projection-manifold's visibility proof to a planar-projection-manifold is determining the order of enumeration. It is clear that points on the planar surface can be reprojected in the order implied by the projected flow lines (i.e. the image's of the spherical epipolar planes), and the result would be consistent with the spherical re-projection. This result would correspond to mapping the original planar projected-image onto a spherical-projected image with the same center-of-projection, then re-projecting the result onto a new spherical frame. While this approach works, it is inconvenient since it involves extra mapping steps. These extra steps can be avoided if the enumeration order is specified in the parametric space of the original projection manifold, as is done in the visibility algorithm presented. Next we show that the combination of the partitioning and enumeration steps specified by the proposed visibility algorithm are equivalent to mapping along the epipolar planes of the spherical-projected surface.

In the spherical-projection case each longitudinal line corresponding to an epipolar plane was considered independently. In the planar case it is necessary to consider all epipolar planes simultaneously. In the following figure a single sheet is shown with two directed isoparametric lines superimposed.



Notice the following properties of these directed isoparametric lines. Each lines' direction is consistent with the spherical flow field's projection onto the corresponding parametric axis. This direction is the same for all isoparametric lines within a given sheet since both components of the flow field's slope have a constant sign. A given isoparametric line will cross the projected image of each epipolar plane exactly once. If a series of these isoparametric lines are enumerated in the direction of the flow field's transverse component, then the overall image of the spherical-projected flow field is preserved. Furthermore, the relative ordering of the two parameters is arbitrary.

By Theorem 1, topological multiplicities are constrained to occur only within epipolar planes. The projected-spherical flow field is the image of these epipolar planes. If a series of isoparametric lines are mapped in the order determined by the sheet's slope component in the transverse direction to the isoparametric lines, then each point along the epipolar plane will be re-projected in order of decreasing θ . This relative ordering is independent of the u or v parameter choice. Theorem 2 proves that such an ordering is occlusion compatible. Therefore, the resulting planar enumeration is equivalent to mapping the planar-projected surface onto a spherical manifold and performing the re-projection there. Thus, the planar algorithm given is also occlusion compatible.

The various partitioning and enumeration cases of the planar visibility algorithm all arise from the structure induced on the infinite planar manifold by the parameterization. The choice of the parametric domain determines both the subdivision boundaries as well as the number of sheets. The enumeration direction relative to the epipolar ray's image is determined by whether that image is of the positive or negative epipolar ray. The complementary epipole's image appears at infinity. Thus, the enumeration order of any projective surface, which is topologically equivalent to some spherical region, is always from the negative epipole toward the positive one.

Now, consider the case where the projection plane is parallel to the epipolar ray. Since the projected image of all the epipolar rays onto the planar manifold are parallel, it can be considered a special case of the re-projection where the entire infinite plane consists of a single sheet. The parameterization still determines the enumeration order along isoparametric lines based on the slope of the flow field induced by the image of the spherical frames' lines of longitude. Even though both of the epipolar rays project to infinity on the planar manifold, the direction from the negative to the positive epipolar ray's image is still well defined since all of the epipolar plane images are parallel. The visibility algorithm will handle this correctly if interaction toward epipolar rays at infinity is handled consistently.

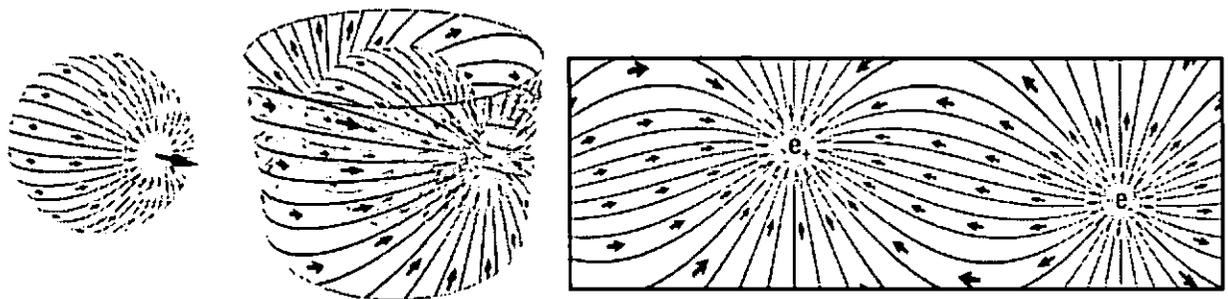
4.0 Discussion and Conclusions

Special cases of the visibility algorithm presented in this paper have been discovered previously by Anderson [Anderson82] and Adelson and Hodges [Adelson93a]. Anderson's original hidden-line algorithm handled the special case where the initial surface was defined in an euclidean space rather than a projective one. This corresponds to setting the initial center of projection at infinity. This proves to be a useful representation for bivariate functions of the sort that Anderson was interested in displaying. It is also notable that Anderson's algorithm enumerates the surface in an order opposite of the presented algorithm. This requires a buffer to be maintained where the minimum and maximum extents of the projected surface are stored. Anderson's painting rule, which

requires that a surface never backtrack upon itself (i.e. it can never overwrite a surface point which has already been written) is essentially the inverse of maintaining only the last surface written at each point on the desired re-projection. Anderson's enumeration order has the advantage that it never generates points that are subsequently overwritten. This makes a lot of sense when considering that his algorithm was developed for use in line plotter applications where, unlike a modern computer graphics display, once ink is drawn on the paper it cannot be overwritten with satisfactory results. This reduction in the number of writes, however, comes at the cost of maintaining the auxiliary data structure to represent the surface boundaries' extents. Depending on the implementation, the representation of the boundary buffer can itself introduce problems. If the extents of the boundaries are represented only along a discrete set of isoparametric lines, then quantization artifacts will be introduced that can create numerical precision problems for future comparisons. If the boundary curves are represented explicitly as line segments, then the storage requirements are potentially very large.

Adelson and Hodges proposed a method for generating stereo pairs from a single planar projective image by re-projecting the right eye's image from the left eye's image. The algorithm they describe is equivalent to the special case of the given visibility algorithm where the epipolar ray is both parallel to the view plane and it is oriented along an isoparametric line of the planar projection. This unique geometry allows for the right eye's image to be generated by enumerating and re-projecting the left eye's image along all scan lines from left to right.

The given visibility algorithm can also be extended to other projection manifolds, using the same procedure of mapping the problem from the original projection manifold to a spherical manifold, re-projecting it, and then mapping it back onto the original surface. The following figure shows this progression with a cylindrical projection manifold.



Once again the longitudinal lines of the spherical projection correspond to the projected images of the epipolar planes that are induced by a change in center of projection in the direction of the epipolar ray. Images of these epipolar planes can be mapped onto a cylindrical projection manifold that shares the same center of projection. The cylindrical projected surface can then be re-projected along these lines of longitude in an order progressing from the negative to the positive epipolar rays. The resulting enumeration

would be equivalent to the spherical re-projection which has been proven to be occlusion compatible.

Using the simple methodology outlined above, visibility algorithms can be established for any projection manifold that is topologically equivalent to a sphere (i.e. its range function is strictly single valued). While this is the case for all projected surfaces, it does not obviously extend to more general surface representations. This leads to many interesting avenues for future exploration. Consider this: it is a simple matter to represent arbitrary objects in a projective space by arbitrarily choosing some center of projection for them (for example the object's geometric center or the object's center of mass). The resulting representation is essentially the same as the projective mapping, $\tilde{P}_i: R_W^3 \rightarrow R_{S_i}^3$, defined in Section 3. If the proposed visibility algorithm could be simply extended to properly handle surfaces of this type, the resulting algorithm would be truly general purpose.

Acknowledgments

This research was done as a course project requirement for COMP 257 in the fall of 1994. The course was taught by Stephen M. Pizer.

5.0 References

- [Adelson93a] Adelson, S. J., and L. F. Hodges, "*Stereoscopic Ray-Tracing*", *The Visual Computer*, Springer-Verlag, 1993, pp. 127-144.
- [Adelson93b] Adelson, S. J., and L. F. Hodges, "*Exact Ray-Traced Animation Frames Generated by Reprojection*", *Technical Report GIT-GVU-93-30*, Georgia Institute of Technology, (Atlanta, GA), July 1993.
- [Anderson82] Anderson, D., "*Hidden Line Elimination in Projected Grid Surfaces*," *ACM Transactions on Graphics*, October 1982.
- [Chen93] Chen, S. E. and L. Williams. "*View Interpolation for Image Synthesis*," *Computer Graphics (SIGGRAPH'93 Proceedings)*, pp. 279-288, July 1993.
- [Laveau94] Laveau, S. and O. Faugeras, "*3-D Scene Representation as a Collection of Images and Fundamental Matrices*," *INRIA, Technical Report No. 2205*, February, 1994.
- [McMillan95a] McMillan, L., and G. Bishop, "*Head-Tracked Stereo Display Using Image Warping*," *1995 IS&T/SPIE Symposium on Electronic Imaging Science and Technology*, SPIE Proceedings #2409A, (San Jose, CA) February 5-10, 1995, pp. 21-30.

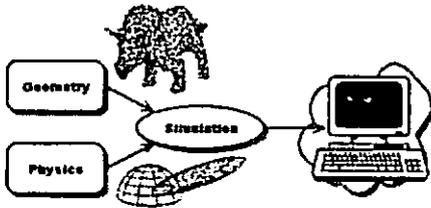
- [McMillan95b] McMillan, L., "A List-Priority Rendering Algorithm for Redisplaying Projected Surfaces," UNC Technical Report TR95-005, University of North Carolina, 1995.
- [McMillan95c] McMillan, L., and G. Bishop, "Plenoptic Modeling: An Image-Based Rendering System," Proceedings of SIGGRAPH 95, (Los Angeles, CA) August 6-11, 1995, pp. 39-46.
- [Prazdny83] Prazdny, K., "On the Information in Optical Flows," *Computer Vision, Graphics, and Image Processing*, Vol. 22, No. 9, 1983, pp 239-259.
- [Rogers85] Rogers, D.F., *Procedural Elements for Computer Graphics*, McGraw-Hill, New York, NY, 1985.
- [Sutherland74] Sutherland, I. E., R. F. Sproull, and R. A. Schumacker, "A Characterization of Ten Hidden-Surface Algorithms," *ACM Computing Surveys*, 6(1), March 1974, 1-55. (Also in J. C. Beatty and K. S. Booth, eds, *Tutorial: Computer Graphics*, Second Edition, IEEE Comp. Soc. Press, Silver Spring, MD, 1982 pp. 384-441)
- [Wang90] Wang S.C. and J. Staudhammer, "Visibility Determination on Projected Grid Surfaces," *IEEE Computer Graphics and Applications*, July 1990.
- [Wolberg90] Wolberg, G., *Digital Image Warping*, IEEE Computer Society Press, Los Alamitos, CA, 1990.
- [Wright73] Wright, T., "A Two-Space Solution to the Hidden Line Problem for Plotting Functions of Two Variables," *IEEE Transactions on Computers*, January 1973.

Image-Based Rendering Using Image Warping

Leonard McMillan
LCS Computer Graphics Group
MIT

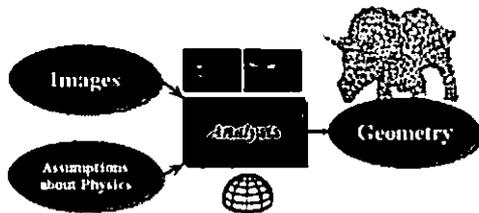
Conventional 3-D Graphics

✓ Simulation



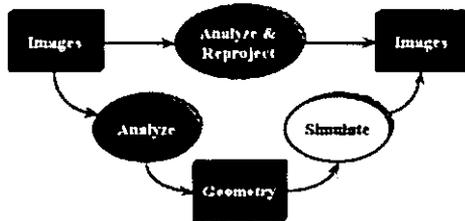
Computer Vision

✓ Analysis

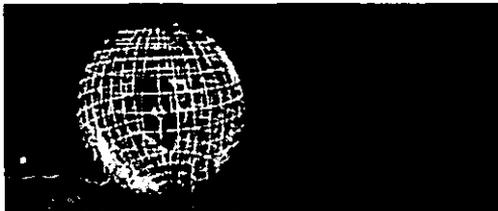


The Image-Based Approach

✓ Transformation



Images as a Collection of Rays



A image is a subset of the rays seen from a given point
- this "space" of rays occupies two dimensions

The Plenoptic Function

✓ The set of rays seen from all points ...



$$p = P(\theta, \phi, x, y, z, \lambda, t)$$

Image-based rendering is about

...reconstructing the plenoptic function from a set of samples taken from it.

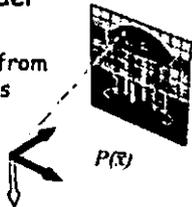


- ✓ Ignoring time, and selecting a discrete set of wavelengths, yields a 5 dimensional plenoptic function

Where to begin?

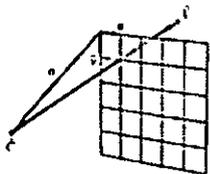
- ✓ Pinhole camera model

- Defines a mapping from image points to rays in space



From Rays to Points

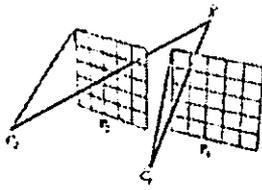
- ✓ Simple Derivation



$$P = \begin{bmatrix} u_x & v_x & o_x \\ u_y & v_y & o_y \\ u_z & v_z & o_z \end{bmatrix}$$

$$\vec{X} = \vec{C} + t P \vec{x}$$

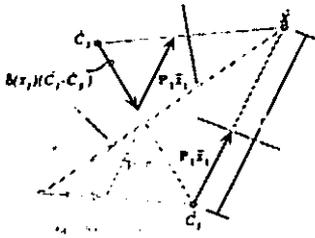
Correspondence



$$\begin{aligned} C_2 + l_2 P_2 \bar{x}_2 &= C_1 + l_1 P_1 \bar{x}_1 \\ l_2 P_2 \bar{x}_2 &= C_1 - C_2 + l_1 P_1 \bar{x}_1 \\ l_2 \bar{x}_2 &= P_2^{-1}(C_1 - C_2) + l_1 P_2^{-1} P_1 \bar{x}_1 \\ \frac{l_2}{l_1} \bar{x}_2 &= \frac{l_1}{l_2} P_2^{-1}(C_1 - C_2) + P_2^{-1} P_1 \bar{x}_1 \\ \bar{x}_2 &= \frac{l_1}{l_2} P_2^{-1}(C_1 - C_2) + \frac{P_2^{-1} P_1}{H_u} \bar{x}_1 \end{aligned}$$

Planar Warping Equation

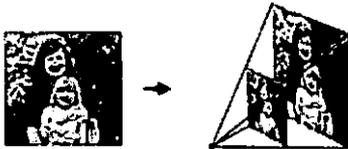
$$\bar{x}_2 = \delta(\bar{x}_1) P_2^{-1}(C_1 - C_2) + P_2^{-1} P_1 \bar{x}_1$$



Resulting Warping Function

✓ A perturbed planar warp ...

$$\bar{x}_2 = \delta \bar{x}_1 + H_{21} \bar{x}_1$$



Special Case

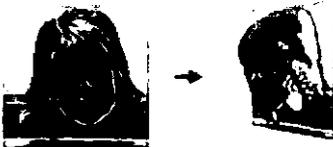
✓ A simple Planar warp

$$\bar{x}_2 = H_{21}\bar{x}_1$$



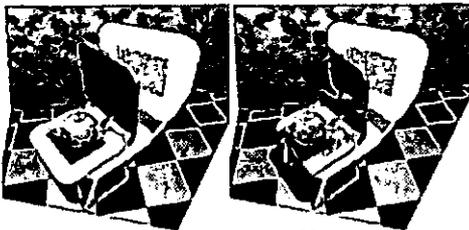
Warping in Action

✓ A 3D Warp



Visibility

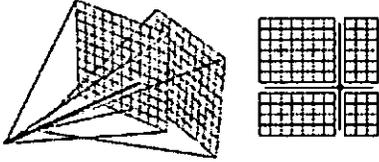
✓ The warping equation determines where points go...



... but that is not sufficient

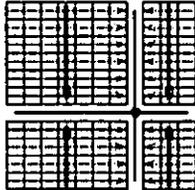
Partition Reference Image

- ✓ Project the *desired* center-of-projection onto the reference image



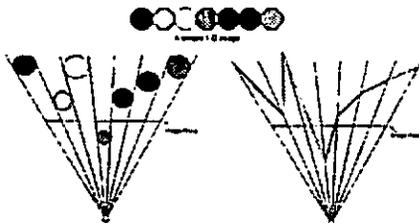
Enumeration

- ✓ Drawing toward the projected point guarantees an *occlusion compatible* ordering
- ✓ Ordering is consistent with a painter's algorithm
- ✓ Independent of the scene's contents
- ✓ Easily generalized to other viewing surfaces
- ✓ No auxiliary information required



Reconstruction

- ✓ Typical images are discrete, not continuous
- ✓ An image can be formed by different geometries



Gaussian Cloud Model



- ✓ Represents samples as Gaussian cloud densities
- ✓ Excessive exposure errors

Bilinear Patch Model



- ✓ Fits a bilinear patch through grid points in reference image
- ✓ Excessive occlusion errors

Comparison of Models

✓ Gaussian-Cloud Model

✓ Bilinear-Patch Model

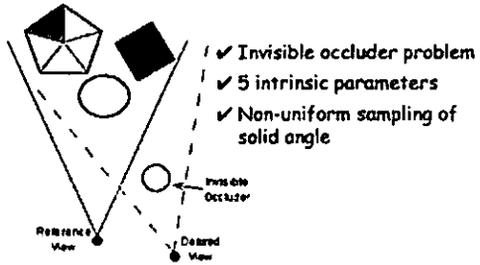


- Excessive exposure errors
- Pinhole problems
- Generally preferred



- Excessive occlusion errors
- Rasterization H/W
- Difficult to navigate

Problems with Planar Cameras



Panoramic Cameras



- ✓ Warping equation can be easily adapted
- ✓ Visibility algorithm works
- ✓ Nonlinear mapping functions

Examples

- ✓ Cylindrical camera



Redundant Structure



Comparing Approaches

✓ Geometry Based

- Forward Mapping (Graphics Pipeline)
- Inverse Mapping (Ray tracing)

✓ Image based

- Greater spatial coherence
- Lower depth complexity

Image-Based Pipeline

Geometry-Based Rendering Pipeline



Inverse Evaluation

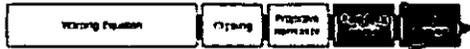
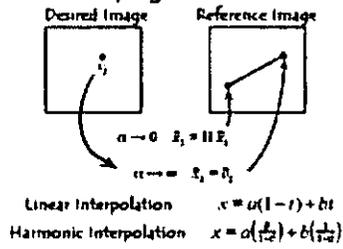


Image-Based Rendering Pipeline

Image-Based Ray Tracing

✓ Inverse warping

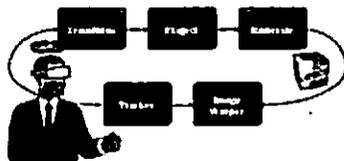


Algorithm Properties

- ✓ Search is confined to a line
- ✓ First intersection is closest point
- ✓ Incremental line drawing (DDA)
- ✓ Reconstruction occurs in reference image

Applications of IBR

- ✓ IBR combined with traditional methods
- ✓ Decouples rendering from tracking
- ✓ Latency compensation



Conclusions

✓ New representation for 3D graphics

- easy to acquire
- allows efficient rendering
- scalable performance
 - depends on number of pixels in desired image rather than the number of geometric primitives
- amenable to HW acceleration

Recovering Geometry I

SIGGRAPH Course on
Image-Based Modeling and Rendering

Richard Szeliski

Microsoft Research

szeliski@microsoft.com

July 20, 1998

Computer Vision

Computer Vision is the *inverse* of
Computer Graphics:

- computer graphics:
 - given a 3D model, render it
- computer vision
 - given some images, create a 3D model

This talk will describes some techniques for recovering 3D geometry from images.

Motivation

- model building for virtual reality, animation, and CAD is slow and tedious
- animators and designers want photo-realistic (texture-mapped) models
- video input, display, and processing hardware becoming ubiquitous (multimedia)
- computer vision algorithms becoming more mature and reliable

Applications

- recover camera location to superimpose graphics on image
- extract texture-maps from real world
- create a 3-D model object or world model, without extensive interactive modeling

Outline

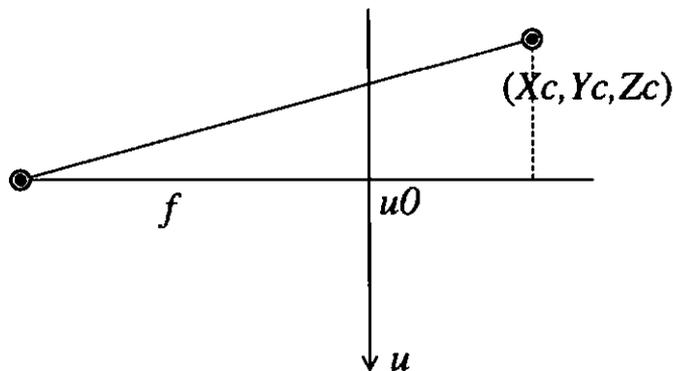
- camera calibration
- pose estimation (view correlation)
- triangulation
- structure from motion
- feature matching (correlation)
- stereo matching (dense shape estimates)
- volumes (octrees) from silhouettes
- surface curves from profiles
- inverse texture mapping
- applications

Camera calibration

- determine camera *internal* (focal length) and *external* (pose) parameters, either separately or simultaneously, from known 3D points
- forward projection equations

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = [\mathbf{R}]_{3 \times 3} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \mathbf{t}$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} U \\ V \\ W \end{bmatrix} = \begin{bmatrix} f & 0 & u_c \\ 0 & f & v_c \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$



Camera matrix calibration

- directly estimate 11 unknowns in 3×4 matrix projecting 3D \rightarrow 2D

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \sim \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- solve set of linear equations

$$\begin{aligned} u_i(m_{20}X_i + m_{21}Y_i + m_{22}Z_i + 1) &= \\ & m_{00}X_i + m_{01}Y_i + m_{02}Z_i + m_{03} \\ v_i(m_{20}X_i + m_{21}Y_i + m_{22}Z_i + 1) &= \\ & m_{10}X_i + m_{11}Y_i + m_{12}Z_i + m_{13} \end{aligned}$$

Camera matrix calibration (con't)

Advantages:

- very simple to formulate and solve

Disadvantages:

- doesn't compute internal parameters
- more unknowns than true degrees of freedom
- need a separate camera matrix for each new view

Pose estimation

- once the internal camera parameters are known, can compute camera pose (position and orientation) from known markers
- application: superimpose 3D graphics onto video

possible solution techniques:

- use standard calibration code [Tsa87]
- use *view correlation* [Bog91]
- use *through the lens camera control* [GW92]
- other techniques from computer vision textbooks

Triangulation (Stereo)

- given some points in *correspondence* across two or more images (taken from calibrated cameras), $\{(u_j, v_j)\}$, compute the 3D location \mathbf{X}

Method I: intersect viewing rays in 3D

- minimize

$$\arg \min_{\mathbf{X}} \sum_j \min_{s_j} \|\mathbf{C}_j + s_j \mathbf{V}_j - \mathbf{X}\|^2$$

where \mathbf{X} is the unknown 3D point, \mathbf{C}_j is the optical center of camera j , \mathbf{V}_j is the *viewing ray* corresponding to pixel (u_j, v_j) , and s_j is unknown distance along \mathbf{V}_j .

- advantage: geometrically intuitive

Triangulation (con't)

Method II: solve linear equations in \mathbf{X}

$$\begin{aligned}u_j(m_{20}^{(j)}X + m_{21}^{(j)}Y + m_{22}^{(j)}Z + 1) &= \\ & m_{00}^{(j)}X + m_{01}^{(j)}Y + m_{02}^{(j)}Z + m_{03}^{(j)} \\ v_j(m_{20}^{(j)}X + m_{21}^{(j)}Y + m_{22}^{(j)}Z + 1) &= \\ & m_{10}^{(j)}X + m_{11}^{(j)}Y + m_{12}^{(j)}Z + m_{13}^{(j)}\end{aligned}$$

- advantage: very simple

Method III: non-linear minimization

- minimize $\sum_j (u_j - \hat{u}_j)^2 + (v_j - \hat{v}_j)^2$ where

$$\begin{aligned}\hat{u}_j &= \frac{m_{00}^{(j)}X + m_{01}^{(j)}Y + m_{02}^{(j)}Z + m_{03}^{(j)}}{m_{20}^{(j)}X + m_{21}^{(j)}Y + m_{22}^{(j)}Z + 1} \\ \hat{v}_j &= \frac{m_{10}^{(j)}X + m_{11}^{(j)}Y + m_{12}^{(j)}Z + m_{13}^{(j)}}{m_{20}^{(j)}X + m_{21}^{(j)}Y + m_{22}^{(j)}Z + 1}\end{aligned}$$

- advantage: most accurate (image plane error)

Structure from motion

- given many points in *correspondence* across several images, $\{(u_{ij}, v_{ij})\}$, simultaneously compute the 3D location \mathbf{X}_i and camera (or *motion*) parameters \mathbf{M}_j
- two main variants: calibrated, and uncalibrated (sometimes associated with Euclidean and projective reconstructions)
- long history of research algorithms [LH81, TK92, WHA93, SK94, BTZ96]

Structure from motion (con't)

- simple iterative algorithm used for face reconstruction [P⁺98] assuming roughly known geometry and pose
- assume $(u_c, v_c) = (0, 0)$, but f is unknown

$$u_{ij} = s_j \frac{\mathbf{r}_j^x \cdot \mathbf{X}_i + t_j^x}{1 + \eta_j \mathbf{r}_j^z \cdot \mathbf{X}_i}$$

$$v_{ij} = s_j \frac{\mathbf{r}_j^y \cdot \mathbf{X}_i + t_j^y}{1 + \eta_j \mathbf{r}_j^z \cdot \mathbf{X}_i}$$

where $\eta_j = 1/t_j^z$ is the inverse distance to object, and $s_j = f_j/t_j^z$ is a world-pixel scale factor

- *advantage*: works well for narrow fields of view when f and t_j^z are hard to estimate

Structure from motion (con't)

- bring denominator over to l.h.s.

$$u_{ij} + u_{ij}\eta_j(\mathbf{r}_j^z \cdot \mathbf{X}_i + t_i^z) - s_j(\mathbf{r}_j^x \cdot \mathbf{X}_i + t_j^x) = 0$$

$$v_{ij} + v_{ij}\eta_j(\mathbf{r}_j^z \cdot \mathbf{X}_i + t_i^z) - s_j(\mathbf{r}_j^y \cdot \mathbf{X}_i + t_j^y) = 0$$

- iteratively solve for

1. s_j

2. \mathbf{X}_i

3. \mathbf{R}_j

4. t_j^x and t_j^y

5. η_j

- all equations are linear, except for \mathbf{R}_j , which is linearized by using a small angle (instantaneous velocity) approximation

$$\mathbf{R}'_j \approx \begin{bmatrix} 1 & -\omega_z & \omega_y \\ \omega_z & 1 & -\omega_x \\ -\omega_y & \omega_x & 1 \end{bmatrix} \mathbf{R}_j$$

Structure from motion (example)

- automatically track points in video sequence, validate consistent matches, and build 3D structure from point tracks [BTZ96]
- uses both points and lines for reconstruction
- final output is texture-mapped model

Structure from motion: limitations

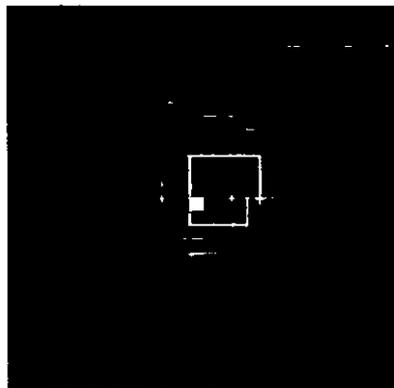
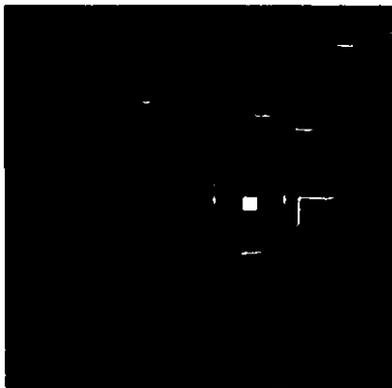
- very difficult to reliably estimate structure and motion unless:
 - large (x or y) rotation *or*
 - large field of view and depth variation
- camera calibration important for Euclidean reconstructions
- need good feature trackers
- postprocessing: what to do with the resulting 3-D points?

Feature matching (correlation)

- find corresponding points in image video sequence
- one simple technique: find two patches with minimal summed squared error [Ana89]

$$E_{ij}(u, v) = \sum_{k=x-w}^{x+w} \sum_{l=y-w}^{y+w} [I_1(k+u, l+v) - I_0(k, l)]^2$$

where (u, v) is *displacement*



- can speed up computation using Fourier transforms

Feature matching (optic flow)

- need *sub-pixel* precision to get best registration
- solution: Taylor series expansion of image function [LK81]

$$\begin{aligned} E(\mathbf{u} + \delta\mathbf{u}) &= \sum_{(\mathbf{x})} [I_1(\mathbf{x}') + \nabla I_1(\mathbf{x}') \cdot \delta\mathbf{u} - I_0(\mathbf{x})]^2 \\ &= \sum_i (e_i + \mathbf{g}_i \cdot \delta\mathbf{u})^2 \end{aligned}$$

where $\mathbf{x}' = \mathbf{x} + \mathbf{u}$, $e_i = I_1(\mathbf{x}') - I_0(\mathbf{x})$,
 $\mathbf{g}_i = \nabla I_1(\mathbf{x}')$

- solve 2×2 system

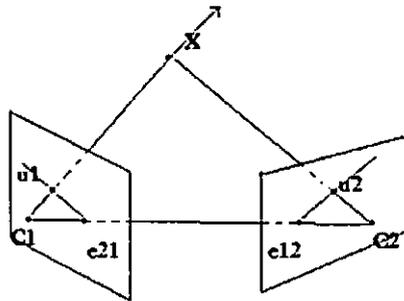
$$\left[\sum_i \mathbf{g}_i \mathbf{g}_i^T \right] \delta\mathbf{u} = - \left[\sum_i e_i \mathbf{g}_i \right]$$

- use a *coarse-to-fine* pyramid to speed up search [BAHH92]
- related to *Brightness Constancy Equation* [HS81]

$$I_x u + I_y v - I_t = 0$$

Stereo: epipolar geometry

- for any pixel in one image, can find a corresponding *epipolar line* in any other image
 - simply connect the projection of the camera center C_j and viewing ray V_j



- for *two* images (or collection of images with collinear centers of projection), can find corresponding epipolar lines
- family of epipolar lines is the projection of the *pencil* of planes passing through the centers of projection
- **rectification**: warping the input images (perspective transformation) so that epipolar lines are horizontal [Fau93, p. 188]

Stereo: extracting dense depth

- apply feature matching criterion at *all* pixel simultaneously
- search only over epipolar lines (many fewer candidate positions)



- can also match features such as lines
- recovered depth map can be used for *view interpolation* [CW93, SD96]
- *weakness*: can only match in textured areas; what to do when part is *occluded* in other image(s)?

Stereo matching: limitations

- too many errors
- problems at and near occlusions
- incorrect color extraction
- no partial occupancy in (mixed) border pixels

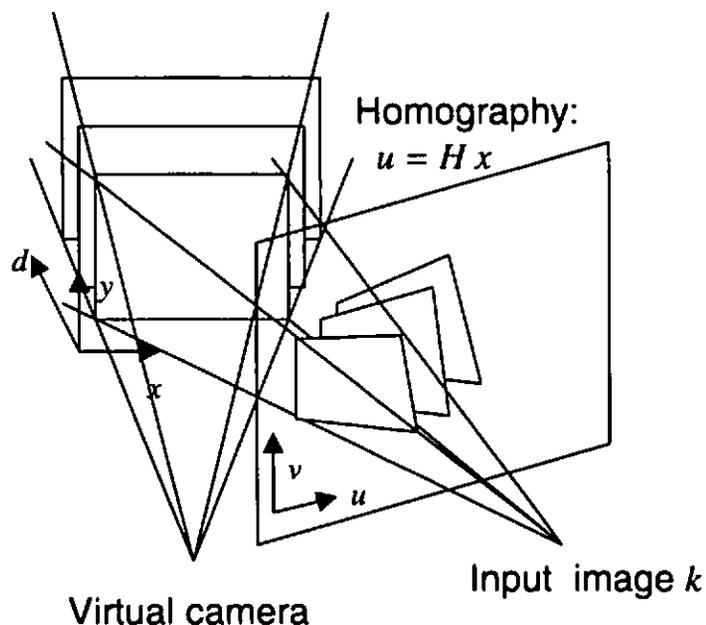


- solution: simultaneously recover *disparities*, *colors*, and *opacities*

Generalized disparity space

Projective mapping (*collineation*) of 3-D space

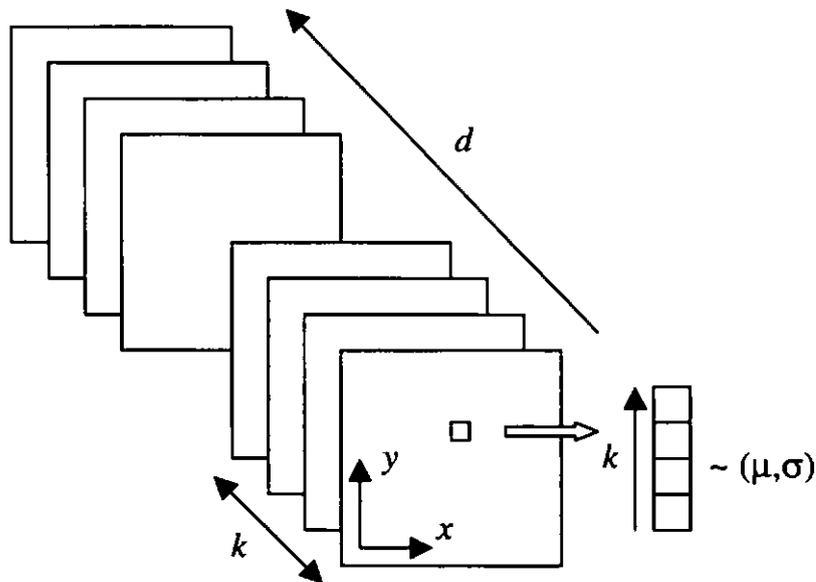
$$(x, y, d, 1) \sim \hat{M}_0(X, Y, Z, 1)$$



- *virtual camera* can be one of the inputs, or at arbitrary location (e.g., ∞)
- *disparity* spacing can be uniform or projective (e.g., $1/Z$)

Generalized disparity space (con't)

Collect all K (warped) images into “stack” in disparity space:



- variance across k for a given pixel $(x, y, d) \Rightarrow$ confidence in match
- *true multi-image* matching (space sweep [Col96])

Local evidence aggregation

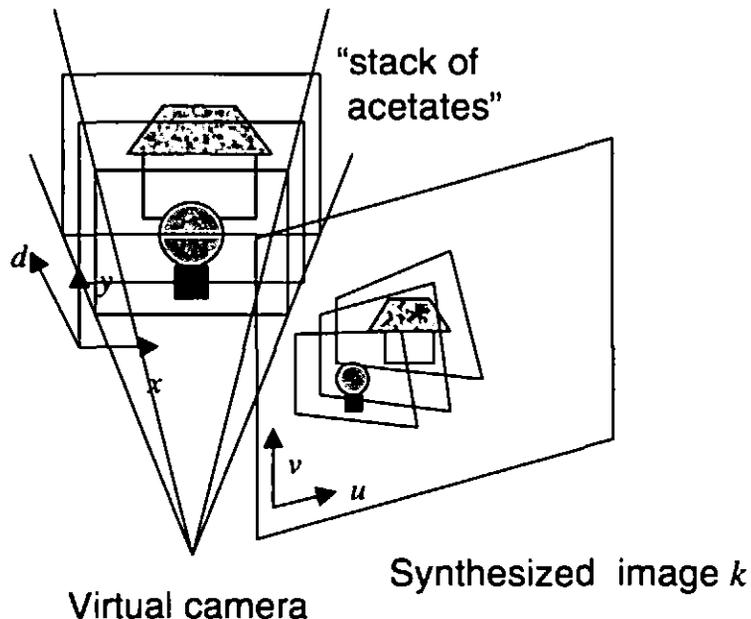
1. warp all input images to all disparities
2. compute (μ, σ) at each (x, y, d)
3. accumulate evidence through (possibly non-linear) smoothing and/or diffusion
4. pick clear winner in each column

Problems:

- misses some partially occluded surfaces
- makes errors in partially occluded areas
- no modeling of translucent (mixed) border pixels

Iterative re-projection

“Stack of acetates” model [SG98]



- warp each layer back into input image
- composite back-to-front using *over* operator [PD84]

$$f \wedge b \equiv f + (1 - \alpha_f)b,$$

f and b are premultiplied colors, α_f is opacity

Iterative re-projection (con't)

- composite (re-synthesize) image:

$$\begin{aligned}\tilde{\mathbf{c}}_k(u, v) &= \bigwedge_{d=d_{\max}}^{d_{\min}} \tilde{\mathbf{c}}_k(u, v, d) \\ &= \tilde{\mathbf{c}}_k(u, v, d_{\max}) \wedge \cdots \wedge \tilde{\mathbf{c}}_k(u, v, d_{\min})\end{aligned}$$

- visibility map for each disparity

$$\begin{aligned}V_k(u, v, d - 1) &= V_k(u, v, d) (1 - \tilde{\alpha}_k(u, v, d)) \\ &= \prod_{d'=d}^{d_{\max}} (1 - \tilde{\alpha}_k(u, v, d')),\end{aligned}$$

$$\tilde{\mathbf{c}}_k(u, v) = \sum_{d=d_{\min}}^{d_{\max}} \tilde{\mathbf{c}}_k(u, v, d) V_k(u, v, d).$$

- re-compute (μ, σ) from *visible* colors

$$\mathbf{c}_k(u, v, d) = \mathbf{c}_k(u, v) V_k(u, v, d)$$

- find new winners and iterate

Layered Stereo

- generalize stack of acetates model to use (fewer) oriented planes [BSA98]

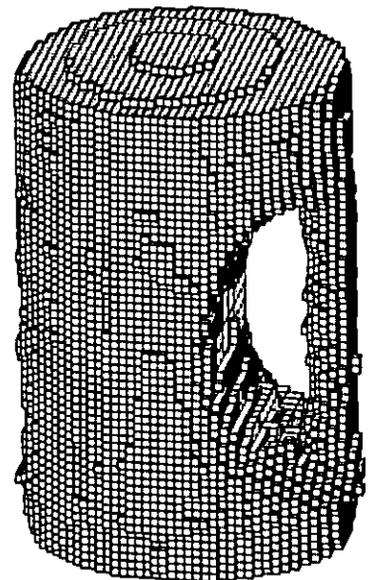
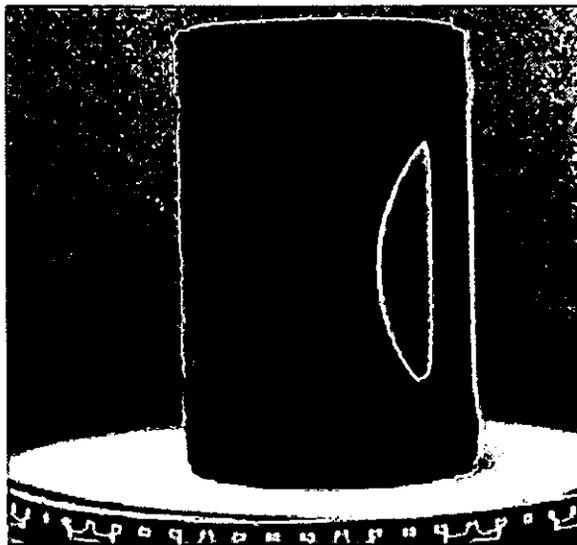


- compute out-of-plane residual depth



Volumes from silhouettes

- extract binary *silhouette* of object photographed against known background
- each silhouette + camera center defines conic region of space which must enclose object
- intersection of cones gives bounding volume



- use octree representation of volume for efficiency [Sze93]

Volumes from silhouettes

Advantages:

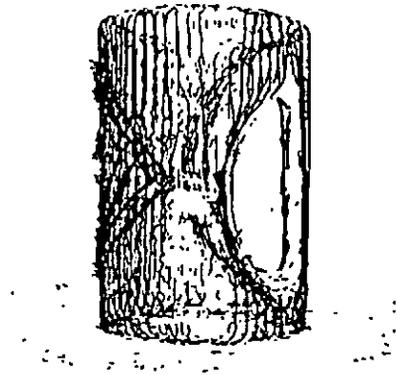
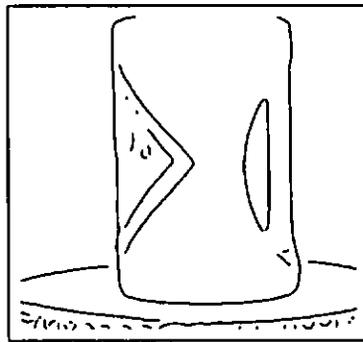
- simple to implement, fairly robust
- fast execution
- complete (closed) surface

Disadvantages:

- only produces *line hull*
- limited resolution
- sensitive to classification (thresholding)

Surface curves from profiles

- extract and link edges in each image
- match edges across image sequence



- infer 3-D location from 2 or more matched edges:
 - for *stationary edge* (surface marking, sharp crease), use regular triangulation
 - for smooth self-occluding *profile* (limb), use 3 or more edges, fit circular arc [SW94]

Surface curves from profiles

Advantages:

- correct estimates at occluding contours
- good for smoothly curved objects
- provides intrinsic surface estimates, piecewise continuous surface mesh
- works on interior surface markings

Disadvantages:

- fails in highly textured regions
- fails in textureless *interior* areas
- incomplete surface (not closed)

Inverse texture mapping (photometry)

- recover color distribution over shape
- undo shading effects:
 - diffuse illumination
 - single source Lambertian
- weight contribution by surface normal
- smooth (and sharpen) results

Application: 3D face model building

- take several photos of a face from different views
- identify key points (eye and mouth corners, nose tip, ...) in each image
- recover camera position and coarse geometry using structure from motion
- add more correspondences, refine geometry, and interpolate to the rest of the mesh (scattered data interpolation)
- recover texture map (cylindrical coordinates)
- refine shape estimates using stereo
- animate by *morphing* between expressions

[P⁺98]

Applications

- industrial applications
 - CAD/CAM
 - “3D Fax” : collaborative design
 - architecture
 - biomedical (surgery, prostheses)
 - special effects (FX), virtual studio
 - fashion & clothing
- consumer applications:
 - 3D world building (travel, home sales, home page, ...)
 - 3D model construction (art, hobby, ..)
 - 3D avatar construction (heads)
 - “3D videophone”

To find out more

- general references on computer vision:
[BB82, Hor86, Fau93, NaI93]
- recent survey of (some) 3D modeling techniques [Sze97]
- Computer Vision Home Page:
<http://www.cs.cmu.edu/afs/cs/project/cil/ftp/html/vision.html>
- Workshop on Image-Based Modeling and Rendering:
<http://graphics.stanford.edu/workshops/ibr98/>

References

- [Ana89] P. Anandan. A computational framework and an algorithm for the measurement of visual motion. *International Journal of Computer Vision*, 2(3):283–310, January 1989.
- [BAHH92] J. R. Bergen, P. Anandan, K. J. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *Second European Conference on Computer Vision (ECCV'92)*, pages 237–252, Santa Margherita Liguere, Italy, May 1992. Springer-Verlag.
- [BB82] D. H. Ballard and C. M. Brown. *Computer Vision*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.
- [Bog91] R. G. Bogart. View correlation. In J. Arvo, editor, *Graphics Gems II*, page 181190. Academic Press, Boston, 1991.
- [BSA98] S. Baker, R. Szeliski, and P. Anandan. A layered approach to stereo reconstruction. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'98)*, Santa Barbara, June 1998.
- [BTZ96] P. Beardsley, P. Torr, and A. Zisserman. 3D model acquisition from extended image sequences. In *Fourth European Conference on Computer Vision (ECCV'96)*, volume 2, pages 683–695, Cambridge, England, April 1996. Springer-Verlag.

- [Col96] R. T. Collins. A space-sweep approach to true multi-image matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 358–363, San Francisco, California, June 1996.
- [CW93] S. Chen and L. Williams. View interpolation for image synthesis. *Computer Graphics (SIGGRAPH'93)*, pages 279–288, August 1993.
- [Fau93] O. Faugeras. *Three-dimensional computer vision: A geometric viewpoint*. MIT Press, Cambridge, Massachusetts, 1993.
- [GW92] M. Gleicher and A. Witkin. Through-the-lens camera control. *Computer Graphics (SIGGRAPH'92)*, 26(2):331–340, July 1992.
- [Hor86] B. K. P. Horn. *Robot Vision*. MIT Press, Cambridge, Massachusetts, 1986.
- [HS81] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, 1981.
- [LH81] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, 1981.
- [LK81] B. D. Lucas and T. Kanade. An iterative image registration technique with an application in stereo vision. In *Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, pages 674–679, Vancouver, 1981.

- [NaI93] V. S. Nalwa. *A Guided Tour of Computer Vision*. Addison-Wesley, Reading, MA, 1993.
- [P⁺98] F. Pighin et al. Modeling and animating realistic facial expressions from photographs. In *Computer Graphics (SIGGRAPH'98) Proceedings*, Orlando, July 1998. ACM SIGGRAPH.
- [PD84] T. Porter and T. Duff. Compositing digital images. *Computer Graphics (SIGGRAPH'84)*, 18(3):253–259, July 1984.
- [SD96] S. M. Seitz and C. M. Dyer. View morphing. In *Computer Graphics Proceedings, Annual Conference Series*, pages 21–30, Proc. SIGGRAPH'96 (New Orleans), August 1996. ACM SIGGRAPH.
- [SG98] R. Szeliski and P. Golland. Stereo matching with transparency and matting. In *Sixth International Conference on Computer Vision (ICCV'98)*, pages 517–524, Bombay, January 1998.
- [SK94] R. Szeliski and S. B. Kang. Recovering 3D shape and motion from image streams using nonlinear least squares. *Journal of Visual Communication and Image Representation*, 5(1):10–28, March 1994.
- [SW94] R. Szeliski and R. Weiss. Robust shape recovery from occluding contours using a linear smoother. In C. M. Brown and D. Terzopoulos, editors, *Real-time Computer Vision*, pages 141–165. Cambridge University Press, Cambridge, England, 1994.

- [Sze93] R. Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Understanding*, 58(1):23–32, July 1993.
- [Sze97] R. Szeliski. From images to models (and beyond): a personal retrospective. In *Vision Interface '97*, pages 126–137, Kelowna, British Columbia, May 1997. Canadian Image Processing and Pattern Recognition Society.
- [TK92] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: A factorization method. *International Journal of Computer Vision*, 9(2):137–154, November 1992.
- [Tsa87] R. Y. Tsai. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal of Robotics and Automation*, RA-3(4):323–344, August 1987.
- [WHA93] J. Weng, T. S. Huang, and N. Ahuja. *Motion and Structure from Image Sequences*. Springer-Verlag, Berlin, 1993.

Non-parametric Local Transforms for Computing Visual Correspondence

Ramin Zabih¹ and John Woodfill²

¹ Computer Science Department, Cornell University, Ithaca NY 14853-7501, USA

² Interval Research Corporation, 1801-C Page Mill Road, Palo Alto CA 94304, USA

Abstract. We propose a new approach to the correspondence problem that makes use of non-parametric local transforms as the basis for correlation. Non-parametric local transforms rely on the relative ordering of local intensity values, and not on the intensity values themselves. Correlation using such transforms can tolerate a significant number of outliers. This can result in improved performance near object boundaries when compared with conventional methods such as normalized correlation. We introduce two non-parametric local transforms: the *rank transform*, which measures local intensity, and the *census transform*, which summarizes local image structure. We describe some properties of these transforms, and demonstrate their utility on both synthetic and real data.

1 Introduction

The correspondence problem is a fundamental problem in vision, as it forms the basis for stereo depth computation and most optical flow algorithms. Given two images of the same scene, a pixel in one image corresponds to a pixel in the other if both pixels are projections along lines of sight of the same physical scene element. If the two images are temporally consecutive, then computing correspondence determines motion. If the two images are spatially separated but simultaneous, then computing correspondence determines stereo depth. *Area-based* approaches to the correspondence problem [4] find a dense solution, usually by relying on some kind of statistical correlation between local intensity regions.

In this paper we propose a new area-based approach to the correspondence problem, based on non-parametric local transforms followed by correlation. We begin by motivating our approach, then show how non-parametric local transforms can be used to determine correspondence. In section 3 we introduce the *rank* and *census* transforms, and describe their properties. We give empirical evidence of the performance of our methods in section 4, using both natural and synthetic images. Finally, in section 5 we survey related work and discuss some planned extensions.

2 Non-parametric local transforms

Our approach to the correspondence problem is first to apply a local transform to the image, and then to use correlation. In this respect, our work is similar

to that of Nishihara [12] and Seitz [14, 1]. Nishihara's transform is the sign bit of the image after convolution with a Laplacian, while Seitz's transform is the direction of the intensity gradient.

Most approaches to the correspondence problem have difficulty near discontinuities in disparity, which occur at the boundaries of objects. Near such a boundary, the pixels in a local region represent scene elements from two distinct intensity populations. Some of the pixels come from the object, and some from other parts of the scene. As a result, the local pixel distribution will in general be multimodal near a boundary. This poses a problem for many correspondence algorithms, such as normalized correlation [6].

Correspondence algorithms are usually based on standard statistical methods, which are best suited to a single population. Parametric measures, such as the mean or variance, do not behave well in the presence of distinct subpopulations, each with its own coherent parameters. This problem, which we will refer to as *factionalism*, is a major issue in computer vision, and has been addressed with a variety of methods, including robust statistics [2, 3], Markov Random Fields [5] and regularization [13].

The fundamental idea behind our approach is to define a local image transform that tolerates factionalism. Correspondence can be computed by transforming both images and then using correlation. For this approach to succeed, the transform must result in significant local variation within a given image; in addition, it must give similar results near corresponding points between the two images. (Marr and Nishihara [10] refer to these two properties as *sensitivity* and *stability*.) Finally, to handle stereo imagery, the transform should be invariant under changes in image gain and bias.

Our approach relies on local transforms based on non-parametric measures that are designed to tolerate factionalism. Non-parametric statistics [9] is distinguished by the use of ordering information among data, rather than the data values themselves. Non-parametric local transforms, which we introduced in [15], are local image transformations that rely on the relative ordering of intensities, and not on the intensity values themselves.

3 The rank transform and the census transform

We next describe two non-parametric local transforms. The first, called the *rank transform*, is a non-parametric measure of local intensity. The second, called the *census transform*, is a non-parametric summary of local spatial structure.

Let P be a pixel, $I(P)$ its intensity (usually an 8-bit integer), and $N(P)$ the set of pixels in some square neighborhood of diameter d surrounding P . All non-parametric transforms depend upon the comparative intensities of P versus the pixels in the neighborhood $N(P)$. The transforms we will discuss only depend on the sign of the comparison. Define $\xi(P, P')$ to be 1 if $I(P') < I(P)$ and 0 otherwise. The non-parametric local transforms depend solely on the set of pixel

comparisons, which is the set of ordered pairs

$$\Xi(P) = \bigcup_{P' \in N(P)} (P', \xi(P, P')).$$

They differ in terms of their exact reliance on Ξ .

The first non-parametric local transform is called the *rank transform*, and is defined as the number of pixels in the local region whose intensity is less than the intensity of the center pixel. Formally, the rank transform $R(P)$ is

$$R(P) = \|\{P' \in N(P) \mid I(P') < I(P)\}\|.$$

Note that $R(P)$ is not an intensity at all, but rather an integer in the range $\{0, \dots, d^2 - 1\}$. This distinguishes the rank transform from other attempts to use non-parametric measures such as median filters, mode filters or rank filters [7]. To compute correspondence, we have used L_1 correlation (minimizing the sum of absolute values of differences) on the rank-transformed images.

The second non-parametric transform is named the *census transform*. $R_r(P)$ maps the local neighborhood surrounding a pixel P to a bit string representing the set of neighboring pixels whose intensity is less than that of P . Let $N(P) = P \oplus D$, where \oplus is the Minkowski sum and D is a set of displacements, and let \otimes denote concatenation. The census transform can then be specified,

$$R_r(P) = \bigotimes_{[i,j] \in D} \xi(P, P + [i, j]).$$

Two pixels of census transformed images are compared for similarity using the Hamming distance, i.e. the number of bits that differ in the two bit strings. To compute correspondence, we have minimized the Hamming distance after applying the census transform.

These local transforms rely solely upon the set of comparisons Ξ , and are therefore invariant under changes in gain or bias. The tolerance of these transforms for factionalism also results from their reliance upon Ξ . If a minority of pixels in a local neighborhood has a very different intensity distribution than the majority, only comparisons involving a member of the minority are affected. Such pixels do not make a contribution proportional to their intensity, but proportional to their number. This limited dependence on the minority's intensity values is a major distinction between our approach and parametric measures.

To illustrate the manner in which these transforms tolerate factionalism, consider a three-by-three region of an image whose intensities are

```

127 127 129
126 128 129
127 131 A

```

for some value $0 \leq A < 256$. Consider the effect on various parametric and non-parametric measures, computed at the center of this region, as A varies over its

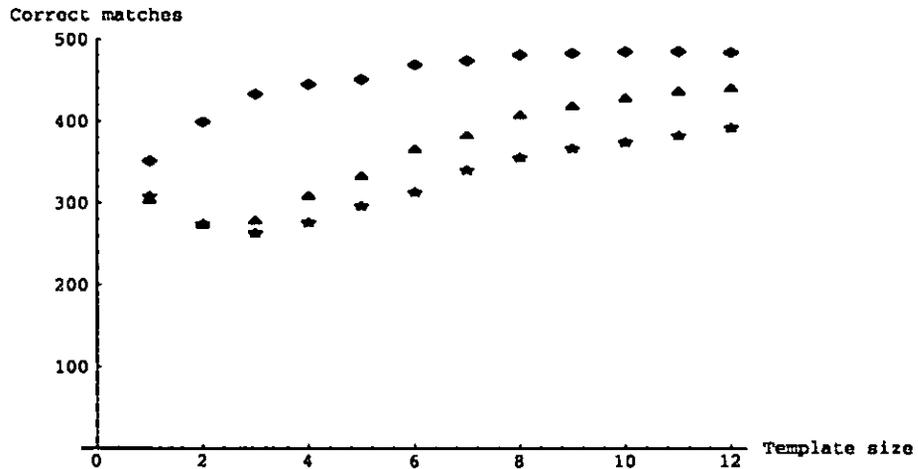


Fig. 1. Comparison of rank (♦), normalized (Δ) and SSD (+) correlation on Aschwanden data-set with salt-and-pepper noise

256 possible values. The mean³ of this region varies from 114 to 142, while the variance ranges from 2 to 1823. These parametric measures exhibit continuous variation over a substantial range as A changes.

Non-parametric transforms are more stable, however. All the elements of Ξ except one will remain fixed as A changes. Ξ will be

$$\begin{matrix} 1 & 1 & 0 \\ 1 & 0 & \\ 1 & 0 & a \end{matrix}$$

where a is 1 if $A < 128$, and otherwise 0. The census transform simply results in the bits of Ξ in some canonical ordering, such as $\{1, 1, 0, 1, 0, 1, 0, a\}$. The rank transform will give 5 if $A < 128$, and otherwise 4.

This comparison shows the tolerance that non-parametric measures have for factionalism. A minority of pixels can have a very different value, but the effect on the rank and census transforms is limited by the size of the minority.

4 Empirical results

We have implemented these non-parametric local transforms, and have explored their behavior on both real and synthetic imagery. The motivation for our approach was to obtain better results near the edges of objects. We have obtained comparative results on synthetic data which show that our methods can outperform normalized correlation.

In [1], Aschwanden and Guggenbühl have described the performance of a number of area-based stereo algorithms under several different noise models.

³ For convenience, we are rounding the actual values

Figure 1 compares correlation with the rank transform against two standard stereo algorithms, namely normalized correlation and sum of squared differences (SSD) correlation. Performance is measured as function of template radius, as described in [1].

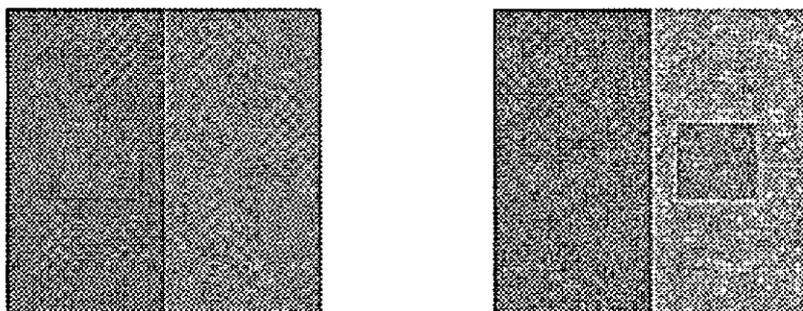


Fig. 2. Right and left random-dot stereograms

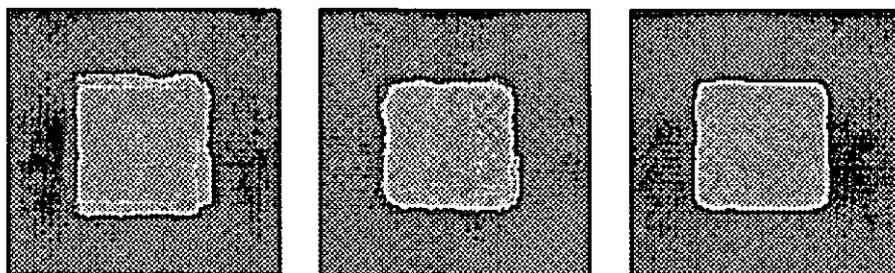


Fig. 3. Disparities from normalized correlation, rank and census transforms

Another way to compare correlation methods is with random dot imagery. Figure 2 shows a random dot stereogram of a square floating in front of a flat surface, on which there is a vertical intensity edge. The images are noise-free, but the intensities differ by fixed gain and bias.

Figure 3 shows the disparities computed from normalized correlation and from correlation with the rank and census transforms. There should only be 2 disparities in this scene: one for the background surface (which is at disparity 0), and one for the foreground square (which is at disparity 104). Notice the comparatively poor performance of normalized correlation near the edges, where it introduces spurious disparities. The performance of our approach can be seen by counting the pixels with incorrect disparities, as shown below.

Algorithm	Incorrect matches
Normalized	1385
Rank transform	609
Census transform	407

On this example, the non-parametric local transforms appear to exhibit better performance than normalized correlation.

The best evidence in favor of the non-parametric local transforms is their performance on real images. We have used the rank transform and the census transform on a number of different images to obtain stereo depth. Depth maps are shown with lighter shades indicating larger disparities and thus nearer scene elements. All the depth maps shown were generated with the same parameters (a transform radius of 7 pixels, and a correlation radius of 4 pixels).

Figure 4 shows a beam-splitter image of a puppet (Elmo from the television show "Sesame Street"). The depth results of the non-parametric local transforms are shown in figure 5. Figure 6 shows an image from a tree sequence⁴ captured by moving a camera along a rail, and the depth results from the transforms.

5 Related work and planned extensions

The algorithms we describe are related to non-parametric measures of association, such as Spearman's correlation coefficient r , or Kendall's τ . These are measures of association of paired data that are based upon comparisons. However, such measures are very expensive to compute, and do not capture the spatial structure of images.

Probably the most similar approach to ours is the work based on robust statistics [2, 11, 3]. Robust statistics differs from our approach in that they emphasize reducing the influence of outliers. Implicit in this work is the assumption that outliers are distributed randomly. However, at the edges of objects, factionalism produces outliers with consistent distributions. Our approach tolerates outliers with consistent distributions, and does not allow pixels from a small faction to contribute in a manner proportional to their intensity.

One limitation of the non-parametric transforms we have described is that the amount of information they associate with a pixel is not very large. We hope to address this shortcoming by combining a number of different non-parametric transforms into a vector of measures associated with a pixel. Ultimately, we would like to avoid the correlation phase altogether and simply match pixels according to a set of semi-independent measures, in a manner similar to that proposed by Kass [8].

Another limitation of our approach is that the local measures rely heavily upon the intensity of the center pixel. This has not been an issue in practice, but we propose to address it by doing comparisons from a local median intensity instead of $I(P)$. An additional idea we intend to pursue is to generalize Ξ , which currently uses the sign of the intensity differences. We plan to explore using higher-order differences, as well as the information contained in the total ordering of the local pixel intensities.

We are also interested in efficient algorithms for implementing such transforms. [15] describes a number of fast algorithms for computing the rank trans-

⁴ The tree imagery appears courtesy of Harlyn Baker and Bob Bolles

form based on dynamic programming. We have recently implemented an approximation of the census transform on a Sun workstation, which produces stereo depth with 24 disparities on 640 by 240 images at 1–2 frames per second.

Acknowledgements

Portions of this work were done while the first author was at the Computer Science Department at Stanford University, supported by a fellowship from the Fannie and John Hertz Foundation. We wish to thank SRI for the use of their Connection Machine.

References

1. P. Aschwanden and W. Guggenbühl. Experimental results from a comparative study on correlation-type registration algorithms. In Förstner and Ruwedel, editors, *Robust Computer Vision*, pages 268–289. Wichmann, 1993.
2. Paul Besl, Jeffrey Birch, and Layne Watson. Robust window operators. In *International Conference on Computer Vision*, pages 591–600, 1988.
3. Michael Black and P Anandan. A framework for the robust estimation of optical flow. In *International Conference on Computer Vision*, pages 231–236, 1993.
4. U. Dhond and J. Aggarwal. Structure from stereo — a review. *IEEE Transactions on Systems, Man and Cybernetics*, 19(6), 1989.
5. Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE PAMI*, 6:721–741, 1984.
6. Marsha Jo Hanna. *Computer Matching of Areas in Stereo Images*. PhD thesis, Stanford, 1974.
7. R. Hodgson, D. Bailey, M. Naylor, A. Ng, and S. McNeill. Properties, implementations and applications of rank filters. *Journal of Image and Vision Computing*, 3(1):3–14, February 1985.
8. Michael Kass. Computing visual correspondence. *DARPA Image Understanding Proceedings*, pages 54–60, 1983.
9. E. L. Lehman. *Nonparametrics: statistical methods based on ranks*. Holden-Day, 1975.
10. David Marr and Keith Nishihara. Representation and recognition of the spatial organization of three-dimensional shapes. *Proceedings of the Royal Society of London B*, 200:269–294, 1978.
11. Peter Meer, Doron Mintz, Azriel Rosenfeld, and Dong Yoon Kim. Robust regression methods for computer vision: A review. *International Journal of Computer Vision*, 6(1):59–70, 1991.
12. H. Keith Nishihara. Practical real-time imaging stereo matcher. *Optical Engineering*, 23(5):536–545, Sept–Oct 1984.
13. Tomaso Poggio, Vincent Torre, and Christof Koch. Computational vision and regularization theory. *Nature*, 317:314–319, 1985.
14. Peter Seitz. Using local orientational information as image primitive for robust object recognition. *SPIE proceedings*, 1199:1630–1639, 1989.
15. Ramin Zabih. *Individuating Unknown Objects by Combining Motion and Stereo*. PhD thesis, Stanford University, 1994 (forthcoming).

This article was processed using the \LaTeX macro package with LLNCS style

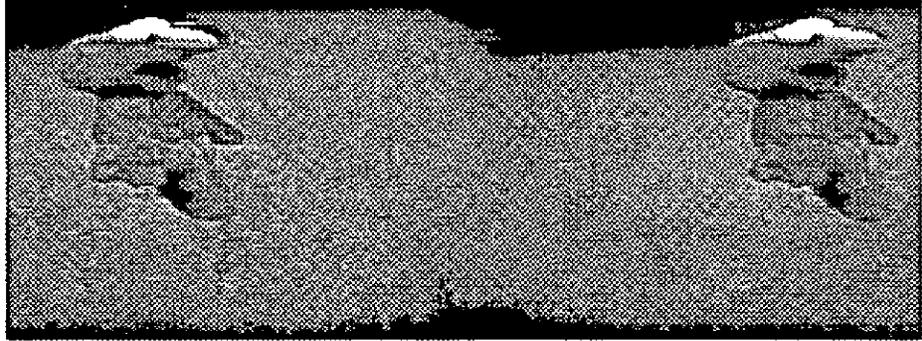


Fig. 4. Elmo stereo pair from beam-splitter

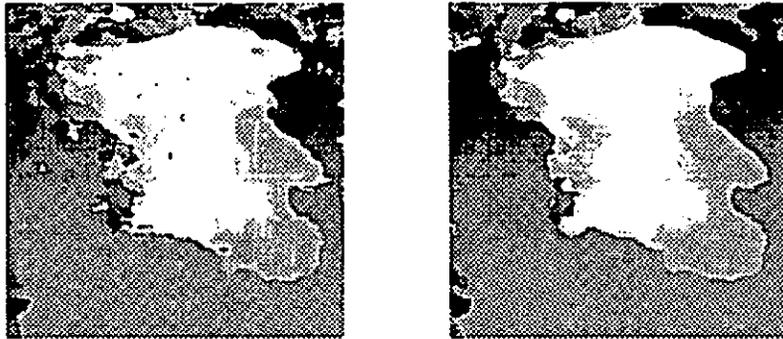


Fig. 5. Rank and census results on Elmo

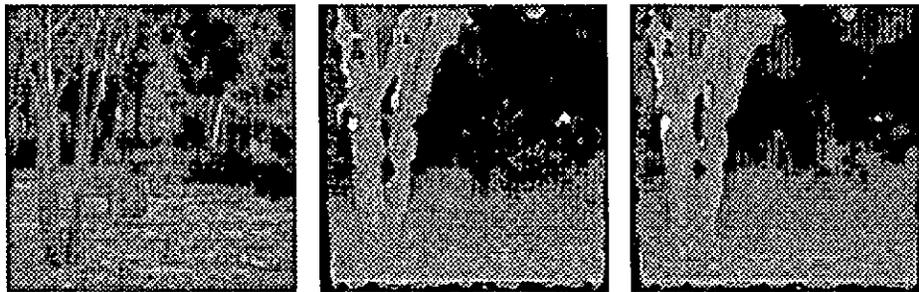


Fig. 6. Tree image with rank and census correlation results

Stereo Matching with Transparency and Matting

Richard Szeliski and Polina Golland

May, 1997

Technical Report

MSR-TR-97-13

Microsoft Research

One Microsoft Way

Redmond, WA 98052

<http://www.research.microsoft.com/>

Abstract

This paper formulates and solves a new variant of the stereo correspondence problem: simultaneously recovering the disparities, true colors, and opacities of visible surface elements. This problem arises in newer applications of stereo reconstruction, such as view interpolation and the layering of real imagery with synthetic graphics for special effects and virtual studio applications. While this problem is intrinsically more difficult than traditional stereo correspondence, where only the disparities are being recovered, it provides a principled way of dealing with commonly occurring problems such as occlusions and the handling of mixed (foreground/background) pixels near depth discontinuities. It also provides a novel means for separating foreground and background objects (matting), without the use of a special blue screen. We formulate the problem as the recovery of colors and opacities in a generalized 3-D (x, y, d) disparity space, and solve the problem using a combination of initial evidence aggregation followed by iterative energy minimization.

keywords: Stereo matching, multiple-view geometry, 3D reconstruction, 3D representations, vision in multi-media.

1 Introduction

Stereo matching has long been one of the central research problems in computer vision. Early work was motivated by the desire to recover depth maps and shape models for robotics and object recognition applications. More recently, depth maps obtained from stereo have been painted with *texture maps* extracted from input images in order to create realistic 3-D scenes and environments for virtual reality and *virtual studio* applications [MB95, SK95, K⁺96, B⁺96]. Unfortunately, the quality and resolution of most stereo algorithms falls quite short of that demanded by these new applications, where even isolated errors in the depth map become readily visible when composited with synthetic graphical elements.

One of the most common errors made by most stereo algorithms is a systematic “fattening” of depth layers near occlusion boundaries. Algorithms based on variable window sizes [KO94] or iterative evidence aggregation [SS96] can in many instances mitigate such errors. Another common problem is that disparities are only estimated to the nearest pixel, which is typically not sufficiently accurate for tasks such as view interpolation. Different techniques have been developed for computing sub-pixel estimates, such as using a finer set of disparity hypotheses or finding the analytic minimum of the local error surface [TH86, MSK89].

Unfortunately, for challenging applications such as *z-keying* (the insertion of graphics between different depth layers in video) [PW94, K⁺96, B⁺96], even this is not good enough. Pixels lying near or on occlusion boundaries will typically be *mixed*, i.e., they will contain blends of both foreground and background colors. When such pixels are composited with other images or graphical elements, objectionable “halos” or “color bleeding” may be visible.

The computer graphics and special effects industries faced a similar problem when extracting foreground objects using *blue screen* techniques [SB96]. A variety of techniques were developed for this *matting problem*, all of which model mixed pixels as combinations of foreground and background colors (the latter of which is usually assumed to be known). Practitioners in these fields quickly discovered that it is insufficient to merely label pixels as foreground and background: it is necessary to simultaneously recover both the true color of each pixel and its *transparency* or *opacity* [PD84, Bli94a].

In this paper, we develop a new, multiframe stereo algorithm which simultaneously recovers depth, color, and transparency estimates at each pixel. Unlike traditional blue-screen matting, we cannot use a known background color to perform the color and matte recovery. Instead, we explicitly model a 3-D (x, y, d) *disparity space*, where each cell has an associated color and opacity value. Our task is to estimate the color and opacity values which best predict the appearance of each input image, using prior assumptions about the (piecewise-) continuity of depths, colors, and opacities to make the problem well posed. To our knowledge, this is the first time that the simultaneous recovery of depth, color, and opacity from stereo images has been attempted.

We begin this paper with a review of previous work in stereo matching. In Section 3, we discuss our novel representation for accumulating color samples in a generalized disparity space. We then describe how to compute an initial estimate of the disparities (Section 4), and how to refine this estimate by taking into account occlusions (Section 5). In Section 6, we develop a novel energy minimization algorithm for estimating disparities, colors and opacities. We present some experiments on both synthetic and real images in Section 7. We conclude the paper with a discussion of our results, and a list of topics for future research.

2 Previous Work

Stereo matching (and the more general problem of stereo-based 3-D reconstruction) are fields with very rich histories [BF82, DA89]. In this section, we focus only on previous work related to our central topics of interest: pixel-accurate matching with sub-pixel precision, the handling of occlusion boundaries, and the use of more than two images. We also mention techniques used in computer graphics to composite images with transparencies and to recover matte (transparency) values using traditional blue-screen techniques.

In thinking about stereo algorithms, we have found it useful to subdivide the stereo matching process into three tasks: the initial computation of matching costs, the aggregation of local evidence, and the selection or computation of a disparity value for each pixel [SS96].

The most fundamental element of any correspondence algorithm is a matching cost that measures the similarity of two or more corresponding pixels in different images. Matching costs can be defined

locally (at the pixel level), e.g., as absolute [K⁺96] or squared intensity differences [MSK89], using edges [Bak80] or filtered images [JJT91, JM92]. Alternatively, matching costs may be defined over an area, e.g., using correlation [RGH80] (this can be viewed as a combination of the matching and aggregation stages). In this paper, we use squared intensity differences.

Aggregating support is necessary to disambiguate potential matches. A support region can either be two-dimensional at a fixed disparity (favoring fronto-parallel surfaces), or three-dimensional in (x, y, d) space (allowing slanted surfaces). Two-dimensional evidence aggregation has been done using both fixed square windows (traditional) and windows with adaptive sizes [Arm83, KO94]. Three-dimensional support functions include limited disparity gradient [PMF85], Prazdny's coherence principle [Pra85] (which can be implemented using two diffusion processes [SH85]), local winner-take-all [YYL93], and iterative (non-linear) evidence aggregation [SS96]. In this paper, our initial evidence aggregation uses an iterative technique, with estimates being refined later through a prediction/adjustment mechanism which explicitly models occlusions.

The easiest way of choosing the best disparity is to select at each pixel the minimum aggregated cost across all disparities under consideration ("winner-take-all"). A problem with this is that uniqueness of matches is only enforced for one image (the *reference image*), while points in the other image might get matched to multiple points. Cooperative algorithms employing symmetric uniqueness constraints are one attempt to solve this problem [MP76]. In this paper, we will introduce the concept of a *virtual camera* which is used for the initial winner-take-all stage.

Occlusion is another very important issue in generating high-quality stereo maps. Many approaches ignore the effects of occlusion; others try to minimize them by using a cyclopean disparity representation [Bar89], or try to recover occluded regions after the matching by cross-checking [Fua93]. Several authors have addressed occlusions explicitly, using Bayesian models and dynamic programming [Arm83, OK85, BM92, Cox94, GLY92, IB94]. However, such techniques require the strict enforcement of *ordering constraints* [YP84]. In this paper, we handle occlusion by re-projecting the disparity space into each input image using traditional back-to-front compositing operations [PD84], and eliminating from consideration pixels which are known to be occluded. (A related technique, developed concurrently with ours, traverses the disparity space from front to back [SD97].)

Sub-pixel (fractional) disparity estimates, which are essential for applications such as view interpolation, can be computed by fitting a curve to the matching costs at the discrete disparity levels [LK81, TH86, MSK89, KO94]. This provides an easy way to increase the resolution of a stereo algorithm with little additional computation. However, to work well, the intensities being matched must vary smoothly. In this paper, we present two different representations for fractional disparity estimates.

Multiframe stereo algorithms use more than two images to increase the stability of the algorithm [BBM87, MSK89, KWZK95, Col96]. In this paper, we present a new framework for formulating the multiframe stereo problem based on the concept of a *virtual camera* and a projective *generalized disparity space*, which includes as special cases the *multiple baseline stereo* models of [OK93, KWZK95, Col96].

Finally, the topic of transparent surfaces has not received much study in the context of computational stereo [Pra85, SH85, Wei89]. Relatively more work has been done in the context of transparent motion estimation [BBHP92, SM91b, SM91a, JBJ96, DP91]. However, these techniques are limited to extracting a small number of dominant motions or planar surfaces. None of these techniques explicitly recover a per-pixel transparency value along with a corrected color value, as we do in this paper.

Our stereo algorithm has also been inspired by work in computer graphics, especially in image compositing [PD84, Bli94a] and blue screen techniques [VT93, SB96]. While traditional blue-screen techniques assume that the background is of a known color, we solve for the more difficult case of each partially transparent surface pixel being the combination of two (or more) unknown colors.

3 Disparity space representation

To formulate our (potentially multiframe) stereo problem, we use a *generalized disparity space* which can be any projective sampling (collineation) of 3-D space. More concretely, we first choose a *virtual camera* position and orientation. This virtual camera may be coincident with one of the input images, or it can be chosen based on the application demands and the desired accuracy of

the results. For instance, if we wish to regularly sample a volume of 3-D space, we can make the camera orthographic, with the camera's (x, y, d) axes being orthogonal and evenly sampled (as in [SD97]). As another example, we may wish to use a *skewed camera model* for constructing a Lumigraph [GGSC96].

Having chosen a virtual camera position, we can also choose the orientation and spacing of the *disparity planes*, i.e., the constant d planes. The relationship between d and 3-D space can be projective. For example, we can choose d to be inversely proportional to depth, which is the usual meaning of disparity [OK93]. The information about the virtual camera's position and disparity plane orientation and spacing can be captured in a single 4×4 matrix \hat{M}_0 , which represents a collineation of 3-D space. The matrix \hat{M}_0 can also capture the sampling information inherent in our disparity space, e.g, if we define disparity space (x, y, d) to be an integer valued sampling of the mapping $\hat{M}_0\mathbf{x}$, where \mathbf{x} represents point in 3-D (Euclidean) space.

An example of a possible disparity space representation is the standard epipolar geometry for two or more cameras placed in a plane perpendicular to their optical axes, in which case a natural choice for disparity is inverse depth (since this corresponds to uniform steps in inter-camera displacements, i.e., the quantity which can be measured accurately) [OK93]. Other choices include the traditional *cyclopean camera* placed symmetrically between two verged cameras, or a uniform sampling of 3-D which is useful in a true verged multi-camera environment [SD97] or for motion stereo. Note that in all of these situations, integral steps in disparity may correspond to fractional shifts in displacement, which may be desirable for optimal accuracy.

Regardless of the disparity space selected, it is always possible to project each of the input images onto the $d = 0$ plane through a simple homography (2-D perspective transform), and to work with such re-projected (*rectified*) images as the inputs to the stereo algorithm. What are the possible advantages of such a rectification step? For two or more cameras whose optical centers are collinear, it is always possible to find a rectification in which corresponding epipolar lines are horizontal, greatly simplifying the stereo algorithm's implementation. For three or more cameras which are coplanar, after rectification, displacements away from the $d = 0$ plane (i.e., changes in disparity) will correspond to uniform steps along fixed directions for each camera (e.g., horizontal and vertical under a suitable camera geometry). Finally, for cameras in general position, steps in

disparity will correspond to zooms (scalings) and sub-pixel shifts of the rectified images, which is quicker (and potentially more accurate) than general perspective resampling [Col96]. A potential disadvantage of pre-rectification is a slight loss in input image quality due to multiple re-samplings, but this can be mitigated using higher-order (e.g., bicubic) sampling filters, and potentially re-sampling the rectified images at higher resolution. Appendix A derives the equations for mapping between input image (both rectified and not) and disparity space.

In this paper, we introduce a generalization of the (x, y, d) space. If we consider each of the $k = 1 \dots K$ images as being samples along a fictitious “camera” dimension, we end up with a 4-D (x, y, d, k) space. In this space, the values in a given (x, y, d) cell as k varies can be thought of as the color distributions at a given location in space, assuming that this location is actually on the surface of the object. We will use these distributions as the inputs to our first stage of processing, i.e., by computing mean and variance statistics. A different slice through (x, y, d, k) space, this time by fixing k , gives the series of shifted images seen by one camera. In particular, compositing these images in a back-to-front order, taking into account each voxel’s opacity, should reconstruct what is seen by a given (rectified) input image (see Section 5).¹

Figure 1 shows a set of sample images, together with an (x, d, k) slice through the 4-D space (y is fixed at a given scanline), where color samples varying in k are grouped together.

4 Estimating an initial disparity surface

The first step in stereo matching is to compute some initial evidence for a surface existing at (or near) a location (x, y, d) in disparity space. We do this by conceptually populating the entire 4-D (x, y, d, k) space with colors obtained by resampling the K input images,

$$\mathbf{c}(x, y, d, k) = \mathcal{W}_f(\mathbf{c}_k(u, v); \mathbf{H}_k + \mathbf{t}_k[0 \ 0 \ d]), \quad (1)$$

¹Note that this 4-D space is *not* the same as that used in the Lumigraph [GGSC96], where the description is one of rays in 3-D, as opposed to color distributions across multiple cameras in 3-D. It is also not the same as an epipolar-plane image (EPI) volume [BBM87], which is a simple concatenation of warped input images.

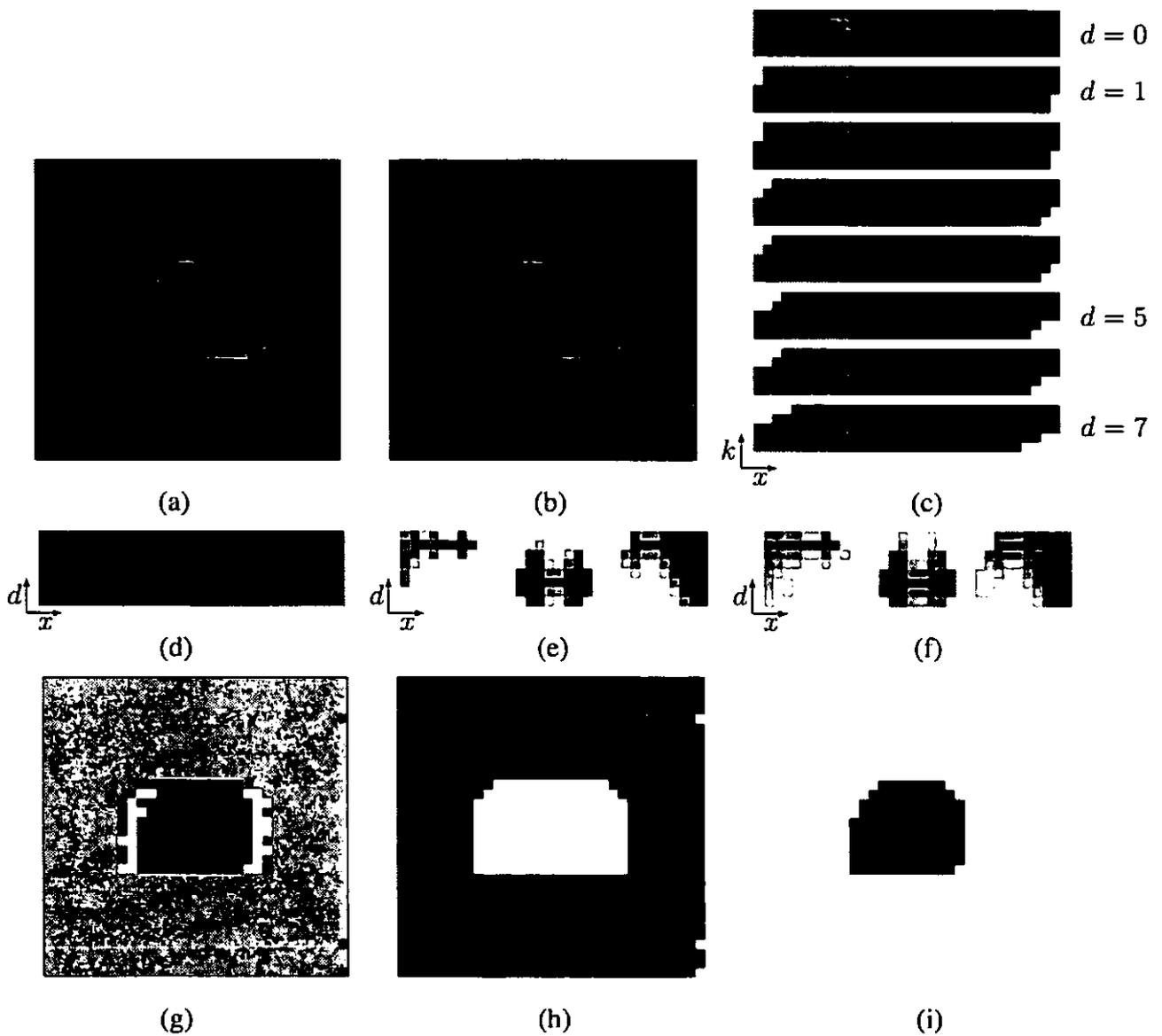


Figure 1: Sample slices through a 4-D disparity space: (a–b) sample input images (arranged for free fusion), (c) (x, d, k) slice for scanline 17, (d) means and (e) variances as a function of (x, d) (smaller variances are darker), (f) variances after evidence accumulation, (g) results of winner-takes-all for whole image (undecided columns in white), (h–i) colors and opacities at disparities 1 and 5. For easier interpretation, all images have been composited over an opaque white background.

where $c_k(u, v)$ is the k th input image,² $H_k + t_k[0 \ 0 \ d]$ is the homography mapping this image to disparity plane d (see Appendix A), \mathcal{W}_f is the forward warping operator,³ and $c(x, y, d, k)$ is the pixel mapped into the 4-D generalized disparity space. Algorithmically, this can be achieved either by first rectifying each image onto the $d = 0$ plane, or by directly using a homography (planar perspective transform) to compute each (d, k) slice.⁴ Note that at this stage, not all (x, y, d, k) cells will be populated, as some of these may map to pixels which are outside some of the input images.

Once we have a collection of color (or luminance) values at a given (x, y, d) cell, we can compute some initial statistics over the K (or fewer) colors, e.g., the sample mean μ and variance σ .⁵ Robust estimates of sample mean and variance are also possible (e.g., [SS96]). Examples of the mean and variance values for our sample image are shown in Figures 1d and 1e, where darker values indicate smaller variances.

After accumulating the local evidence, we usually do not have enough information to determine the correct disparities in the scene (unless each pixel has a unique color). While pixels at the correct disparity should in theory have zero variance, this is not true in the presence of image noise, fractional disparity shifts, and photometric variations (e.g., specularities). The variance may also be arbitrarily high in occluded regions, where pixels which actually belong to a different disparity level will nevertheless vote, often leading to gross errors. For example, in Figure 1c, the middle (red) group of pixels at $d = 5$ should all have the same color in any given column, but they do not because of resampling errors. This effect is especially pronounced near the edge of the red square, where the red color has been severely contaminated by the background blue. This contamination is one of the reasons why most stereo algorithms make systematic errors in the vicinity of depth discontinuities.

²The color values c can be replaced with gray-level intensity values without affecting the validity of our analysis.

³In our current implementation, the warping (resampling) algorithm uses bi-linear interpolation of the pixel colors and opacities.

⁴For certain epipolar geometries, even more efficient algorithms are possible, e.g., by simply shifting along epipolar lines [K⁺96].

⁵In many traditional stereo algorithms, it is common to effectively set the mean to be just the value in one image, which makes these algorithms not truly multiframe [Col96]. The sample variance then corresponds to the squared difference or sum of squared differences [OK93].

To help disambiguate matches, we can use local evidence aggregation. The most common form is averaging using square windows, which results in the traditional sum of squared difference (SSD and SSSD) algorithms [OK93]. To obtain results with better quality near discontinuities, it is preferable to use adaptive windows [KO94] or iterative evidence accumulation [SS96]. In the latter case, we may wish to accumulate an evidence measure which is not simply summed error (e.g., the probability of a correct match [SS96]). Continuing our simple example, Figure 1f shows the results of an evidence accumulation stage, where more certain depths are darker. To generate these results, we aggregate evidence using a variant of the algorithm described in [SS96],

$$\sigma_i^{t+1} \leftarrow a \hat{\sigma}_i^t + b \sum_{j \in \mathcal{N}_4(i)} \hat{\sigma}_j^t + c \sigma_i^0, \quad \hat{\sigma}_i^t = \min(\sigma_i^t, \sigma_{\max}), \quad (2)$$

where σ_i^t is the variance of pixel i at iteration t , $\hat{\sigma}_i^t$ is a robustified (limited) version of the variance, and \mathcal{N}_4 are the usual four nearest neighbors. For the results in Figure 1, we use $(a, b, c) = (0.1, 0.15, 0.3)$ and $\sigma_{\max} = 16$.

At this stage, most stereo matching algorithms pick a winning disparity in each (x, y) column, and call this the final correspondence map. Optionally, they may also compute a fractional disparity value by fitting an analytic curve to the error surface around the winning disparity and then finding its minimum [MSK89, OK93]. Unfortunately, this does nothing to resolve several problems: occluded pixels may not be handled correctly (since they have “inconsistent” color values at the correct disparity), and it is difficult to recover the true (unmixed) color values of surface elements (or their opacities, in the case of pixels near discontinuities).

Our solution to this problem is to use the initial disparity map as the input to a refinement stage which simultaneously estimates the disparities, colors, and opacities which best match the input images while conforming to some prior expectations on smoothness. To start this procedure, we initially pick only winners in each column where the answer is fairly certain, i.e., where the variance (“scatter” in color values) is below a threshold and is a clear winner with respect to the other candidate disparities.⁶ A new (x, y, d) volume is created, where each cell now contains a

⁶To account for resampling errors which occur near rapid color or luminance changes, we set the threshold proportional to the local image variation within a 3×3 window. In our experiments, the threshold is set to $\theta = \theta_{\min} + \theta_s \text{Var}_{3 \times 3}$, with $\theta_{\min} = 10$ and $\theta_s = 0.02$.

color value, initially set to the mean color computed in the first stage, and the opacity is set to 1 for cells which are winners, and 0 otherwise.⁷

5 Computing visibilities through re-projection

Once we have an initial (x, y, d) volume containing estimated RGBA (color and 0/1 opacity) values, we can re-project this volume into each of the input cameras using the known transformation

$$\mathbf{x}_k = \mathbf{M}_k \hat{\mathbf{M}}_0^{-1} \hat{\mathbf{x}}_0 \quad (3)$$

(see (14) in Appendix A), where $\hat{\mathbf{x}}_0$ is a (homogeneous) coordinate in (x, y, d) space, $\hat{\mathbf{M}}_0$ is the complete camera matrix corresponding to the virtual camera, \mathbf{M}_k is the k th camera matrix, and \mathbf{x}_k are the image coordinates in the k th image. There are several techniques possible for performing this projection, including classical *volume rendering* techniques [Lev90, LL94]. In our approach, we interpret the (x, y, d) volume as a set of (potentially) transparent acetates stacked at different d levels. Each acetate is first warped into a given input camera’s frame using the known homography

$$\mathbf{x}_k = \mathbf{H}_k \mathbf{x}_0 + \mathbf{t}_k d = (\mathbf{H}_k + \mathbf{t}_k [0 \ 0 \ d]) \mathbf{x}_0 \quad (4)$$

and the layers are then composited back-to-front (this is called a shear-warp algorithm [LL94]).⁸

The resampling procedure for a given layer d into the coordinate system of camera k can be written as

$$\bar{\mathbf{c}}_k(u, v, d) = \mathcal{W}_b(\hat{\mathbf{c}}(x, y, d); \mathbf{H}_k + \mathbf{t}_k [0 \ 0 \ d]), \quad (5)$$

where $\hat{\mathbf{c}} = [r \ g \ b \ \alpha]^T$ is the current color and opacity estimate at a given location (x, y, d) , $\bar{\mathbf{c}}_k$ is the resampled layer d in camera k ’s coordinate system, and \mathcal{W}_b is the resampling operation induced by the homography (4).⁹ Note that the warping function is *linear* in the colors and opacities being

⁷We may, for computational reasons, choose to represent this volume using colors premultiplied by their opacities (*associated colors* [PD84, Bli94a]), in which case voxels for which alpha (opacity) is 0 should have their color or intensity values set to 0. See [Bli94a, Bli94b] for a discussion of the advantages of using premultiplied colors.

⁸If the input images have been rectified, or under certain imaging geometries, this homography will be a simple scale and/or shift (Appendix A).

⁹This is the inverse of the warp specified in (1).

resampled, i.e., the $\tilde{c}_k(u, v, d)$ can be expressed as a linear function of the $\hat{c}(x, y, d)$, e.g., through a sparse matrix multiplication.

Once the layers have been resampled, they are then composited using the standard *over* operator [PD84],

$$f \wedge b \equiv f + (1 - \alpha_f)b,$$

where f and b are the premultiplied foreground and background colors, and α_f is the opacity of the foreground [PD84, Bli94a]. Using the over operator, we can form a composite image

$$\tilde{c}_k(u, v) = \bigwedge_{d=d_{\max}}^{d_{\min}} \tilde{c}_k(u, v, d) = \tilde{c}_k(u, v, d_{\max}) \wedge \cdots \wedge \tilde{c}_k(u, v, d_{\min}) \quad (6)$$

(note that the over operator is associative but not commutative, and that d_{\max} is the layer closest to the camera).

After the re-projection step, we refine the disparity estimates by preventing visible surface pixels from voting for potential disparities in the regions they occlude. More precisely, we build an (x, y, d, k) *visibility map*, which indicates whether a given camera k can see a voxel at location (x, y, d) . A simple way to construct such a visibility map is to record the disparity value d_{top} for each (u, v) pixel which corresponds to the topmost opaque pixel seen during the compositing step.¹⁰ The visibility value can then be defined as

$$V_k(u, v, d) = \text{if } d \geq d_{\text{top}}(u, v) \text{ then } 1 \text{ else } 0.$$

The visibility and opacity (alpha) values taken together can be interpreted as follows:

$$V_k = 1, \tilde{\alpha}_k = 0:$$

$$V_k = 1, \tilde{\alpha}_k = 1:$$

$$V_k = 0, \tilde{\alpha}_k = ?:$$

A more principled way of defining visibility, which takes into account partially opaque voxels, uses a recursive front-to-back algorithm

$$\begin{aligned} V_k(u, v, d - 1) &= V_k(u, v, d) (1 - \tilde{\alpha}_k(u, v, d)) \\ &= \prod_{d'=d}^{d_{\max}} (1 - \tilde{\alpha}_k(u, v, d')), \end{aligned} \quad (7)$$

¹⁰Note that it is not possible to compute visibility in (x, y, d) disparity space, as several opaque pixels in disparity space may potentially project to the same input camera pixel.

with the initial visibilities all being set to 1, $V_k(u, v, d_{\max}) = 1$. We now have a very simple (linear) expression for the compositing operation,

$$\tilde{c}_k(u, v) = \sum_{d=d_{\min}}^{d_{\max}} \tilde{c}_k(u, v, d) V_k(u, v, d). \quad (8)$$

Once we have computed the visibility volumes for each input camera, we can update the list of color samples we originally used to get our initial disparity estimates. Let

$$c_k(u, v, d) = c_k(u, v) V_k(u, v, d)$$

be the input color image multiplied by its visibility at disparity d . If we substitute $c_k(u, v, d)$ for $c_k(u, v)$ in (1), we obtain a distribution of colors in (x, y, d, k) where each color has an associated visibility value (Figure 2c). Voxels which are occluded by surfaces lying in front in a given view k will now have fewer (or potentially no) votes in their local color distributions. We can therefore recompute the local mean and variance estimates using weighted statistics, where the visibilities $V(x, y, d, k)$ provide the weights (Figures 2d and 2e).

With these new statistics, we are now in position to refine the disparity map. In particular, voxels in disparity space which previously had an inconsistent set of color votes (large variance) may now have a consistent set of votes, because voxels in (partially occluded) regions will now only receive votes from input pixels which are not already assigned to nearer surfaces (Figure 2c–f). Figure 2g–i show the results after one iteration of this algorithm.

6 Refining color and transparency estimates

While the above process of computing visibilities and refining disparity estimates will in general lead to a higher quality disparity map (and better quality mean colors, i.e., texture maps), we have not yet addressed the issue of recovering true colors and transparencies in *mixed pixels*, e.g., near depth discontinuities, which is one of the main goals of this research.

A simple way to approach this problem is to take the binary opacity maps produced by our stereo matching algorithm, and to make them real-valued using a low-pass filter. Another possibility might be to recover the transparency information by looking at the magnitude of the intensity gradient [MYT95], assuming that we can isolate regions which belong to different disparity levels.

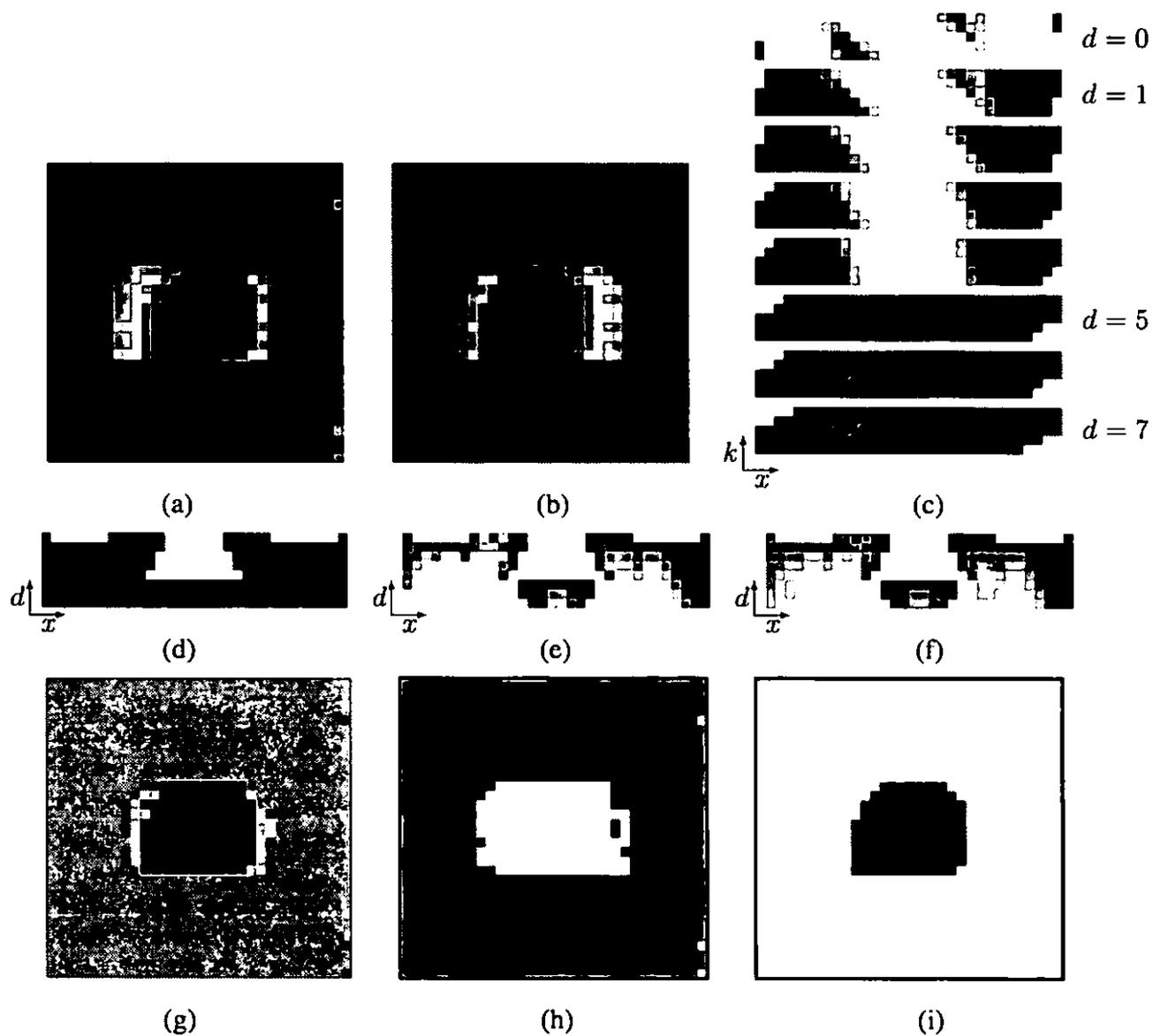


Figure 2: After modifying input images by visibility $V_k(u, v, d)$: (a–b) re-synthesized views of sample images, (c) (x, d, k) slice for scanline 17, (d) means and (e) variances as a function of (x, d) , (f) variances after evidence accumulation, (g) results of winner-takes-all for whole image, and (h–i) colors and opacities at disparities 1 and 5 after one iteration of the reprojection algorithm.

In our work, we have chosen instead to adjust the opacity and color values $\hat{c}(x, y, d)$ to match the input images (after re-projection), while favoring continuity in the color and opacity values. This can be formulated as a non-linear minimization problem, where the cost function has three parts:

1. a weighted error norm on the difference between the re-projected images $\tilde{c}_k(u, v)$ and the original (or rectified) input images $c_k(u, v)$

$$C_1 = \sum_{(u,v)} w_k(u, v) \rho_1(\tilde{c}_k(u, v) - c_k(u, v)), \quad (9)$$

where the weights $w_k(u, v)$ may depend on the position of camera k relative to the virtual camera;¹¹

2. a (weak) smoothness constraint on the colors and opacities,

$$C_2 = \sum_{(x,y,d)} \sum_{(x',y',d') \in \mathcal{N}(x,y,d)} \rho_2(\hat{c}(x', y', d') - \hat{c}(x, y, d)); \quad (10)$$

3. a prior distribution on the opacities,

$$C_3 = \sum_{(x,y,d)} \phi(\alpha(x, y, d)). \quad (11)$$

In the above equations, ρ_1 and ρ_2 are either quadratic functions or robust penalty functions [Hub81], and ϕ is a function which encourages opacities to be 0 or 1, e.g., $\phi(x) = x(1 - x)$.¹²

The smoothness constraint on colors makes more sense with non-premultiplied colors. For example, a voxel lying on a depth discontinuity will be partially transparent, and yet should have the same non-premultiplied color as its neighbors. An alternative, which allows us to work with premultiplied colors, is to use a smoothness constraint of the form

$$C'_2 = \sum_{(x,y,d)} \sum_{(x',y',d') \in \mathcal{N}(x,y,d)} \rho_2(\alpha(x, y, d)c(x', y', d') - \alpha(x', y', d')c(x, y, d)). \quad (12)$$

¹¹More precisely, we may wish to measure the angle between the viewing ray corresponding to (u, v) in the two cameras. However, the ray corresponding to (u, v) in the virtual camera depends on the disparity d .

¹²All color and opacity values are, of course, constrained to lie in the range $[0, 1]$, making this a constrained optimization problem.

To minimize the total cost function

$$\mathcal{C} = \lambda_1 \mathcal{C}_1 + \lambda_2 \mathcal{C}_2 + \lambda_3 \mathcal{C}_3, \quad (13)$$

we use a preconditioned gradient descent algorithm. Appendix B contains details on how to compute the required gradients and Hessians.

7 Experiments

To study the properties of our new stereo correspondence algorithm, we ran a small set of experiments on some synthetic stereo datasets, both to evaluate the basic behavior of the algorithm (aggregation, visibility-based refinement, and energy minimization), and to study its performance on mixed (boundary) pixels. Being able to visualize opacities/transparencies is very important for understanding and validating our algorithm. For this reason, we chose color stimuli (the background is blue-green, and the foreground is red). Pixels which are partially transparent will show up as “pale” colors, while fully transparent pixels will be white. We should emphasize that our algorithm does not require colored images as inputs (see Figure 5), nor does it require the use of standard epipolar geometries.

The first stimulus we generated was a traditional random-dot stereogram, where the choice of camera geometry and filled disparity planes results in integral pixel shifts. This example also contains no partially transparent pixels. Figure 3 shows the results on this stimulus. The first eight columns are the eight disparity planes in (x, y, d) space, showing the estimated colors and opacities (smaller opacities are shown as lighter colors, since the RGBA colors are composited over a white background). The ninth and tenth column are two re-synthesized views (leftmost and middle). The last column is the re-synthesized middle view with a synthetic light-gray square inserted at disparity $d = 3$.

As we can see in Figure 3, the basic iterative aggregation algorithm results in a “perfect” reconstruction, although only one pixel is chosen in each column. For this reason, the re-synthesized leftmost view (ninth column) contains a large “gap”.

Figure 3b shows the results of using only the first \mathcal{C}_1 term in our cost function, i.e., only matching re-synthesized views with input images. The re-synthesized view in column nine is now much better,

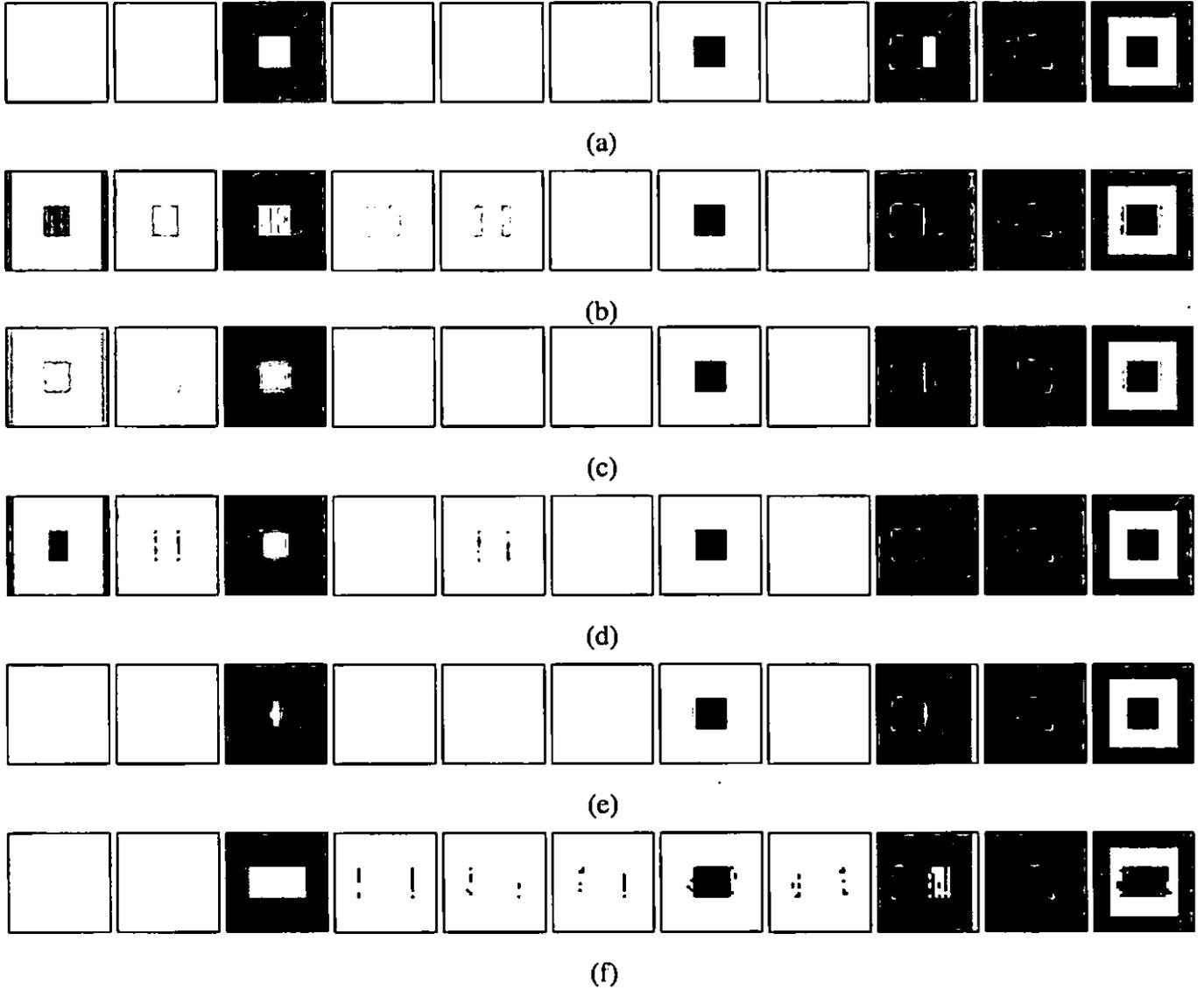


Figure 3: Traditional synthetic RDS results: (a) after iterative aggregation but before gradient descent, (b) without smoothness or opacity constraint, $\lambda_1 = 1, \lambda_2 = \lambda_3 = 0$, (c) without opacity constraint, $\lambda_1 = \lambda_2 = 1, \lambda_3 = 0$, (d) with all three constraints, $\lambda_1 = 50, \lambda_2 = 1, \lambda_3 = 50$, (e) with all three constraints, $\lambda_1 = 50, \lambda_2 = 1, \lambda_3 = 100$, (f) simple winner-take-all (shown for comparison). The first eight columns are the disparity layers, $d = 0 \dots 7$. The ninth and tenth columns are re-synthesized sample views. The last column is a re-synthesized view with a synthetic gray square inserted at disparity $d = 3$.

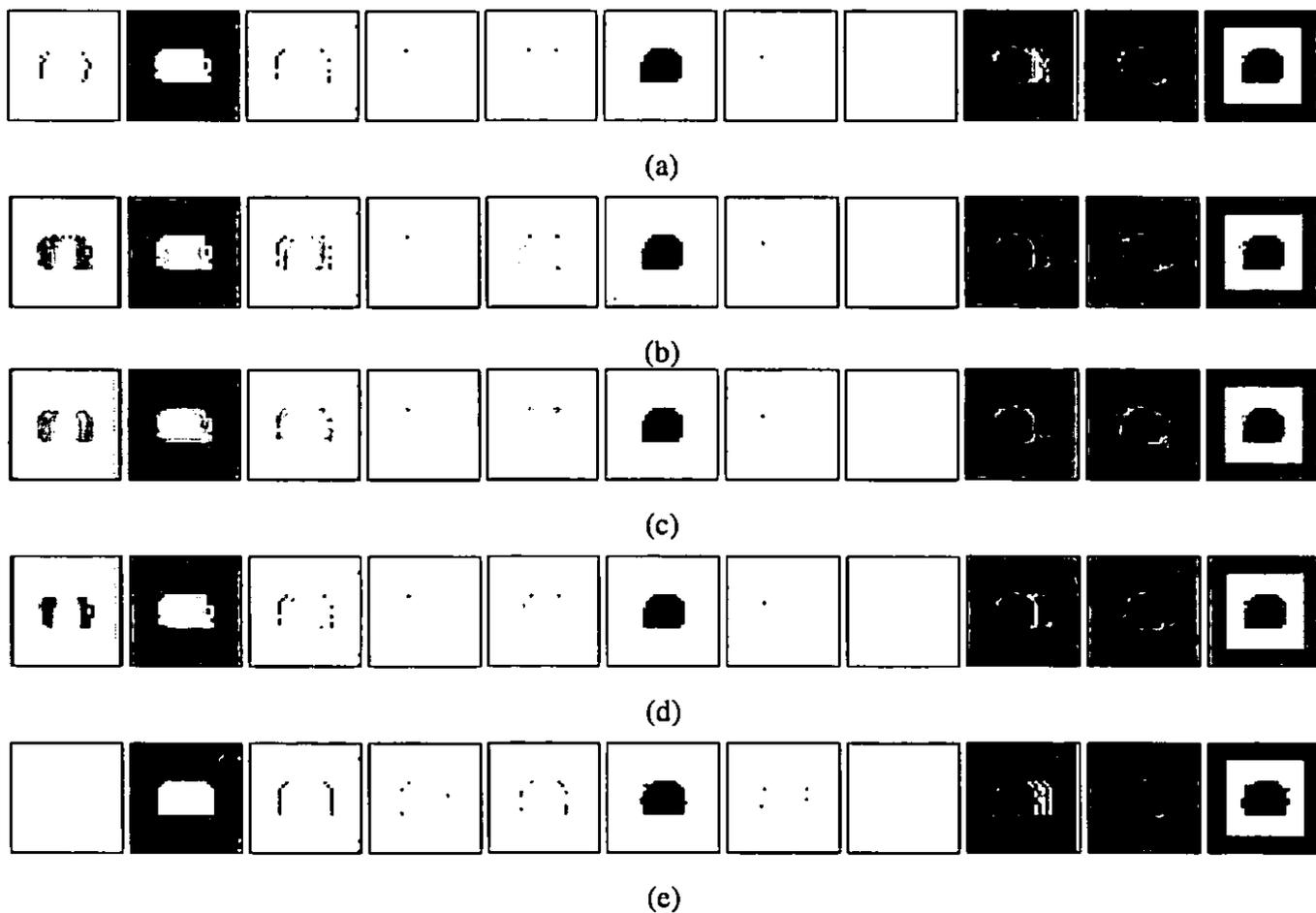


Figure 4: More challenging synthetic RDS results: (a) after iterative aggregation but before gradient descent, (b) without smoothness or opacity constraint, $\lambda_1 = 1, \lambda_2 = \lambda_3 = 0$, (c) without opacity constraint, $\lambda_1 = \lambda_2 = 1, \lambda_3 = 0$, (d) with all three constraints, $\lambda_1 = 50, \lambda_2 = 1, \lambda_3 = 50$, (e) simple winner-take-all (shown for comparison). The first eight columns are the disparity layers, $d = 0 \dots 7$. The ninth and tenth columns are re-synthesized sample views. The last column is the re-synthesized view with a synthetic gray square inserted at disparity $d = 3$.

although we see that a bit of the background has bled into the foreground layers, and that the pixels near the depth discontinuity are spread over several disparities.

Adding the smoothness constraint \mathcal{C}_2 (Figure 3c) ameliorates both of these problems. Adding the (weak) 0/1 opacity constraint \mathcal{C}_3 (Figure 3d–e) further removes stray pixels at wrong disparity levels. Figure 3d shows a “softer” variant of the opacity constraint ($\lambda_3 = 50 = \lambda_1$), where more levels end up being filled in, but the re-synthesized views are very good. Figure 3e shows a “harder” constraint ($\lambda_3 = 100 = 2\lambda_1$), where only pixels adjacent to initial estimates are filled in, at the cost of a gap in some re-synthesized views.

For comparison, Figure 3f shows the results of a traditional winner-take-all algorithm (the same as Figure 3a with a very large θ_{\min} and no occluded pixel removal). We can clearly see the effects of background colors being pulled into the foreground layer, as well as increased errors in the occluded regions.

Our second set of experiments uses the same synthetic stereo dataset as shown in Figures 1 and 2. Here, because the background layer is at an odd disparity, we get significant re-sampling errors (because we currently use bilinear interpolation) and mixed pixels. The stimulus also has partially transparent pixels along the edge of the top half-circle in the foreground shape. This stereo dataset is significantly more difficult to match than previous random-dot stereograms.

Figure 4a shows the results of applying only our iterative aggregation algorithm, without any energy minimization. The set of estimated disparities are insufficient to completely reconstruct the input images (this could be changed by adjusting the thresholds θ_{\min} and θ_s), and several pixels are incorrectly assigned to the $d = 0$ layer (due to difficulties in disambiguating depths in partially occluded regions).

Figure 4b shows the results of using only the first \mathcal{C}_1 term in our cost function, i.e., only matching re-synthesized views with input images. The re-synthesized view in column nine is now much better, although we see that a bit of the background has bled into the foreground layers, and that the pixels near the depth discontinuity are spread over several disparities.

Adding the smoothness constraint \mathcal{C}_2 (Figure 4c) ameliorates both of these problems. Adding the (weak) 0/1 opacity constraint \mathcal{C}_3 (Figure 4d) further removes stray pixels at wrong disparity levels, but at the cost of an incompletely reconstructed image (this is less of a problem if the foreground

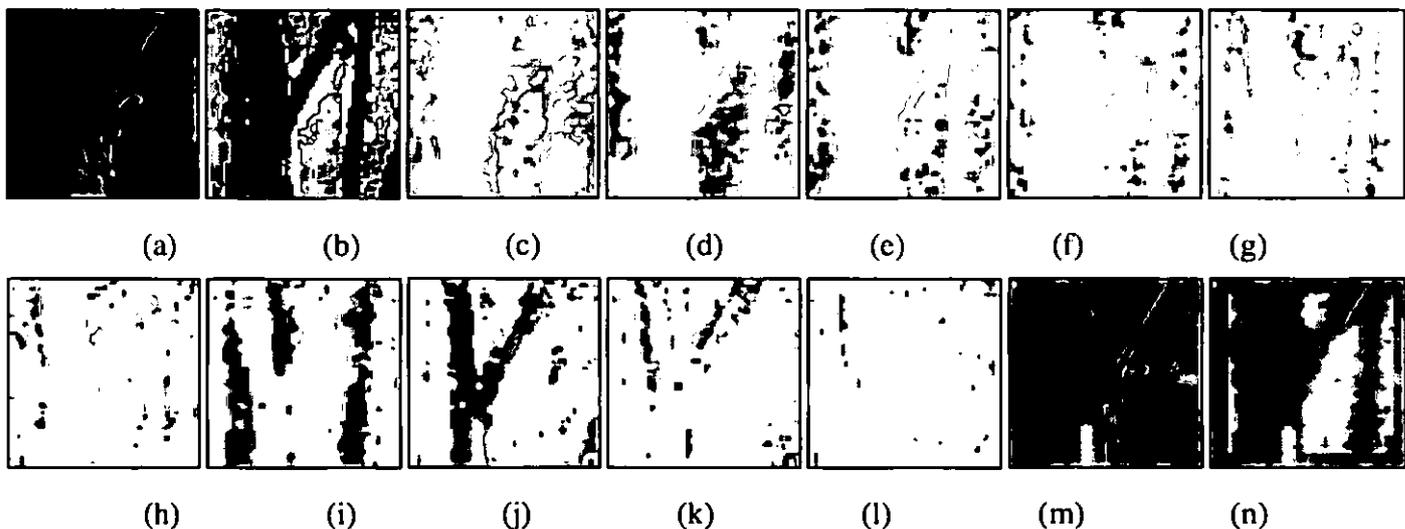


Figure 5: Real image example: (a) cropped subimage from *SRI Trees* data set, (b) depth map after initial aggregation stage, (c–l) disparity layers $d = 0 \dots 9$, (m) re-synthesized input image, (n) with inserted $d = 4$ blue layer.

is being layered on a synthetic background, as in the last column). As before, Figure 4e shows the results of a traditional winner-take-all algorithm.

Figure 5 shows the results on a cropped portion of the *SRI Trees* multibaseline stereo dataset. A small region (64×64 pixels) was selected in order to better visualize pixel-level errors. While the overall reconstruction is somewhat noisy, the final reconstruction with a synthetic blue layer inserted shows that the algorithm has done a reasonable job of assigning pixel depths and computing partial transparencies near the tree boundaries.

From these examples, it is apparent that the algorithm is currently sensitive to the choice of parameters used to control both the initial aggregation stage and the energy minimization phase. Setting these parameters automatically will be an important area for further research.

8 Discussion

While our preliminary experimental results are encouraging, the simultaneous recovery of accurate depth, color, and opacity estimates remains a challenging problem. Traditional stereo algorithms search for a unique disparity value at each pixel in a given reference image. Our approach, on

the other hand, is to recover a sparsely populated volume of colors and opacities. This has the advantage of correctly modeling mixed pixels and occlusion effects, and allows us to merge images from very disparate points of view. Unfortunately, it also makes the estimation problem much more difficult, since the number of free parameters often exceeds the number of measurements, hence necessitating smoothness constraints and other prior models.

Partially occluded areas are problematic because very few samples may be available to disambiguate depth. A more careful analysis of the interaction between the measurement, smoothness, and opacity constraints will be required to solve this problem. Other problems occur near depth discontinuities, and in general near rapid intensity (albedo) changes, where the scatter in color samples may be large because of re-sampling errors. Better imaging and sensor models, or perhaps working on a higher resolution image grid, might be required to solve these problems.

8.1 Future work

There are many additional topics related to transparent stereo and matting which we plan to investigate. For example, we plan to try our algorithm on data sets with true transparency (not just mixed pixels), such as traditional transparent random dot stereograms [Pra85, Wei89] and reflections in windows [BBHP92].

Estimating disparities to sub-integer precision should improve the quality of our reconstructions. Such fractional disparity estimates can be obtained by interpolating a variance vs. disparity curve $\sigma(d)$, e.g., by fitting a parabola to the lowest variance and its two neighbors [TH86, MSK89]. Alternatively, we can linearly interpolate individual color errors $c(x, y, d, k) - \mu(x, y, d)$ between disparity levels, and find the minimum of the summed squared error (which will be a quadratic function of the fractional disparity).

Instead of representing our color volume $\hat{c}(x, y, d)$ using colors pre-multiplied by their opacities [Bli94a], we could keep these quantities separate. Thus, colors could “bleed” into areas which are transparent, which may be a more natural representation for color smoothness (e.g., for surfaces with small holes). Different color representations such as hue, saturation, intensity (HSV) may also be more suitable for performing correspondence [GB95], and they would permit us to reason more directly about underlying physical processes (shadows, shading, *etc.*).

We plan to investigate the relationship of our new disparity space model to more traditional layered motion models [BBHP92, SM91b, SM91a, DP91, JBJ96, SA96]. We also plan to make more principled use of robust statistics, and investigate alternative search algorithms such as multiresolution (pyramidal) continuation methods and stochastic (Monte Carlo) gradient descent techniques.

9 Conclusions

In this paper, we have developed a new framework for simultaneously recovering disparities, colors, and opacities from multiple images. This framework enables us to deal with many commonly occurring problems in stereo matching, such as partially occluded regions and pixels which contain mixtures of foreground and background colors. Furthermore, it promises to deliver better quality (sub-pixel accurate) color and opacity estimates, which can be used for foreground object extraction and mixing live and synthetic imagery.

To set the problem in as general a framework as possible, we have introduced the notion of a virtual camera which defines a generalized disparity space, which can be any regular projective sampling of 3-D. We represent the output of our algorithm as a collection of color and opacity values lying on this sampled grid. Any input image can (in principle) be re-synthesized by warping each disparity layer using a simple homography and compositing the images. This representation can support a much wider range of synthetic viewpoints in view interpolation applications than a single texture-mapped depth image.

To solve the correspondence problem, we first compute mean and variance estimates at each cell in our (x, y, d) grid. We then pick a subset of the cells which are likely to lie on the reconstructed surface using a thresholded winner-take-all scheme. The mean and variance estimates are then refined by removing from consideration cells which are in the occluded (shadow) region of each current surface element, and this process is repeated.

Starting from this rough estimate, we formulate an energy minimization problem consisting of an input matching criterion, a smoothness criterion, and a prior on likely opacities. This criterion is then minimized using an iterative preconditioned gradient descent algorithm.

While our preliminary experimental results look encouraging, there remains much work to

be done in developing truly accurate and robust correspondence algorithms. We believe that the development of such algorithms will be crucial in promoting a wider use of stereo-based imaging in novel applications such as special effects, virtual reality modeling, and virtual studio productions.

References

- [Arn83] R. D. Arnold. Automated stereo perception. Technical Report AIM-351, Artificial Intelligence Laboratory, Stanford University, March 1983.
- [B⁺96] L. Blonde et al. A virtual studio for live broadcasting: The Mona Lisa project. *IEEE Multimedia*, 3(2):18–29, Summer 1996.
- [Bak80] H. H. Baker. Edge based stereo correlation. In L. S. Baumann, editor, *Image Understanding Workshop*, pages 168–175. Science Applications International Corporation, April 1980.
- [Bar89] S. T. Barnard. Stochastic stereo matching over scale. *International Journal of Computer Vision*, 3(1):17–32, 1989.
- [BBHP92] J. R. Bergen, P. J. Burt, R. Hingorani, and S. Peleg. A three-frame algorithm for estimating two-component image motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(9):886–896, September 1992.
- [BBM87] R. C. Bolles, H. H. Baker, and D. H. Marimont. Epipolar-plane image analysis: An approach to determining structure from motion. *International Journal of Computer Vision*, 1:7–55, 1987.
- [BF82] S. T. Barnard and M. A. Fischler. Computational stereo. *Computing Surveys*, 14(4):553–572, December 1982.
- [Bli94a] J. F. Blinn. Jim Blinn's corner: Compositing, part 1: Theory. *IEEE Computer Graphics and Applications*, 14(5):83–87, September 1994.
- [Bli94b] J. F. Blinn. Jim Blinn's corner: Compositing, part 2: Practice. *IEEE Computer Graphics and Applications*, 14(6):78–82, November 1994.
- [BM92] P. N. Belhumeur and D. Mumford. A Bayesian treatment of the stereo correspondence problem using half-occluded regions. In *Computer Vision and Pattern Recognition*, pages 506–512, Champaign-Urbana, Illinois, 1992.
- [Col96] R. T. Collins. A space-sweep approach to true multi-image matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 358–363, San Francisco, California, June 1996.

- [Cox94] I. J. Cox. A maximum likelihood n -camera stereo algorithm. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'94)*, pages 733–739, Seattle, Washington, June 1994. IEEE Computer Society.
- [DA89] U. R. Dhond and J. K. Aggarwal. Structure from stereo—a review. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6):1489–1510, November/December 1989.
- [DP91] T. Darrell and A. Pentland. Robust estimation of a multi-layered motion representation. In *IEEE Workshop on Visual Motion*, pages 173–178, Princeton, New Jersey, October 1991. IEEE Computer Society Press.
- [Fua93] P. Fua. A parallel stereo algorithm that produces dense depth maps and preserves image features. *Machine Vision and Applications*, 6:35–49, 1993.
- [GB95] P. Golland and A.M. Bruckstein. Motion from color. Technical Report 9513, IS Lab, CS Department, Technion, Haifa, Israel, 1995.
- [GGSC96] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *Computer Graphics Proceedings, Annual Conference Series*, pages 43–54, Proc. SIGGRAPH'96 (New Orleans), August 1996. ACM SIGGRAPH.
- [GLY92] D. Geiger, B. Ladendorf, and A. Yuille. Occlusions and binocular stereo. In *Second European Conference on Computer Vision (ECCV'92)*, pages 425–433, Santa Margherita Liguere, Italy, May 1992. Springer-Verlag.
- [Hub81] P. J. Huber. *Robust Statistics*. John Wiley & Sons, New York, New York, 1981.
- [IB94] S. S. Intille and A. F. Bobick. Disparity-space images and large occlusion stereo. In *Proc. Third European Conference on Computer Vision (ECCV'94)*, volume 1, Stockholm, Sweden, May 1994. Springer-Verlag.
- [JBJ96] S. X. Ju, M. J. Black, and A. D. Jepson. Skin and bones: Multi-layer, locally affine, optical flow and regularization with transparency. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 307–314, San Francisco, California, June 1996.
- [JJT91] M. R. M. Jenkin, A. D. Jepson, and J. K. Tsotsos. Techniques for disparity measurement. *CVGIP: Image Understanding*, 53(1):14–30, January 1991.
- [JM92] D. G. Jones and J. Malik. A computational framework for determining stereo correspondence from a set of linear spatial filters. In *Second European Conference on Computer Vision (ECCV'92)*, pages 397–410, Santa Margherita Liguere, Italy, May 1992. Springer-Verlag.

- [K⁺96] T. Kanade et al. A stereo machine for video-rate dense depth mapping and its new applications. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 196–202, San Francisco, California, June 1996.
- [KO94] T. Kanade and M. Okutomi. A stereo matching algorithm with an adaptive window: Theory and experiment. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(9):920–932, September 1994.
- [KWZK95] S. B. Kang, J. Webb, L. Zitnick, and T. Kanade. A multibaseline stereo system with active illumination and real-time image acquisition. In *Fifth International Conference on Computer Vision (ICCV'95)*, pages 88–93, Cambridge, Massachusetts, June 1995.
- [Lev90] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, July 1990.
- [LK81] B. D. Lucas and T. Kanade. An iterative image registration technique with an application in stereo vision. In *Seventh International Joint Conference on Artificial Intelligence (IJCAI-81)*, pages 674–679, Vancouver, 1981.
- [LL94] P. Lacroute and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. *Computer Graphics (SIGGRAPH'94)*, pages 451–457, July 1994.
- [MB95] L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. *Computer Graphics (SIGGRAPH'95)*, pages 39–46, August 1995.
- [MP76] D. Marr and T. Poggio. Cooperative computation of stereo disparity. *Science*, 194:283–287, October 1976.
- [MSK89] L. H. Matthies, R. Szeliski, and T. Kanade. Kalman filter-based algorithms for estimating depth from image sequences. *International Journal of Computer Vision*, 3:209–236, 1989.
- [MYT95] T. Mitsunaga, T. Yokoyama, and T. Totsuka. Autokey: Human assisted key extraction. In *Computer Graphics Proceedings, Annual Conference Series*, pages 265–272, Proc. SIGGRAPH'95 (Los Angeles), August 1995. ACM SIGGRAPH.
- [OK85] Y. Ohta and T. Kanade. Stereo by intra- and inter-scanline search using dynamic programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-7(2):139–154, March 1985.
- [OK93] M. Okutomi and T. Kanade. A multiple baseline stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(4):353–363, April 1993.
- [PD84] T. Porter and T. Duff. Compositing digital images. *Computer Graphics (SIGGRAPH'84)*,

18(3):253–259, July 1984.

- [PFTV92] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, England, second edition, 1992.
- [PMF85] S. B. Pollard, J. E. W. Mayhew, and J. P. Frisby. PMF: A stereo correspondence algorithm using a disparity gradient limit. *Perception*, 14:449–470, 1985.
- [Pra85] K. Prazdny. Detection of binocular disparities. *Biological Cybernetics*, 52:93–99, 1985.
- [PW94] Y. Paker and S. Wilbur, editors. *Image Processing for Broadcast and Video Production, Hamburg, 1994*, Workshops in Computing, Hamburg, 1994. Springer. Proceedings of the European Workshop on Combined Real and Synthetic Image Processing for Broadcast and Video Production, Hamburg, 23-24 November, 1994.
- [RGH80] T. W. Ryan, R. T. Gray, and B. R. Hunt. Prediction of correlation errors in stereo-pair images. *Optical Engineering*, 19(3):312–322, May/June 1980.
- [SA96] H. S. Sawhney and S. Ayer. Compact representation of videos through dominant multiple motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(8):814–830, August 1996.
- [SB96] A. R. Smith and J. F. Blinn. Blue screen matting. In *Computer Graphics Proceedings, Annual Conference Series*, pages 259–268, Proc. SIGGRAPH'96 (New Orleans), August 1996. ACM SIGGRAPH.
- [SD97] S. M. Seitz and C. M. Dyer. Photorealistic scene reconstruction by space coloring. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'97)*, San Juan, Puerto Rico, June 1997.
- [SH85] R. Szeliski and G. Hinton. Solving random-dot stereograms using the heat equation. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'85)*, pages 284–288, San Francisco, California, June 1985. IEEE Computer Society Press.
- [SK95] R. Szeliski and S. B. Kang. Direct methods for visual scene reconstruction. In *IEEE Workshop on Representations of Visual Scenes*, pages 26–33, Cambridge, Massachusetts, June 1995.
- [SM91a] M. Shizawa and K. Mase. Principle of superposition: A common computational framework for analysis of multiple motion. In *IEEE Workshop on Visual Motion*, pages 164–172, Princeton, New Jersey, October 1991. IEEE Computer Society Press.
- [SM91b] M. Shizawa and K. Mase. A unified computational theory of motion transparency and motion

- boundaries based on eigenenergy analysis. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'91)*, pages 289–295, Maui, Hawaii, June 1991. IEEE Computer Society Press.
- [SS96] D. Scharstein and R. Szeliski. Stereo matching with non-linear diffusion. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'96)*, pages 343–350, San Francisco, California, June 1996.
- [TH86] Q. Tian and M. N. Huhns. Algorithms for subpixel registration. *Computer Vision, Graphics, and Image Processing*, 35:220–233, 1986.
- [VT93] P. Vlahos and B. Taylor. Traveling matte composite photography. In *American Cinematographer Manual*, pages 430–445. American Society of Cinematographers, Hollywood, 7 edition, 1993.
- [Wei89] D. Weinshall. Perception of multiple transparent planes in stereo vision. *Nature*, 341:737–739, 26 October 1989.
- [YP84] A. L. Yuille and T. Poggio. A generalized ordering constraint for stereo correspondence. A. I. Memo 777, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, May 1984.
- [YYL93] Y. Yang, A. Yuille, and J. Lu. Local, global, and multilevel stereo matching. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'93)*, pages 274–279, New York, New York, June 1993. IEEE Computer Society.

A Camera models, disparity space, and induced homographies

The homographies mapping input images (rectified or not) to planes in disparity space can be derived directly from the camera matrices involved. Throughout this appendix, we use projective coordinates, i.e., equality is defined only up to a scale factor.

Let M_k be the 3×4 *camera matrix* which maps real-world coordinates $\mathbf{x} = [X \ Y \ Z \ 1]^T$ into a camera's screen coordinates $\mathbf{x}_k = [u \ v \ 1]^T$, $\mathbf{x}_k = M_k \mathbf{x}$. Similarly, let \hat{M}_0 be the 4×4 collineation which maps world coordinates \mathbf{x} into disparity space coordinates $\hat{\mathbf{x}}_0 = [x \ y \ 1 \ d]^T$, $\hat{\mathbf{x}}_0 = \hat{M}_0 \mathbf{x}$.

We can write the mapping between a pixel in the d th disparity plane, $\mathbf{x}_0 = [x \ y \ 1]^T$, and its

corresponding coordinate \mathbf{x}_k in the k th input image as

$$\mathbf{x}_k = \mathbf{M}_k \hat{\mathbf{M}}_0^{-1} \hat{\mathbf{x}}_0 = \mathbf{H}_k \mathbf{x}_0 + \mathbf{t}_k d, = (\mathbf{H}_k + \mathbf{t}_k [0 \ 0 \ d]) \mathbf{x}_0, \quad (14)$$

where \mathbf{H}_k is the homography relating the rectified and non-rectified version of input image k (i.e., the homography for $d = 0$), and \mathbf{t}_k is the image of the virtual camera's center of projection in image k , i.e., the epipole (this can be seen by setting $d \rightarrow \infty$).

If we first rectify an input image, we can re-project it into a new disparity plane d using

$$\mathbf{x}_k = \mathbf{H}_k \mathbf{x}'_0 = \mathbf{H}_k \mathbf{x}_0 + \mathbf{t}_k d$$

where \mathbf{x}'_0 is the new coordinate corresponding to \mathbf{x}_0 at $d = 0$. From this,

$$\mathbf{x}'_0 = \mathbf{x}_0 + \hat{\mathbf{t}}_k d = (\mathbf{I} + \hat{\mathbf{t}}_k [0 \ 0 \ d]) \mathbf{x}_0 = \hat{\mathbf{H}}_k \mathbf{x}_0,$$

where $\hat{\mathbf{t}}_k = \mathbf{H}_k^{-1} \mathbf{t}_k$ is the focus of expansion, and the new homography $\hat{\mathbf{H}}_k = \mathbf{I} + \hat{\mathbf{t}}_k [0 \ 0 \ d]$ represents a simple shift and scale. It can be shown [Col96] that the first two elements of $\hat{\mathbf{t}}_k$ depend on the horizontal and vertical displacements between the virtual camera and the k th camera, whereas the third element is proportional to the displacement in depth (perpendicular to the d plane). Thus, if all of the cameras are coplanar (regardless of their vergence), and if the d planes are parallel to this common plane, then the re-mappings of rectified images to a new disparity correspond to pure shifts.

Note that in the body of the paper, when we speak of the homography (14) parameterized by \mathbf{H}_k and \mathbf{t}_k , we can replace \mathbf{H}_k and \mathbf{t}_k by \mathbf{I} and $\hat{\mathbf{t}}_k$ if the input images have been pre-rectified.

B Gradient descent algorithm

To implement our gradient descent algorithm, we need to compute the partial derivatives of the cost functions $\mathcal{C}_1 \dots \mathcal{C}_3$ with respect to all of the unknowns, i.e., the colors and opacities $\hat{\mathbf{c}}(x, y, d)$. In this section, we will use $\hat{\mathbf{c}} = [r \ g \ b \ \alpha]^T$ to indicate the four-element vector of colors and opacities, and α to indicate just the opacity channel. In addition to computing the partial derivatives, we will compute the diagonal of the approximate Hessian matrix [PFTV92, pp. 681-685], i.e., the square of the derivative of the term inside the ρ function.

The derivative of \mathcal{C}_1 given in (9) can be computed by first expressing $\tilde{c}_k(u, v)$ in terms of $\bar{c}_k(u, v, d)$,

$$\begin{aligned}\tilde{c}_k(u, v) &= \sum_{d=d_{\min}}^{d_{\max}} \bar{c}_k(u, v, d) V_k(u, v, d) \\ &= \sum_{d'=d}^{d_{\max}} \bar{c}_k(u, v, d') V_k(u, v, d') + (1 - \bar{\alpha}_k(u, v, d)) \tilde{a}_k(u, v, d - 1),\end{aligned}$$

where

$$\tilde{a}_k(u, v, d) = \sum_{d'=d_{\min}}^d \bar{c}_k(u, v, d') V_k(u, v, d')$$

is the *accumulated color/opacity* in layer d , with $\tilde{c}_k(u, v) = \tilde{a}_k(u, v, d_{\max})$. We obtain

$$\frac{\partial \tau_k(u, v)}{\partial \tau_k(u, v, d)} = \frac{\partial g_k(u, v)}{\partial g_k(u, v, d)} = \frac{\partial b_k(u, v)}{\partial b_k(u, v, d)} = V_k(u, v, d)$$

and

$$\frac{\partial \tilde{c}_k(u, v)}{\partial \tilde{\alpha}_k(u, v, d)} = [0 \ 0 \ 0 \ V_k(u, v, d)]^T - \tilde{a}_k(u, v, d - 1).$$

Let $\mathbf{e}_k(u, v) = \tilde{c}_k(u, v) - \mathbf{c}_k(u, v)$ be the color error in image k , and assume for now that $w_k = 1$ and $\rho_1(\mathbf{e}_k) = \|\mathbf{e}_k\|^2$ in (9). The gradient of \mathcal{C}_1 w.r.t. $\tilde{c}_k(u, v, d)$ is thus

$$\tilde{\mathbf{g}}_k(u, v, d) = V_k(u, v, d) \left(\mathbf{e}_k(u, v) - [0 \ 0 \ 0 \ \mathbf{e}_k(u, v) \cdot \tilde{a}_k(u, v, d - 1)]^T \right)$$

while the diagonal of the Hessian is

$$\tilde{\mathbf{h}}_k(u, v, d) = V_k(u, v, d) [1 \ 1 \ 1 \ 1 - \|\tilde{a}_k(u, v, d - 1)\|^2]^T.$$

Once we have computed the derivatives w.r.t. the *warped* predicted color values $\tilde{c}_k(u, v, d)$, we need to convert this to the gradient w.r.t. the disparity space colors $\hat{c}(x, y, d)$. This can be done using the transpose of the linear mapping induced by the backward warp \mathcal{W}_b used in (5). For certain cases (pure shifts), this is the same as warping the gradient $\tilde{\mathbf{g}}_k(u, v, d)$ and Hessian $\tilde{\mathbf{h}}_k(u, v, d)$ through the forward warp \mathcal{W}_f ,

$$\begin{aligned}\hat{\mathbf{g}}_1(x, y, d, k) &= \mathcal{W}_f(\tilde{\mathbf{g}}_k(u, v, d); \mathbf{H}_k + \mathbf{t}_k[0 \ 0 \ d]), \\ \hat{\mathbf{h}}_1(x, y, d, k) &= \mathcal{W}_f(\tilde{\mathbf{h}}_k(u, v, d); \mathbf{H}_k + \mathbf{t}_k[0 \ 0 \ d]).\end{aligned}$$

For many other cases (moderate scaling and shear), this is still a good approximation, so it is the approach we currently use.

Computing the gradient of \mathcal{C}_2 w.r.t. $\hat{c}(x, y, d)$ is much more straightforward,

$$\hat{\mathbf{g}}_2(x, y, d) = \sum_{(x', y', d') \in \mathcal{N}_4(x, y, d)} \rho_2(\mathbf{c}(x', y', d') - \mathbf{c}(x, y, d)),$$

where ρ_2 is applied to each color component separately. The Hessian will be a constant for a quadratic penalty function; for a non-quadratic function, the secant approximation $\dot{\rho}(r)/r$ can be used [SA96].

Finally, the derivative of the opacity penalty function \mathcal{C}_3 can easily be computed for $\phi = x(1-x)$,

$$\hat{\mathbf{g}}_3(x, y, d) = [0 \ 0 \ 0 \ (1 - 2\alpha(x, y, d))]^T.$$

To ensure that the Hessian is positive, we set $\hat{\mathbf{h}}_3(x, y, d) = [0 \ 0 \ 0 \ 1]^T$.

The gradients for the three cost functions can now be combined

$$\begin{aligned} \hat{\mathbf{g}}(x, y, d) &= \lambda_1 \sum_{k=1}^K \hat{\mathbf{g}}_1(x, y, d, k) + \lambda_2 \hat{\mathbf{g}}_2(x, y, d) + \lambda_3 \hat{\mathbf{g}}_3(x, y, d), \\ \hat{\mathbf{h}}(x, y, d) &= \lambda_1 \sum_{k=1}^K \hat{\mathbf{h}}_1(x, y, d, k) + \lambda_2 \hat{\mathbf{h}}_2(x, y, d) + \lambda_3 \hat{\mathbf{h}}_3(x, y, d), \end{aligned}$$

and a gradient descent step can be performed,

$$\hat{c}(x, y, d) \leftarrow \hat{c}(x, y, d) + \epsilon_1 \hat{\mathbf{g}}(x, y, d) / (\hat{\mathbf{h}}(x, y, d) + \epsilon_2). \quad (15)$$

In our current experiments, we use $\epsilon_1 = \epsilon_2 = 0.5$.¹³

¹³A more sophisticated Levenberg-Marquardt minimization technique could also be implemented by adding an extra stabilization parameter [PFTV92]. However, implementing a full Levenberg-Marquardt with off-diagonal Hessian elements would greatly complicate the implementation.

1996 Paul Debevec, Camillo Taylor, Jitendra Malik, UC Berkeley
Modeling and Rendering Architecture from Photographs



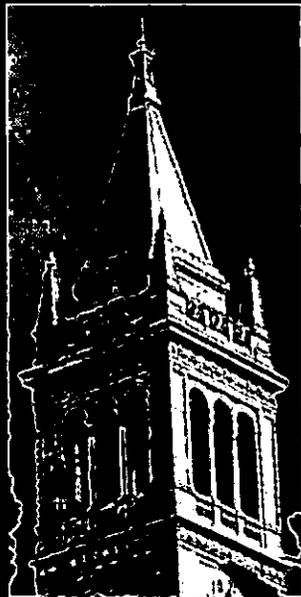
Modeling and Rendering Architecture from Photographs



Paul Debevec
Camillo Taylor
Jitendra Malik

*George Borshukov
Yizhou Yu*

Computer Vision Group
Computer Science Division
University of California at Berkeley



Photograph



Synthetic Rendering

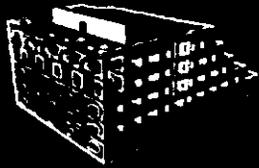
Geometry-Based Modeling and Rendering

■ Advantages:

- Produces freely navigable, easily rendered models

■ Disadvantages:

- Labor Intensive
- Not photorealistic



Soda Hall Walkthru Project
University of California at Berkeley

Giza Plateau Computer Model
University of Chicago

Amiens Cathedral
Columbia University

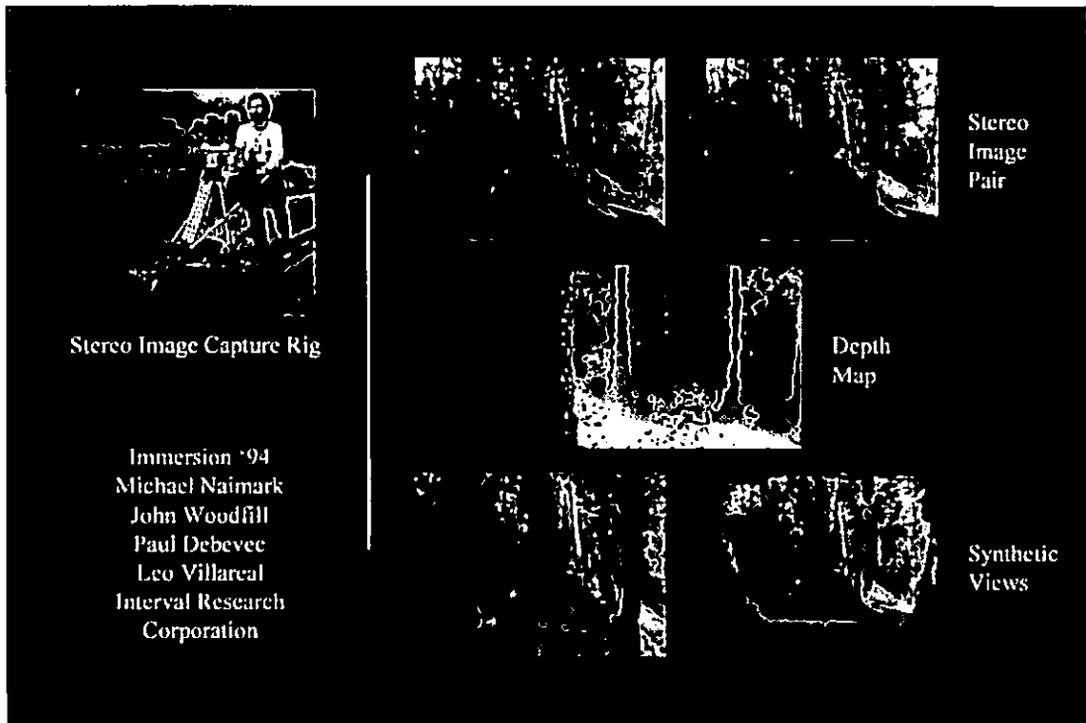
Image-Based Modeling and Rendering

- Chen & Williams '93
View Interpolation for Image Synthesis
- Laveau & Faugeras '94
3-D Scene Representation as a Collection of Images
- Szeliski '94
Image Mosaicing for Tele-Reality Applications
- McMillan & Bishop '95
Plenoptic Modeling

Image-Based Modeling and Rendering

- Advantages:
 - Photorealistic
 - Models any geometry
- Disadvantages:
 - Stereo is weak → Novel views must be near original photographs

1996 Paul Debevec, Camillo Taylor, Jitendra Malik, UC Berkeley
Modeling and Rendering Architecture from Photographs



Overview

- Photogrammetric Modeling
 - ⊙ Allows the user to construct a volumetric model of the scene directly from photographs
- View-Dependent Texture-Mapping
 - ⊙ Composites the original photographs onto the model
- Model-Based Stereo
 - ⊙ Recovers remaining geometric detail through stereo correspondence

Photogrammetric Modeling

■ Based on **Structure from Motion**:

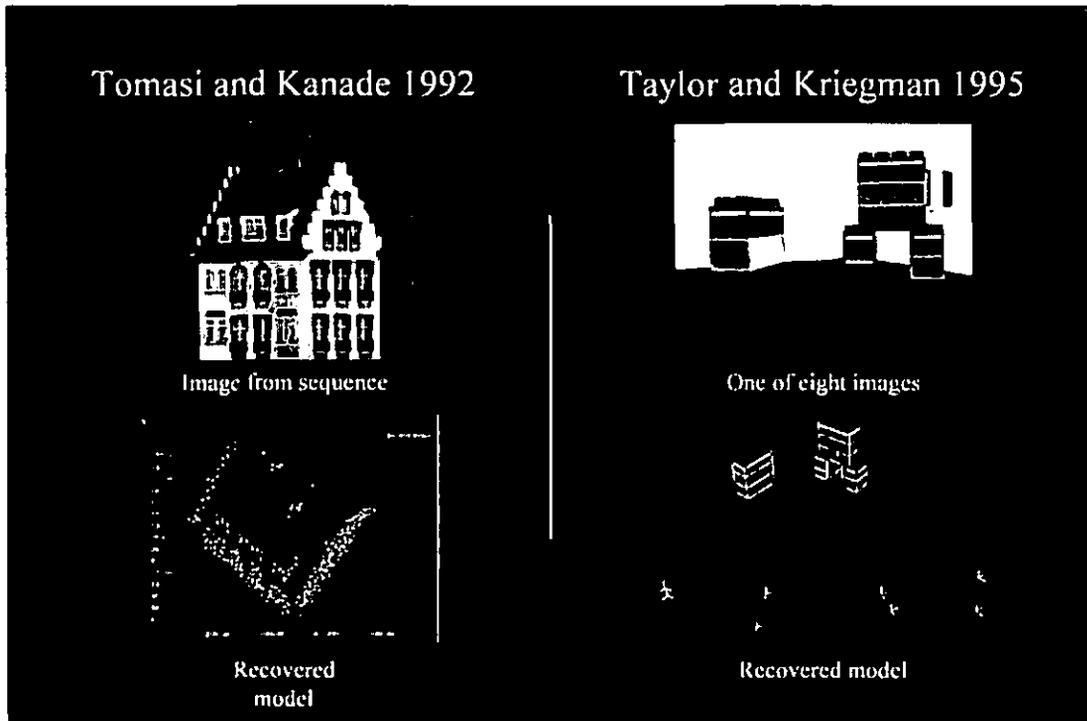
From a set of corresponding points (or lines) observed in several images, one can determine:

- Scene structure
- Camera positions

Structure from Motion: Limitations

- Requires many correspondences
- Difficult to solve
- Recovers only a cloud of points or lines

Solution: Take advantage of the nature of architectural scenes

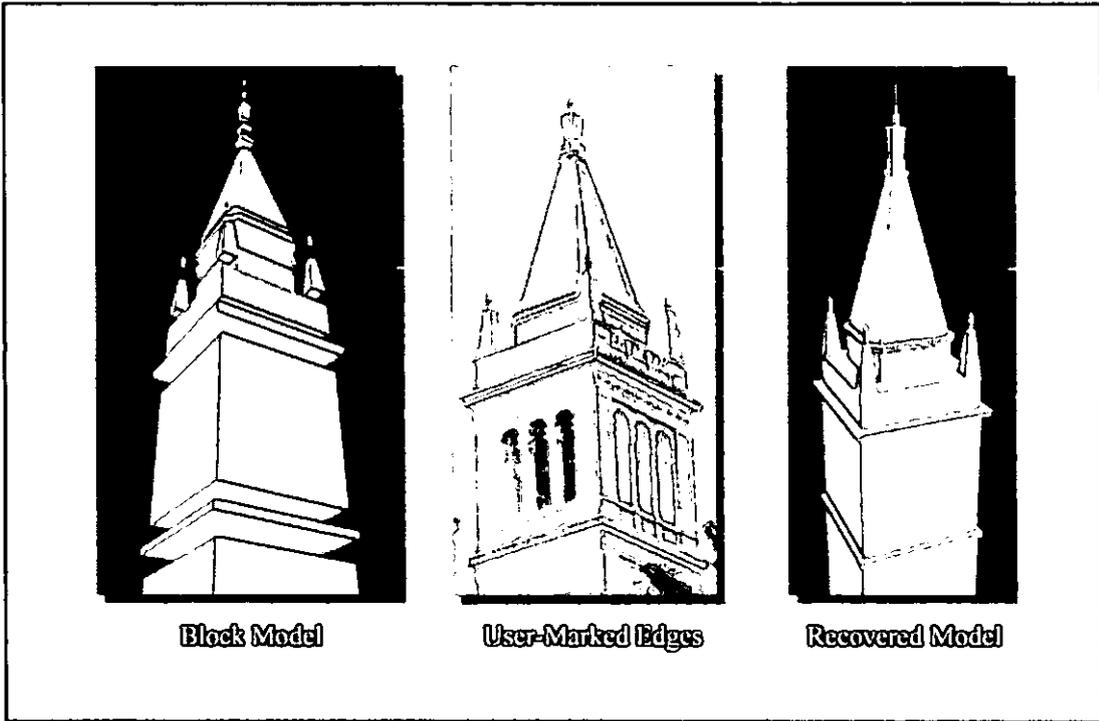
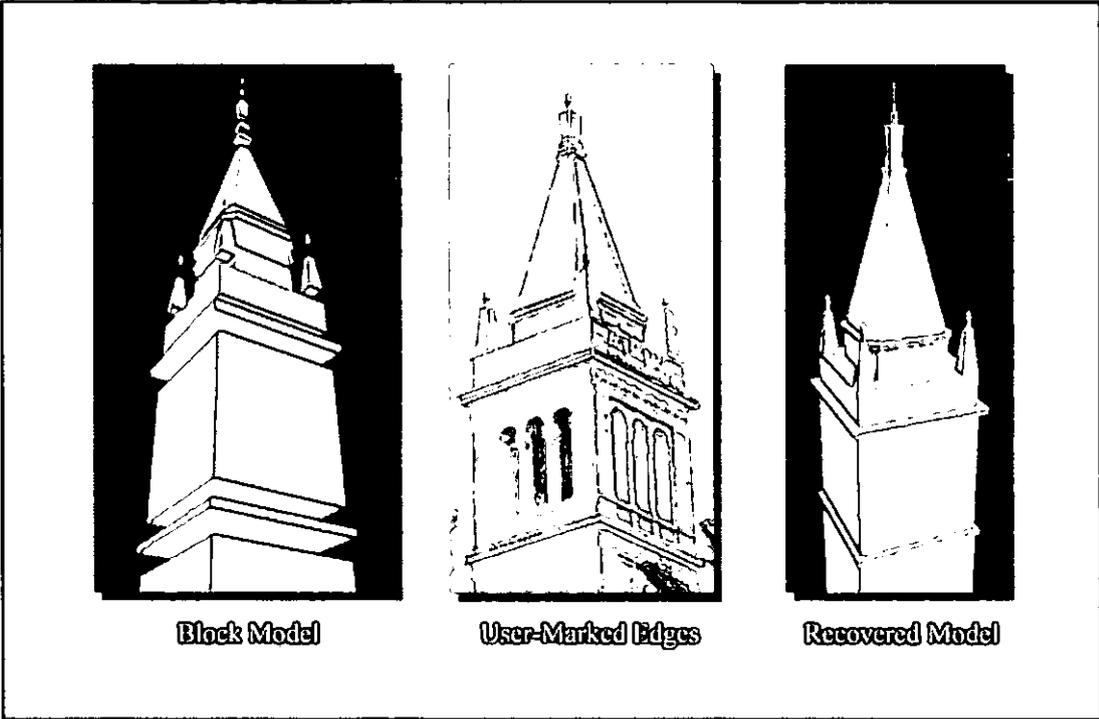


Our Modeling Method:

- **The user** represents the scene as a collection of blocks
- **The computer** solves for the sizes and positions of the blocks according to user-supplied edge correspondences

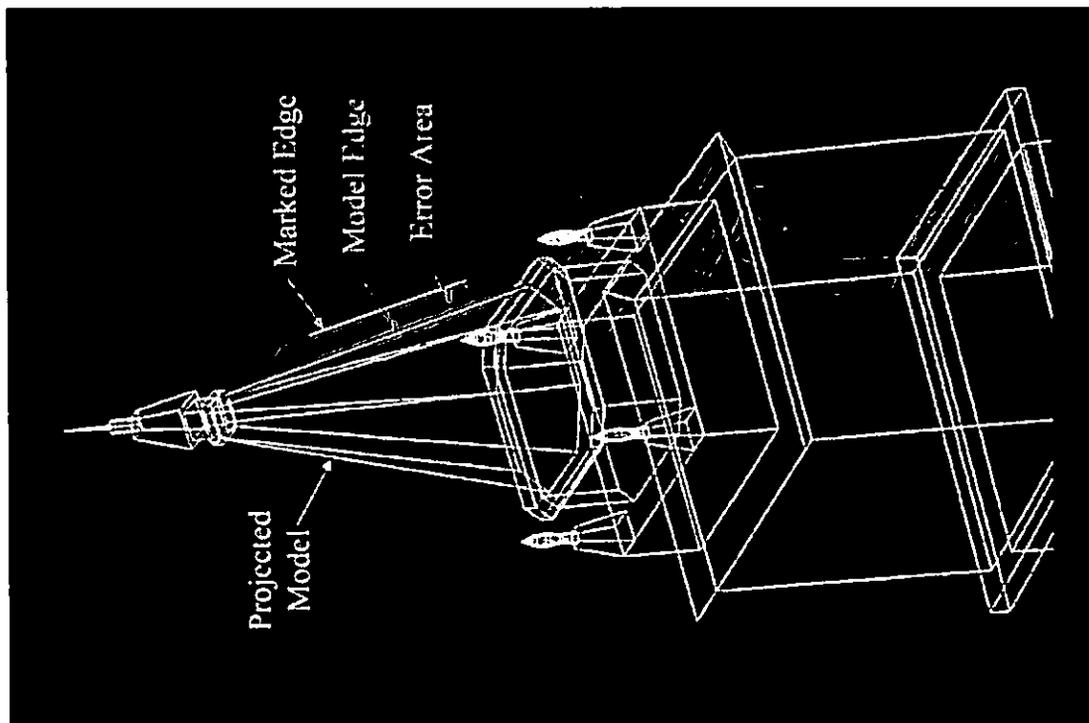
Related Work: Becker and Bove 1995

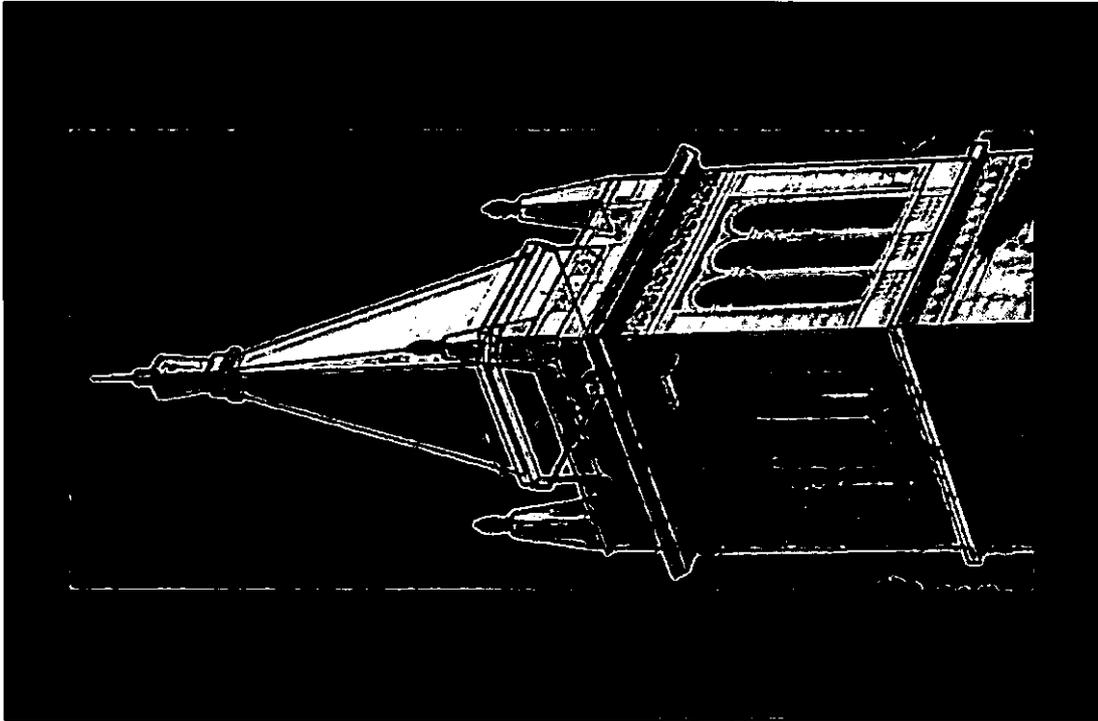
1996 Paul Debevec, Camillo Taylor, Jitendra Malik, UC Berkeley
Modeling and Rendering Architecture from Photographs



Reconstruction Algorithm

- An objective function O measures the misalignment between the marked edges and the corresponding projected edges of the model
- O is minimized with respect to the model parameters and camera positions
- An initial estimate is obtained by a separate procedure





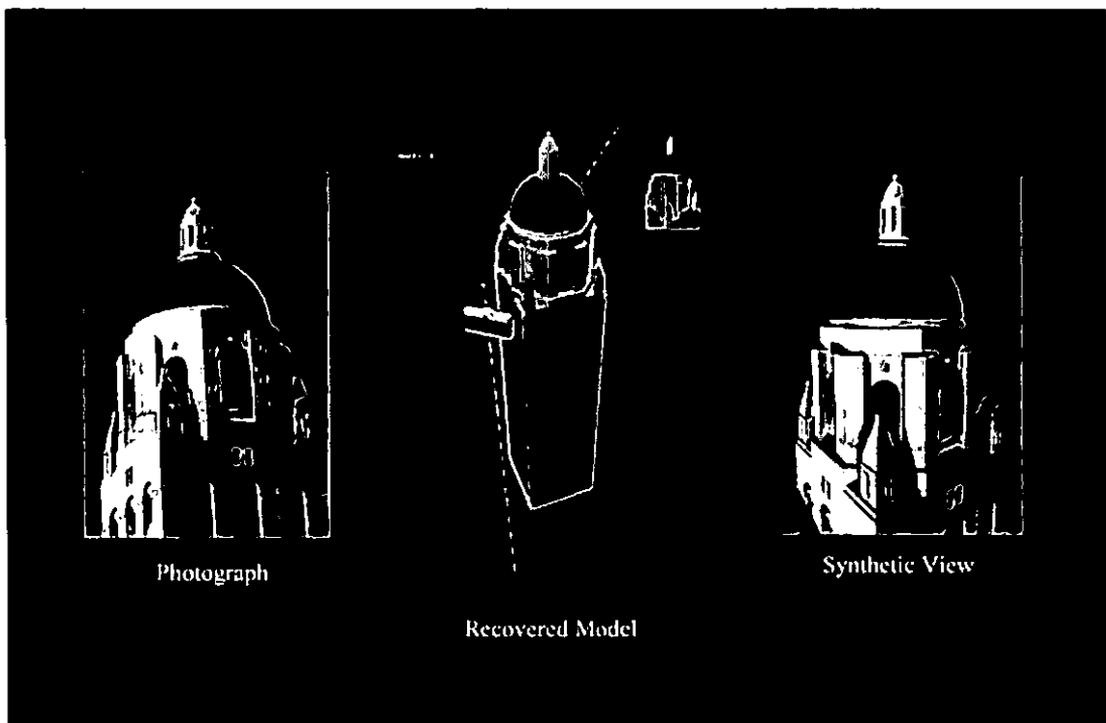
Photogrammetric Modeling Summary

Modeling with blocks works because:

- Convenient for architecture
 - Reduces number of model parameters, e.g.
The Campanile model has:
 - 2,896 parameters as independent edges
 - 240 parameters as independent blocks
 - 33 parameters as constrained blocks
- ⊙ → Few marked features required
 - → Easier to solve

Surfaces of Revolution

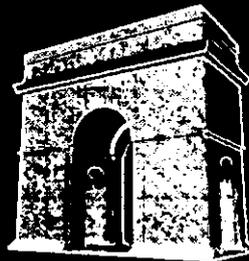
- Can be recovered from silhouette edges



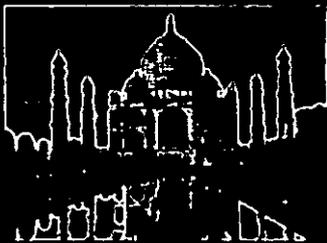


Arc de Triomphe

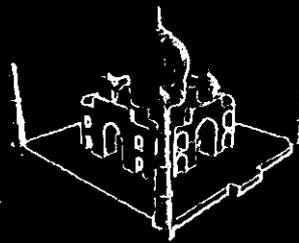
Modeled from five photographs
by George Borshukov



Surfaces of Revolution



Taj Mahal
modeled from
one photograph
by G. Borshukov



Projecting Images onto the Model

- Any image is easily projected since camera positions are known
- Occlusions determined with shadow buffer
- Multiple images composited to render entire model
- Areas never hit are filled in algorithmically

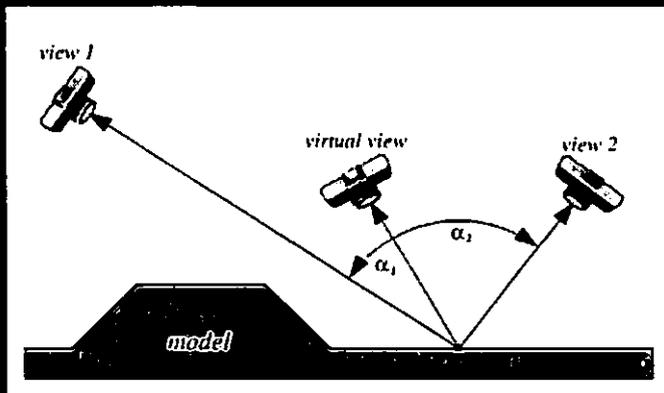


View-Dependent Texture Mapping

- When images overlap, which is best to use?

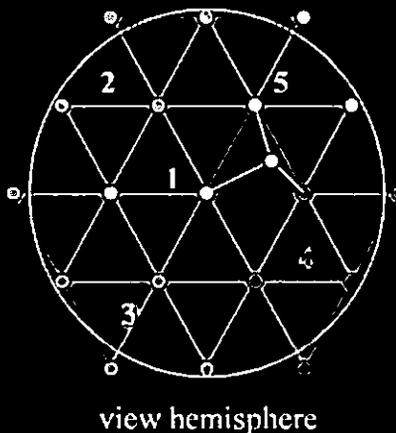
Answer depends on virtual viewpoint due to:

- Specular reflectance
 - Inaccuracies in the model
- Composite the best images using a view-dependent weighting function



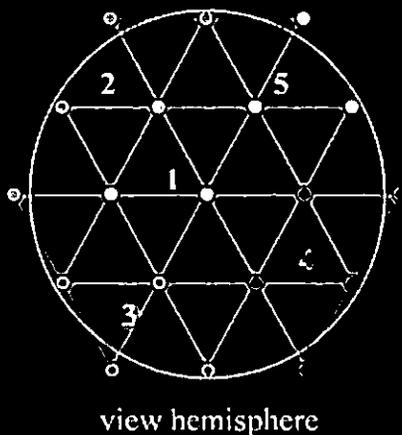
View-Dependent Weighting Function

Rendering with View-Dependent Texture Mapping



- To render, determine to which triangle the viewpoint belongs
- Compute Barycentric weights for the triangle vertices
- Render the polygon with a weighted average of the three vertex images

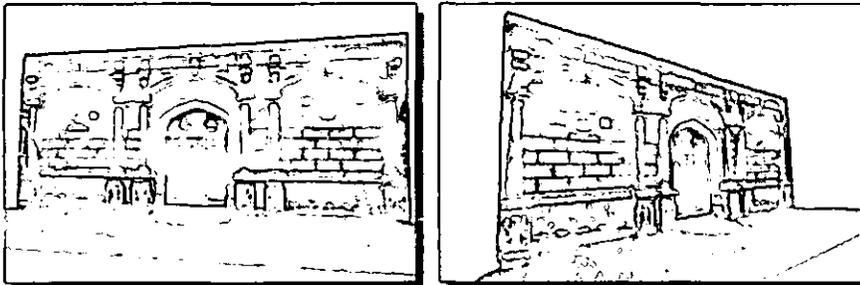
Rendering with View-Dependent Texture Mapping



- Triangulate the view hemisphere
- For each polygon, determine which images viewed it from which angles
- Label each triangle vertex according to best viewed image

Recovering Additional Detail with Model-Based Stereo

- ❑ Scenes will have geometric detail not captured in the model
- ❑ This detail can be recovered automatically through model-based stereo



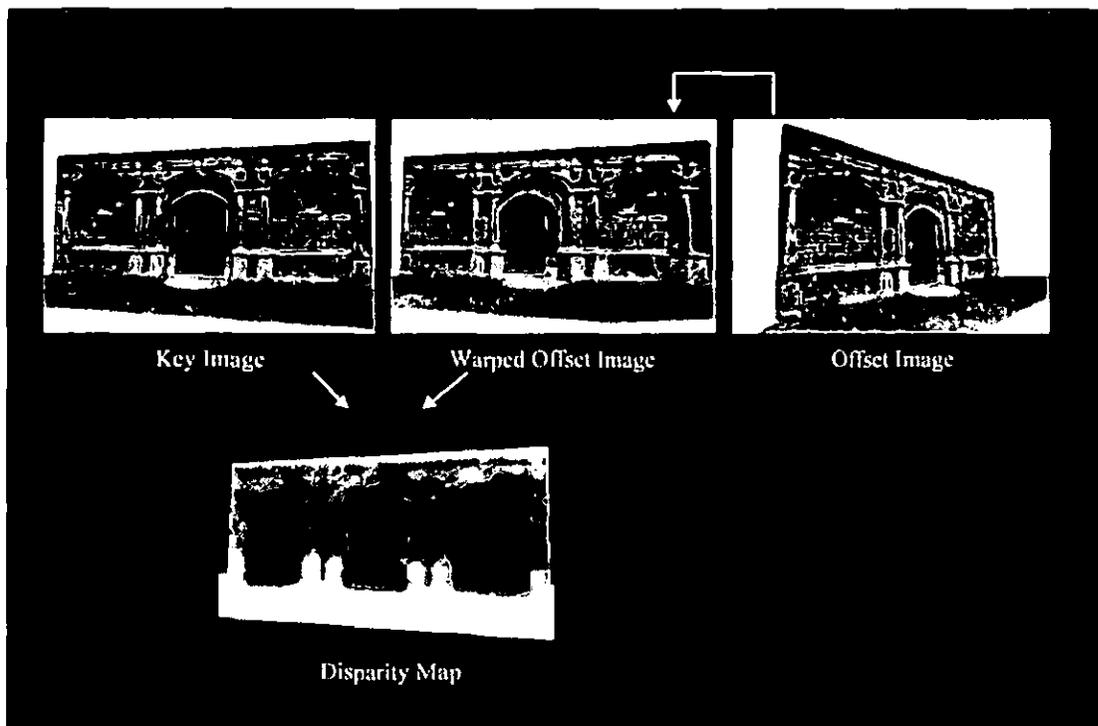
Scene with Geometric Detail



Approximate Block Model

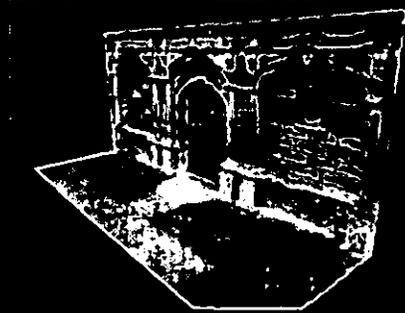
Model-Based Stereo

- Given a key and an offset image,
 - Project the offset image onto the model
 - View the model through the key camera
→ Warped offset image
- Stereo becomes feasible between key and warped offset images because:
 - Disparities are small
 - Foreshortening is greatly reduced



Synthetic Views of Refined Model

Four images composited with
View-Dependent Texture Mapping



Conclusion

- Basic models of architectural scenes can be recovered through photogrammetric modeling from a few photographs
- Remaining geometric detail can be recovered with model-based stereo
- Synthetic views can be rendered with view-dependent texture mapping

Sponsors

-  Interval Research Corporation
-  National Science Foundation
 - California MICRO Program
 - Joint Services Electronics Program

See also:

<http://www.cs.berkeley.edu/~debevec/Research>

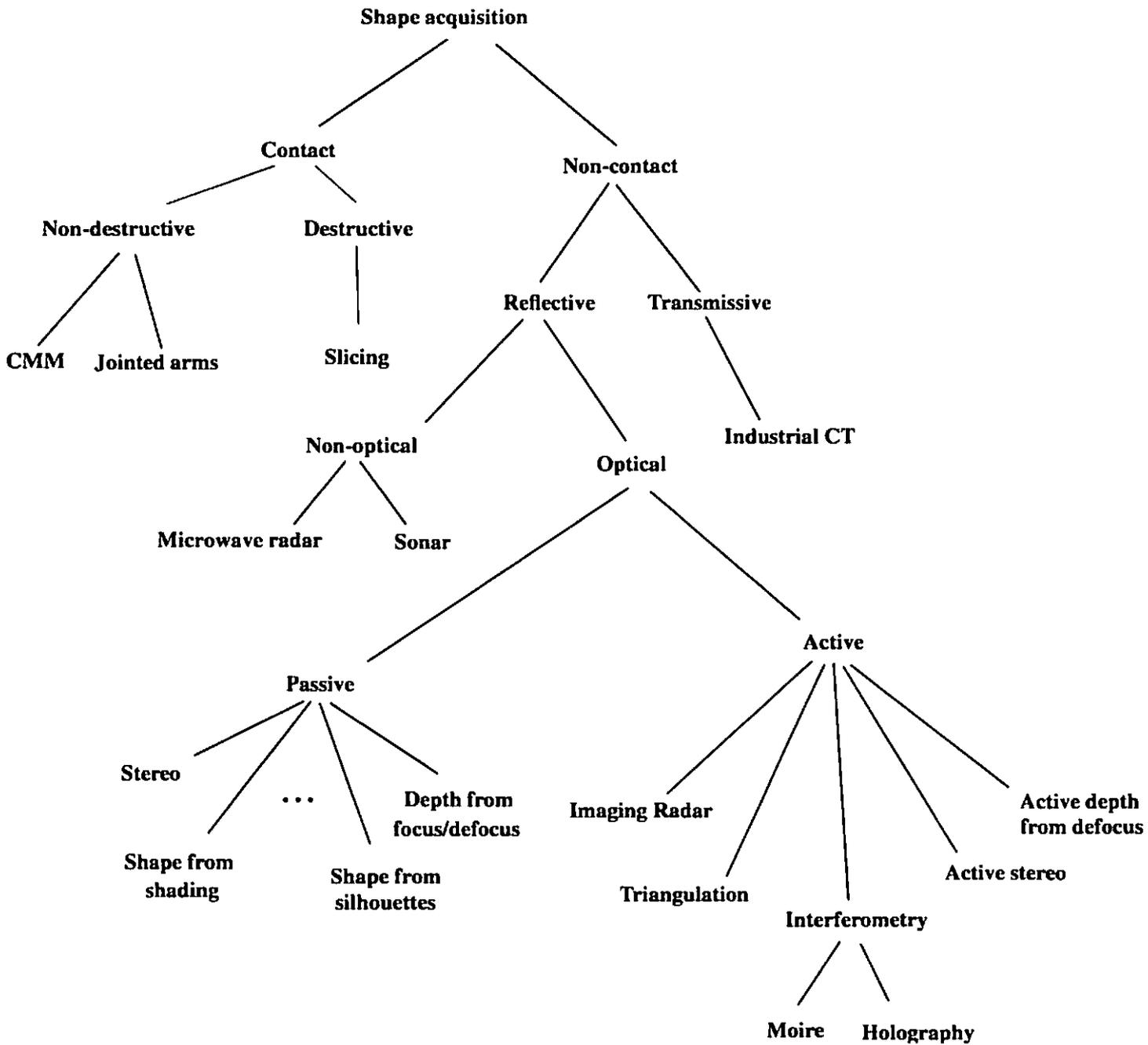
Active shape acquisition

Brian Curless
Univeristy of Washington
April 1998

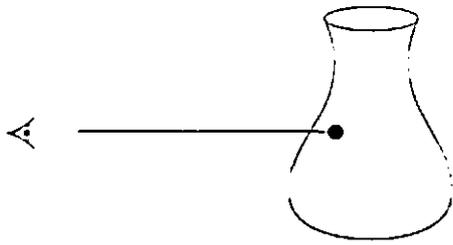
Outline:

1. Introduction
2. Active optical shape acquisition
 - a. Illumination
 - b. Imaging radar
 - c. Triangulation
 - d. Moire
 - e. Active stereo
 - f. Active depth-from-defocus
3. Contact methods
4. Industrial CT

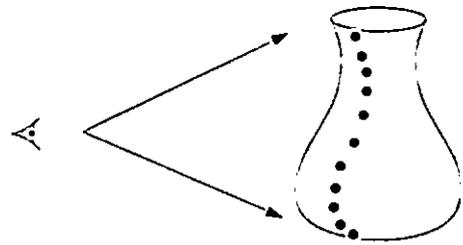
A taxonomy of shape acquisition methods



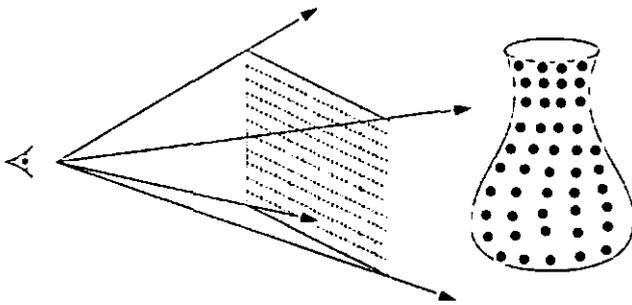
Structure of data



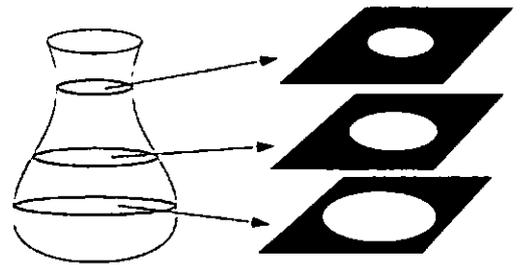
Point



Profile



Range image



Volumetric

Quality Measures

1. Resolution

- The smallest change in range or lateral position that the range sensor can report. How many bits of quantization? How far apart are samples?

2. Repeatability

- Statistical variations as a range sensor makes repeated measurements of the same point over time. Do the results tend to drift with usage?

3. Accuracy

- Statistical variations as a range sensor makes repeated measurements of a known true value. Usually characterized as standard deviation, σ . Manufacturers may report σ , 2σ , or 3σ . Specifications are frequently "not to be exceeded".

4. Environmental sensitivity

- Does accuracy vary with temperature? Wind speed?

5. Speed

- Points per second (seconds per point??)
- Off-line real-time
- True real-time
- What would you do with a real-time range imaging sensor?

Active optical shape acquisition

Strengths:

- non-contact
- safe
- can be inexpensive
- fast (many points per second).

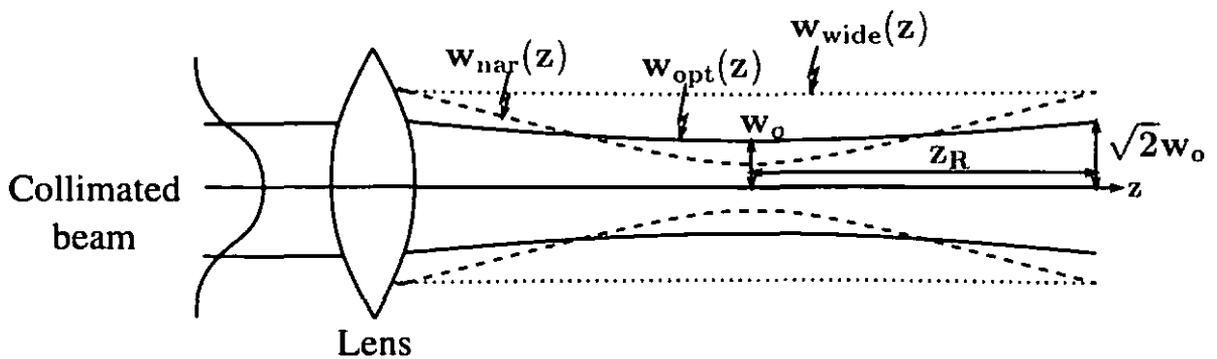
Limitations:

- Can only acquire visible portions of the surface
- Sensitive to surface properties:
 - transparency
 - shininess
 - abrupt changes in color
 - darkness (zero reflection)
- Can be confused by interreflections

Illumination

Why are lasers a good idea?

- Compact (laser diodes)
- Low power
- Single wavelength means the ability to discriminate laser from other lighting
- Single wavelength means no chromatic aberration
- Can be held in tight focus over long distance

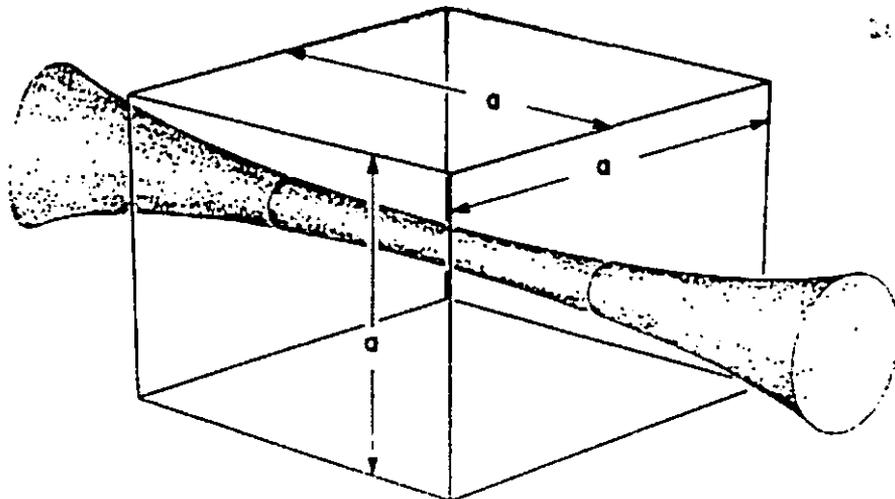


$$z_r = \frac{\pi w_0^2}{\lambda}$$

z_r = Rayleigh range (distance where $w = \sqrt{2}w_0$)

w_0 = beam waist (narrowest laser width)

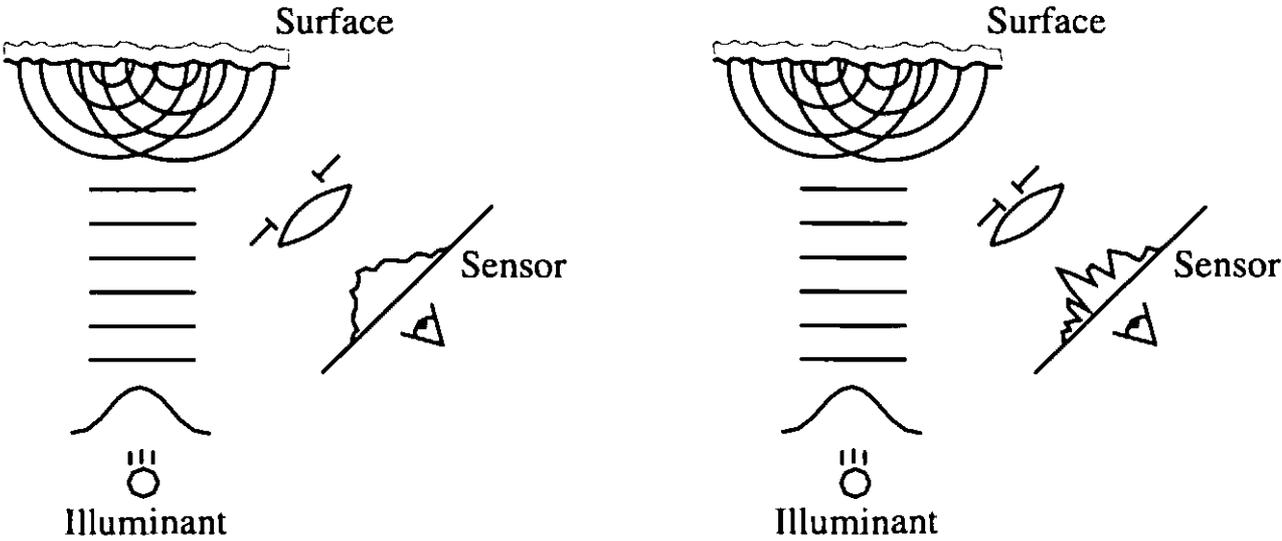
λ = wavelength of laser



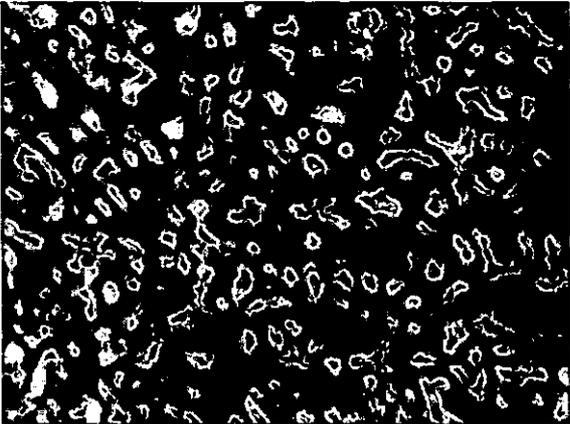
[Figure of 3D beam from Rioux 1987]

Limitations of lasers:

- Eye safety concerns
- Laser speckle adds noise – narrowing the aperture increases the noise



Speckle variation with aperture



[Image of speckle from Goodman 1984]

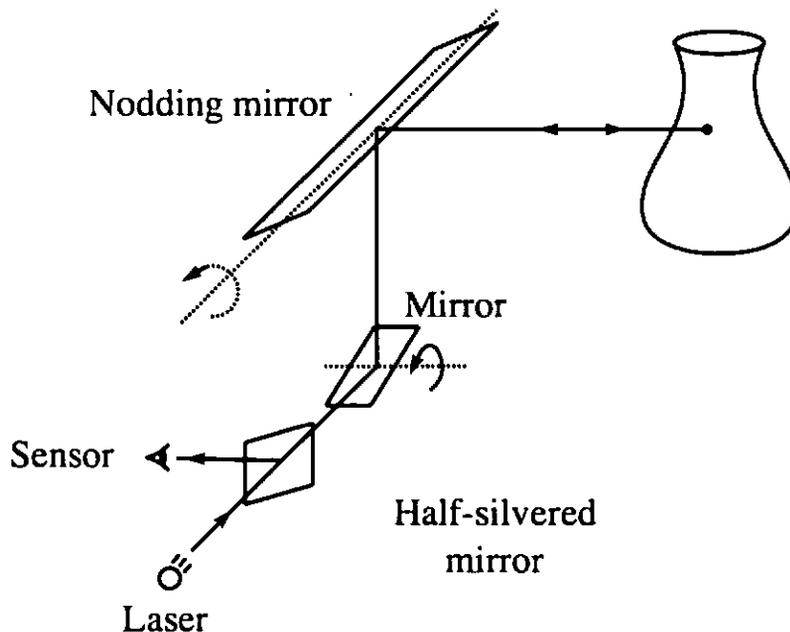
Imaging radar: Time of flight

Principle:

A pulse of light is emitted, and the time of the reflected pulse is measured

$$c t = 2 r = \text{roundtrip distance}$$

Typical scanning configuration is two rotating mirrors to derive a range image.



[The following statistics are for the *Cyra scanner*.]

Working volume:

50 meters range and about 60 degree field of view

Accuracy:

Aproximately 0.5 cm => 1 part in 10,000.

Notes:

- 0.5 cm accuracy requires timing resolution down to 10's of picoseconds.
- Limitation is in detecting absolute time of flight – does not scale down to smaller working volumes.
- show video...

Imaging radar: Amplitude Modulation (AM)

Principle:

A laser diode is driven at frequency:

$$f_{AM} = c / \lambda_{AM}$$

where c is the speed of light and λ_{AM} is the wavelength of the modulated signal. We can compute the range based on the phase difference between the outgoing and incoming signal:

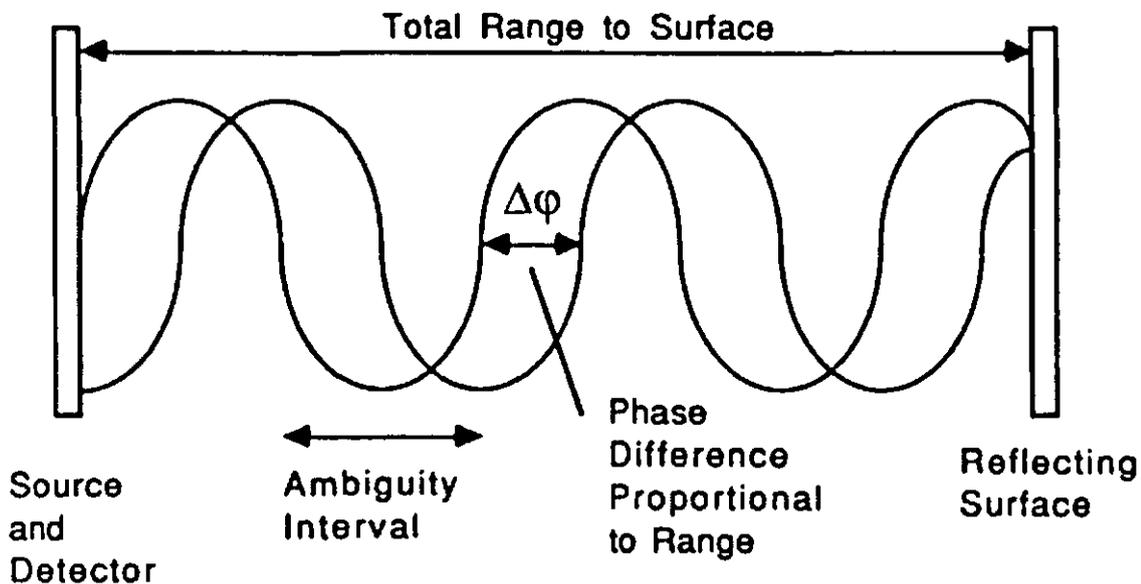
$$r(\Delta\phi) = \frac{1}{2} \lambda_{AM} (\Delta\phi \pm 2\pi n) / 2\pi$$

roundtrip correction wavelength of signal fraction of interval

Note the phase ambiguity due to the $\pm 2\pi n$. This translates to a range ambiguity:

$$r_{ambig} = n \lambda_{AM} / 2$$

Typical scanning configuration is two rotating mirrors to derive a range image.



[Figure from Besl 1989]

Imaging radar: AM (cont'd)

[The following statistics are for the *Perceptron scanner* as reported by R. Pito at U. Penn.]

Working volume:
0.6 m

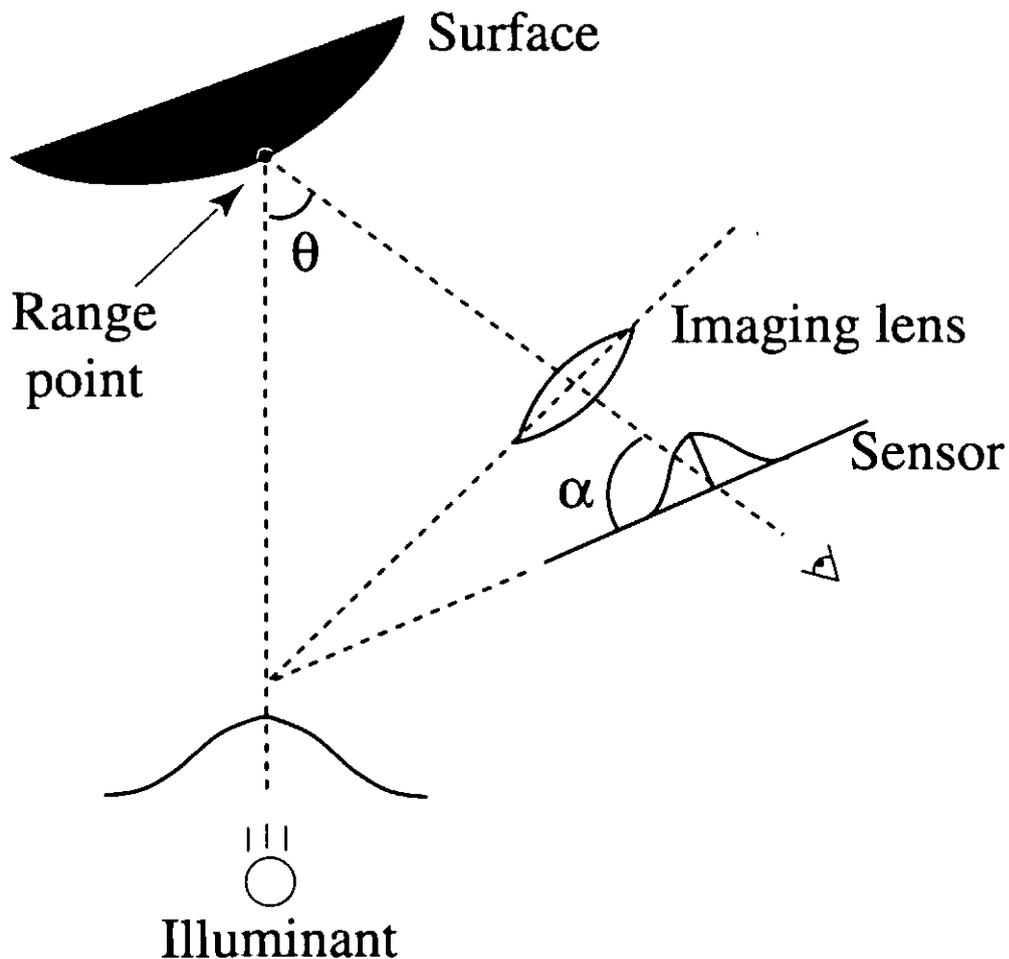
Accuracy:
1 mm => 1 part in 600

Notes:
– How can you resolve the phase ambiguity problem?

Optical triangulation

Principle:

A narrow beam of light strikes a surface, and the reflection is imaged by an off-axis sensor. The "center" of the imaged reflection approximately maps to the center of the illumination. The intersection of these two lines of sight correspond to a range point.



A lens maps a line to a line and a plane to a plane. If the line or plane is tilted, then the image plane is also tilted. The image tilt is related to the object tilt through the *Scheimpflug condition*:

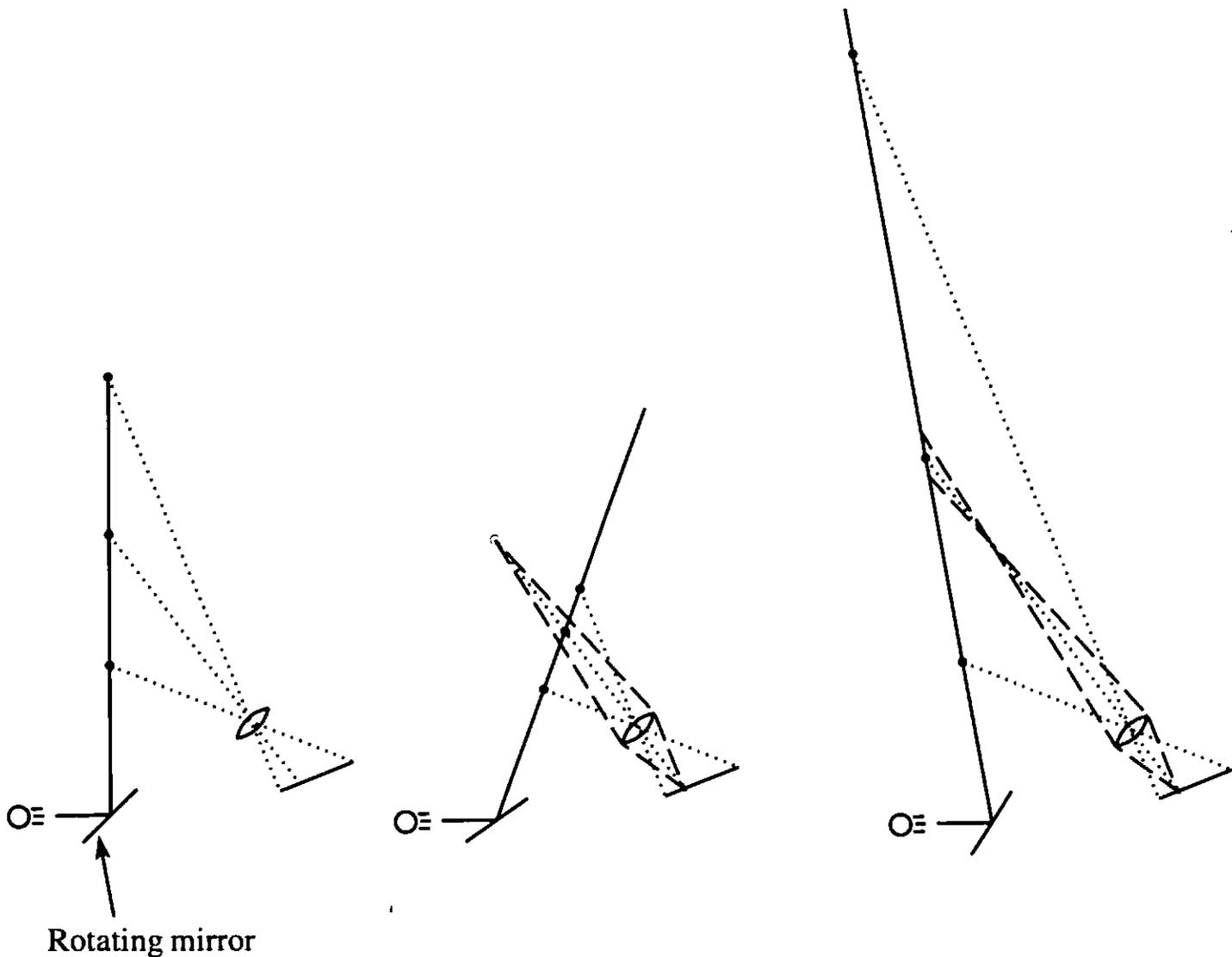
$$\tan\alpha = 1/M \tan\theta$$

where M is the on-axis magnification (d_o/d_i for a thin lens).

Optical triangulation: scanning configurations

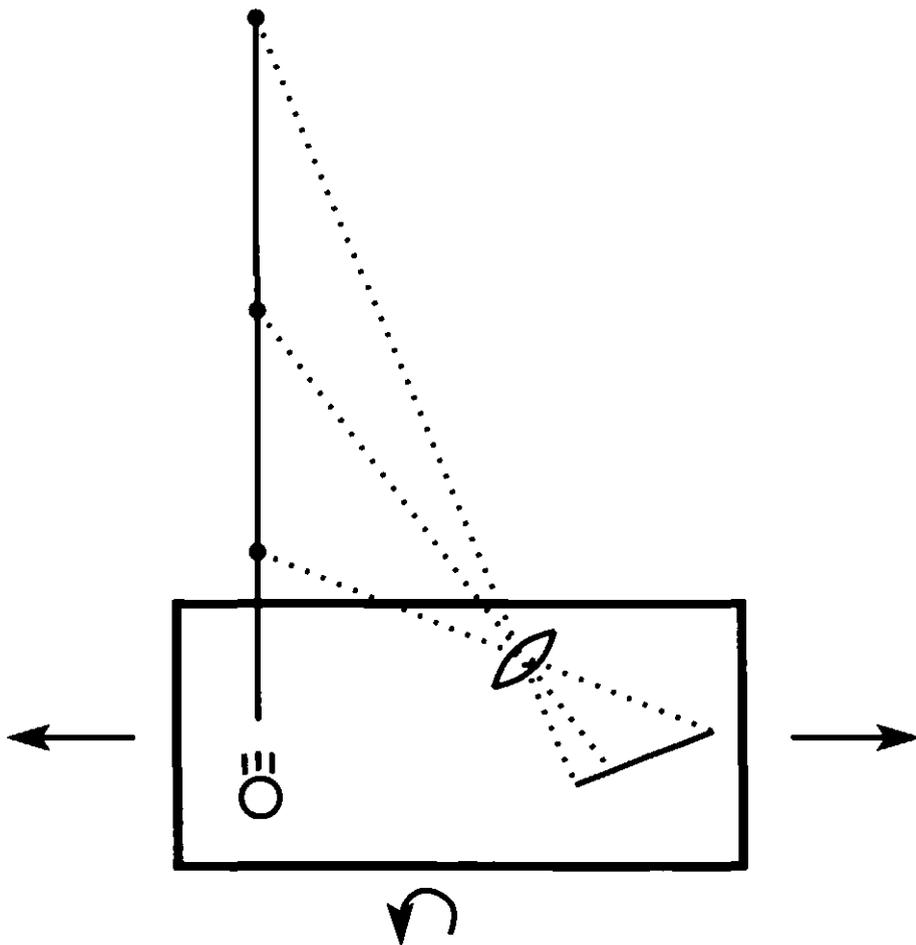
To scan a scene with a triangulation scanner, one approach is to keep the sensor stationary and sweep the laser across the scene. This approach has several problems:

- Loss of resolution due to defocus
- Field of view varies widely
- Resolution varies widely

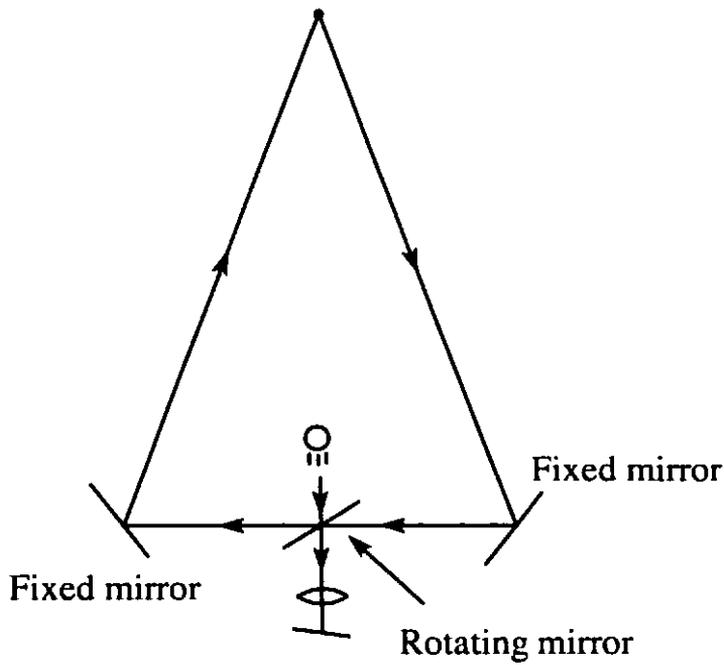


Instead of moving the laser independently of the sensor, we can move the laser and sensor as a unit. This will maintain focus and more consistent resolution.

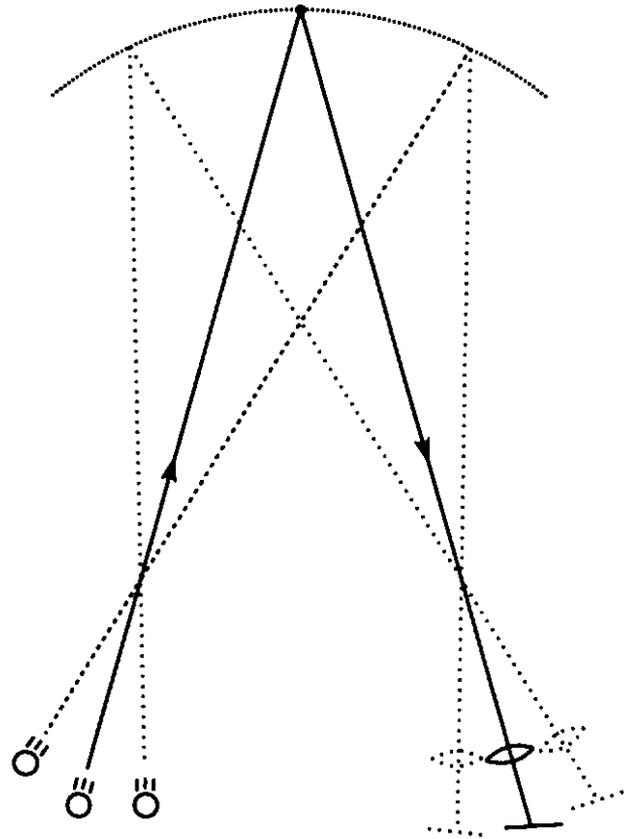
Scanning is achieved by translating and/or rotating the triangulation unit.



A novel design has been created (and patented) by Marc Rioux at the NRC in Canada. This design scans the laser and the sensor *simultaneously* over a scene.

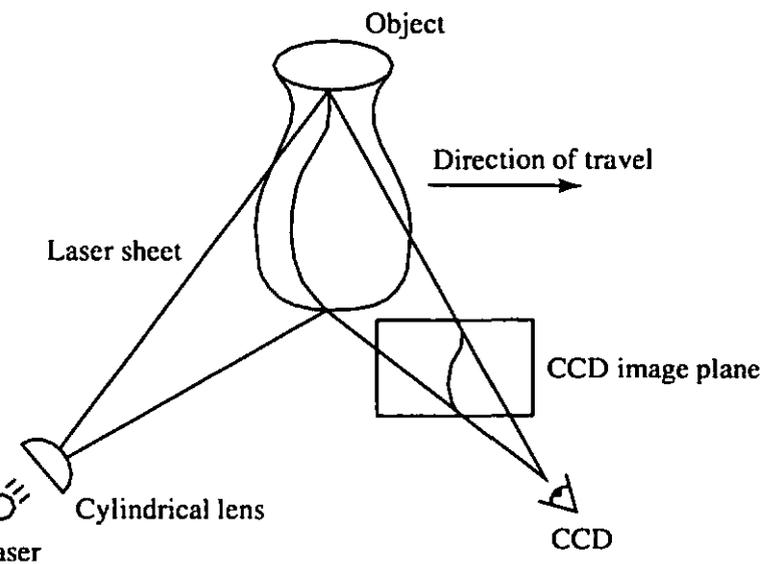


Rioux scanner

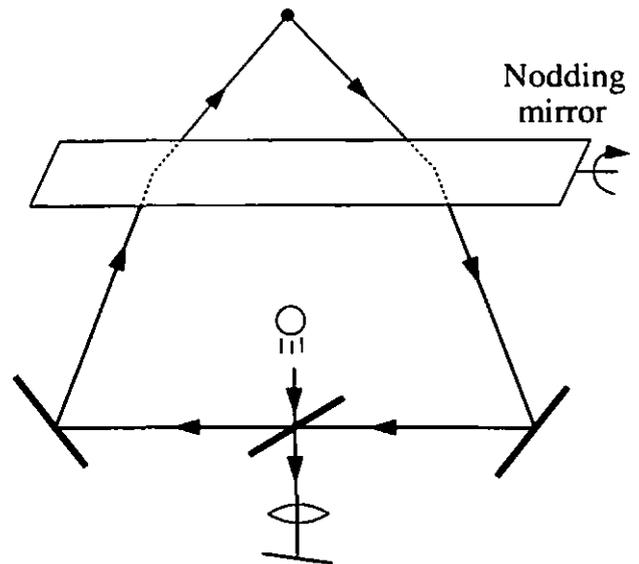


After unfolding the optics

Extension into 3D can be achieved by spreading the light beam into a plane of line to make a stripe scanner. Alternatively, a nodding mirror can be added to a flying spot scanner.



E.g., Cyberware scanner



Rioux scanner

Cyberware specs for MS3030

Working volume:
30 cm

Triangulation angle:
30 degrees

Accuracy:
0.3 mm => 1 part in 1000

Speed:
15,000 pts/sec

(show video)

Hymarc (Rioux) scanner specs:

Working volume:
8 cm

Triangulation angle:
~15 degrees

Accuracy:
50 microns => 1 part in 1500

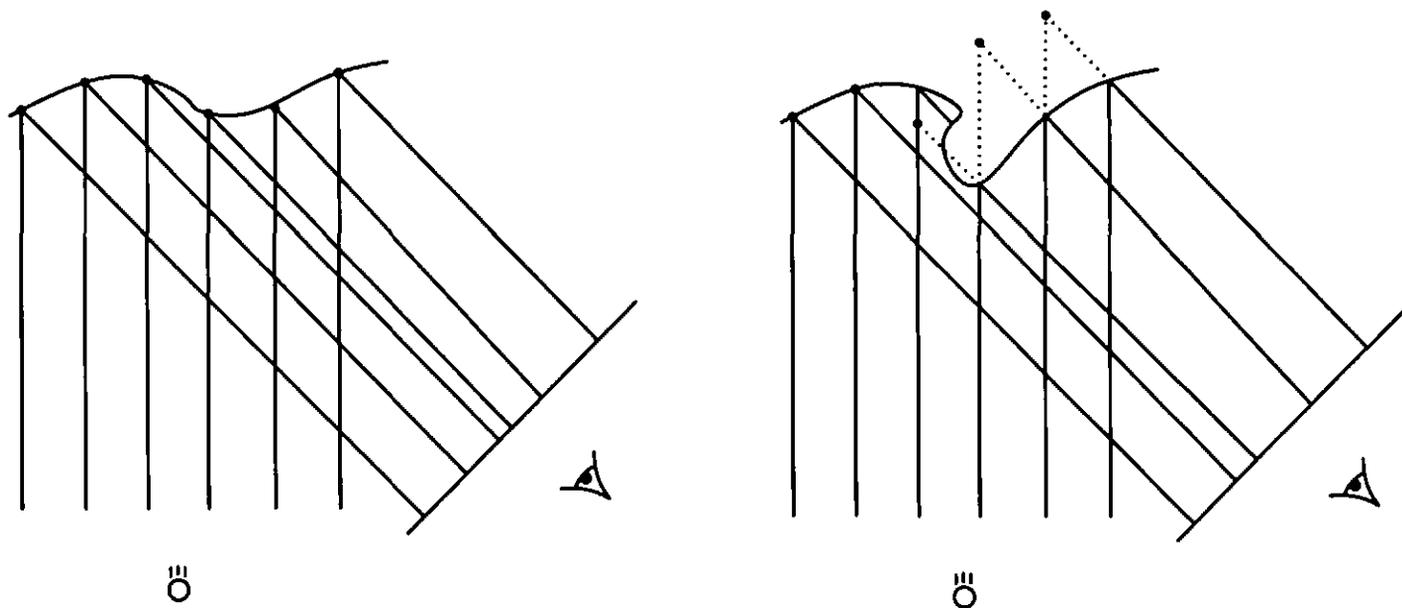
Speed:
10,000 pts/sec

Multi-spot and multi-stripe triangulation

To achieve faster acquisition, some scanners use multiple spots or stripes. Such systems have the same limitations as described with moving illuminant and stationary sensor.

Still, they are attractive because of the potential for acquiring many range points in a short period of time.

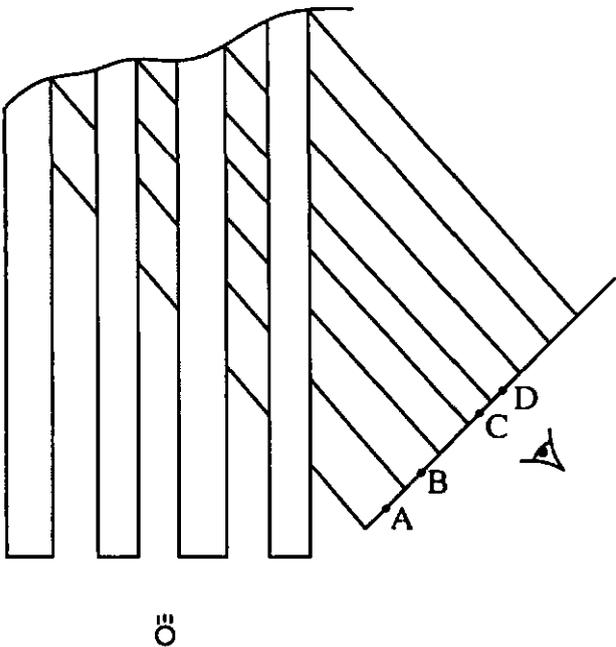
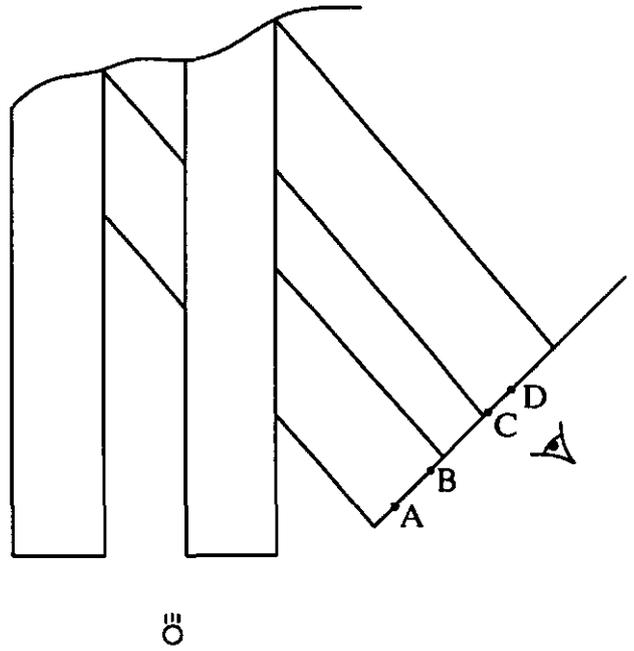
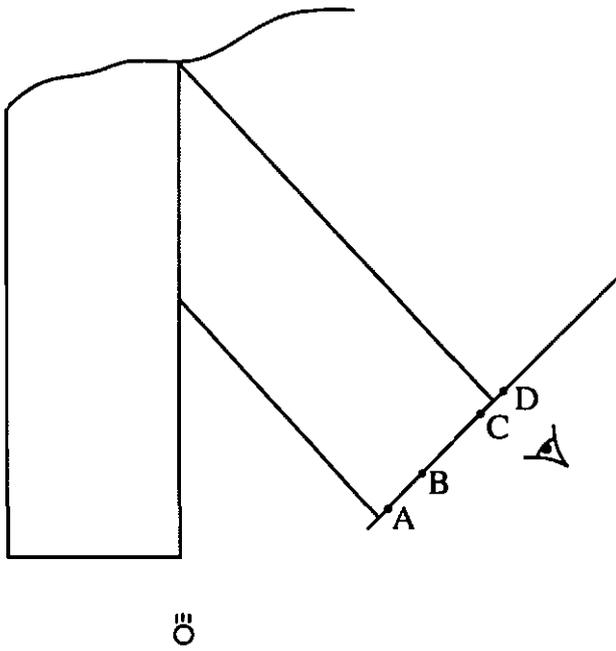
There is, however, one additional problem: ambiguity.



How can we resolve this ambiguity?

Binary coded multiple illumination

One approach to solving the ambiguity problem without projecting N stripes separately, is to project a binary pattern of light and dark regions. This method, illustrated below, resolves the ambiguity with $\log_2 N$ steps.



Binary codes:

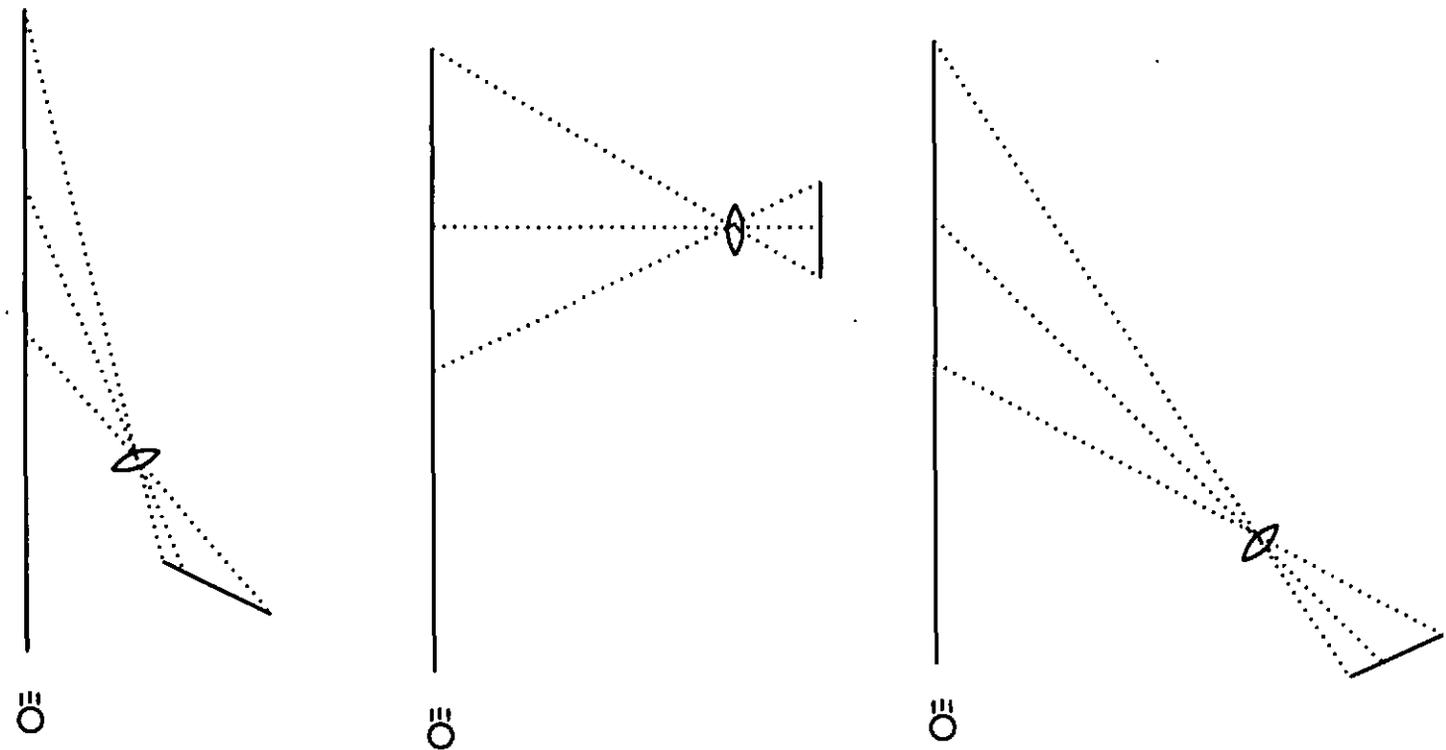
- A = 1 1 1
- B = 1 1 0
- C = 1 0 0
- D = 0 1 1

Notes on triangulation

Scalable to very small working volumes.

Not easily scalable to large working volumes – baseline requirements become prohibitive.

Smaller triangulation angles mean less self-occlusion, but smaller angles mean trading off depth resolution at the back of the field of view.

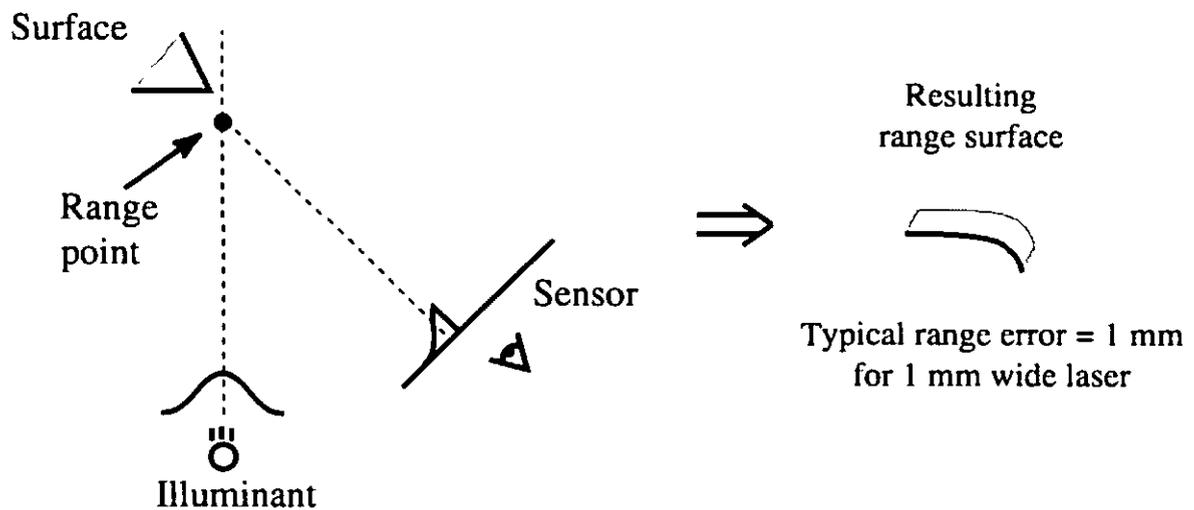


Triangulation Errors

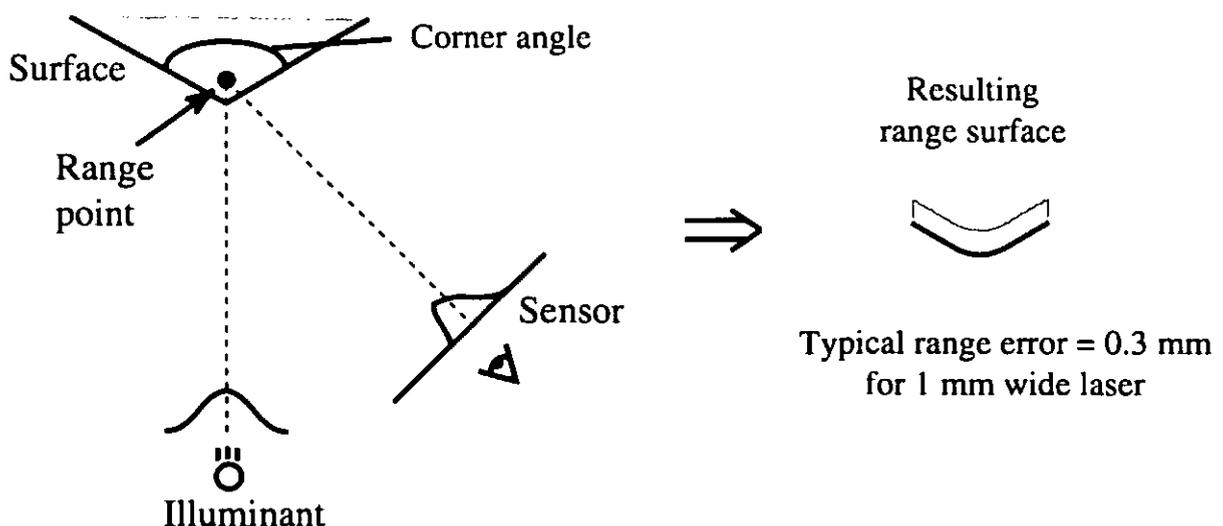
Finding the center of the imaged light is the tricky part. If light reflects off of a planar, diffuse surface, then results are accurate (up to speckle noise).

If the surface exhibits variations in reflectance and shape, then the accuracy is ultimately limited by the width of the laser sheet... unless *spacetime analysis* is applicable.

Edges

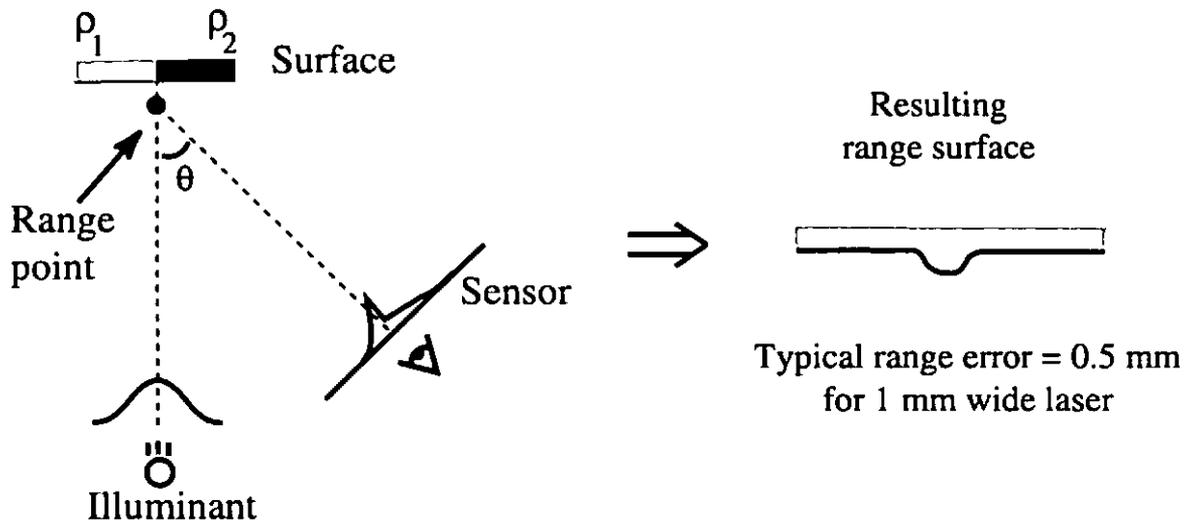


Shape variations

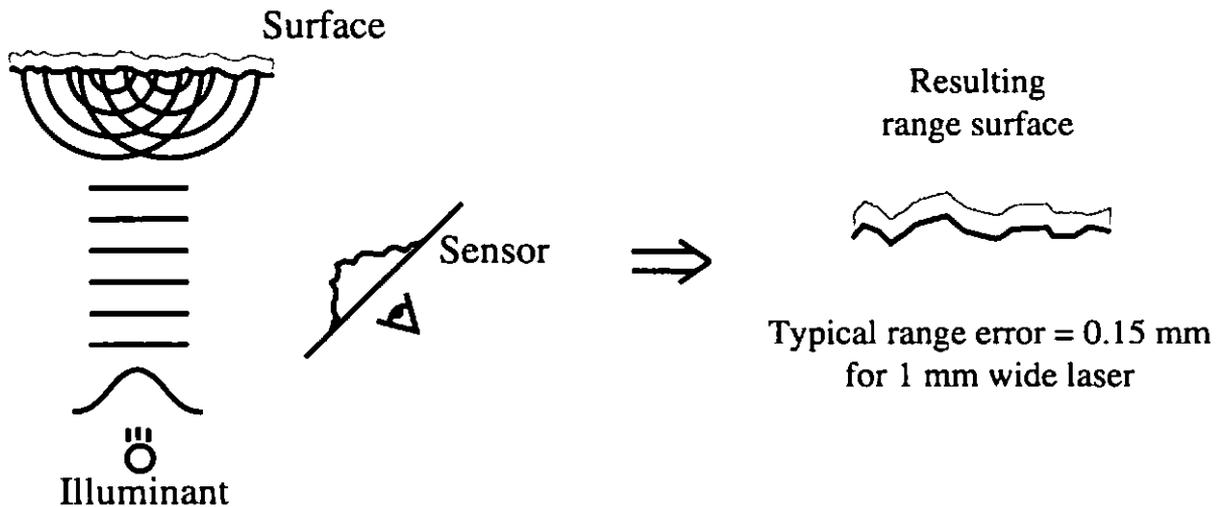


Triangulation Errors

Reflectance variations



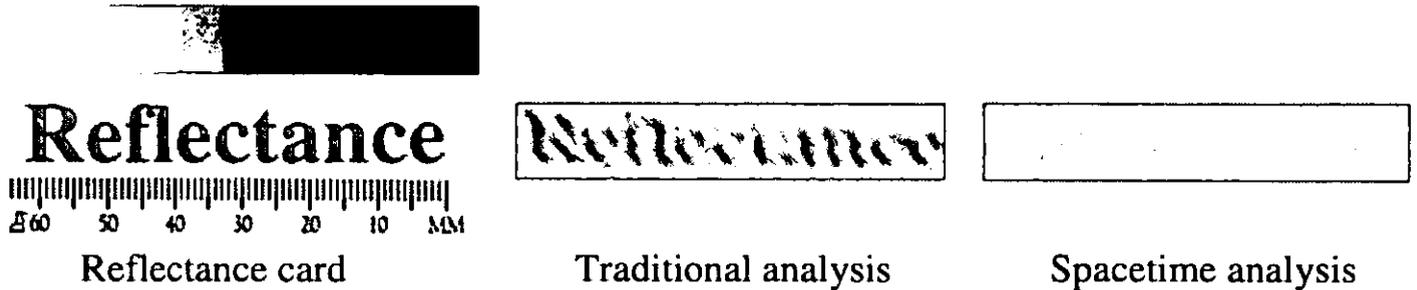
Laser speckle



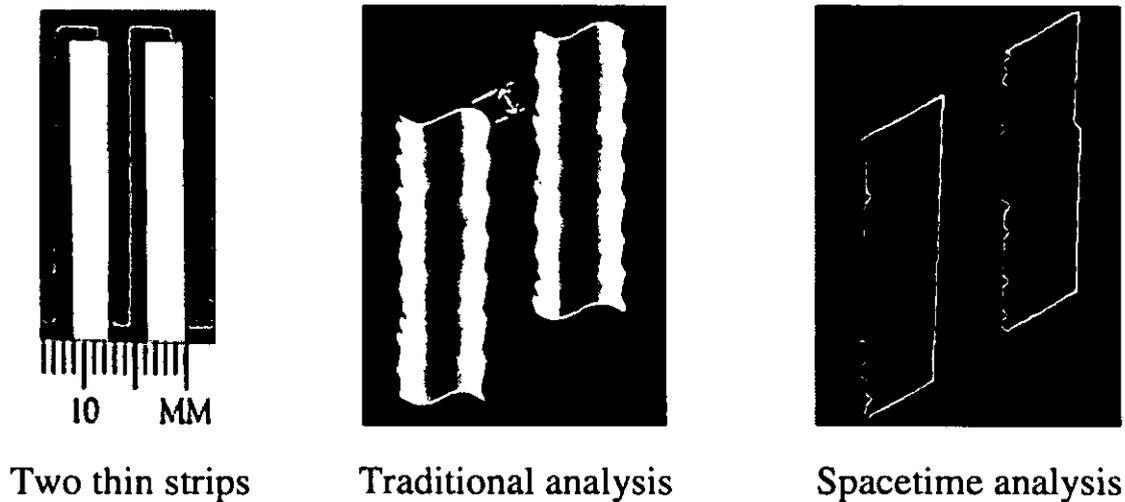
Errors and corrections through spacetime analysis

For triangulation *scanners*, the same point on an object is imaged multiple times across sensor frames. Using a method called *spacetime analysis*, it is possible to extract accurate shape by observing the time evolution of reflected light. See [Curless 1995] for details.

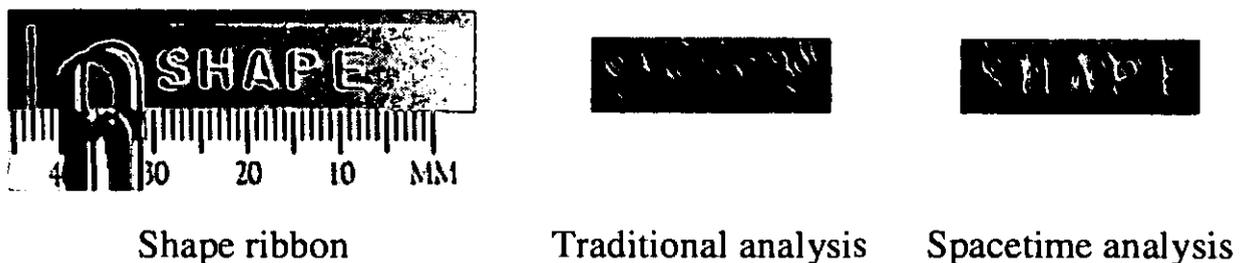
Reflectance correction



Edge curl reduction



Improved shape extraction



Moire

Principle:

Illuminate a surface at an angle with a periodic grating such as a sinusoid. Image the reflection from a different angle. The reflected image is of the form:

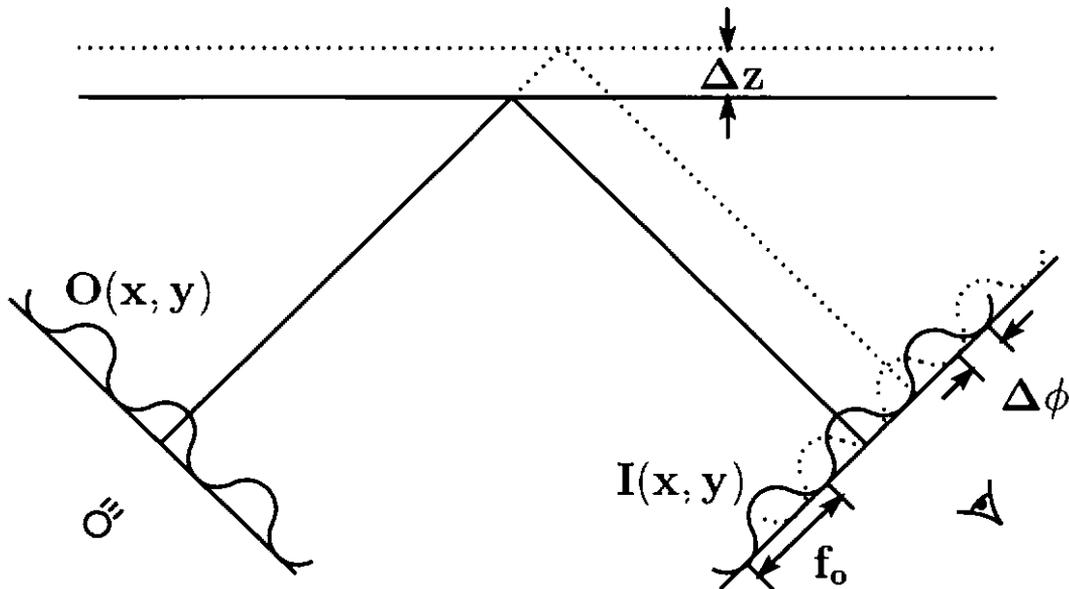
$$I(x, y) = a(x, y) \cos[2\pi f_0 x + \phi(x, y)]$$

where:

$a(x, y)$ is the amount of light reflected from the illuminant to the sensor, modulated by the illumination function.

f_0 is the period of the image of the grating when reflecting from a flat surface

$\phi(x, y)$ is the phase difference imparted by variations in the shape of the surface



It is easy to show that:

$$\Delta\phi(x, y) \sim \Delta z(x, y)$$

Quadrature Multiplicative Moire

To recover the shape, we can "demodulate" the signal; i.e., extract the phase. To do so, we multiply by a sinusoid of the same frequency:

$$\begin{aligned} I(x, y) \cos[2\pi f_0 x] \\ &= a(x, y) \cos[2\pi f_0 x + \phi(x, y)] \cos[2\pi f_0 x] \\ &= 1/2 a(x, y) \{ \cos[\phi(x, y)] + \cos[4\pi f_0 x + \phi(x, y)] \} \end{aligned}$$

Assuming $a(x, y)$ and $\phi(x, y)$ are slowly varying, we can low-pass filter (LPF) the image:

$$M_1(x, y) = \text{LPF}\{ I(x, y) \cos[2\pi f_0 x] \} = 1/2 a(x, y) \cos[\phi(x, y)]$$

Next, we multiply by a phase shifted sinusoid:

$$\begin{aligned} I(x, y) \sin[2\pi f_0 x] \\ &= a(x, y) \cos[2\pi f_0 x + \phi(x, y)] \cos[2\pi f_0 x + \pi/2] \\ &= 1/2 a(x, y) \{ \sin[\phi(x, y)] - \sin[4\pi f_0 x + \phi(x, y)] \} \end{aligned}$$

After filtering:

$$M_2(x, y) = \text{LPF}\{ I(x, y) \cos[2\pi f_0 x + \pi/2] \} = 1/2 a(x, y) \sin[\phi(x, y)]$$

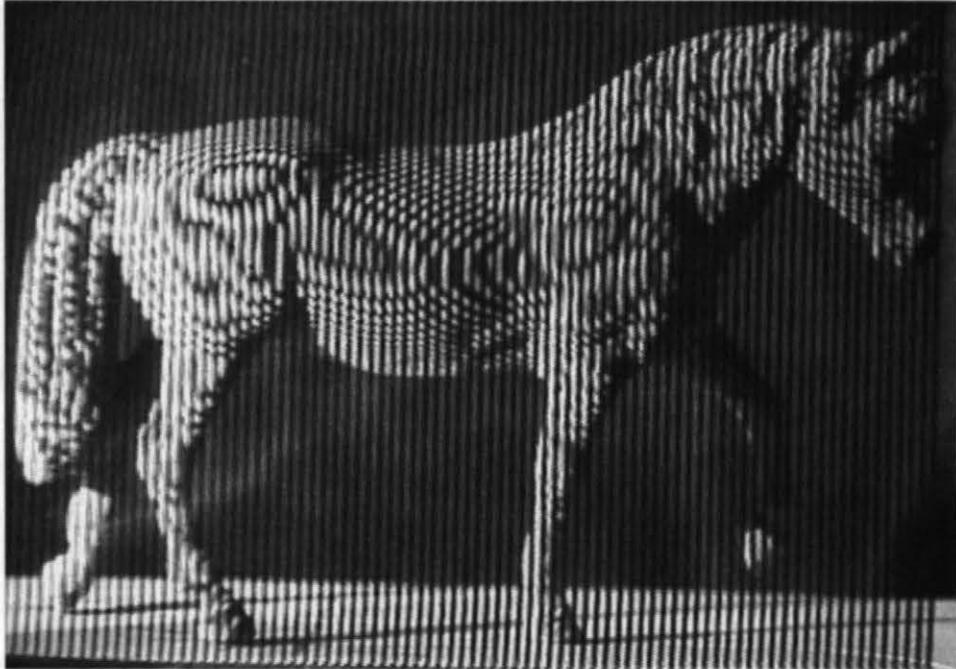
Thus, we have

$$\tan[\phi(x, y)] = M_1(x, y) / M_2(x, y)$$

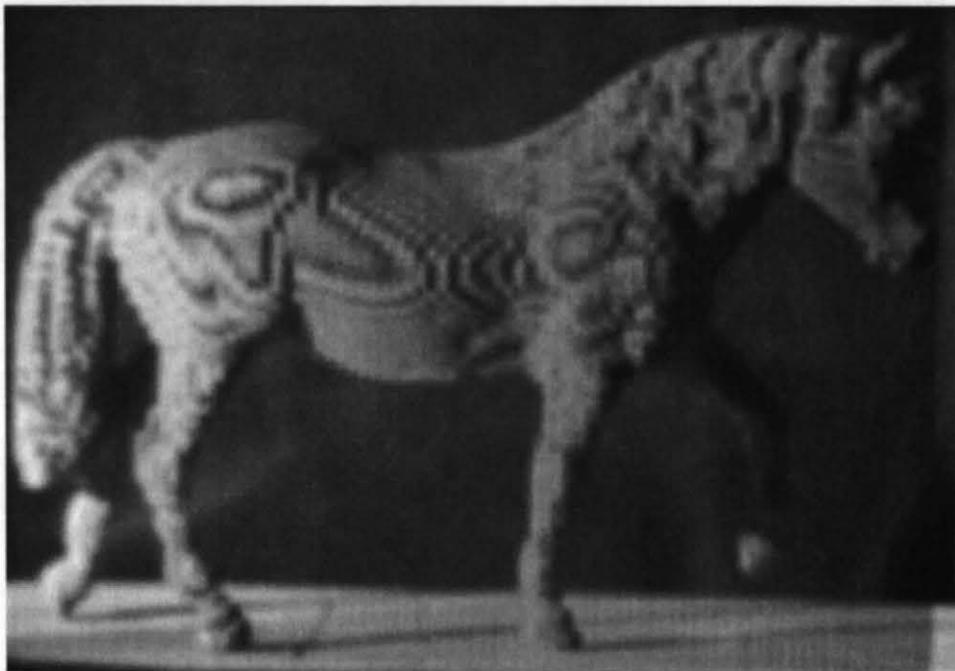
The phase corresponds to the range values to be extracted. In order for the phase term not to be filtered out, it cannot change rapidly. This requires that the surface be fairly smooth.

Example: Shadow Moire

We can place a grating near a surface and then illuminate it. When viewing it from a different angle, the grating is the demodulating signal and creates an interference pattern. Instant Moire!



Shadow Moire



Filtered to extract the phase term

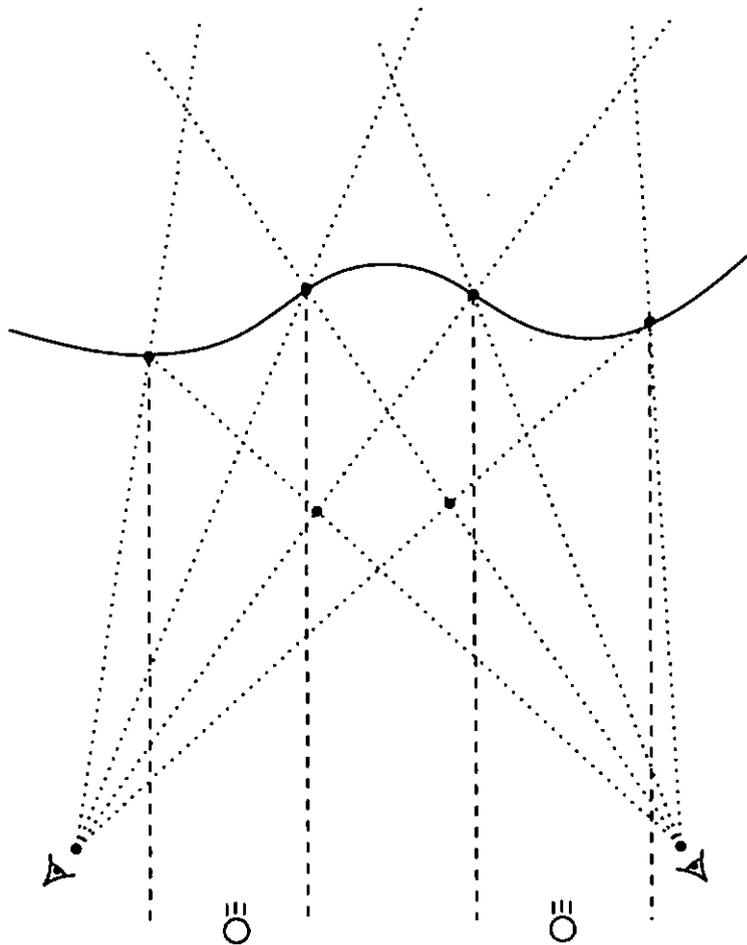
Active stereo

Principle:

Stereo methods use two offset cameras and match features in one image to features in another image. The lines of sight of matching features meet at a point on the surface.

Active stereo uses projected illumination to avoid the problem of finding textures.

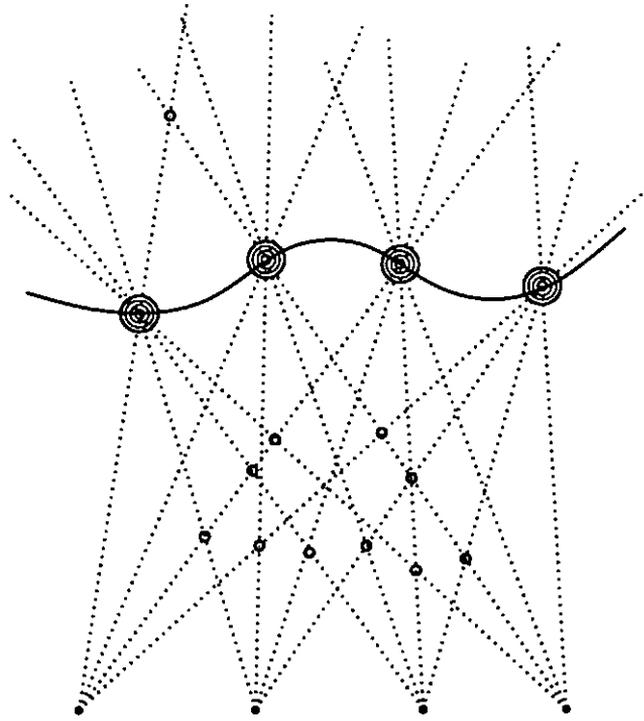
Distinguishing accurate matches from false matches is not always robust.



Note that for a stripe projected texture (shown here) we can limit matches to those sharing the same gradient directions on the image plane; i.e., don't match light-dark edges with dark-light edges.

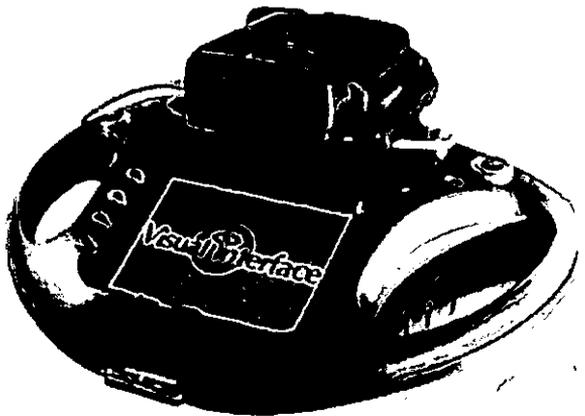
Active multi-baseline stereo

Using multiple cameras helps reduce the likelihood of false matches.



All possible matches

A new product has been released by Visual Interface, started by Jon Webb from CMU.



6 cameras
1 light stripe pattern projector
1 color camera

Working volume:
1 meter

Accuracy (typical):
1 mm => 1 part in 1000

[Image from Visual Interface web pages – <http://www.visint.com>]

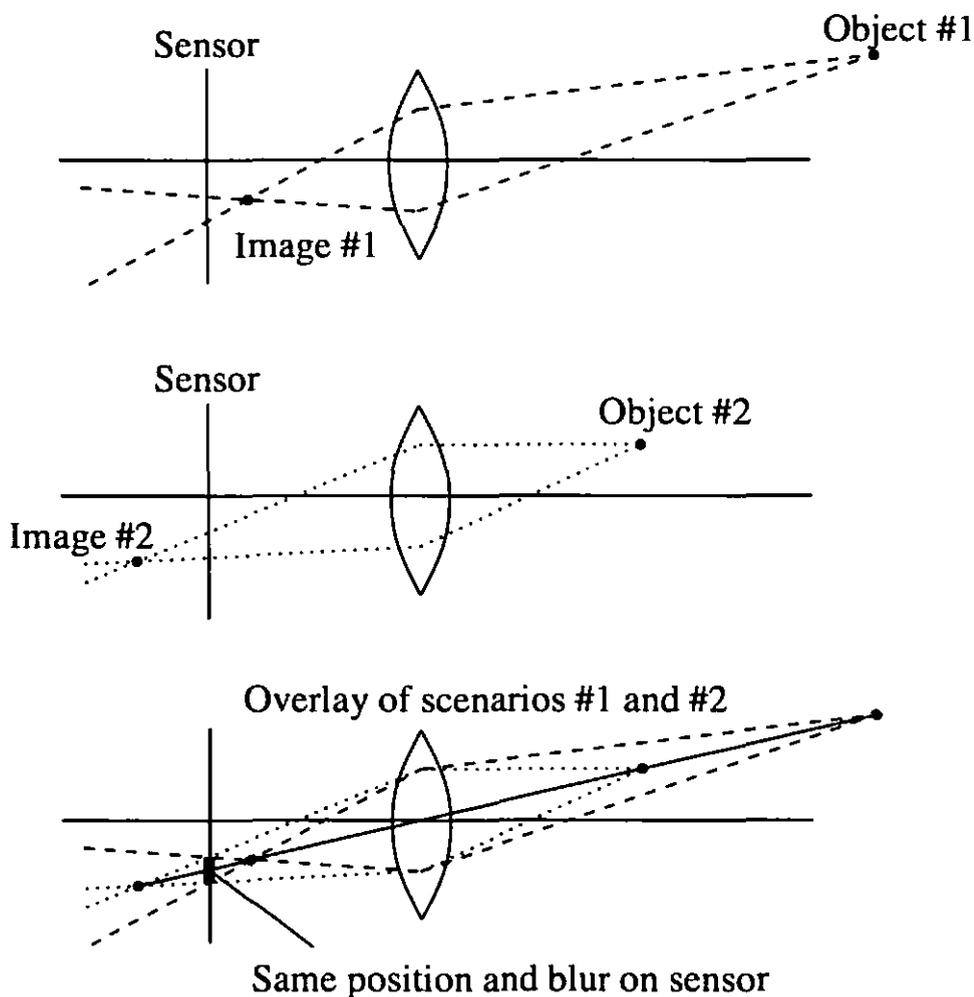
Active depth from defocus

Principle:

When an imaging sensor is placed in a plane other than the image plane corresponding to an object point, the image of that point will be blurred. The amount of blur corresponds to the distance to the object.

The blur arises from rays that are *converging* to an image point *behind* the sensor plane *or* from rays that are *diverging* from an image point *in front* of the sensor plane. These scenarios correspond to two different object points, so at least two sensor positions are needed to resolve the ambiguity.

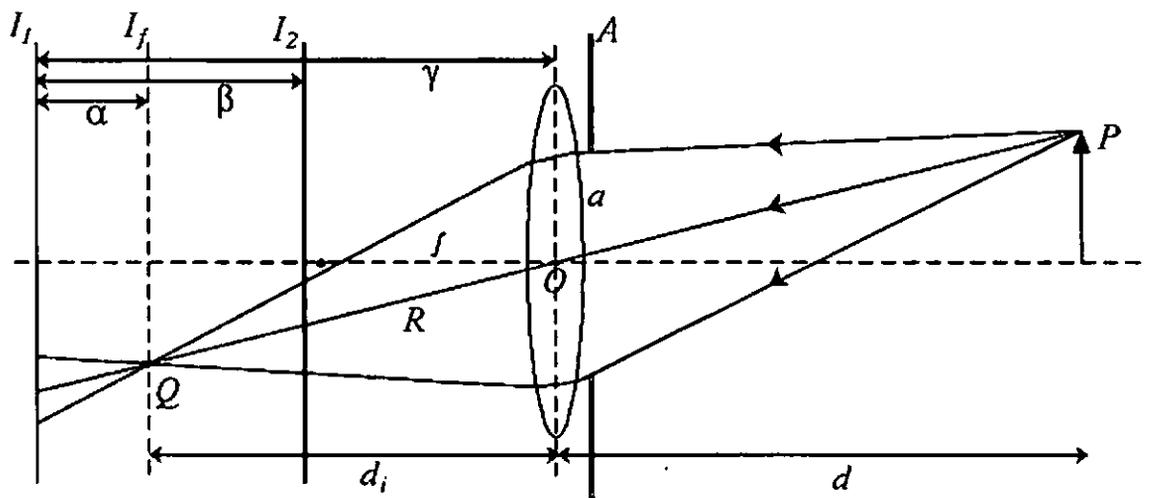
To avoid the problem of finding blurry textures already on the object, an active depth from defocus system projects light onto the scene.



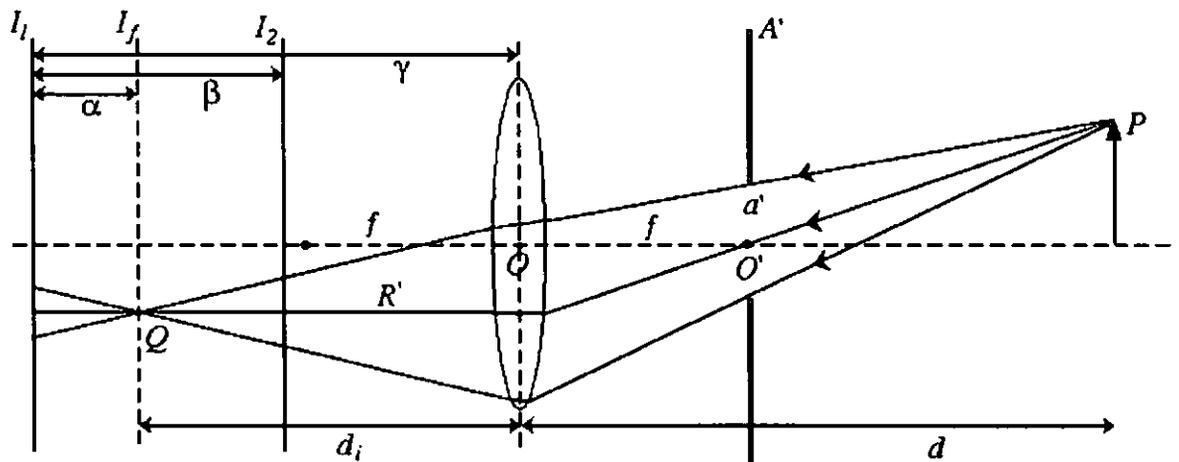
Example: Nayar's real-time system

Highlights:

- Telecentric optics for constant magnification between the image planes
- Checkerboard illumination pattern that follows the same optical path as the cameras – no occlusions
- Careful consideration of all the sampling and filtering issues
- Simple "focus operator" that returns the amount of defocus at each point
- Accuracy to 1 part in 300 over 30 cm
- Real-time!



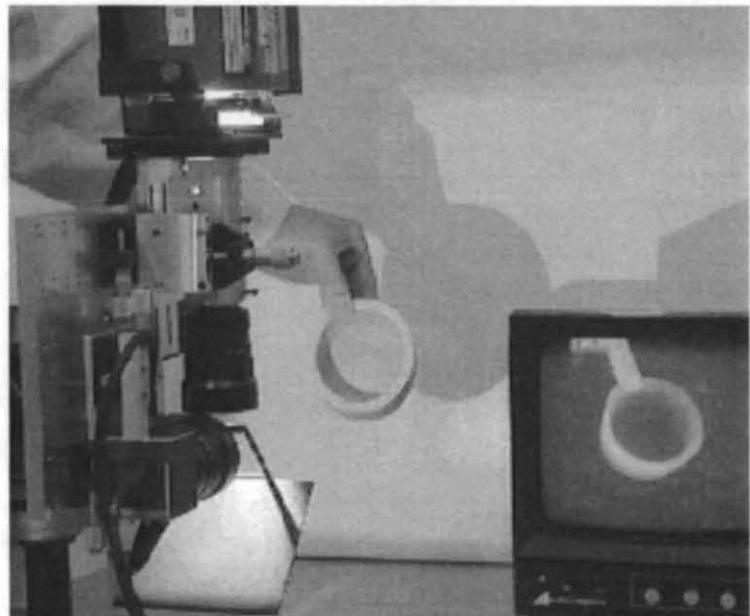
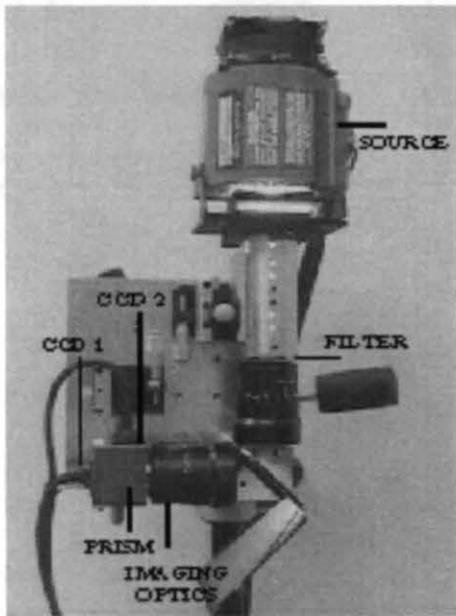
Traditional optics



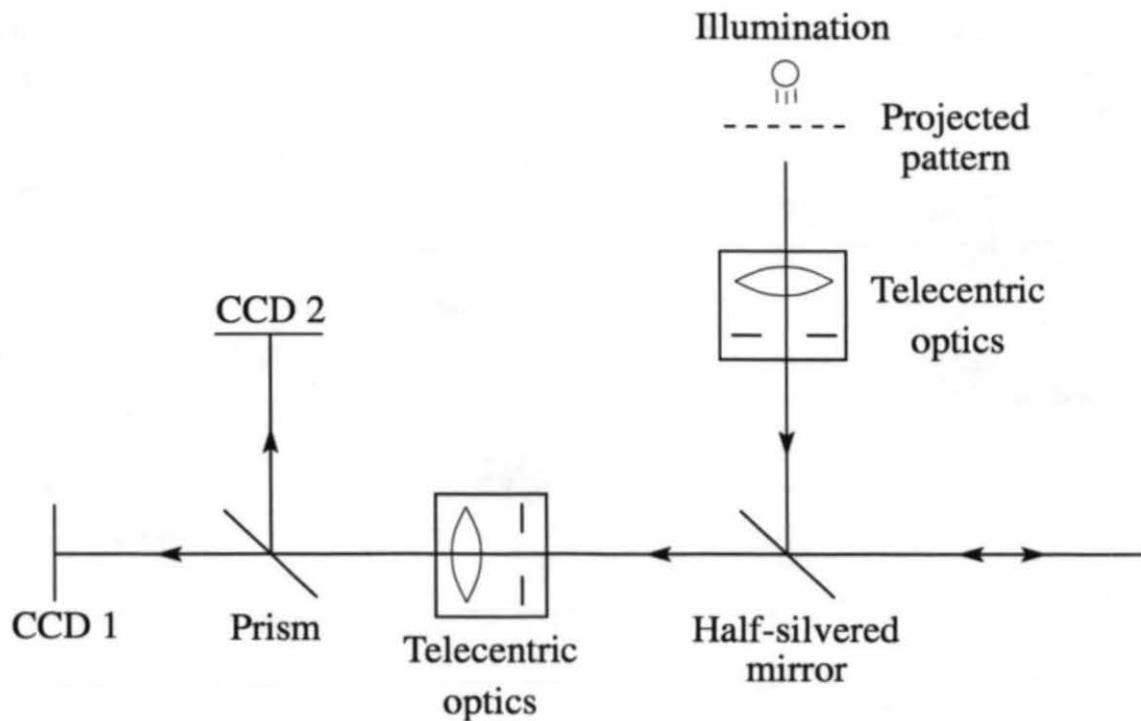
Telecentric optics

[Figures from Nayar 1995]

Nayar's optical configuration



The hardware: 2 CCD's, 1 prism, 1 HS mirror, 2 identical lens systems



[Upper figure from Nayar 1995]

Contact Methods: Mechanical

Coordinate Measuring Machine (CMM)

Principle:

Typically use X,Y,Z translation assembly with optical gratings to measure displacements. The probe may have a rotational degree of freedom and is specially calibrated to measure point of contact.

Working volume:

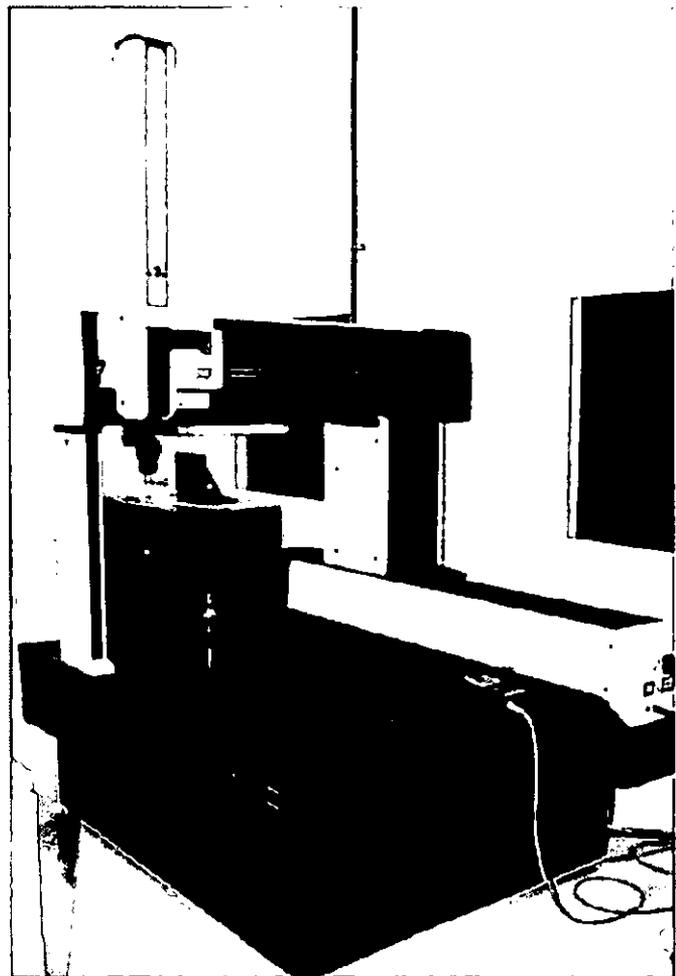
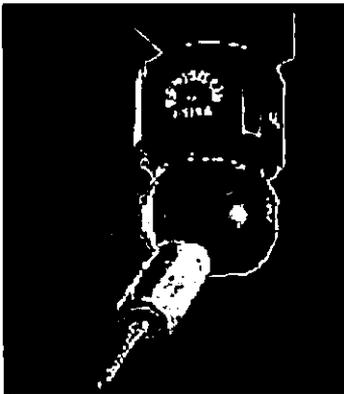
1–5 meters

Accuracy:

10 microns in 1 meter => 1 part in 100,000 accuracy!

Notes:

- Expensive
- Slow



Overall view of coordinate measuring machine

Contact Methods: Mechanical

Jointed arms

Principle:

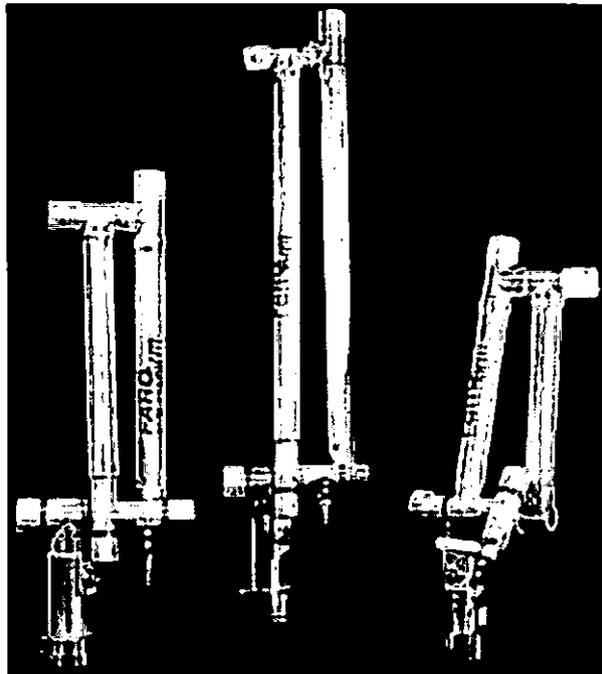
A series of rotating joints provide freedom of movement. A sequence of joint to joint transformations provide position and orientation of the stylus.

[The following are statistics for a *FaroArm*.]

Working volume:
2.4 meters

Accuracy (2σ):
0.1 mm or 1 part in 24,000 (Silver series)
0.3 mm or 1 part in 8,000 (Bronze series – in our lab)

Notes:
– Much less expensive than CMM's
– Slow



[Image from Faro's web pages – <http://www.faro.com>]

Contact Methods: Magnetic trackers

Principle:

A transmitter sends multiple magnetic field pulses, and a receiver detects the pulses. Based on the strengths of the receiver responses, the position and orientation of the transmitter source is computed.

[The following are statistics for the *Polhemus FastTrak*.]

Working volume: 3–10 meters

Accuracy:

0.75 mm and 0.15 degrees => ~1 part in 5000 positional accuracy

Notes:

- Wireless versions available for motion tracking
- Sensitive to metals and magnetic sources (monitors)

Transmissive: Industrial CT

Principle:

A series of line X-rays are taken at many angles within a plane to obtain a density profile of one object slice. The object is translated through the scanner one slice at a time. The result is a volumetric description of the object.

The surface is an implicit function, a level set at $F(x) = C$ within the volume. Isosurface extraction algorithms can obtain the desired surface.

Working volume:

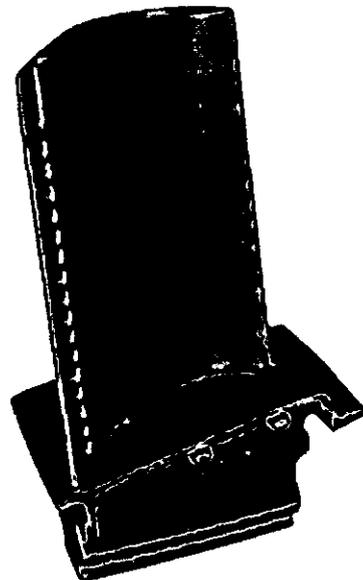
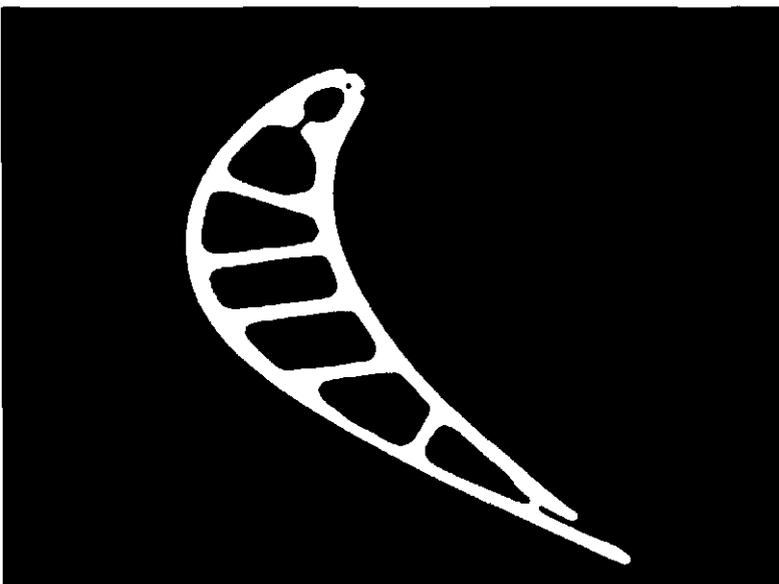
0.2 – 2 meters

Accuracy:

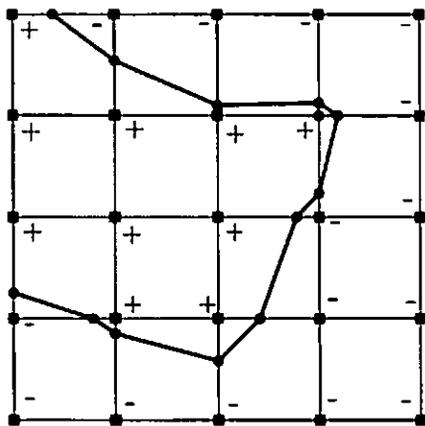
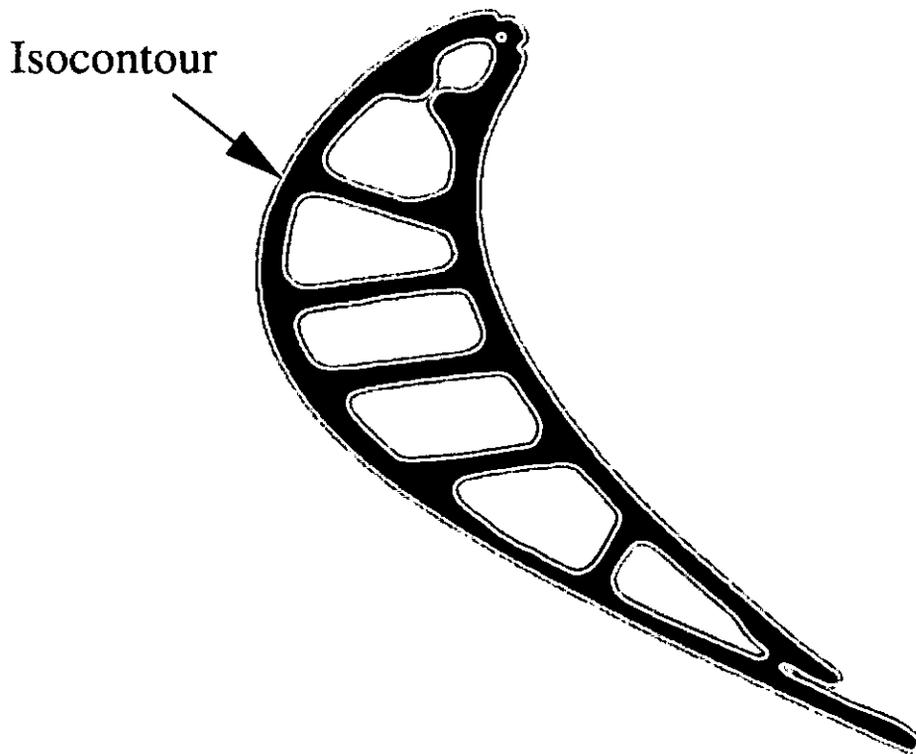
0.25 mm voxel spacing (in 0.2 meters)

Notes:

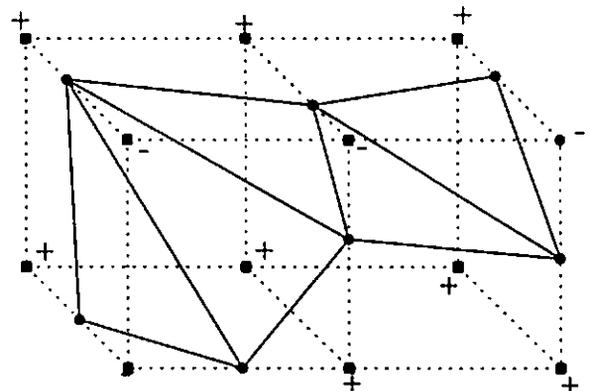
- Acquires internal cavities
- Very expensive
- Uses hazardous radiation
- Accuracy varies with large variations in material densities – e.g., wood glued on metal



[Images courtesy Bill Lorensen, General Electric Corporate R&D]



Isocontour from "marching squares"



Isocontour from "marching cubes"

[Image courtesy Bill Lorensen, General Electric Corporate R&D]

Bibliography

Besl, P. *Advances in Machine Vision. Chapter 1: Active Optical Range Imaging Sensors.* Sanz, J., editor, Springer-Verlag, New York, 1989.

Curless, B. and Levoy, M., "Better optical triangulation through spacetime analysis," Fifth International Conference on Computer Vision (1995), pp. 987-994.

Goodman, J. *Laser Speckle and Related Phenomena. Chapter 1: Statistical properties of laser speckle patterns.* Dainty, J., editor. Springer-Verlag, New York, 1984.

Nayar, S.K., Watanabe, M., and Noguchi, M. "Real-time focus range sensor", Fifth International Conference on Computer Vision (1995), pp. 995-1001.

Rioux, M., Bechthold, G., Taylor, D., and Duggan, M. "Design of a large depth of view three-dimensional camera for robot vision," *Optical Engineering* (1987), vol. 26, no. 12, pp. 1245-1250.

Modeling and Rendering Architecture from Photographs: A hybrid geometry- and image-based approach

Paul E. Debevec

Camillo J. Taylor

Jitendra Malik

University of California at Berkeley¹

ABSTRACT

We present a new approach for modeling and rendering existing architectural scenes from a sparse set of still photographs. Our modeling approach, which combines both geometry-based and image-based techniques, has two components. The first component is a *photogrammetric modeling* method which facilitates the recovery of the basic geometry of the photographed scene. Our photogrammetric modeling approach is effective, convenient, and robust because it exploits the constraints that are characteristic of architectural scenes. The second component is a *model-based stereo* algorithm, which recovers how the real scene deviates from the basic model. By making use of the model, our stereo technique robustly recovers accurate depth from widely-spaced image pairs. Consequently, our approach can model large architectural environments with far fewer photographs than current image-based modeling approaches. For producing renderings, we present *view-dependent texture mapping*, a method of compositing multiple views of a scene that better simulates geometric detail on basic models. Our approach can be used to recover models for use in either geometry-based or image-based rendering systems. We present results that demonstrate our approach's ability to create realistic renderings of architectural scenes from viewpoints far from the original photographs.

CR Descriptors: I.2.10 [Artificial Intelligence]: Vision and Scene Understanding - *Modeling and recovery of physical attributes*; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism - *Color, shading, shadowing, and texture* I.4.8 [Image Processing]: Scene Analysis - *Stereo*; J.6 [Computer-Aided Engineering]: Computer-aided design (CAD).

1 INTRODUCTION

Efforts to model the appearance and dynamics of the real world have produced some of the most compelling imagery in computer graphics. In particular, efforts to model architectural scenes, from the Amiens Cathedral to the Giza Pyramids to Berkeley's Soda Hall, have produced impressive walk-throughs and inspiring fly-bys. Clearly, it is an attractive application to be able to explore the world's architecture unencumbered by fences, gravity, customs, or jetlag.

¹Computer Science Division, University of California at Berkeley, Berkeley, CA 94720-1776. {debevec,camillo,malik}@cs.berkeley.edu. See also <http://www.cs.berkeley.edu/~debevec/Research>

Unfortunately, current geometry-based methods (Fig. 1a) of modeling existing architecture, in which a modeling program is used to manually position the elements of the scene, have several drawbacks. First, the process is extremely labor-intensive, typically involving surveying the site, locating and digitizing architectural plans (if available), or converting existing CAD data (again, if available). Second, it is difficult to verify whether the resulting model is accurate. Most disappointing, though, is that the renderings of the resulting models are noticeably computer-generated; even those that employ liberal texture-mapping generally fail to resemble real photographs.

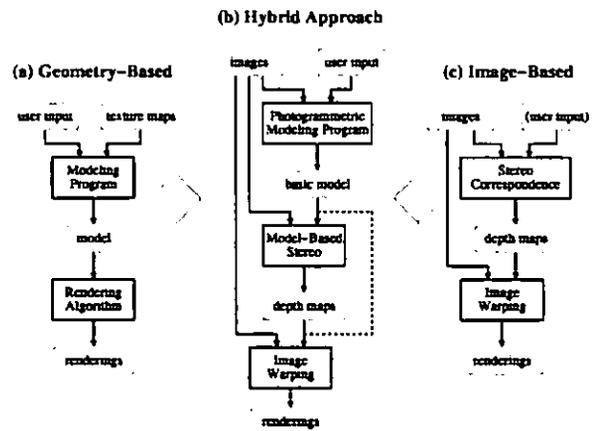


Figure 1: Schematic of how our hybrid approach combines geometry-based and image-based approaches to modeling and rendering architecture from photographs.

Recently, creating models directly from photographs has received increased interest in computer graphics. Since real images are used as input, such an image-based system (Fig. 1c) has an advantage in producing photorealistic renderings as output. Some of the most promising of these systems [16, 13] rely on the computer vision technique of computational stereopsis to automatically determine the structure of the scene from the multiple photographs available. As a consequence, however, these systems are only as strong as the underlying stereo algorithms. This has caused problems because state-of-the-art stereo algorithms have a number of significant weaknesses; in particular, the photographs need to appear very similar for reliable results to be obtained. Because of this, current image-based techniques must use many closely spaced images, and in some cases employ significant amounts of user input for each image pair to supervise the stereo algorithm. In this framework, capturing the data for a realistically renderable model would require an impractical number of closely spaced photographs, and deriving the depth from the photographs could require an impractical amount of user input. These concessions to the weakness of stereo algorithms bode poorly for creating large-scale, freely navigable virtual environments from photographs.

Our research aims to make the process of modeling architectural

scenes more convenient, more accurate, and more photorealistic than the methods currently available. To do this, we have developed a new approach that draws on the strengths of both geometry-based and image-based methods, as illustrated in Fig. 1b. The result is that our approach to modeling and rendering architecture requires only a sparse set of photographs and can produce realistic renderings from arbitrary viewpoints. In our approach, a basic geometric model of the architecture is recovered interactively with an easy-to-use photogrammetric modeling system, novel views are created using view-dependent texture mapping, and additional geometric detail can be recovered automatically through stereo correspondence. The final images can be rendered with current image-based rendering techniques. Because only photographs are required, our approach to modeling architecture is neither invasive nor does it require architectural plans, CAD models, or specialized instrumentation such as surveying equipment, GPS sensors or range scanners.

1.1 Background and Related Work

The process of recovering 3D structure from 2D images has been a central endeavor within computer vision, and the process of rendering such recovered structures is a subject receiving increased interest in computer graphics. Although no general technique exists to derive models from images, four particular areas of research have provided results that are applicable to the problem of modeling and rendering architectural scenes. They are: Camera Calibration, Structure from Motion, Stereo Correspondence, and Image-Based Rendering.

1.1.1 Camera Calibration

Recovering 3D structure from images becomes a simpler problem when the cameras used are *calibrated*, that is, the mapping between image coordinates and directions relative to each camera is known. This mapping is determined by, among other parameters, the camera's focal length and its pattern of radial distortion. Camera calibration is a well-studied problem both in photogrammetry and computer vision; some successful methods include [20] and [5]. While there has been recent progress in the use of uncalibrated views for 3D reconstruction [7], we have found camera calibration to be a straightforward process that considerably simplifies the problem.

1.1.2 Structure from Motion

Given the 2D projection of a point in the world, its position in 3D space could be anywhere on a ray extending out in a particular direction from the camera's optical center. However, when the projections of a sufficient number of points in the world are observed in multiple images from different positions, it is theoretically possible to deduce the 3D locations of the points as well as the positions of the original cameras, up to an unknown factor of scale.

This problem has been studied in the area of photogrammetry for the principal purpose of producing topographic maps. In 1913, Kruppa [10] proved the fundamental result that given two views of five distinct points, one could recover the rotation and translation between the two camera positions as well as the 3D locations of the points (up to a scale factor). Since then, the problem's mathematical and algorithmic aspects have been explored starting from the fundamental work of Ullman [21] and Longuet-Higgins [11], in the early 1980s. Faugeras's book [6] overviews the state of the art as of 1992. So far, a key realization has been that the recovery of structure is very sensitive to noise in image measurements when the translation between the available camera positions is small.

Attention has turned to using more than two views with image stream methods such as [19] or recursive approaches (e.g. [1]). [19] shows excellent results for the case of orthographic cameras, but direct solutions for the perspective case remain elusive. In general, linear algorithms for the problem fail to make use of all available

information while nonlinear minimization methods are prone to difficulties arising from local minima in the parameter space. An alternative formulation of the problem [17] uses lines rather than points as image measurements, but the previously stated concerns were shown to remain largely valid. For purposes of computer graphics, there is yet another problem: the models recovered by these algorithms consist of sparse point fields or individual line segments, which are not directly renderable as solid 3D models.

In our approach, we exploit the fact that we are trying to recover geometric models of architectural scenes, not arbitrary three-dimensional point sets. This enables us to include additional constraints not typically available to structure from motion algorithms and to overcome the problems of numerical instability that plague such approaches. Our approach is demonstrated in a useful interactive system for building architectural models from photographs.

1.1.3 Stereo Correspondence

The geometrical theory of structure from motion assumes that one is able to solve the *correspondence* problem, which is to identify the points in two or more images that are projections of the same point in the world. In humans, corresponding points in the two slightly differing images on the retinas are determined by the visual cortex in the process called binocular stereopsis.

Years of research (e.g. [2, 4, 8, 9, 12, 15]) have shown that determining stereo correspondences by computer is difficult problem. In general, current methods are successful only when the images are similar in appearance, as in the case of human vision, which is usually obtained by using cameras that are closely spaced relative to the objects in the scene. When the distance between the cameras (often called the *baseline*) becomes large, surfaces in the images exhibit different degrees of foreshortening, different patterns of occlusion, and large disparities in their locations in the two images, all of which makes it much more difficult for the computer to determine correct stereo correspondences. Unfortunately, the alternative of improving stereo correspondence by using images taken from nearby locations has the disadvantage that computing depth becomes very sensitive to noise in image measurements.

In this paper, we show that having an approximate model of the photographed scene makes it possible to robustly determine stereo correspondences from images taken from widely varying viewpoints. Specifically, the model enables us to warp the images to eliminate unequal foreshortening and to predict major instances of occlusion *before* trying to find correspondences.

1.1.4 Image-Based Rendering

In an image-based rendering system, the model consists of a set of images of a scene and their corresponding depth maps. When the depth of every point in an image is known, the image can be re-rendered from any nearby point of view by projecting the pixels of the image to their proper 3D locations and reprojecting them onto a new image plane. Thus, a new image of the scene is created by warping the images according to their depth maps. A principal attraction of image-based rendering is that it offers a method of rendering arbitrarily complex scenes with a constant amount of computation required per pixel. Using this property, [23] demonstrated how regularly spaced synthetic images (with their computed depth maps) could be warped and composited in real time to produce a virtual environment.

More recently, [13] presented a real-time image-based rendering system that used panoramic photographs with depth computed, in part, from stereo correspondence. One finding of the paper was that extracting reliable depth estimates from stereo is "very difficult". The method was nonetheless able to obtain acceptable results for nearby views using user input to aid the stereo depth recovery: the correspondence map for each image pair was seeded with 100 to 500 user-supplied point correspondences and also post-processed. Even

with user assistance, the images used still had to be closely spaced; the largest baseline described in the paper was five feet.

The requirement that samples be close together is a serious limitation to generating a freely navigable virtual environment. Covering the size of just one city block would require thousands of panoramic images spaced five feet apart. Clearly, acquiring so many photographs is impractical. Moreover, even a dense lattice of ground-based photographs would only allow renderings to be generated from within a few feet of the original camera level, precluding any virtual fly-bys of the scene. Extending the dense lattice of photographs into three dimensions would clearly make the acquisition process even more difficult. The approach described in this paper takes advantage of the structure in architectural scenes so that it requires only a sparse set of photographs. For example, our approach has yielded a virtual fly-around of a building from just twelve standard photographs.

1.2 Overview

In this paper we present three new modeling and rendering techniques: photogrammetric modeling, view-dependent texture mapping, and model-based stereo. We show how these techniques can be used in conjunction to yield a convenient, accurate, and photo-realistic method of modeling and rendering architecture from photographs. In our approach, the photogrammetric modeling program is used to create a basic volumetric model of the scene, which is then used to constrain stereo matching. Our rendering method composites information from multiple images with view-dependent texture-mapping. Our approach is successful because it splits the task of modeling from images into tasks which are easily accomplished by a person (but not a computer algorithm), and tasks which are easily performed by a computer algorithm (but not a person).

In Section 2, we present our **photogrammetric modeling** method. In essence, we have recast the structure from motion problem not as the recovery of individual point coordinates, but as the recovery of the parameters of a constrained hierarchy of parametric primitives. The result is that accurate architectural models can be recovered robustly from just a few photographs and with a minimal number of user-supplied correspondences.

In Section 3, we present **view-dependent texture mapping**, and show how it can be used to realistically render the recovered model. Unlike traditional texture-mapping, in which a single static image is used to color in each face of the model, view-dependent texture mapping interpolates between the available photographs of the scene depending on the user's point of view. This results in more lifelike animations that better capture surface specularities and unmodeled geometric detail.

Lastly, in Section 4, we present **model-based stereo**, which is used to automatically refine a basic model of a photographed scene. This technique can be used to recover the structure of architectural ornamentation that would be difficult to recover with photogrammetric modeling. In particular, we show that projecting pairs of images onto an initial approximate model allows conventional stereo techniques to robustly recover very accurate depth measurements from images with widely varying viewpoints.

2 Photogrammetric Modeling

In this section we present our method for photogrammetric modeling, in which the computer determines the parameters of a hierarchical model of parametric polyhedral primitives to reconstruct the architectural scene. We have implemented this method in *Façade*, an easy-to-use interactive modeling program that allows the user to construct a geometric model of a scene from digitized photographs. We first overview *Façade* from the point of view of the user, then we describe our model representation, and then we explain our reconstruction algorithm. Lastly, we present results from using *Façade* to reconstruct several architectural scenes.

2.1 The User's View

Constructing a geometric model of an architectural scene using *Façade* is an incremental and straightforward process. Typically, the user selects a small number of photographs to begin with, and models the scene one piece at a time. The user may refine the model and include more images in the project until the model meets the desired level of detail.

Fig. 2(a) and (b) shows the two types of windows used in *Façade*: image viewers and model viewers. The user instantiates the components of the model, marks edges in the images, and corresponds the edges in the images to the edges in the model. When instructed, *Façade* computes the sizes and relative positions of the model components that best fit the edges marked in the photographs.

Components of the model, called *blocks*, are parameterized geometric primitives such as boxes, prisms, and surfaces of revolution. A box, for example, is parameterized by its length, width, and height. The user models the scene as a collection of such blocks, creating new block classes as desired. Of course, the user does not need to specify numerical values for the blocks' parameters, since these are recovered by the program.

The user may choose to constrain the sizes and positions of any of the blocks. In Fig. 2(b), most of the blocks have been constrained to have equal length and width. Additionally, the four pinnacles have been constrained to have the same shape. Blocks may also be placed in constrained relations to one other. For example, many of the blocks in Fig. 2(b) have been constrained to sit centered and on top of the block below. Such constraints are specified using a graphical 3D interface. When such constraints are provided, they are used to simplify the reconstruction problem.

The user marks edge features in the images using a point-and-click interface; a gradient-based technique as in [14] can be used to align the edges with sub-pixel accuracy. We use edge rather than point features since they are easier to localize and less likely to be completely obscured. Only a section of each edge needs to be marked, making it possible to use partially visible edges. For each marked edge, the user also indicates the corresponding edge in the model. Generally, accurate reconstructions are obtained if there are as many correspondences in the images as there are free camera and model parameters. Thus, *Façade* reconstructs scenes accurately even when just a portion of the visible edges are marked in the images, and when just a portion of the model edges are given correspondences.

At any time, the user may instruct the computer to reconstruct the scene. The computer then solves for the parameters of the model that cause it to align with the marked features in the images. During the reconstruction, the computer computes and displays the locations from which the photographs were taken. For simple models consisting of just a few blocks, a full reconstruction takes only a few seconds; for more complex models, it can take a few minutes. For this reason, the user can instruct the computer to employ faster but less precise reconstruction algorithms (see Sec. 2.4) during the intermediate stages of modeling.

To verify the accuracy of the recovered model and camera positions, *Façade* can project the model into the original photographs. Typically, the projected model deviates from the photographs by less than a pixel. Fig. 2(c) shows the results of projecting the edges of the model in Fig. 2(b) into the original photograph.

Lastly, the user may generate novel views of the model by positioning a virtual camera at any desired location. *Façade* will then use the view-dependent texture-mapping method of Section 3 to render a novel view of the scene from the desired location. Fig. 2(d) shows an aerial rendering of the tower model.

2.2 Model Representation

The purpose of our choice of model representation is to represent the scene as a surface model with as few parameters as possible: when

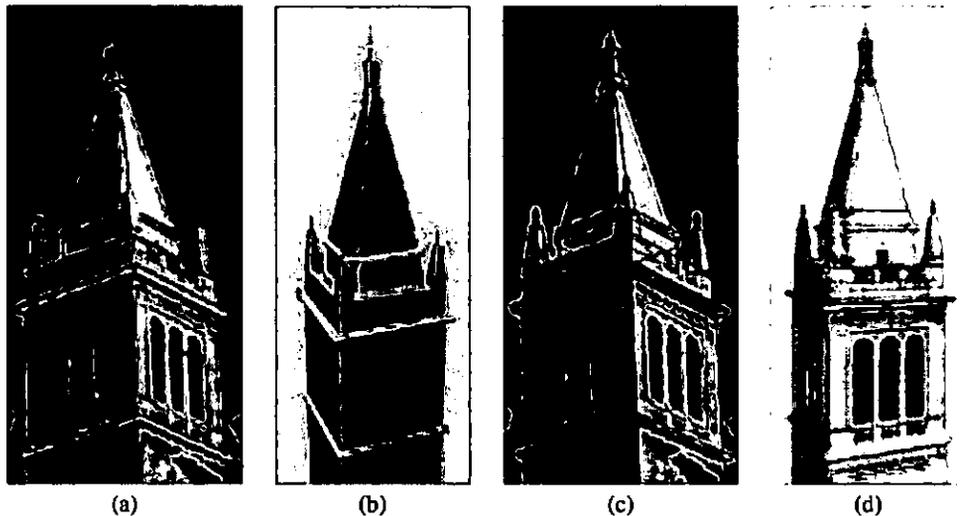


Figure 2: (a) A photograph of the Campanile, Berkeley's clock tower, with marked edges shown in green. (b) The model recovered by our photogrammetric modeling method. Although only the left pinnacle was marked, the remaining three (including one not visible) were recovered from symmetrical constraints in the model. Our method allows any number of images to be used, but in this case constraints of symmetry made it possible to recover an accurate 3D model from a single photograph. (c) The accuracy of the model is verified by reprojecting it into the original photograph through the recovered camera position. (d) A synthetic view of the Campanile generated using the view-dependent texture-mapping method described in Section 3. A real photograph from this position would be difficult to take since the camera position is 250 feet above the ground.

the model has fewer parameters, the user needs to specify fewer correspondences, and the computer can reconstruct the model more efficiently. In Façade, the scene is represented as a constrained hierarchical model of parametric polyhedral primitives, called *blocks*. Each block has a small set of parameters which serve to define its size and shape. Each coordinate of each vertex of the block is then expressed as linear combination of the block's parameters, relative to an internal coordinate frame. For example, for the wedge block in Fig. 3, the coordinates of the vertex P_0 are written in terms of the block parameters *width*, *height*, and *length* as $P_0 = (-width, -height, length)^T$. Each block is also given an associated bounding box.

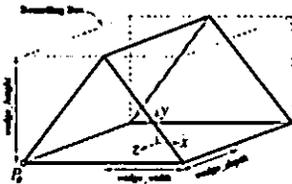


Figure 3: A wedge block with its parameters and bounding box.

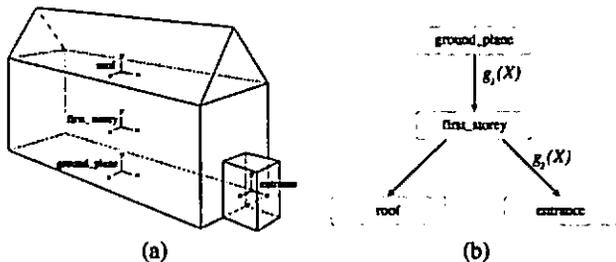


Figure 4: (a) A geometric model of a simple building. (b) The model's hierarchical representation. The nodes in the tree represent parametric primitives (called blocks) while the links contain the spatial relationships between the blocks.

The blocks in Façade are organized in a hierarchical tree structure

as shown in Fig. 4(b). Each node of the tree represents an individual block, while the links in the tree contain the spatial relationships between blocks, called *relations*. Such hierarchical structures are also used in traditional modeling systems.

The relation between a block and its parent is most generally represented as a rotation matrix R and a translation vector t . This representation requires six parameters: three each for R and t . In architectural scenes, however, the relationship between two blocks usually has a simple form that can be represented with fewer parameters, and Façade allows the user to build such constraints on R and t into the model. The rotation R between a block and its parent can be specified in one of three ways: first, as an unconstrained rotation, requiring three parameters; second, as a rotation about a particular coordinate axis, requiring just one parameter; or third, as a fixed or null rotation, requiring no parameters.

Likewise, Façade allows for constraints to be placed on each component of the translation vector t . Specifically, the user can constrain the bounding boxes of two blocks to align themselves in some manner along each dimension. For example, in order to ensure that the roof block in Fig. 4 lies on top of the first story block, the user can require that the maximum y extent of the first story block be equal to the minimum y extent of the roof block. With this constraint, the translation along the y axis is computed ($t_y = (first_story_y^{MAX} - roof_y^{MIN})$) rather than represented as a parameter of the model.

Each parameter of each instantiated block is actually a reference to a named symbolic variable, as illustrated in Fig. 5. As a result, two parameters of different blocks (or of the same block) can be equated by having each parameter reference the same symbol. This facility allows the user to equate two or more of the dimensions in a model, which makes modeling symmetrical blocks and repeated structure more convenient. Importantly, these constraints reduce the number of degrees of freedom in the model, which, as we will show, simplifies the structure recovery problem.

Once the blocks and their relations have been parameterized, it is straightforward to derive expressions for the world coordinates of the block vertices. Consider the set of edges which link a specific block in the model to the ground plane as shown in Fig. 4.

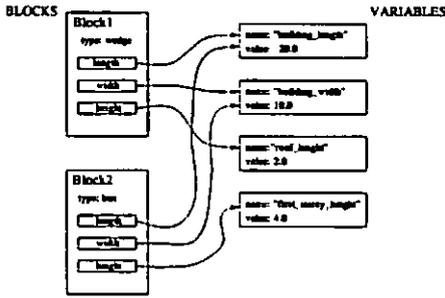


Figure 5: Representation of block parameters as symbol references. A single variable can be referenced by the model in multiple places, allowing constraints of symmetry to be embedded in the model.

Let $g_1(X), \dots, g_n(X)$ represent the rigid transformations associated with each of these links, where X represents the vector of all the model parameters. The world coordinates $P_w(X)$ of a particular block vertex $P(X)$ is then:

$$P_w(X) = g_1(X) \dots g_n(X) P(X) \quad (1)$$

Similarly, the world orientation $v_w(X)$ of a particular line segment $v(X)$ is:

$$v_w(X) = g_1(X) \dots g_n(X) v(X) \quad (2)$$

In these equations, the point vectors P and P_w and the orientation vectors v and v_w are represented in homogeneous coordinates.

Modeling the scene with polyhedral blocks, as opposed to points, line segments, surface patches, or polygons, is advantageous for a number of reasons:

- Most architectural scenes are well modeled by an arrangement of geometric primitives.
- Blocks implicitly contain common architectural elements such as parallel lines and right angles.
- Manipulating block primitives is convenient since they are at a suitably high level of abstraction; individual features such as points and lines are less manageable.
- A surface model of the scene is readily obtained from the blocks, so there is no need to infer surfaces from discrete features.
- Modeling in terms of blocks and relationships greatly reduces the number of parameters that the reconstruction algorithm needs to recover.

The last point is crucial to the robustness of our reconstruction algorithm and the viability of our modeling system, and is illustrated best with an example. The model in Fig. 2 is parameterized by just 33 variables (the unknown camera position adds six more). If each block in the scene were unconstrained (in its dimensions and position), the model would have 240 parameters; if each line segment in the scene were treated independently, the model would have 2,896 parameters. This reduction in the number of parameters greatly enhances the robustness and efficiency of the method as compared to traditional structure from motion algorithms. Lastly, since the number of correspondences needed to suitably overconstrain the minimization is roughly proportional to the number of parameters in the model, this reduction means that the number of correspondences required of the user is manageable.

2.3 Reconstruction Algorithm

Our reconstruction algorithm works by minimizing an objective function \mathcal{O} that sums the disparity between the projected edges of the model and the edges marked in the images, i.e. $\mathcal{O} = \sum Err_i$; where Err_i represents the disparity computed for edge feature i .

Thus, the unknown model parameters and camera positions are computed by minimizing \mathcal{O} with respect to these variables. Our system uses the error function Err_i from [17], described below.

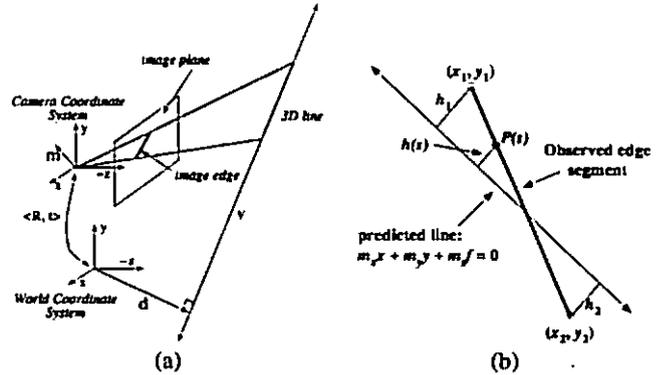


Figure 6: (a) Projection of a straight line onto a camera's image plane. (b) The error function used in the reconstruction algorithm. The heavy line represents the observed edge segment (marked by the user) and the lighter line represents the model edge predicted by the current camera and model parameters.

Fig. 6(a) shows how a straight line in the model projects onto the image plane of a camera. The straight line can be defined by a pair of vectors (v, d) where v represents the direction of the line and d represents a point on the line. These vectors can be computed from equations 2 and 1 respectively. The position of the camera with respect to world coordinates is given in terms of a rotation matrix R_j and a translation vector t_j . The normal vector denoted by m in the figure is computed from the following expression:

$$m = R_j(v \times (d - t_j)) \quad (3)$$

The projection of the line onto the image plane is simply the intersection of the plane defined by m with the image plane, located at $z = -f$ where f is the focal length of the camera. Thus, the image edge is defined by the equation $m_x x + m_y y - m_z f = 0$.

Fig. 6(b) shows how the error between the observed image edge $\{(x_1, y_1), (x_2, y_2)\}$ and the predicted image line is calculated for each correspondence. Points on the observed edge segment can be parameterized by a single scalar variable $s \in [0, l]$ where l is the length of the edge. We let $h(s)$ be the function that returns the shortest distance from a point on the segment, $p(s)$, to the predicted edge.

With these definitions, the total error between the observed edge segment and the predicted edge is calculated as:

$$Err_i = \int_0^l h^2(s) ds = \frac{l}{3} (h_1^2 + h_1 h_2 + h_2^2) = m^T (A^T B A) m \quad (4)$$

where:

$$m = (m_x, m_y, m_z)^T$$

$$A = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{pmatrix}$$

$$B = \frac{l}{3(m_x^2 + m_y^2)} \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix}$$

The final objective function \mathcal{O} is the sum of the error terms resulting from each correspondence. We minimize \mathcal{O} using a variant of the Newton-Raphson method, which involves calculating the gradient and Hessian of \mathcal{O} with respect to the parameters of the camera

and the model. As we have shown, it is simple to construct symbolic expressions for m in terms of the unknown model parameters. The minimization algorithm differentiates these expressions symbolically to evaluate the gradient and Hessian after each iteration. The procedure is inexpensive since the expressions for d and v in Equations 2 and 1 have a particularly simple form.

2.4 Computing an Initial Estimate

The objective function described in Section 2.3 section is non-linear with respect to the model and camera parameters and consequently can have local minima. If the algorithm begins at a random location in the parameter space, it stands little chance of converging to the correct solution. To overcome this problem we have developed a method to directly compute a good initial estimate for the model parameters and camera positions that is near the correct solution. In practice, our initial estimate method consistently enables the non-linear minimization algorithm to converge to the correct solution.

Our initial estimate method consists of two procedures performed in sequence. The first procedure estimates the camera rotations while the second estimates the camera translations and the parameters of the model. Both initial estimate procedures are based upon an examination of Equation 3. From this equation the following constraints can be deduced:

$$m^T R_j v = 0 \quad (5)$$

$$m^T R_j (d - t_j) = 0 \quad (6)$$

Given an observed edge u_j , the measured normal m_j' to the plane passing through the camera center is:

$$m_j' = \begin{pmatrix} x_1 \\ y_1 \\ -f \end{pmatrix} \times \begin{pmatrix} x_2 \\ y_2 \\ -f \end{pmatrix} \quad (7)$$

From these equations, we see that any model edges of known orientation constrain the possible values for R_j . Since most architectural models contain many such edges (e.g. horizontal and vertical lines), each camera rotation can be usually be estimated from the model independent of the model parameters and independent of the camera's location in space. Our method does this by minimizing the following objective function \mathcal{O}_1 that sums the extents to which the rotations R_j violate the constraints arising from Equation 5:

$$\mathcal{O}_1 = \sum_i (m^T R_j v_i)^2, \quad v_i \in \{\hat{x}, \hat{y}, \hat{z}\} \quad (8)$$

Once initial estimates for the camera rotations are computed, Equation 6 is used to obtain initial estimates of the model parameters and camera locations. Equation 6 reflects the constraint that all of the points on the line defined by the tuple (v, d) should lie on the plane with normal vector m passing through the camera center. This constraint is expressed in the following objective function \mathcal{O}_2 where $P_i(X)$ and $Q_i(X)$ are expressions for the vertices of an edge of the model.

$$\mathcal{O}_2 = \sum_i (m^T R_j (P_i(X) - t_j))^2 + (m^T R_j (Q_i(X) - t_j))^2 \quad (9)$$

In the special case where all of the block relations in the model have a known rotation, this objective function becomes a simple quadratic form which is easily minimized by solving a set of linear equations.

Once the initial estimate is obtained, the non-linear minimization over the entire parameter space is applied to produce the best possible reconstruction. Typically, the minimization requires fewer than ten iterations and adjusts the parameters of the model by at most a few percent from the initial estimates. The edges of the recovered models typically conform to the original photographs to within a pixel.



Figure 7: Three of twelve photographs used to reconstruct the entire exterior of University High School in Urbana, Illinois. The superimposed lines indicate the edges the user has marked.

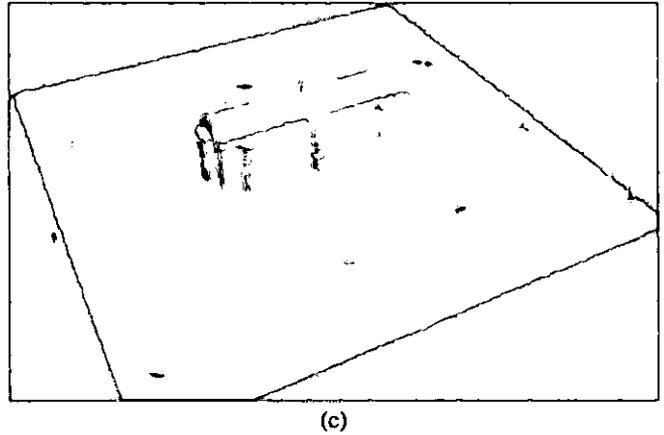
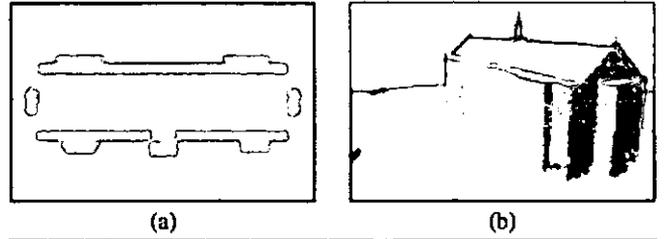


Figure 8: The high school model, reconstructed from twelve photographs. (a) Overhead view. (b) Rear view. (c) Aerial view showing the recovered camera positions. Two nearly coincident cameras can be observed in front of the building; their photographs were taken from the second story of a building across the street.



Figure 9: A synthetic view of University High School. This is a frame from an animation of flying around the entire building.

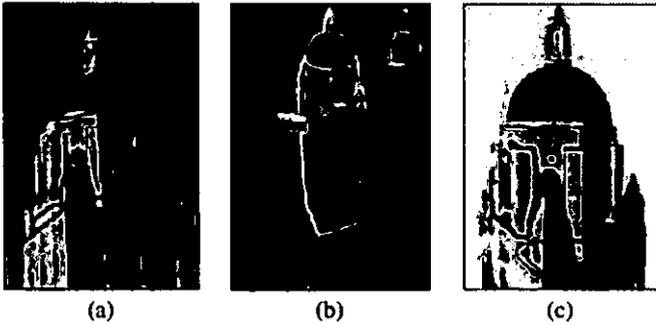


Figure 10: Reconstruction of Hoover Tower, Stanford, CA (a) Original photograph, with marked edges indicated. (b) Model recovered from the single photograph shown in (a). (c) Texture-mapped aerial view from the virtual camera position indicated in (b). Regions not seen in (a) are indicated in blue.

2.5 Results

Fig. 2 showed the results of using Façade to reconstruct a clock tower from a single image. Figs. 7 and 8 show the results of using Façade to reconstruct a high school building from twelve photographs. (The model was originally constructed from just five images; the remaining images were added to the project for purposes of generating renderings using the techniques of Section 3.) The photographs were taken with a calibrated 35mm still camera with a standard 50mm lens and digitized with the PhotoCD process. Images at the 1536×1024 pixel resolution were processed to correct for lens distortion, then filtered down to 768×512 pixels for use in the modeling system. Fig. 8 shows some views of the recovered model and camera positions, and Fig. 9 shows a synthetic view of the building generated by the technique in Sec. 3.

Fig. 10 shows the reconstruction of another tower from a single photograph. The dome was modeled specially since the reconstruction algorithm does not recover curved surfaces. The user constrained a two-parameter hemisphere block to sit centered on top of the tower, and manually adjusted its height and width to align with the photograph. Each of the models presented took approximately four hours to create.

3 View-Dependent Texture-Mapping

In this section we present view-dependent texture-mapping, an effective method of rendering the scene that involves projecting the original photographs onto the model. This form of texture-mapping is most effective when the model conforms closely to the actual structure of the scene, and when the original photographs show the scene in similar lighting conditions. In Section 4 we will show how view-dependent texture-mapping can be used in conjunction with model-based stereo to produce realistic renderings when the recovered model only approximately models the structure of the scene.

Since the camera positions of the original photographs are recovered during the modeling phase, projecting the images onto the model is straightforward. In this section we first describe how we project a single image onto the model, and then how we merge several image projections to render the entire model. Unlike traditional texture-mapping, our method projects different images onto the model depending on the user's viewpoint. As a result, our view-dependent texture mapping can give a better illusion of additional geometric detail in the model.

3.1 Projecting a Single Image

The process of texture-mapping a single image onto the model can be thought of as replacing each camera with a slide projector that projects the original image onto the model. When the model is not

convex, it is possible that some parts of the model will shadow others with respect to the camera. While such shadowed regions could be determined using an object-space visible surface algorithm, or an image-space ray casting algorithm, we use an image-space shadow map algorithm based on [22] since it is efficiently implemented using z-buffer hardware.

Fig. 11, upper left, shows the results of mapping a single image onto the high school building model. The recovered camera position for the projected image is indicated in the lower left corner of the image. Because of self-shadowing, not every point on the model within the camera's viewing frustum is mapped.

3.2 Compositing Multiple Images

In general, each photograph will view only a piece of the model. Thus, it is usually necessary to use multiple images in order to render the entire model from a novel point of view. The top images of Fig. 11 show two different images mapped onto the model and rendered from a novel viewpoint. Some pixels are colored in just one of the renderings, while some are colored in both. These two renderings can be merged into a composite rendering by considering the corresponding pixels in the rendered views. If a pixel is mapped in only one rendering, its value from that rendering is used in the composite. If it is mapped in more than one rendering, the renderer has to decide which image (or combination of images) to use.

It would be convenient, of course, if the projected images would agree perfectly where they overlap. However, the images will not necessarily agree if there is unmodeled geometric detail in the building, or if the surfaces of the building exhibit non-Lambertian reflection. In this case, the best image to use is clearly the one with the viewing angle closest to that of the rendered view. However, using the image closest in angle at every pixel means that neighboring rendered pixels may be sampled from different original images. When this happens, specularly and unmodeled geometric detail can cause visible seams in the rendering. To avoid this problem, we smooth these transitions through weighted averaging as in Fig. 12.



Figure 11: The process of assembling projected images to form a composite rendering. The top two pictures show two images projected onto the model from their respective recovered camera positions. The lower left picture shows the results of compositing these two renderings using our view-dependent weighting function. The lower right picture shows the results of compositing renderings of all twelve original images. Some pixels near the front edge of the roof not seen in any image have been filled in with the hole-filling algorithm from [23].

Even with this weighting, neighboring pixels can still be sampled from different views at the boundary of a projected image, since the contribution of an image must be zero outside its boundary. To

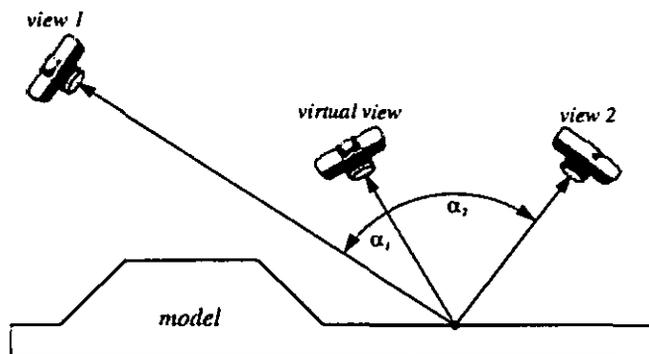


Figure 12: The weighting function used in view-dependent texture mapping. The pixel in the virtual view corresponding to the point on the model is assigned a weighted average of the corresponding pixels in actual views 1 and 2. The weights w_1 and w_2 are inversely proportional to the magnitude of angles α_1 and α_2 . Alternately, more sophisticated weighting functions based on expected foreshortening and image resampling can be used.

address this, the pixel weights are ramped down near the boundary of the projected images. Although this method does not guarantee smooth transitions in all cases, we have found that it eliminates most artifacts in renderings and animations arising from such seams.

If an original photograph features an unwanted car, tourist, or other object in front of the architecture of interest, the unwanted object will be projected onto the surface of the model. To prevent this from happening, the user may mask out the object by painting over the obstruction with a reserved color. The rendering algorithm will then set the weights for any pixels corresponding to the masked regions to zero, and decrease the weights of the pixels near the boundary as before to minimize seams. Any regions in the composite image which are occluded in every projected image are filled in using the hole-filling method from [23].

In the discussion so far, projected image weights are computed at every pixel of every projected rendering. Since the weighting function is smooth (though not constant) across flat surfaces, it is not generally necessary to compute it for every pixel of every face of the model. For example, using a single weight for each face of the model, computed at the face's center, produces acceptable results. By coarsely subdividing large faces, the results are visually indistinguishable from the case where a unique weight is computed for every pixel. Importantly, this technique suggests a real-time implementation of view-dependent texture mapping using a texture-mapping graphics pipeline to render the projected views, and α -channel blending to composite them.

For complex models where most images are entirely occluded for the typical view, it can be very inefficient to project every original photograph to the novel viewpoint. Some efficient techniques to determine such visibility *a priori* in architectural scenes through spatial partitioning are presented in [18].

4 Model-Based Stereopsis

The modeling system described in Section 2 allows the user to create a basic model of a scene, but in general the scene will have additional geometric detail (such as friezes and cornices) not captured in the model. In this section we present a new method of recovering such additional geometric detail automatically through stereo correspondence, which we call *model-based stereo*. Model-based stereo differs from traditional stereo in that it measures how the actual scene deviates from the approximate model, rather than trying to measure the structure of the scene without any prior information. The model serves to place the images into a common frame of reference that makes the stereo correspondence possible even for im-

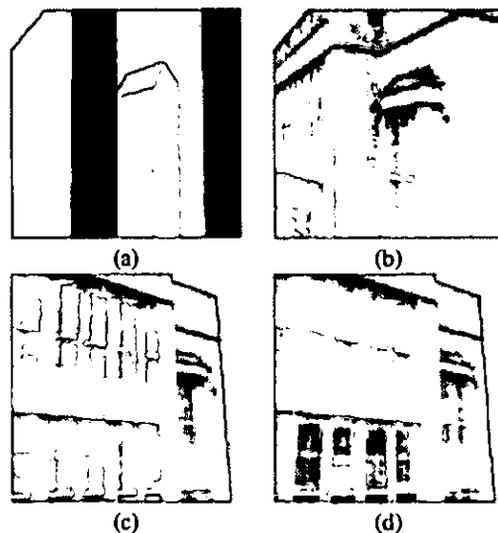


Figure 13: View-dependent texture mapping. (a) A detail view of the high school model. (b) A rendering of the model from the same position using view-dependent texture mapping. Note that although the model does not capture the slightly recessed windows, the windows appear properly recessed because the texture map is sampled primarily from a photograph which viewed the windows from approximately the same direction. (c) The same piece of the model viewed from a different angle, using the same texture map as in (b). Since the texture is not selected from an image that viewed the model from approximately the same angle, the recessed windows appear unnatural. (d) A more natural result obtained by using view-dependent texture mapping. Since the angle of view in (d) is different than in (b), a different composition of original images is used to texture-map the model.

ages taken from relatively far apart. The stereo correspondence information can then be used to render novel views of the scene using image-based rendering techniques.

As in traditional stereo, given two images (which we call the *key* and *offset*), model-based stereo computes the associated depth map for the key image by determining corresponding points in the key and offset images. Like many stereo algorithms, our method is *correlation-based*, in that it attempts to determine the corresponding point in the offset image by comparing small pixel neighborhoods around the points. As such, correlation-based stereo algorithms generally require the neighborhood of each point in the key image to resemble the neighborhood of its corresponding point in the offset image.

The problem we face is that when the key and offset images are taken from relatively far apart, as is the case for our modeling method, corresponding pixel neighborhoods can be foreshortened very differently. In Figs. 14(a) and (c), pixel neighborhoods toward the right of the key image are foreshortened horizontally by nearly a factor of four in the offset image.

The key observation in model-based stereo is that even though two images of the same scene may appear very different, they appear similar after being projected onto an approximate model of the scene. In particular, projecting the offset image onto the model and viewing it from the position of the key image produces what we call the *warped offset* image, which appears very similar to the key image. The geometrically detailed scene in Fig. 14 was modeled as two flat surfaces with our modeling program, which also determined the relative camera positions. As expected, the warped offset image (Fig. 14(b)) exhibits the same pattern of foreshortening as the key image.

In model-based stereo, pixel neighborhoods are compared between the key and warped offset images rather than the key and off-

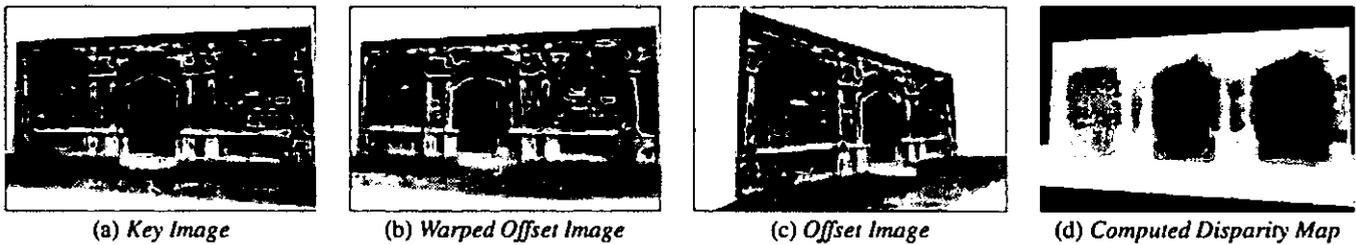


Figure 14: (a) and (c) Two images of the entrance to Peterhouse chapel in Cambridge, UK. The Façade program was used to model the façade and ground as a flat surfaces and to recover the relative camera positions. (b) The warped offset image, produced by projecting the offset image onto the approximate model and viewing it from the position of the key camera. This projection eliminates most of the disparity and foreshortening with respect to the key image, greatly simplifying stereo correspondence. (d) An unedited disparity map produced by our model-based stereo algorithm.

set images. When a correspondence is found, it is simple to convert its disparity to the corresponding disparity between the key and offset images, from which the point's depth is easily calculated. Fig. 14(d) shows a disparity map computed for the key image in (a).

The reduction of differences in foreshortening is just one of several ways that the warped offset image simplifies stereo correspondence. Some other desirable properties of the warped offset image are:

- Any point in the scene which lies on the approximate model will have zero disparity between the key image and the warped offset image.
- Disparities between the key and warped offset images are easily converted to a depth map for the key image.
- Depth estimates are far less sensitive to noise in image measurements since images taken from relatively far apart can be compared.
- Places where the model occludes itself relative to the key image can be detected and indicated in the warped offset image.
- A linear epipolar geometry (Sec. 4.1) exists between the key and warped offset images, despite the warping. In fact, the epipolar lines of the warped offset image coincide with the epipolar lines of the key image.

4.1 Model-Based Epipolar Geometry

In traditional stereo, the *epipolar constraint* (see [6]) is often used to constrain the search for corresponding points in the offset image to searching along an epipolar line. This constraint simplifies stereo not only by reducing the search for each correspondence to one dimension, but also by reducing the chance of selecting a false matches. In this section we show that taking advantage of the epipolar constraint is no more difficult in model-based stereo case, despite the fact that the offset image is non-uniformly warped.

Fig. 15 shows the epipolar geometry for model-based stereo. If we consider a point P in the scene, there is a unique *epipolar plane* which passes through P and the centers of the key and offset cameras. This epipolar plane intersects the key and offset image planes in *epipolar lines* e_k and e_o . If we consider the projection p_k of P onto the key image plane, the epipolar constraint states that the corresponding point in the offset image must lie somewhere along the offset image's epipolar line.

In model-based stereo, neighborhoods in the key image are compared to the warped offset image rather than the offset image. Thus, to make use of the epipolar constraint, it is necessary to determine where the pixels on the offset image's epipolar line project to in the warped offset image. The warped offset image is formed by projecting the offset image onto the model, and then reprojecting the model onto the image plane of the key camera. Thus, the projection p_o of P in the offset image projects onto the model at Q , and then reprojects to q_k in the warped offset image. Since each of these projections occurs within the epipolar plane, any possible correspondence

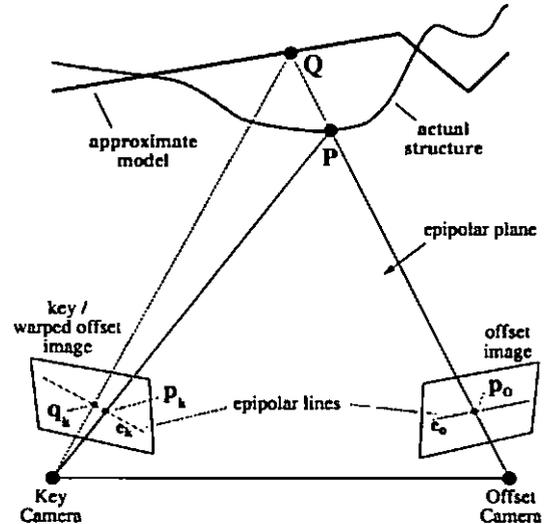


Figure 15: Epipolar geometry for model-based stereo.

for p_k in the key image must lie on the key image's epipolar line in the warped offset image. In the case where the actual structure and the model coincide at P , p_o is projected to P and then reprojected to p_k , yielding a correspondence with zero disparity.

The fact that the epipolar geometry remains linear after the warping step also facilitates the use of the ordering constraint [2, 6] through a dynamic programming technique.

4.2 Stereo Results and Rerendering

While the warping step makes it dramatically easier to determine stereo correspondences, a stereo algorithm is still necessary to actually determine them. The algorithm we developed to produce the images in this paper is described in [3].

Once a depth map has been computed for a particular image, we can rerender the scene from novel viewpoints using the methods described in [23, 16, 13]. Furthermore, when several images and their corresponding depth maps are available, we can use the view-dependent texture-mapping method of Section 3 to composite the multiple renderings. The novel views of the chapel façade in Fig. 16 were produced through such compositing of four images.

5 Conclusion and Future Work

To conclude, we have presented a new, photograph-based approach to modeling and rendering architectural scenes. Our modeling approach, which combines both geometry-based and image-based modeling techniques, is built from two components that we have developed. The first component is an easy-to-use photogrammet-

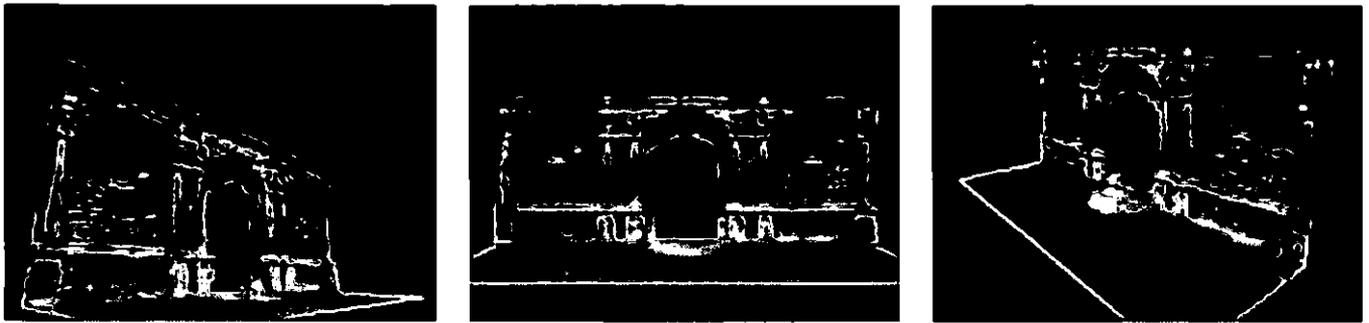


Figure 16: Novel views of the scene generated from four original photographs. These are frames from an animated movie in which the façade rotates continuously. The depth is computed from model-based stereo and the frames are made by compositing image-based renderings with view-dependent texture-mapping.

ric modeling system which facilitates the recovery of a basic geometric model of the photographed scene. The second component is a model-based stereo algorithm, which recovers precisely how the real scene differs from the basic model. For rendering, we have presented view-dependent texture-mapping, which produces images by warping and compositing multiple views of the scene. Through judicious use of images, models, and human assistance, our approach is more convenient, more accurate, and more photorealistic than current geometry-based or image-based approaches for modeling and rendering real-world architectural scenes.

There are several improvements and extensions that can be made to our approach. First, surfaces of revolution represent an important component of architecture (e.g. domes, columns, and minarets) that are not recovered in our photogrammetric modeling approach. (As noted, the dome in Fig. 10 was manually sized by the user.) Fortunately, there has been much work (e.g. [24]) that presents methods of recovering such structures from image contours. Curved model geometry is also entirely consistent with our approach to recovering additional detail with model-based stereo.

Second, our techniques should be extended to recognize and model the photometric properties of the materials in the scene. The system should be able to make better use of photographs taken in varying lighting conditions, and it should be able to render images of the scene as it would appear at any time of day, in any weather, and with any configuration of artificial light. Already, the recovered model can be used to predict shadowing in the scene with respect to an arbitrary light source. However, a full treatment of the problem will require estimating the photometric properties (i.e. the bidirectional reflectance distribution functions) of the surfaces in the scene.

Third, it is clear that further investigation should be made into the problem of selecting which original images to use when rendering a novel view of the scene. This problem is especially difficult when the available images are taken at arbitrary locations. Our current solution to this problem, the weighting function presented in Section 3, still allows seams to appear in renderings and does not consider issues arising from image resampling. Another form of view selection is required to choose which pairs of images should be matched to recover depth in the model-based stereo algorithm.

Lastly, it will clearly be an attractive application to integrate the models created with the techniques presented in this paper into forthcoming real-time image-based rendering systems.

Acknowledgments

This research was supported by a National Science Foundation Graduate Research Fellowship and grants from Interval Research Corporation, the California MICRO program, and JSEP contract F49620-93-C-0014. The authors also wish to thank Tim Hawkins, Carlo Sequin, David Forsyth, and Jianbo Shi for their valuable help in revising this paper.

References

- [1] Ali Azarbayejani and Alex Pentland. Recursive estimation of motion, structure, and focal length. *IEEE Trans. Pattern Anal. Machine Intell.*, 17(6):562-575, June 1995.
- [2] H. H. Baker and T. O. Binford. Depth from edge and intensity based stereo. In *Proceedings of the Seventh IJCAI, Vancouver, BC*, pages 631-636, 1981.
- [3] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. Technical Report UCB/CSD-96-893, U.C. Berkeley, CS Division, January 1996.
- [4] D.J.Fleet, A.D.Jepson, and M.R.M. Jenkin. Phase-based disparity measurement. *CVGIP: Image Understanding*, 53(2):198-210, 1991.
- [5] Olivier Faugeras and Giorgio Toscani. The calibration problem for stereo. In *Proceedings IEEE CVPR 86*, pages 15-20, 1986.
- [6] Olivier Faugeras. *Three-Dimensional Computer Vision*. MIT Press, 1993.
- [7] Olivier Faugeras, Stephane Laveau, Luc Robert, Gabriella Csurka, and Cyril Zeller. 3-d reconstruction of urban scenes from sequences of images. Technical Report 2572, INRIA, June 1995.
- [8] W. E. L. Grimson. *From Images to Surface*. MIT Press, 1981.
- [9] D. Jones and J. Malik. Computational framework for determining stereo correspondence from a set of linear spatial filters. *Image and Vision Computing*, 10(10):699-708, December 1992.
- [10] E. Kruppa. Zur ermittlung eines objectes aus zwei perspektiven mit innerer orientierung. *Sitz-Ber. Akad. Wiss., Wien. Math. Naturw. Kl. Abt. IIa.*, 122:1939-1948, 1913.
- [11] H.C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133-135, September 1981.
- [12] D. Marr and T. Poggio. A computational theory of human stereo vision. *Proceedings of the Royal Society of London*, 204:301-328, 1979.
- [13] Leonard McMillan and Gary Bishop. Plenoptic modeling: An image-based rendering system. In *SIGGRAPH '95*, 1995.
- [14] Eric N. Mortensen and William A. Barrett. Intelligent scissors for image composition. In *SIGGRAPH '95*, 1995.
- [15] S. B. Pollard, J. E. W. Mayhew, and J. P. Frisby. A stereo correspondence algorithm using a disparity gradient limit. *Perception*, 14:449-470, 1985.
- [16] R. Szeliski. Image mosaicing for tele-reality applications. In *IEEE Computer Graphics and Applications*, 1996.
- [17] Camillo J. Taylor and David J. Kriegman. Structure and motion from line segments in multiple images. *IEEE Trans. Pattern Anal. Machine Intell.*, 17(11), November 1995.
- [18] S. J. Teitel, Celeste Fowler, Thomas Funkhouser, and Pat Hanrahan. Partitioning and ordering large radiosity computations. In *SIGGRAPH '94*, pages 443-450, 1994.
- [19] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9(2):137-154, November 1992.
- [20] Roger Tsai. A versatile camera calibration technique for high accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal of Robotics and Automation*, 3(4):323-344, August 1987.
- [21] S. Ullman. *The Interpretation of Visual Motion*. The MIT Press, Cambridge, MA, 1979.
- [22] L. Williams. Casting curved shadows on curved surfaces. In *SIGGRAPH '78*, pages 270-274, 1978.
- [23] Lance Williams and Eric Chen. View interpolation for image synthesis. In *SIGGRAPH '93*, 1993.
- [24] Mourad Zerroug and Ramakant Nevatia. Segmentation and recovery of shapes from a real intensity image. In *European Conference on Computer Vision*, pages 319-330, 1994.

A Volumetric Method for Building Complex Models from Range Images

Brian Curless and Marc Levoy
Stanford University

Abstract

A number of techniques have been developed for reconstructing surfaces by integrating groups of aligned range images. A desirable set of properties for such algorithms includes: incremental updating, representation of directional uncertainty, the ability to fill gaps in the reconstruction, and robustness in the presence of outliers. Prior algorithms possess subsets of these properties. In this paper, we present a volumetric method for integrating range images that possesses all of these properties.

Our volumetric representation consists of a cumulative weighted signed distance function. Working with one range image at a time, we first scan-convert it to a distance function, then combine this with the data already acquired using a simple additive scheme. To achieve space efficiency, we employ a run-length encoding of the volume. To achieve time efficiency, we resample the range image to align with the voxel grid and traverse the range and voxel scanlines synchronously. We generate the final manifold by extracting an isosurface from the volumetric grid. We show that under certain assumptions, this isosurface is optimal in the least squares sense. To fill gaps in the model, we tessellate over the boundaries between regions seen to be empty and regions never observed.

Using this method, we are able to integrate a large number of range images (as many as 70) yielding seamless, high-detail models of up to 2.6 million triangles.

CR Categories: 1.3.5 [Computer Graphics] Computational Geometry and Object Modeling

Additional keywords: Surface fitting, three-dimensional shape recovery, range image integration, isosurface extraction

1 Introduction

Recent years have witnessed a rise in the availability of fast, accurate range scanners. These range scanners have provided data for applications such as medicine, reverse engineering, and digital film-making. Many of these devices generate *range images*; i.e., they produce depth values on a regular sampling lattice. Figure 1 illustrates how an optical triangulation scanner can be used to acquire a range image. By connecting nearest neighbors with triangular elements, one can construct a *range surface* as shown in Figure 1d. Range images are typically formed by sweeping a 1D or 2D sensor linearly across an object or circularly around it, and generally do not contain enough information to reconstruct the entire object being scanned. Accordingly, we require algorithms that can merge multiple range images into a sin-

gle description of the surface. A set of desirable properties for such a surface reconstruction algorithm includes:

- *Representation of range uncertainty.* The data in range images typically have asymmetric error distributions with primary directions along sensor lines of sight, as illustrated for optical triangulation in Figure 1a. The method of range integration should reflect this fact.
- *Utilization of all range data,* including redundant observations of each object surface. If properly used, this redundancy can reduce sensor noise.
- *Incremental and order independent updating.* Incremental updates allow us to obtain a reconstruction after each scan or small set of scans and allow us to choose the next best orientation for scanning. Order independence is desirable to ensure that results are not biased by earlier scans. Together, they allow for straightforward parallelization.
- *Time and space efficiency.* Complex objects may require many range images in order to build a detailed model. The range images and the model must be represented efficiently and processed quickly to make the algorithm practical.
- *Robustness.* Outliers and systematic range distortions can create challenging situations for reconstruction algorithms. A robust algorithm needs to handle these situations without catastrophic failures such as holes in surfaces and self-intersecting surfaces.
- *No restrictions on topological type.* The algorithm should not assume that the object is of a particular genus. Simplifying assumptions such as "the object is homeomorphic to a sphere" yield useful results in only a restricted class of problems.
- *Ability to fill holes in the reconstruction.* Given a set of range images that do not completely cover the object, the surface reconstruction will necessarily be incomplete. For some objects, no amount of scanning would completely cover the object, because some surfaces may be inaccessible to the sensor. In these cases, we desire an algorithm that can automatically fill these holes with plausible surfaces, yielding a model that is both "watertight" and esthetically pleasing.

In this paper, we present a volumetric method for integrating range images that possesses all of these properties. In the next section, we review some previous work in the area of surface reconstruction. In section 3, we describe the core of our volumetric algorithm. In section 4, we show how this algorithm can be used to fill gaps in the reconstruction using knowledge about the emptiness of space. Next, in section 5, we describe how we implemented our volumetric approach so as to keep time and space costs reasonable. In section 6, we show the results of surface reconstruction from many range images of complex objects. Finally, in section 7 we conclude and discuss limitations and future directions.

2 Previous work

Surface reconstruction from dense range data has been an active area of research for several decades. The strategies have proceeded along two basic directions: reconstruction from unorganized points, and

Authors' Address: Computer Science Department, Stanford University,
Stanford, CA 94305

E-mail: {curless,levoy}@cs.stanford.edu

World Wide Web: <http://www-graphics.stanford.edu>

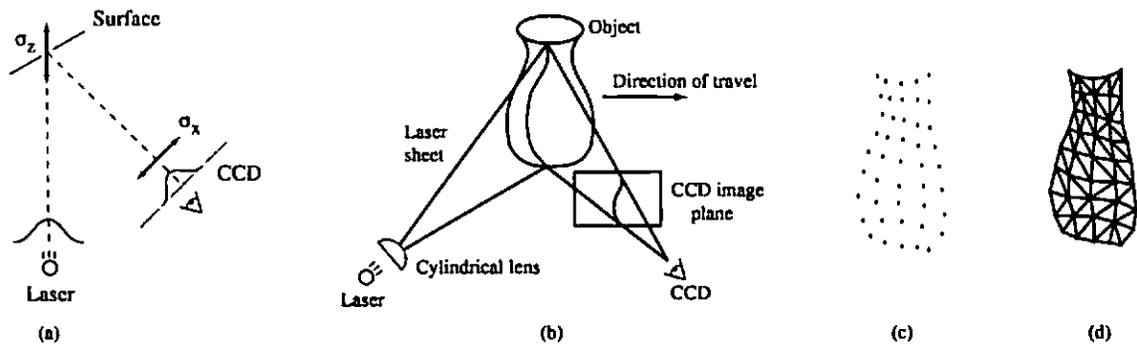


Figure 1. From optical triangulation to a range surface. (a) In 2D, a narrow laser beam illuminates a surface, and a linear sensor images the reflection from an object. The center of the image pulse maps to the center of the laser, yielding a range value. The uncertainty, σ_x , in determining the center of the pulse results in range uncertainty, σ_z along the laser's line of sight. When using the spacetime analysis for optical triangulation [6], the uncertainties run along the lines of sight of the CCD. (b) In 3D, a laser stripe triangulation scanner first spreads the laser beam into a sheet of light with a cylindrical lens. The CCD observes the reflected stripe from which a depth profile is computed. The object sweeps through the field of view, yielding a range image. Other scanner configurations rotate the object to obtain a cylindrical scan or sweep a laser beam or stripe over a stationary object. (c) A range image obtained from the scanner in (b) is a collection of points with regular spacing. (d) By connecting nearest neighbors with triangles, we create a piecewise linear range surface.

reconstruction that exploits the underlying structure of the acquired data. These two strategies can be further subdivided according to whether they operate by reconstructing parametric surfaces or by reconstructing an implicit function.

A major advantage of the unorganized points algorithms is the fact that they do not make any prior assumptions about connectivity of points. In the absence of range images or contours to provide connectivity cues, these algorithms are the only recourse. Among the parametric surface approaches, Boissinat [2] describes a method for Delaunay triangulation of a set of points in 3-space. Edelsbrunner and Mücke [9] generalize the notion of a convex hull to create surfaces called alpha-shapes. Examples of implicit surface reconstruction include the method of Hoppe, et al [16] for generating a signed distance function followed by an isosurface extraction. More recently, Bajaj, et al [1] used alpha-shapes to construct a signed distance function to which they fit implicit polynomials. Although unorganized points algorithms are widely applicable, they discard useful information such as surface normal and reliability estimates. As a result, these algorithms are well-behaved in smooth regions of surfaces, but they are not always robust in regions of high curvature and in the presence of systematic range distortions and outliers.

Among the structured data algorithms, several parametric approaches have been proposed, most of them operating on range images in a polygonal domain. Soucy and Laurendeau [25] describe a method using Venn diagrams to identify overlapping data regions, followed by re-parameterization and merging of regions. Turk and Levoy [30] devised an incremental algorithm that updates a reconstruction by eroding redundant geometry, followed by zipping along the remaining boundaries, and finally a consensus step that reintroduces the original geometry to establish final vertex positions. Rutishauser, et al [24] use errors along the sensor's lines of sight to establish consensus surface positions followed by a re-tessellation that incorporates redundant data. These algorithms typically perform better than unorganized point algorithms, but they can still fail catastrophically in areas of high curvature, as exemplified in Figure 9.

Several algorithms have been proposed for integrating structured data to generate implicit functions. These algorithms can be classified as to whether voxels are assigned one of two (or three) states or are samples of a continuous function. Among the discrete-state volumetric algorithms, Connolly [4] casts rays from a range image accessed as a quad-tree into a voxel grid stored as an octree, and generates results for synthetic data. Chien, et al [3] efficiently generate octree models under the severe assumption that all views are taken from the directions corresponding to the 6 faces of a cube. Li and Crebbin [19] and

Tarbox and Gottschlich [28] also describe methods for generating binary voxel grids from range images. None of these methods has been used to generate surfaces. Further, without an underlying continuous function, there are no mechanism for representing range uncertainty or for combining overlapping, noisy range surfaces.

The last category of our taxonomy consists of implicit function methods that use samples of a continuous function to combine structured data. Our method falls into this category. Previous efforts in this area include the work of Grosso, et al [12], who generate depth maps from stereo and average them into a volume with occupancy ramps of varying slopes corresponding to uncertainty measures; they do not, however, perform a final surface extraction. Succi, et al [26] create depth maps from stereo and optical flow and integrate them volumetrically using a straight average. The details of his method are unclear, but they appear to extract an isosurface at an arbitrary threshold. In both the Grosso and Succi papers, the range maps are sparse, the directions of range uncertainty are not characterized, they use no time or space optimizations, and the final models are of low resolution. Recently, Hilton, et al [14] have developed a method similar to ours in that it uses weighted signed distance functions for merging range images, but it does not address directions of sensor uncertainty, incremental updating, space efficiency, and characterization of the whole space for potential hole filling, all of which we believe are crucial for the success of this approach.

Other relevant work includes the method of probabilistic occupancy grids developed by Elfes and Matthies [10]. Their volumetric space is a scalar probability field which they update using a Bayesian formulation. The results have been used for robot navigation, but not for surface extraction. A difficulty with this technique is the fact that the best description of the surface lies at the peak or ridge of the probability function, and the problem of ridge-finding is not one with robust solutions [8]. This is one of our primary motivations for taking an isosurface approach in the next section: it leverages off of well-behaved surface extraction algorithms.

The discrete-state implicit function algorithms described above also have much in common with the methods of extracting volumes from silhouettes [15] [21] [23] [27]. The idea of using backdrops to help carve out the emptiness of space is one we demonstrate in section 4.

3 Volumetric integration

Our algorithm employs a continuous implicit function, $D(x)$, represented by samples. The function we represent is the weighted signed distance of each point x to the nearest range surface along the line of

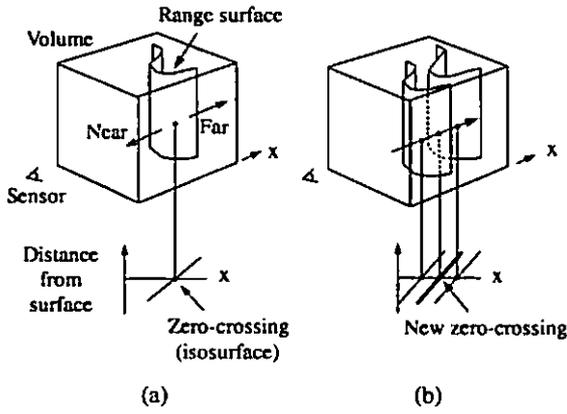


Figure 2. Unweighted signed distance functions in 3D. (a) A range sensor looking down the x -axis observes a range image, shown here as a re-constructed range surface. Following one line of sight down the x -axis, we can generate a signed distance function as shown. The zero crossing of this function is a point on the range surface. (b) The range sensor repeats the measurement, but noise in the range sensing process results in a slightly different range surface. In general, the second surface would interpenetrate the first, but we have shown it as an offset from the first surface for purposes of illustration. Following the same line of sight as before, we obtain another signed distance function. By summing these functions, we arrive at a cumulative function with a new zero crossing positioned midway between the original range measurements.

sight to the sensor. We construct this function by combining signed distance functions $d_1(x)$, $d_2(x)$, ..., $d_n(x)$ and weight functions $w_1(x)$, $w_2(x)$, ..., $w_n(x)$ obtained from range images 1 ... n . Our combining rules give us for each voxel a cumulative signed distance function, $D(x)$, and a cumulative weight $W(x)$. We represent these functions on a discrete voxel grid and extract an isosurface corresponding to $D(x) = 0$. Under a certain set of assumptions, this isosurface is optimal in the least squares sense. A full proof of this optimality is beyond the scope of this paper, but a sketch appears in appendix A.

Figure 2 illustrates the principle of combining unweighted signed distances for the simple case of two range surfaces sampled from the same direction. Note that the resulting isosurface would be the surface created by averaging the two range surfaces along the sensor's lines of sight. In general, however, weights are necessary to represent variations in certainty across the range surfaces. The choice of weights should be specific to the range scanning technology. For optical triangulation scanners, for example, Soucy [25] and Turk [30] make the weight depend on the dot product between each vertex normal and the viewing direction, reflecting greater uncertainty when the illumination is at grazing angles to the surface. Turk also argues that the range data at the boundaries of the mesh typically have greater uncertainty, requiring more down-weighting. We adopt these same weighting schemes for our optical triangulation range data.

Figure 3 illustrates the construction and usage of the signed distance and weight functions in 1D. In Figure 3a, the sensor is positioned at the origin looking down the $+x$ axis and has taken two measurements, r_1 and r_2 . The signed distance profiles, $d_1(x)$ and $d_2(x)$ may extend indefinitely in either direction, but the weight functions, $w_1(x)$ and $w_2(x)$, taper off behind the range points for reasons discussed below.

Figure 3b is the weighted combination of the two profiles. The combination rules are straightforward:

$$D(x) = \frac{\sum w_i(x)d_i(x)}{\sum w_i(x)} \quad (1)$$

$$W(x) = \sum w_i(x) \quad (2)$$

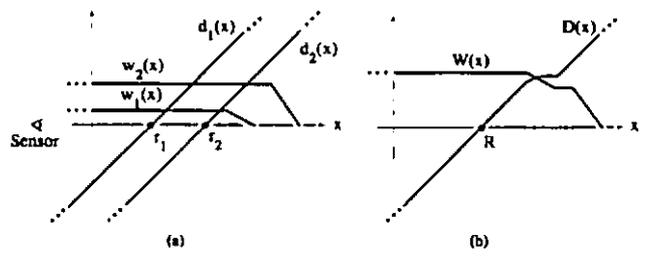


Figure 3. Signed distance and weight functions in one dimension. (a) The sensor looks down the x -axis and takes two measurements, r_1 and r_2 . $d_1(x)$ and $d_2(x)$ are the signed distance profiles, and $w_1(x)$ and $w_2(x)$ are the weight functions. In 1D, we might expect two sensor measurements to have the same weight magnitudes, but we have shown them to be of different magnitude here to illustrate how the profiles combine in the general case. (b) $D(x)$ is a weighted combination of $d_1(x)$ and $d_2(x)$, and $W(x)$ is the sum of the weight functions. Given this formulation, the zero-crossing, R , becomes the weighted combination of r_1 and r_2 and represents our best guess of the location of the surface. In practice, we truncate the distance ramps and weights to the vicinity of the range points.

where, $d_i(x)$ and $w_i(x)$ are the signed distance and weight functions from the i th range image.

Expressed as an incremental calculation, the rules are:

$$D_{i+1}(x) = \frac{W_i(x)D_i(x) + w_{i+1}(x)d_{i+1}(x)}{W_i(x) + w_{i+1}(x)} \quad (3)$$

$$W_{i+1}(x) = W_i(x) + w_{i+1}(x) \quad (4)$$

where $D_i(x)$ and $W_i(x)$ are the cumulative signed distance and weight functions after integrating the i th range image.

In the special case of one dimension, the zero-crossing of the cumulative function is at a range, R given by:

$$R = \frac{\sum w_i r_i}{\sum w_i} \quad (5)$$

i.e., a weighted combination of the acquired range values, which is what one would expect for a least squares minimization.

In principle, the distance and weighting functions should extend indefinitely in either direction. However, to prevent surfaces on opposite sides of the object from interfering with each other, we force the weighting function to taper off behind the surface. There is a trade-off involved in choosing where the weight function tapers off. It should persist far enough behind the surface to ensure that all distance ramps will contribute in the vicinity of the final zero crossing, but, it should also be as narrow as possible to avoid influencing surfaces on the other side. To meet these requirements, we force the weights to fall off at a distance equal to half the maximum uncertainty interval of the range measurements. Similarly, the signed distance and weight functions need not extend far in front of the surface. Restricting the functions to the vicinity of the surface yields a more compact representation and reduces the computational expense of updating the volume.

In two and three dimensions, the range measurements correspond to curves or surfaces with weight functions, and the signed distance ramps have directions that are consistent with the primary directions of sensor uncertainty. The uncertainties that apply to range image integration include errors in alignment between meshes as well as errors inherent in the scanning technology. A number of algorithms for aligning sets of range images have been explored and shown to yield excellent results [11][30]. The remaining error lies in the scanner itself. For optical triangulation scanners, for example, this error has been shown to be ellipsoidal about the range points, with the major axis of the ellipse aligned with the lines of sight of the laser [13][24].

Figure 4 illustrates the two-dimensional case for a range curve derived from a single scan containing a row of range samples. In practice, we use a fixed point representation for the signed distance func-

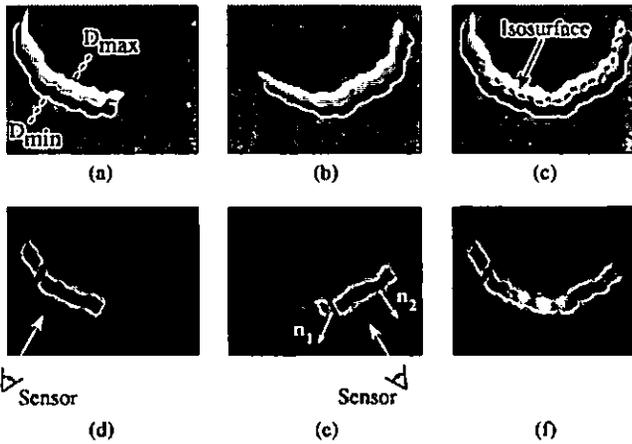


Figure 4. Combination of signed distance and weight functions in two dimensions. (a) and (d) are the signed distance and weight functions, respectively, generated for a range image viewed from the sensor line of sight shown in (d). The signed distance functions are chosen to vary between D_{min} and D_{max} , as shown in (a). The weighting falls off with increasing obliquity to the sensor and at the edges of the meshes as indicated by the darker regions in (e). The normals, n_1 and n_2 shown in (e), are oriented at a grazing angle and facing the sensor, respectively. Note how the weighting is lower (darker) for the grazing normal. (b) and (c) are the signed distance and weight functions for a range image of the same object taken at a 60 degree rotation. (c) is the signed distance function $D(x)$ corresponding to the per voxel weighted combination of (a) and (b) constructed using equations 3 and 4. (f) is the sum of the weights at each voxel, $W(x)$. The dotted green curve in (c) is the isosurface that represents our current estimate of the shape of the object.

tion, which bounds the values to lie between D_{min} and D_{max} as shown in the figure. The values of D_{min} and D_{max} must be negative and positive, respectively, as they are on opposite sides of a signed distance zero-crossing.

For three dimensions, we can summarize the whole algorithm as follows. First, we set all voxel weights to zero, so that new data will overwrite the initial grid values. Next, we tessellate each range image by constructing triangles from nearest neighbors on the sampled lattice. We avoid tessellating over step discontinuities (cliffs in the range map) by discarding triangles with edge lengths that exceed a threshold. We must also compute a weight at each vertex as described above.

Once a range image has been converted to a triangle mesh with a weight at each vertex, we can update the voxel grid. The signed distance contribution is computed by casting a ray from the sensor through each voxel near the range surface and then intersecting it with the triangle mesh, as shown in figure 5. The weight is computed by linearly interpolating the weights stored at the intersection triangle's vertices. Having determined the signed distance and weight we can apply the update formulae described in equations 3 and 4.

At any point during the merging of the range images, we can extract the zero-crossing isosurface from the volumetric grid. We restrict this extraction procedure to skip samples with zero weight, generating triangles only in the regions of observed data. We will relax this restriction in the next section.

4 Hole filling

The algorithm described in the previous section is designed to reconstruct the observed portions of the surface. Unseen portions of the surface will appear as holes in the reconstruction. While this result is an accurate representation of the known surface, the holes are aesthetically unsatisfying and can present a stumbling block to follow-on algorithms that expect continuous meshes. In [17], for example, the authors describe a method for parameterizing patches that entails

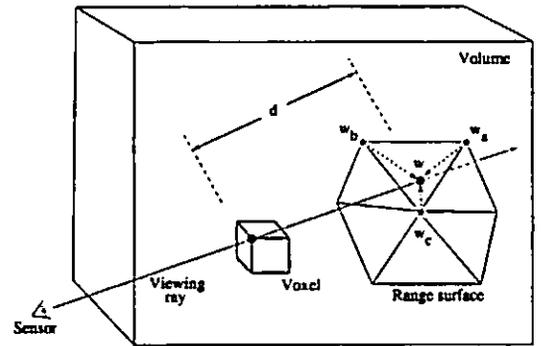


Figure 5. Sampling the range surface to update the volume. We compute the weight, w , and signed distance, d , needed to update the voxel by casting a ray from the sensor, through the voxel onto the range surface. We obtain the weight, w , by linearly interpolating the weights (w_a , w_b , and w_c) stored at neighboring range vertices. Note that for a translating sensor (like our Cyberware scanner), the sensor point is different for each column of range points.

generating evenly spaced grid lines by walking across the edges of a mesh. Gaps in the mesh prevent the algorithm from creating a fair parameterization. As another example, rapid prototyping technologies such as stereolithography typically require a "watertight" model in order to construct a solid replica [7].

One option for filling holes is to operate on the reconstructed mesh. If the regions of the mesh near each hole are very nearly planar, then this approach works well. However, holes in the meshes can be (and frequently are) highly non-planar and may even require connections between unconnected components. Instead, we offer a hole filling approach that operates on our volume, which contains more information than the reconstructed mesh.

The key to our algorithm lies in classifying all points in the volume as being in one of three states: unseen, empty, or near the surface. Holes in the surface are indicated by frontiers between unseen regions and empty regions (see Figure 6). Surfaces placed at these frontiers offer a plausible way to plug these holes (dotted in Figure 6). Obtaining this classification and generating these hole fillers leads to a straightforward extension of the algorithm described in the previous section:

1. Initialize the voxel space to the "unseen" state.
2. Update the voxels near the surface as described in the previous section. As before, these voxels take on continuous signed distance and weight values.
3. Follow the lines of sight back from the observed surface and mark the corresponding voxels as "empty". We refer to this step as *space carving*.
4. Perform an isosurface extraction at the zero-crossing of the signed distance function. Additionally, extract a surface between regions seen to be empty and regions that remain unseen.

In practice, we represent the unseen and empty states using the function and weight fields stored on the voxel lattice. We represent the unseen state with the function values $D(x) = D_{max}$, $W(x) = 0$ and the empty state with the function values $D(x) = D_{min}$, $W(x) = 0$, as shown in Figure 6b. The key advantage of this representation is that we can use the same isosurface extraction algorithm we used in the previous section without the restriction on interpolating voxels of zero weight. This extraction finds both the signed distance and hole fill isosurfaces and connects them naturally where they meet, i.e., at the corners in Figure 6a where the dotted red line meets the dashed green line. Note that the triangles that arise from interpolations across voxels of zero weight are distinct from the others: they are hole fillers.

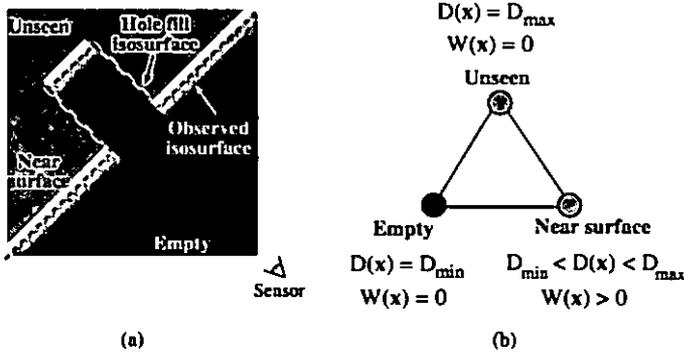


Figure 6. Volumetric grid with space carving and hole filling. (a) The regions in front of the surface are seen as empty, regions in the vicinity of the surface ramp through the zero-crossing, while regions behind remain unseen. The green (dashed) segments are the isosurfaces generated near the observed surface, while the red (dotted) segments are hole fillers, generated by tessellating over the transition from empty to unseen. In (b), we identify the three extremal voxel states with their corresponding function values.

We take advantage of this distinction when smoothing surfaces as described below.

Figure 6 illustrates the method for a single range image, and provides a diagram for the three-state classification scheme. The hole filler isosurfaces are “false” in that they are not representative of the observed surface, but they do derive from observed data. In particular, they correspond to a boundary that confines where the surface could plausibly exist. In practice, we find that many of these hole filler surfaces are generated in crevices that are hard for the sensor to reach.

Because the transition between unseen and empty is discontinuous and hole fill triangles are generated as an isosurface between these binary states, with no smooth transition, we generally observe aliasing artifacts in these areas. These artifacts can be eliminated by prefiltering the transition region before sampling on the voxel lattice using straightforward methods such as analytic filtering or super-sampling and averaging down. In practice, we have obtained satisfactory results by applying another technique: post-filtering the mesh after reconstruction using weighted averages of nearest vertex neighbors as described in [29]. The effect of this filtering step is to blur the hole fill surface. Since we know which triangles correspond to hole fillers, we need only concentrate the surface filtering on these portions of the mesh. This localized filtering preserves the detail in the observed surface reconstruction. To achieve a smooth blend between filtered hole fill vertices and the neighboring “real” surface, we allow the filter weights to extend beyond and taper off into the vicinity of the hole fill boundaries.

We have just seen how “space carving” is a useful operation: it tells us much about the structure of free space, allowing us to fill holes in an intelligent way. However, our algorithm only carves back from observed surfaces. There are numerous situations where more carving would be useful. For example, the interior walls of a hollow cylinder may elude digitization, but by seeing through the hollow portion of the cylinder to a surface placed behind it, we can better approximate its geometry. We can extend the carving paradigm to cover these situations by placing such a backdrop behind the surfaces being scanned. By placing the backdrop outside of the voxel grid, we utilize it purely for carving space without introducing its geometry into the model.

5 Implementation

5.1 Hardware

The examples in this paper were acquired using a Cyberware 3030 MS laser stripe optical triangulation scanner. Figure 1b illustrates the scanning geometry: an object translates through a plane of laser

light while the reflections are triangulated into depth profiles through a CCD camera positioned off axis. To improve the quality of the data, we apply the method of spacetime analysis as described in [6]. The benefits of this analysis include reduced range noise, greater immunity to reflectance changes, and less artifacts near range discontinuities.

When using traditional triangulation analysis implemented in hardware in our Cyberware scanner, the uncertainty in triangulation for our system follows the lines of sight of the expanding laser beam. When using the spacetime analysis, however, the uncertainty follows the lines of sight of the camera. The results described in section 6 of this paper were obtained with one or the other triangulation method. In each case, we adhere to the appropriate lines of sight when laying down signed distance and weight functions.

5.2 Software

The creation of detailed, complex models requires a large amount of input data to be merged into high resolution voxel grids. The examples in the next section include models generated from as many as 70 scans containing up to 12 million input vertices with volumetric grids ranging in size up to 160 million voxels. Clearly, time and space optimizations are critical for merging this data and managing these grids.

5.2.1 Run-length encoding

The core data structure is a run-length encoded (RLE) volume with three run types: empty, unseen, and varying. The varying fields are stored as a stream of varying data, rather than runs of constant value. Typical memory savings vary from 10:1 to 20:1. In fact, the space required to represent one of these voxel grids is usually *less* than the memory required to represent the final mesh as a list of vertices and triangle indices.

5.2.2 Fast volume traversal

Updating the volume from a range image may be likened to inverse volume rendering: instead of reading from a volume and writing to an image, we read from a range image and write to a volume. As a result, we leverage off of a successful idea from the volume rendering community: for best memory system performance, stream through the volume and the image simultaneously in scanline order [18]. In general, however, the scanlines of a range image are not aligned with the scanlines of the voxel grid, as shown in Figure 7a. By suitably resampling the range image, we obtain the desired alignment (Figure 7b). The resampling process consists of a depth rendering of the range surface using the viewing transformation specific to the lines of sight of the range sensor and using an image plane oriented to align with the voxel grid. We assign the weights as vertex “colors” to be linearly interpolated during the rendering step, an approach equivalent to Gouraud shading of triangle colors.

To merge the range data into the voxel grid, we stream through the voxel scanlines in order while stepping through the corresponding scanlines in the resampled range image. We map each voxel scanline to the correct portion of the range scanline as depicted in Figure 7d, and we resample the range data to yield a distance from the range surface. Using the combination rules given by equations 3 and 4, we update the run-length encoded structure. To preserve the linear memory structure of the RLE volume (and thus avoid using linked lists of runs scattered through the memory space), we read the voxel scanlines from the current volume and write the updated scanlines to a second RLE volume; i.e., we double-buffer the voxel grid. Note that depending on the scanner geometry, the mapping from voxels to range image pixels may not be linear, in which case care must be taken to resample appropriately [5].

For the case of merging range data only in the vicinity of the surface, we try to avoid processing voxels distant from the surface. To that end, we construct a binary tree of minimum and maximum depths for every adjacent pair of resampled range image scanlines. Before processing each voxel scanline, we query the binary tree to decide

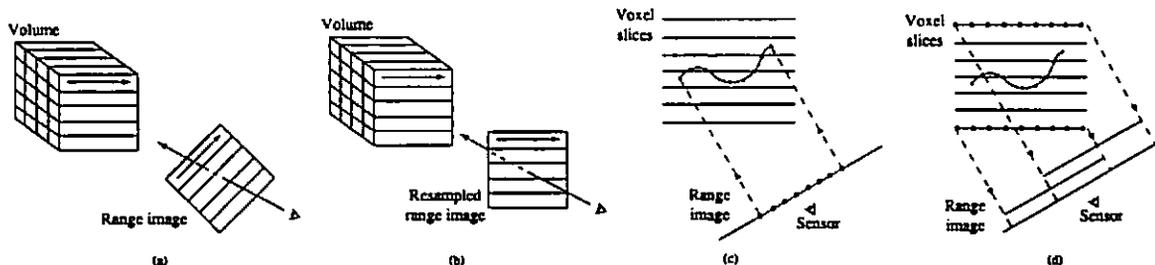


Figure 7. Range image resampling and scanline order voxel updates. (a) Range image scanlines are not in general oriented to allow for coherently streaming through voxel and range scanlines. (b) By resampling the range image, we can obtain the desired range scanline orientation. (c) Casting rays from the pixels on the range image means cutting across scanlines of the voxel grid, resulting in poor memory performance. (d) Instead, we run along scanlines of voxels, mapping them to the correct positions on the resampled range image.

which voxels, if any, are near the range surface. In this way, only relevant pieces of the scanline are processed. In a similar fashion, the space carving steps can be designed to avoid processing voxels that are not seen to be empty for a given range image. The resulting speedups from the binary tree are typically a factor of 15 without carving, and a factor of 5 with carving. We did not implement a brute-force volume update method, however we would expect the overall algorithm described here would be much faster by comparison.

5.2.3 Fast surface extraction

To generate our final surfaces, we employ a Marching Cubes algorithm [20] with a lookup table that resolves ambiguous cases [22]. To reduce computational costs, we only process voxels that have varying data or are at the boundary between empty and unseen.

6 Results

We show results for a number of objects designed to explore the robustness of our algorithm, its ability to fill gaps in the reconstruction, and its attainable level of detail. To explore robustness, we scanned a thin drill bit using the traditional method of optical triangulation. Due to the false edge extensions inherent in data from triangulation scanners [6], this particular object poses a formidable challenge, yet the volumetric method behaves robustly where the zippering method [30] fails catastrophically. The dragon sequence in Figure 11 demonstrates the effectiveness of carving space for hole filling. The use of a backdrop here is particularly effective in filling the gaps in the model. Note that we do not use the backdrop at all times, in part because the range images are much denser and more expensive to process, and also because the backdrop tends to obstruct the path of the object when automatically repositioning it with our motion control platform. Finally, the “Happy Buddha” sequence in Figure 12 shows that our method can be used to generate very detailed, hole-free models suitable for rendering and rapid manufacturing.

Statistics for the reconstruction of the dragon and Buddha models appear in Figure 8. With the optimizations described in the previous section, we were able to reconstruct the observed portions of the surfaces in under an hour on a 250 MHz MIPS R4400 processor. The space carving and hole filling algorithm is not completely optimized, but the execution times are still in the range of 3-5 hours, less than the time spent acquiring and registering the range images. For both models, the RMS distance between points in the original range images and points on the reconstructed surfaces is approximately 0.1 mm. This figure is roughly the same as the accuracy of the scanning technology, indicating a nearly optimal surface reconstruction.

7 Discussion and future work

We have described a new algorithm for volumetric integration of range images, leading to a surface reconstruction without holes. The

algorithm has a number of desirable properties, including the representation of directional sensor uncertainty, incremental and order independent updating, robustness in the presence of sensor errors, and the ability to fill gaps in the reconstruction by carving space. Our use of a run-length encoded representation of the voxel grid and synchronized processing of voxel and resampled range image scanlines make the algorithm efficient. This in turn allows us to acquire and integrate a large number of range images. In particular, we demonstrate the ability to integrate up to 70 scans into a high resolution voxel grid to generate million polygon models in a few hours. These models are free of holes, making them suitable for surface fitting, rapid prototyping, and rendering.

There are a number of limitations that prevent us from generating models from an arbitrary object. Some of these limitations arise from the algorithm while others arise from the limitations of the scanning technology. Among the algorithmic limitations, our method has difficulty bridging sharp corners if no scan spans both surfaces meeting at the corner. This is less of a problem when applying our hole-filling algorithm, but we are also exploring methods that will work without hole filling. Thin surfaces are also problematic. As described in section 3, the influences of observed surfaces extend behind their estimated positions for each range image and can interfere with distance functions originating from scans of the opposite side of a thin surface. In this respect, the apexes of sharp corners also behave like thin surfaces. While we have limited this influence as much as possible, it still places a lower limit on the thickness of surface that we can reliably reconstruct without causing artifacts such as thickening of surfaces or rounding of sharp corners. We are currently working to lift this restriction by considering the estimated normals of surfaces.

Other limitations arise from the scanning technologies themselves. Optical methods such as the one we use in this paper can only provide data for external surfaces; internal cavities are not seen. Further, very complicated objects may require an enormous amount of scanning to cover the surface. Optical triangulation scanning has the additional problem that both the laser and the sensor must observe each point on the surface, further restricting the class of objects that can be scanned completely. The reflectance properties of objects are also a factor. Optical methods generally operate by casting light onto an object, but shiny surfaces can deflect this illumination, dark objects can absorb it, and bright surfaces can lead to interreflections. To minimize these effects, we often paint our objects with a flat, gray paint.

Straightforward extensions to our algorithm include improving the execution time of the space carving portion of the algorithm and demonstrating parallelization of the whole algorithm. In addition, more aggressive space carving may be possible by making inferences about sensor lines of sight that return no range data. In the future, we hope to apply our methods to other scanning technologies and to large scale objects such as terrain and architectural scenes.

Model	Scans	Input triangles	Voxel size (mm)	Volume dimensions	Exec. time (min)	Output triangles	Holes
Dragon	61	15 M	0.35	712x501x322	56	1.7 M	324
Dragon + fill	71	24 M	0.35	712x501x322	257	1.8 M	0
Buddha	48	5 M	0.25	407x957x407	47	2.4 M	670
Buddha + fill	58	9 M	0.25	407x957x407	197	2.6 M	0

Figure 8. Statistics for the reconstruction of the dragon and Buddha models, with and without space carving.

Acknowledgments

We would like to thank Phil Lacroute for his many helpful suggestions in designing the volumetric algorithms. Afra Zomorodian wrote the scripting interface for scanning automation. Homan Igehy wrote the fast scan conversion code, which we used for range image resampling. Thanks to Bill Lorensen for his marching cubes tables and mesh decimation software, and for getting the 3D hardcopy made. Matt Pharr did the accessibility shading used to render the color Buddha, and Pat Hanrahan and Julie Dorscy made helpful suggestions for RenderMan tricks and lighting models. Thanks also to David Addleman and George Dabrowski of Cyberware for their help and for the use of their scanner. This work was supported by the National Science Foundation under contract CCR-9157767 and Interval Research Corporation.

References

- [1] C.L. Bajaj, F. Bernardini, and G. Xu. Automatic reconstruction of surfaces and scalar fields from 3D scans. In *Proceedings of SIGGRAPH '95 (Los Angeles, CA, Aug. 6-11, 1995)*, pages 109-118. ACM Press, August 1995.
- [2] J.-D. Boissonnat. Geometric structures for three-dimensional shape representation. *ACM Transactions on Graphics*, 3(4):266-286, October 1984.
- [3] C.H. Chien, Y.B. Sim, and J.K. Aggarwal. Generation of volume/surface octree from range data. In *The Computer Society Conference on Computer Vision and Pattern Recognition*, pages 254-60, June 1988.
- [4] C. I. Connolly. Cumulative generation of octree models from range data. In *Proceedings, Intl. Conf. Robotics*, pages 25-32, March 1984.
- [5] B. Curless. *Better optical triangulation and volumetric reconstruction of complex models from range images*. PhD thesis, Stanford University, 1996.
- [6] B. Curless and M. Levoy. Better optical triangulation through spacetime analysis. In *Proceedings of IEEE International Conference on Computer Vision*, pages 987-994, June 1995.
- [7] A. Dolenc. Software tools for rapid prototyping technologies in manufacturing. *Acta Polytechnica Scandinavica: Mathematics and Computer Science Series*, Ma62:1-111, 1993.
- [8] D. Eberly, R. Gardner, B. Morse, S. Pizer, and C. Scharlach. Ridges for image analysis. *Journal of Mathematical Imaging and Vision*, 4(4):353-373, Dec 1994.
- [9] H. Edelsbrunner and E.P. Mücke. Three-dimensional alpha shapes. In *Workshop on Volume Visualization*, pages 75-105, October 1992.
- [10] A. Elfes and L. Matthies. Sensor integration for robot navigation: combining sonar and range data in a grid-based representation. In *Proceedings of the 26th IEEE Conference on Decision and Control*, pages 1802-1807, December 1987.
- [11] H. Gagnon, M. Soucy, R. Bergevin, and D. Laurendeau. Registration of multiple range views for automatic 3-D model building. In *Proceedings 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 581-586, June 1994.
- [12] E. Grosso, G. Sandini, and C. Frigato. Extraction of 3D information and volumetric uncertainty from multiple stereo images. In *Proceedings of the 8th European Conference on Artificial Intelligence*, pages 683-688, August 1988.
- [13] P. Hebert, D. Laurendeau, and D. Poussart. Scene reconstruction and description: geometric primitive extraction from multiple viewed scattered data. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 286-292, June 1993.
- [14] A. Hilton, A.J. Toddart, J. Illingworth, and T. Windeatt. Reliable surface reconstruction from multiple range images. In *Fourth European Conference on Com-*

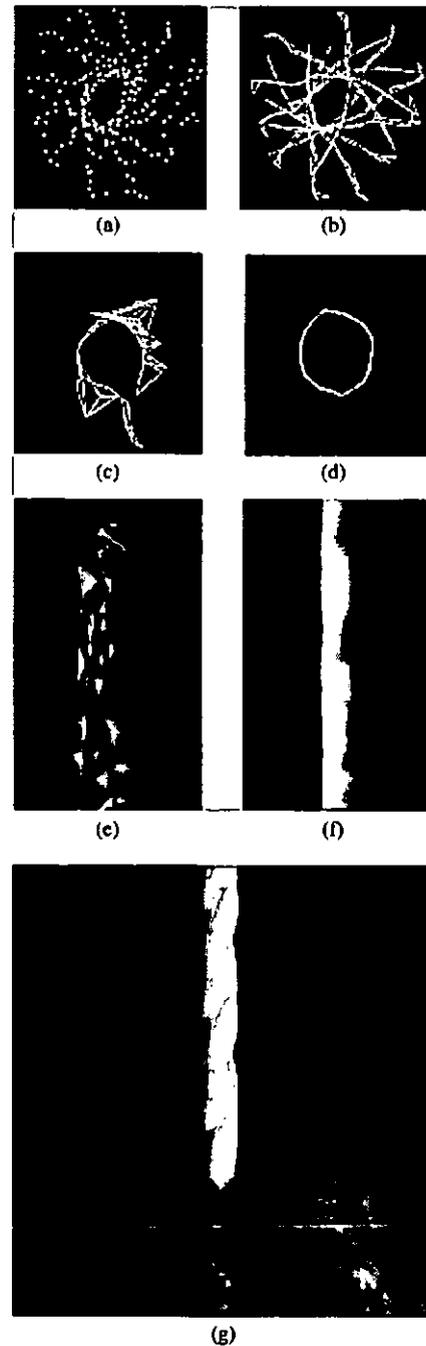


Figure 9. Merging range images of a drill bit. We scanned a 1.6 mm drill bit from 12 orientations at a 30 degree spacing using traditional optical triangulation methods. Illustrations (a) - (d) each show a plan (top) view of a slice taken through the range data and two reconstructions. (a) The range data shown as unorganized points: algorithms that operate on this form of data would likely have difficulty deriving the correct surface. (b) The range data shown as a set of wire frame tessellations of the range data: the false edge extensions pose a challenge to both polygon and volumetric methods. (c) A slice through the reconstructed surface generated by a polygon method: the zippering algorithm of Turk [31]. (d) A slice through the reconstructed surface generated by the volumetric method described in this paper. (e) A rendering of the zippered surface. (f) A rendering of the volumetrically generated surface. Note the catastrophic failure of the zippering algorithm. The volumetric method, however, produces a watertight model. (g) A photograph of the original drill bit. The drill bit was painted white for scanning.

puter Vision, volume 1, pages 117–126, April 1996.

- [15] Tsai-Hong Hong and M. O. Shneier. Describing a robot's workspace using a sequence of views from a moving camera. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 7(6):721–726, November 1985.
- [16] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 71–78, July 1992.
- [17] V. Krishnamurthy and M. Levoy. Fitting smooth surfaces to dense polygon meshes. In these proceedings.
- [18] P. Lacroix and M. Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proceedings of SIGGRAPH '94 (Orlando, FL, July 24-29, 1994)*, pages 451–458. ACM Press, July 1994.
- [19] A. Li and G. Crebbin. Octree encoding of objects from range images. *Pattern Recognition*, 27(5):727–739, May 1994.
- [20] W.E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 163–169, July 1987.
- [21] W.N. Martin and J.K. Aggarwal. Volumetric descriptions of objects from multiple views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(2):150–158, March 1983.
- [22] C. Montani, R. Scateni, and R. Scopigno. A modified look-up table for implicit disambiguation of marching cubes. *Visual Computer*, 10(6):353–355, 1994.
- [23] M. Poumesil. Generating octree models of 3D objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing*, 40(1):1–29, October 1987.
- [24] M. Rutishauser, M. Stricker, and M. Trobina. Merging range images of arbitrarily shaped objects. In *Proceedings 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 573–580, June 1994.
- [25] M. Soucy and D. Laurendeau. A general surface approach to the integration of a set of range views. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(4):344–358, April 1995.
- [26] G. Succi, G. Sandini, E. Grosso, and M. Tistarelli. 3D feature extraction from sequences of range data. In *Robotics Research. Fifth International Symposium*, pages 117–127, August 1990.
- [27] R. Szeliski. Rapid octree construction from image sequences. *CVGIP: Image Understanding*, 58(1):23–32, July 1993.
- [28] G.H. Tarbox and S.N. Gottschlich. IVIS: An integrated volumetric inspection system. In *Proceedings of the 1994 Second CAD-Based Vision Workshop*, pages 220–227, February 1994.
- [29] G. Taubin. A signal processing approach to fair surface design. In *Proceedings of SIGGRAPH '95 (Los Angeles, CA, Aug. 6-11, 1995)*, pages 351–358. ACM Press, August 1995.
- [30] G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Proceedings of SIGGRAPH '94 (Orlando, FL, July 24-29, 1994)*, pages 311–318. ACM Press, July 1994.
- [31] Robert Weinstock. *The Calculus of Variations, with Applications to Physics and Engineering*. Dover Publications, 1974.

A Isosurface as least squares minimizer

It is possible to show that the isosurface of the weighted signed distance function is equivalent to a least squares minimization of squared distances between points on the range surfaces and points on the desired reconstruction. The key assumptions are that the range sensor is orthographic and that the range errors are independently distributed along sensor lines of sight. A full proof is beyond the scope of this paper, but we provide a sketch here. See [5] for details.

Consider a region, R , on the desired surface, f , which is observed by n range images. We define the error between an observed range surface and a possible reconstructed surface as the integral of the weighted squared distances between points on the range surface and the reconstructed surface. These distances are taken along the lines of sight of the sensor, commensurate with the predominant directions of uncertainty (see Figure 10). The total error is the sum of the integrals for the n range images:

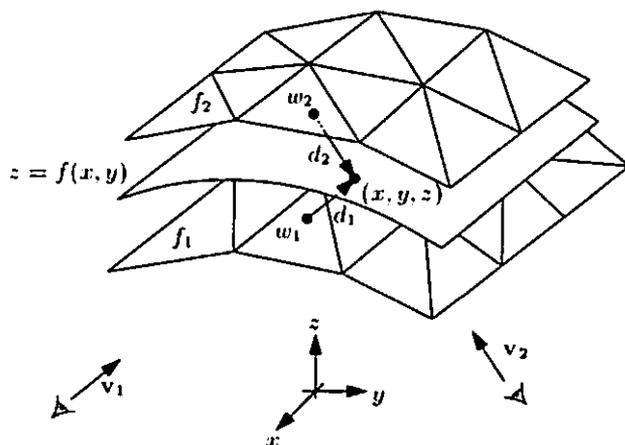


Figure 10. Two range surfaces, f_1 and f_2 , are tessellated range images acquired from directions v_1 and v_2 . The possible range surface, $z = f(x, y)$, is evaluated in terms of the weighted squared distances to points on the range surfaces taken along the lines of sight to the sensor. A point, (x, y, z) , is shown here being evaluated to find its corresponding signed distances, d_1 and d_2 , and weights, w_1 and w_2 .

$$E(f) = \sum_{i=1}^n \iint_{A_i} w_i(s, t, f) d_i(s, t, f)^2 ds dt \quad (6)$$

where each (s, t) corresponds to a particular sensor line of sight for each range image, A_i is the domain of integration for the i 'th range image, and $w_i(s, t, f)$ and $d_i(s, t, f)$ are the weights and signed distances taken along the i 'th range image's lines of sight.

Now, consider a canonical domain, A , on a parameter plane, (x, y) , over which R is a function $z = f(x, y)$. The total error can be rewritten as an integration over the canonical domain:

$$E(z) = \iint_A \sum_{i=1}^n [w_i(x, y, z) d_i(x, y, z)] \left[\mathbf{v}_i \cdot \left(\frac{\partial z}{\partial x}, \frac{\partial z}{\partial y}, -1 \right) \right] dx dy \quad (7)$$

where \mathbf{v}_i is the sensing direction of the i 'th range image, and the weights and distances are evaluated at each point, (x, y, z) , by first mapping them to the lines of sight of the corresponding range image. The dot product represents a correction term that relates differential areas in A to differential areas in A_i . Applying the calculus of variations [31], we can construct a partial differential equation for the z that minimizes this integral. Solving this equation we arrive at the following relation:

$$\sum_{i=1}^n \partial_{\mathbf{v}_i} [w_i(x, y, z) d_i(x, y, z)^2] = 0 \quad (8)$$

where $\partial_{\mathbf{v}_i}$ is the directional derivative along \mathbf{v}_i . Since the weight associated with a line of sight does not vary along that line of sight, and the signed distance has a derivative of unity along the line of sight, we can simplify this equation to:

$$\sum_{i=1}^n w_i(x, y, z) d_i(x, y, z) = 0 \quad (9)$$

This weighted sum of signed distances is the same as what we compute in equations 1 and 2, without the division by the sum of the weights. Since this divisor is always positive, the isosurface we extract in section 3 is exactly the least squares minimizing surface described here.

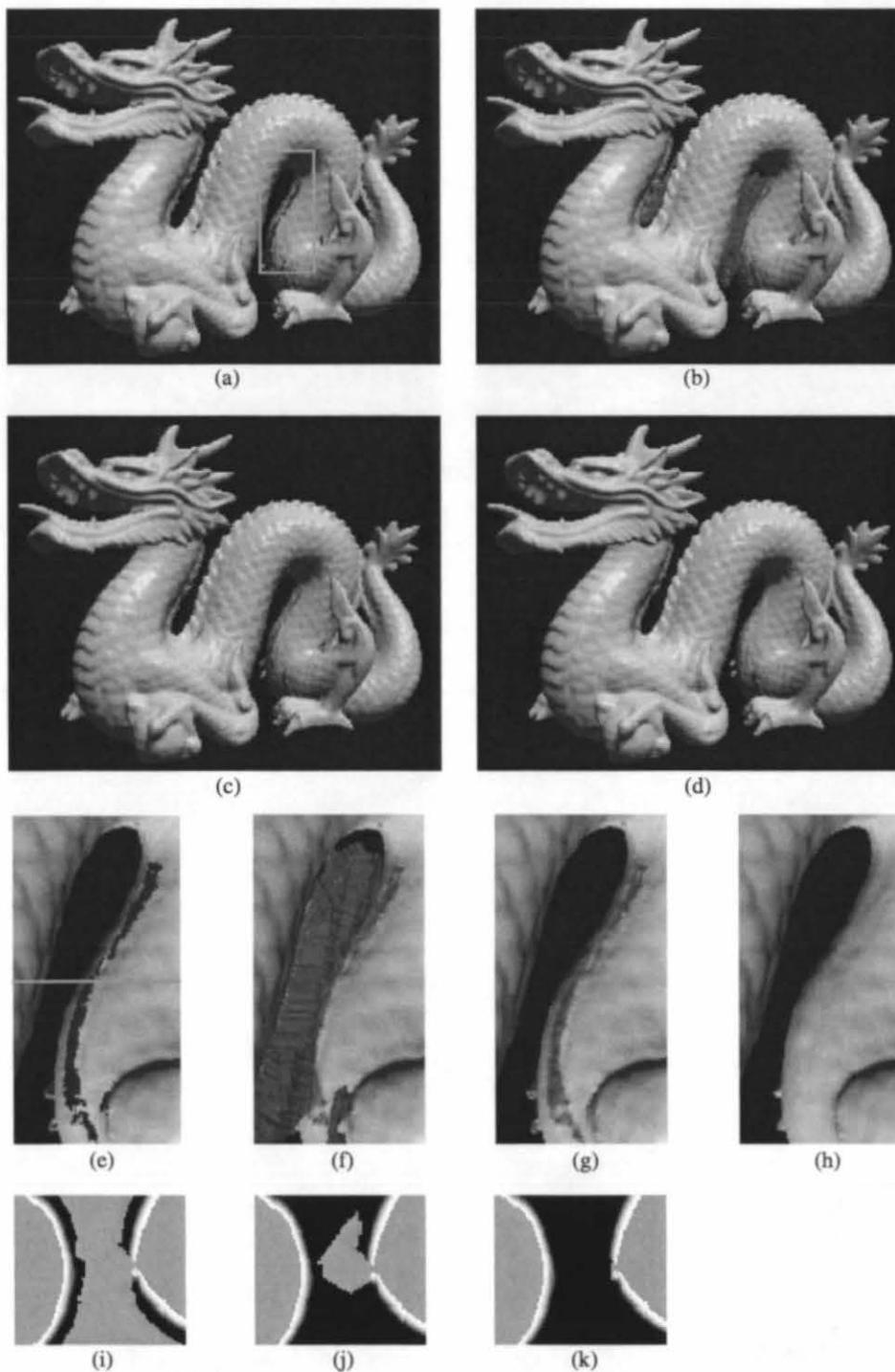


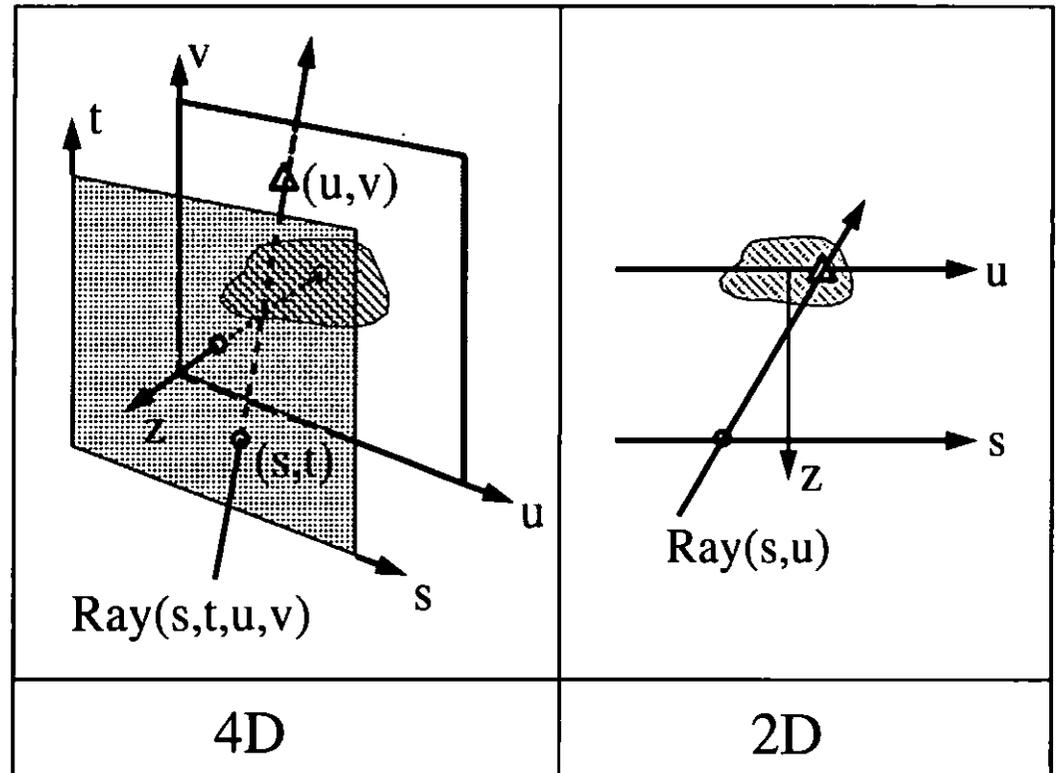
Figure 11. Reconstruction of a dragon. Illustrations (a) - (d) are full views of the dragon. Illustrations (e) - (h) are magnified views of the section highlighted by the green box in (a). Regions shown in red correspond to hole fill triangles. Illustrations (i) - (k) are slices through the corresponding volumetric grids at the level indicated by the green line in (e). (a)(e)(i) Reconstruction from 61 range images without space carving and hole filling. The magnified rendering highlights the holes in the belly. The slice through the volumetric grid shows how the signed distance ramps are maintained close to the surface. The gap in the ramps leads to a hole in the reconstruction. (b)(f)(j) Reconstruction with space carving and hole filling using the same data as in (a). While some holes are filled in a reasonable manner, some large regions of space are left untouched and create extraneous tessellations. The slice through the volumetric grid reveals that the isosurface between the unseen (brown) and empty (black) regions will be connected to the isosurface extracted from the distance ramps, making it part of the connected component of the dragon body and leaving us with a substantial number of false surfaces. (c)(g)(k) Reconstruction with 10 additional range images using "backdrop" surfaces to effect more carving. Notice how the extraneous hole fill triangles nearly vanish. The volumetric slice shows how we have managed to empty out the space near the belly. The bumpiness along the hole fill regions of the belly in (g) corresponds to aliasing artifacts from tessellating over the discontinuous transition between unseen and empty regions. (d)(h) Reconstruction as in (c)(g) with filtering of the hole fill portions of the mesh. The filtering operation blurs out the aliasing artifacts in the hole fill regions while preserving the detail in the rest of the model. Careful examination of (h) reveals a faint ridge in the vicinity of the smoothed hole fill. This ridge is actual geometry present in all of the renderings, (e)-(h). The final model contains 1.8 million polygons and is watertight.



Figure 12. Reconstruction and 3D hardcopy of the "Happy Buddha". The original is a plastic and rosewood statuette that stands 20 cm tall. Note that the camera parameters for each of these images is different, creating a slightly different perspective in each case. (a) Photograph of the original after spray painting it matte gray to simplify scanning. (b) Gouraud-shaded rendering of one range image of the statuette. Scans were acquired using a Cyberware scanner, modified to permit spacetime triangulation [6]. This figure illustrates the limited and fragmentary nature of the information available from a single range image. (c) Gouraud-shaded rendering of the 2.4 million polygon mesh after merging 48 scans, but before hole-filling. Notice that the reconstructed mesh has at least as much detail as the single range image, but is less noisy; this is most apparent around the belly. The hole in the base of the model corresponds to regions that were not observed directly by the range sensor. (d) RenderMan rendering of an 800,000 polygon decimated version of the hole-filled and filtered mesh built from 58 scans. By placing a backdrop behind the model and taking 10 additional scans, we were able to see through the space between the base and the Buddha's garments, allowing us to carve space and fill the holes in the base. (e) Photograph of a hardcopy of the 3D model, manufactured by 3D Systems, Inc., using stereolithography. The computer model was sliced into 500 layers, 150 microns apart, and the hardcopy was built up layer by layer by selectively hardening a liquid resin. The process took about 10 hours. Afterwards, the model was sanded and bead-blasted to remove the stair-step artifacts that arise during layered manufacturing.

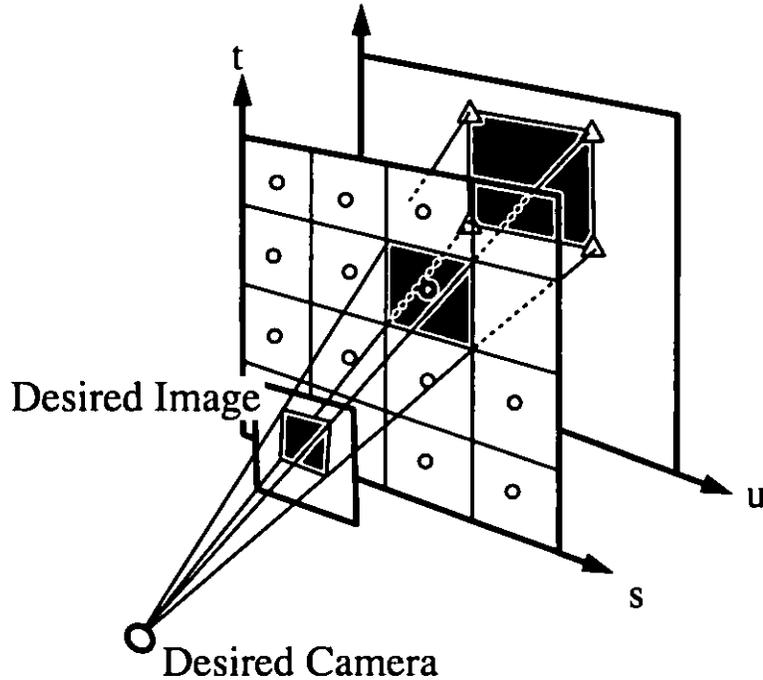
Polyhedral Geometry and the Two-Plane Parameterization

▷ two parallel plane parametrization (2PP)



▷ why use the 2PP

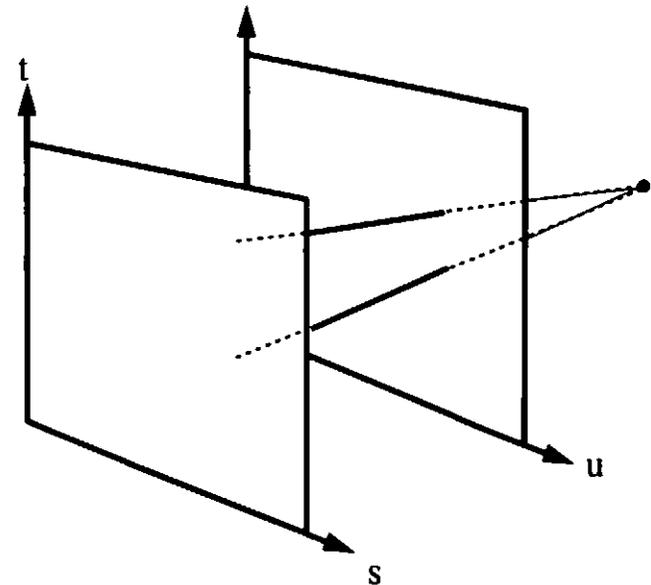
- can parameterize a ray with one reciprocal
- data extraction = texture mapping polygons



- minor disadvantages:
uneven sampling rate
not complete

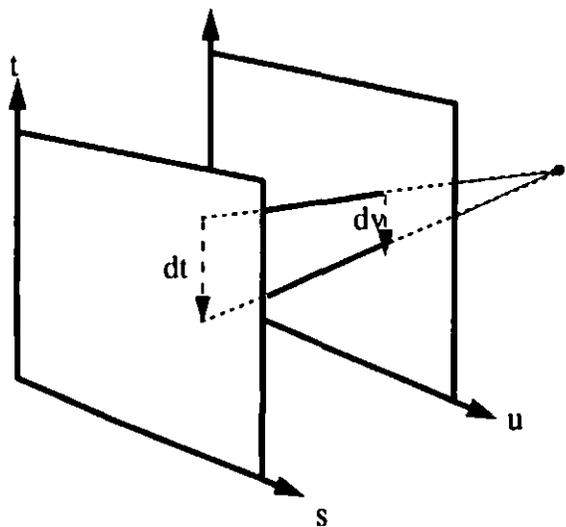
▷ algebra of 2PP

- suppose our scene is black,
with one single white point in the scene
what will the lumigraph look like?
- what is the 2PP subset corresponding
to all lines passing through some point?



▷ algebra of 2PP

- suppose our scene is black, with one single white point in the scene what will the lumigraph look like?
- what is the 2PP subset corresponding to all lines passing through some point?

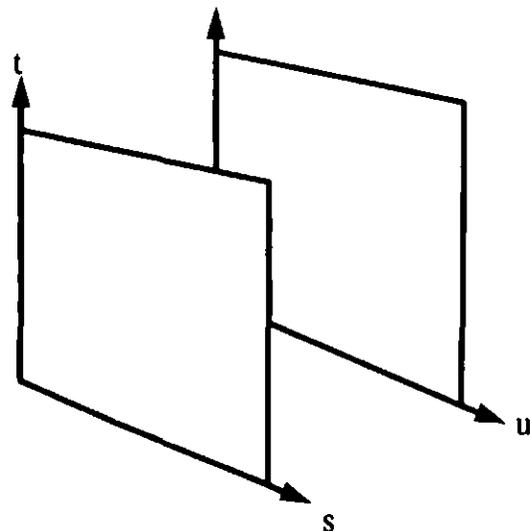


- it has two degrees of freedom (s, t)
- it is affine (flat)

▷ converse questions

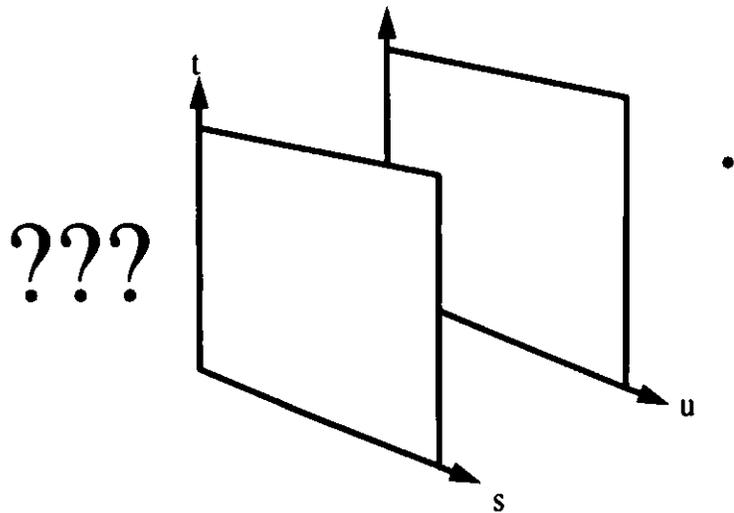
- suppose i give you a lumigraph it is all black except for a white 2d affine subset
- does this correspond to a one-white-dot scene?

???



▷ converse questions

- suppose i give you a lumigraph
it is all black
except for a white 2d affine subset
- does this correspond to a one-white-dot scene?



- how many dof are there for the white point?
- how many dof in choosing a 2d affine subset?

▷ why are we interested in this?
memory layout

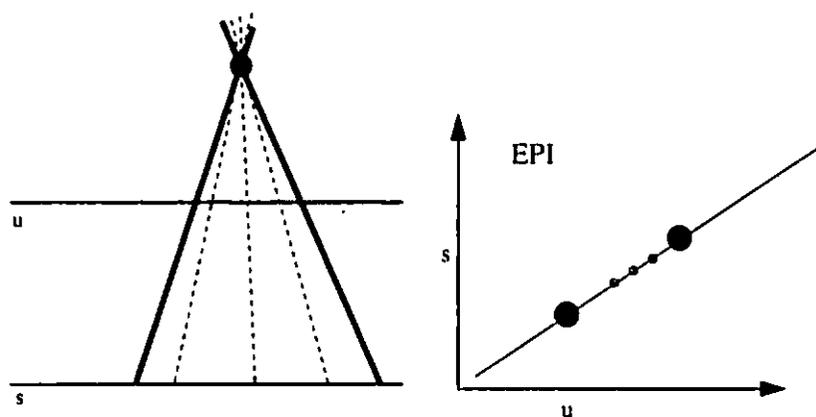
- image extraction constant time
- memory access is the bottleneck
- we wish to understand memory access patterns
optimize performance
- data for extracting an image
all lines through camera center
- requested data will lie in a 3dof family of
2d affine subsets of the 4d data array

▷ why are we interested in this?
compression

- data size is large
hundreds of Mbyte/lumigraph
- 4D data is far from random
- geometry/shading
--> coherence --> compression
- surface region with coherent texture ->
4D region with coherent data
- we want to understand the shapes of
potential 4D regions

▷ why are we interested in this?
generalized EPI

- EPI: move camera along line
for a fixed scan line number
stack up scan lines into an EPI
look for lines in EPI
a line corresponds to a feature point
- limitation: camera must move along line
only 3D set of data
- in 4D setting camera can move freely gathering
ray data
- how to interpret 4D structure



▷ lines through a point

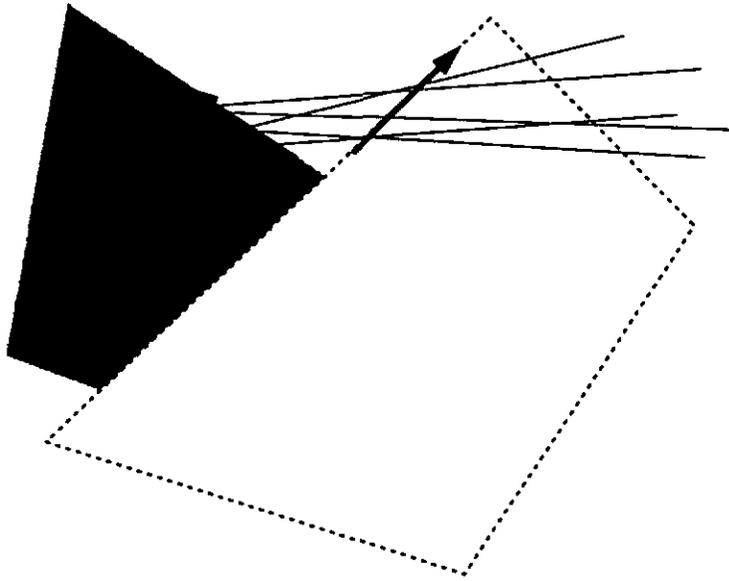
- point = (px, py, pz)
- for each (s,t) solve for the (u,v) such that the line (s,t,u,v) passes through the point
- resulting (s,t,u,v) is:
 $(s, t, s(1+1/pz) - px/pz, t(1+1/pz) - py/pz)$
- class of affine 2d subsets

▷ lines through a line

- parameterize position along the line as λ
- for each (s,t) and λ , solve for the (u,v) such that the line (s,t,u,v) passes through that point on line
- eliminate λ
- if the 2pp paramter of this line is (s_0,t_0,u_0,v_0) then resulting (s,t,u,v) must satisfy

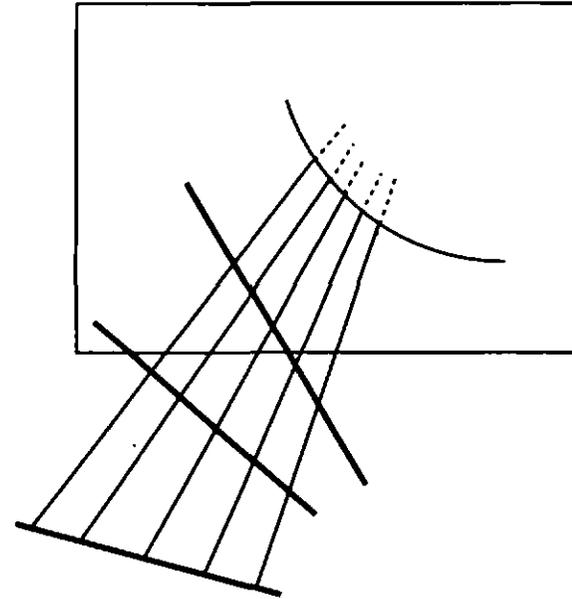
$$0 = (s-s_0)(v-v_0) - (t-t_0)(u-u_0)$$

▷ occlusion



- lines through two lines:
resulting (s,t,u,v) must satisfy
$$0 = (s-s_0)(v-v_0) - (t-t_0)(u-u_0)$$
$$0 = (s-s_1)(v-v_1) - (t-t_1)(u-u_1)$$
- can be reduced to one linear and one quadratic

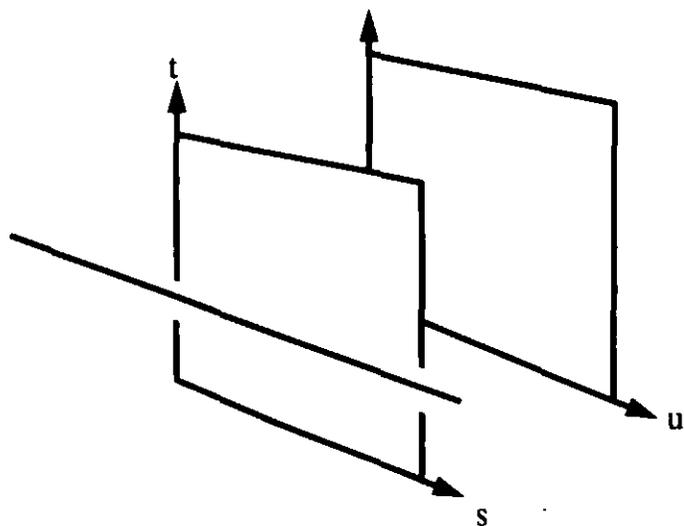
▷ occlusion



- lines through three lines:
resulting (s,t,u,v) must satisfy
$$0 = (s-s_0)(v-v_0) - (t-t_0)(u-u_0)$$
$$0 = (s-s_1)(v-v_1) - (t-t_1)(u-u_1)$$
$$0 = (s-s_2)(v-v_2) - (t-t_2)(u-u_2)$$
- can be reduced to two linear and one quadratic

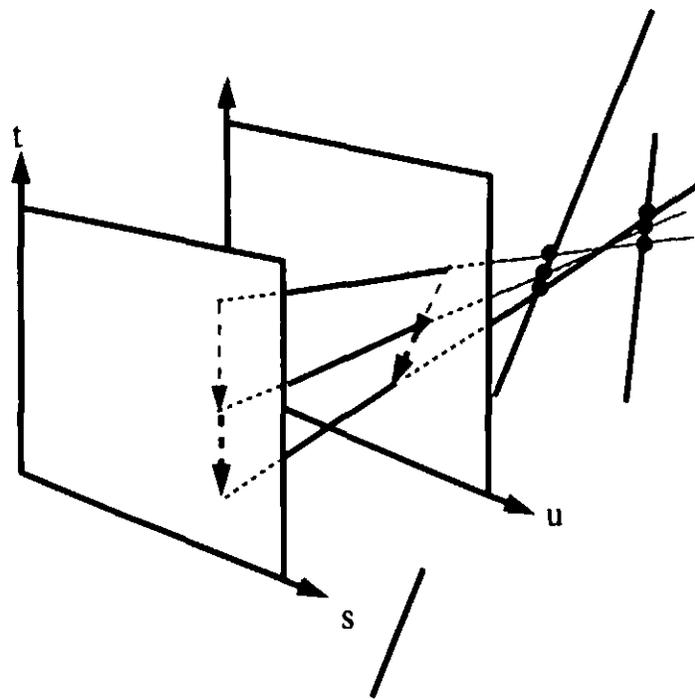
▷ converse questions

- shallow line: line parallel to the two planes
- lines through one shallow line satisfy one linear equation



▷ converse questions

- lines through two shallow lines satisfy two linear equations: some 2D affine subset
- six degrees of freedom



▷ Plücker coordinates

- line parameterization that uses 6 coordinates
- arbitrary scale
- one extra quadratic constraint

- mapping between (s,t,u,v) is non linear

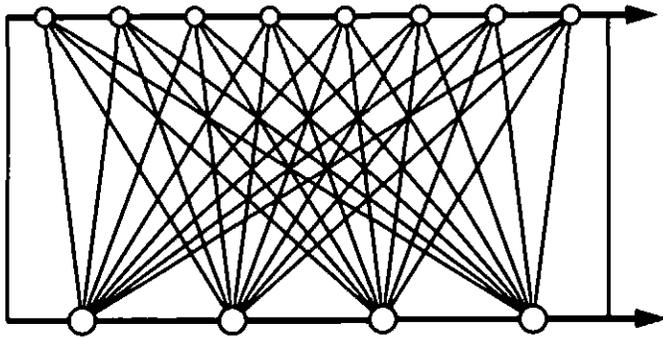
- behaves similarly to 2PP
 - lines through point : affine
 - lines through two lines : quadric
 - lines through three lines : conic

▷ wrap up

- analyzed algebra of polyhedral geometry
- got some feeling for the 4D structure

- future work:
put this understanding to some practical good

The Lumigraph



Steven Gortler

Division of Engineering
and Applied Sciences

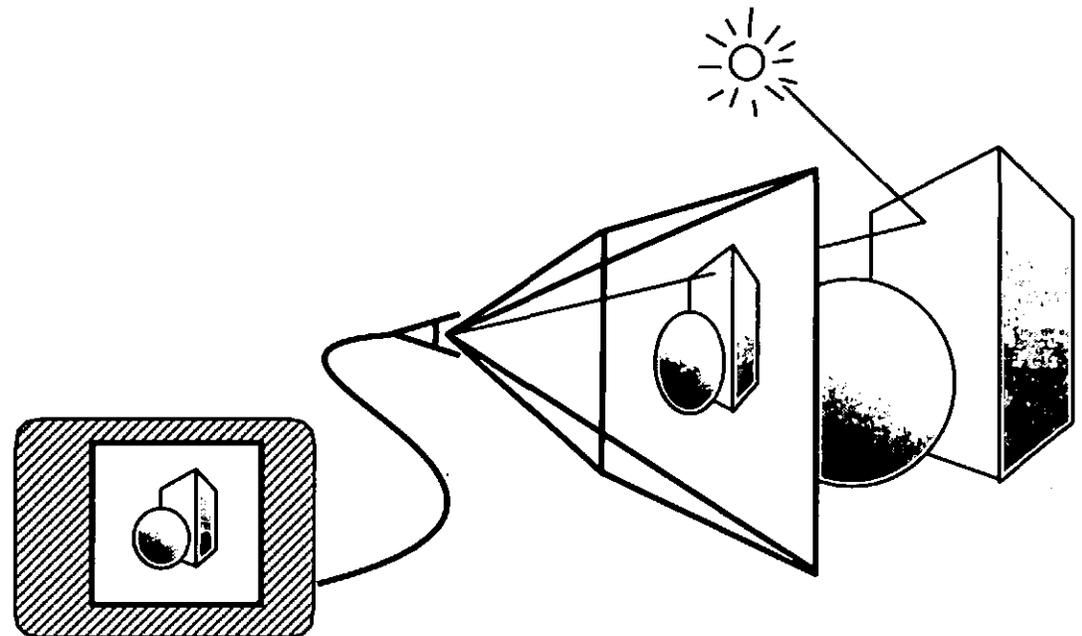
Harvard University

▷ Traditional computer graphics

○ image =
geometric/material/lights model +
camera model

▷ problems:

- modeling difficulty (geometry, materials, lights)
- rendering complexity
 - geometric
 - light simulation
- limited realism



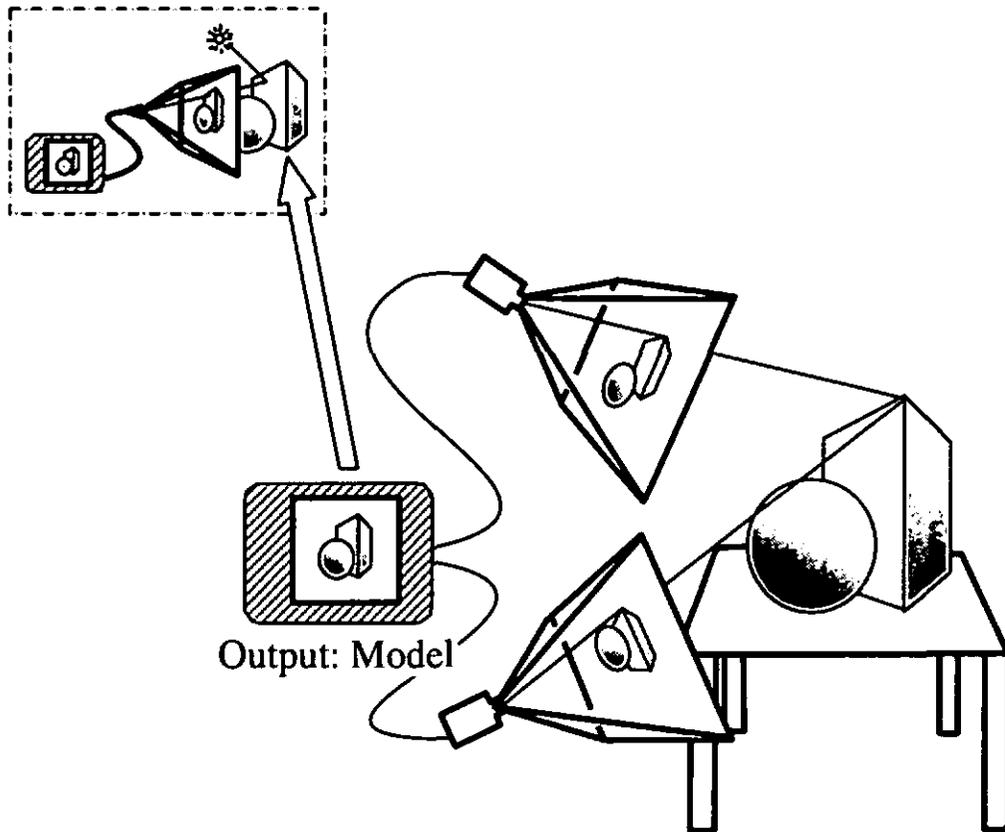
Output: Picture

▷ Traditional object capture (vision)

- use stereo images to deduce depth
- can use structured light methods

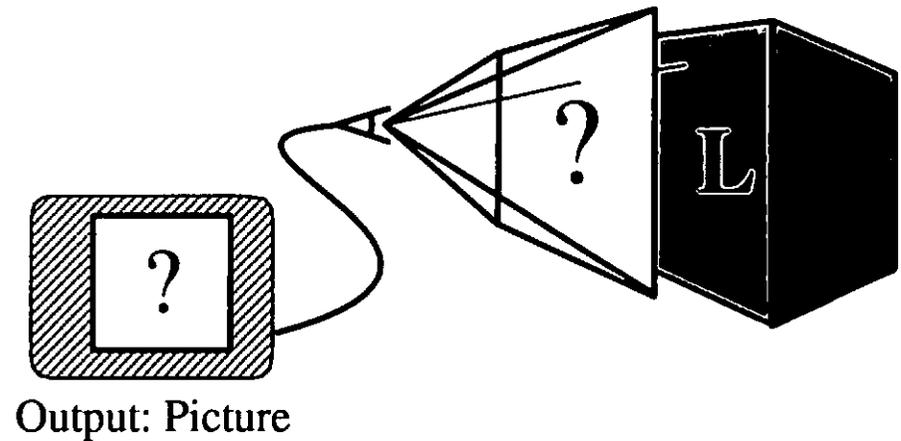
▷ problems:

- not robust
- can't capture fine detail
- can't capture surface properties
- still must render



▷ Rendering from Lumigraph

- image = Lumigraph + camera model
- no penalty for scene complexity



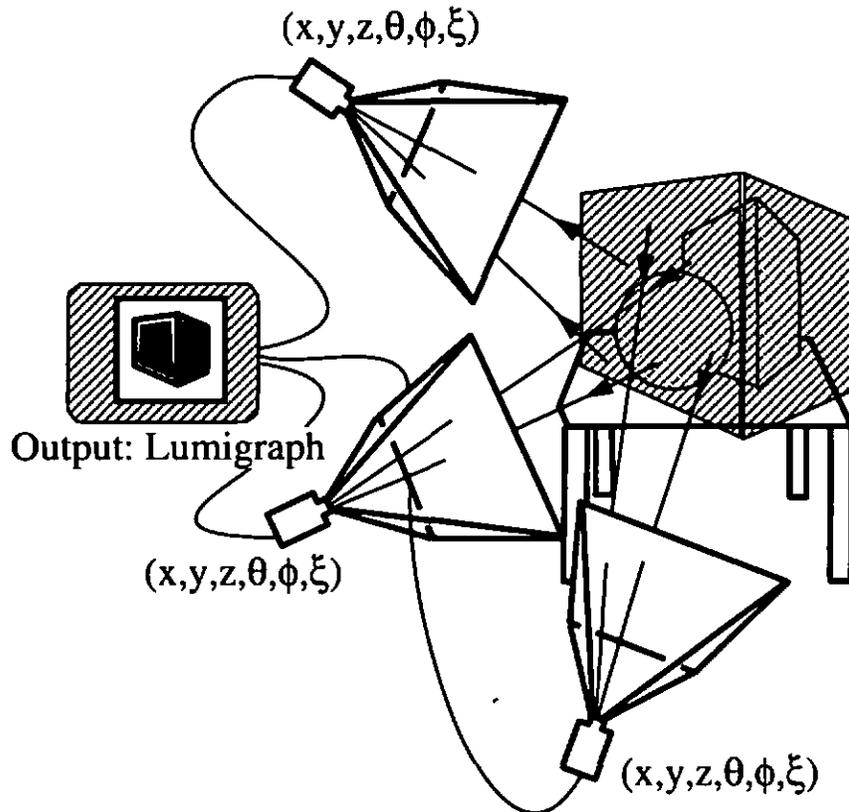
What is a Lumigraph?

▷ Lumigraph:

- a representation of light resulting from a scene

▷ Capturing a Lumigraph

- use multiple images and camera "pose"
- develop directly into Lumigraph function

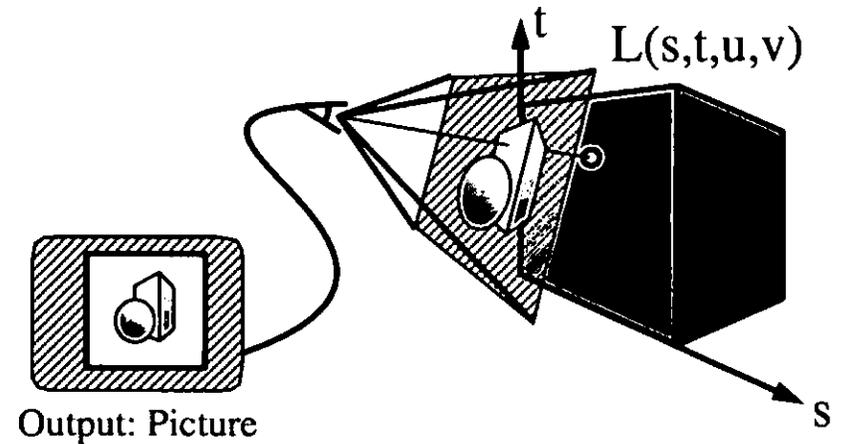


▷ Lumigraph:

- a representation of light resulting from a scene

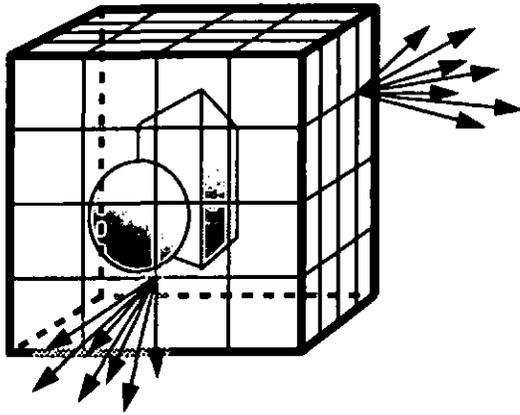
▷ Rendering from Lumigraph

- image = Lumigraph + camera model
- no penalty for scene complexity



- ▷ Lumigraph function:
 - for all points on surrounding surface
 - for all directions
 - intensity of ray

- ▷ our task: represent this 4D function



- ▷ assumptions:
 - air is transparent
 - we are outside the convex hull

- ▷ Related work:

View interpolation
(Chen & Williams)

Fundamental Matrix
(Leveau & Feugeras)

Plenoptic Modeling
(McMillan & Bishop)

▷ Related work:

Stereographic Holograms
(Benton)

Near-field Photometry
(Ashdown)

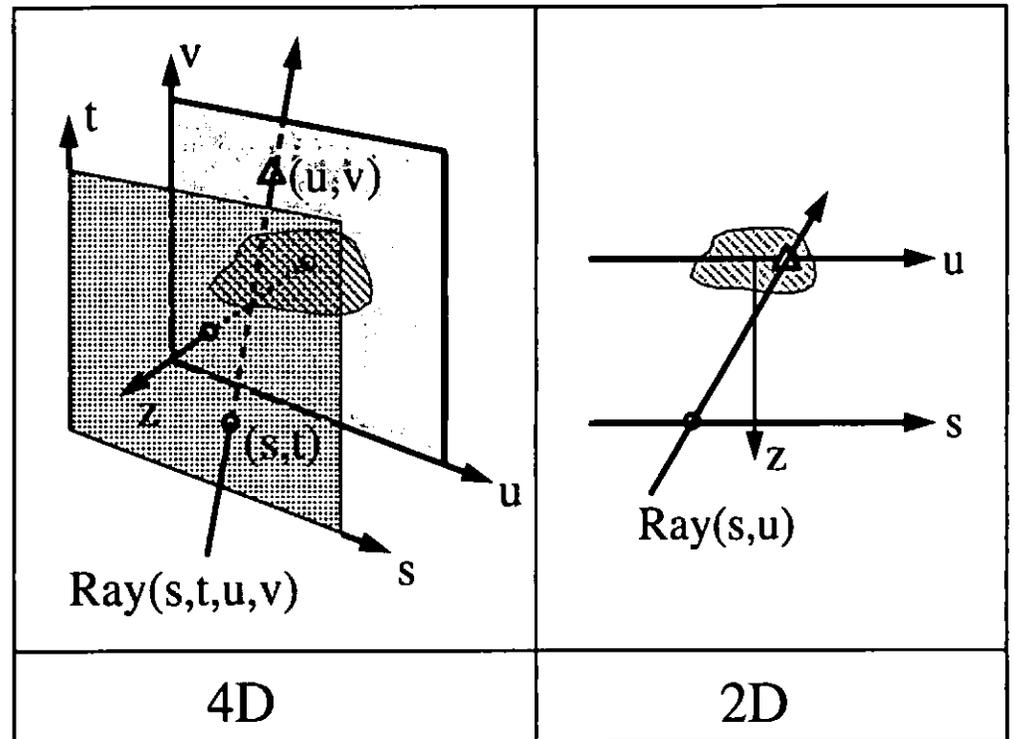
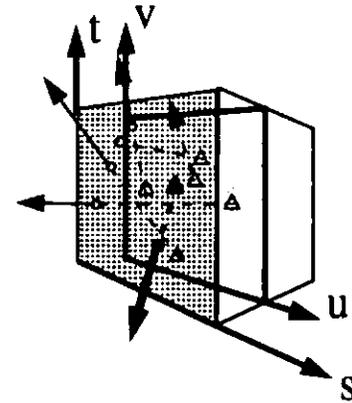
Global Illumination
(Lewis & Fournier)

3D light volumes
(Katayama et. al)

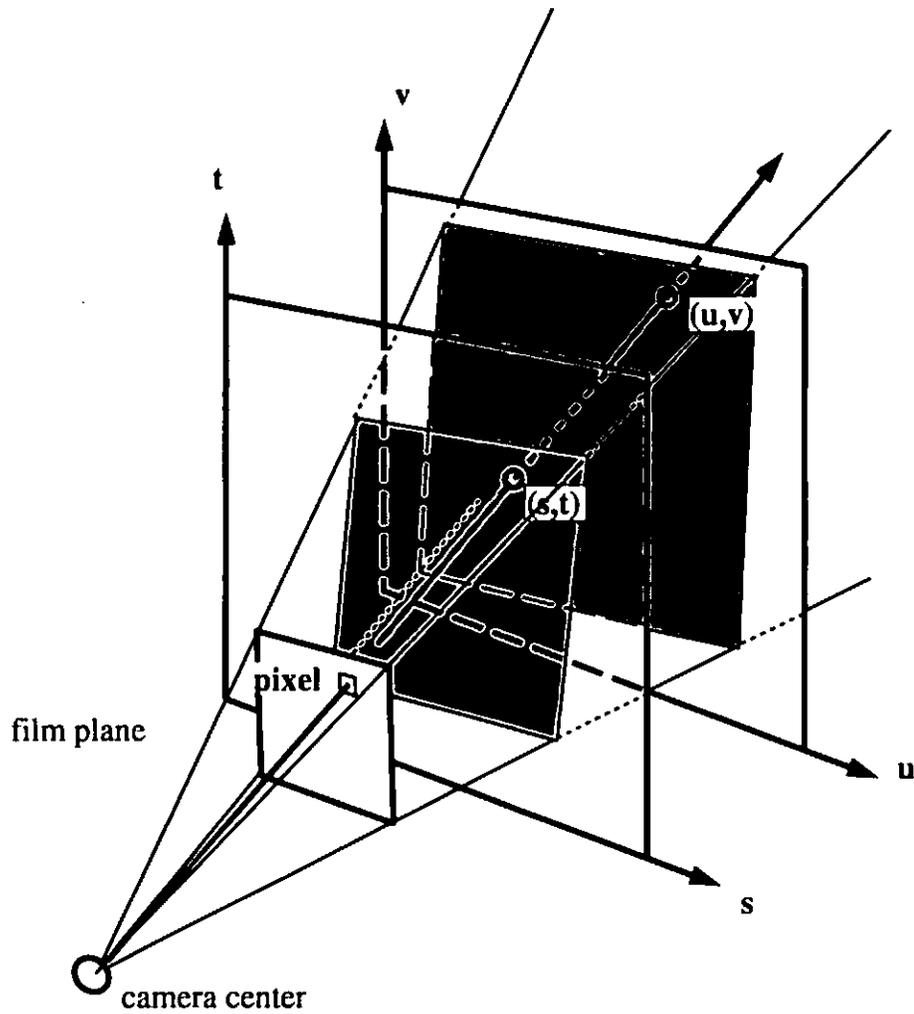
QuickTime VR
(Chen)

4D light field
(Levoy & Hanrahan)

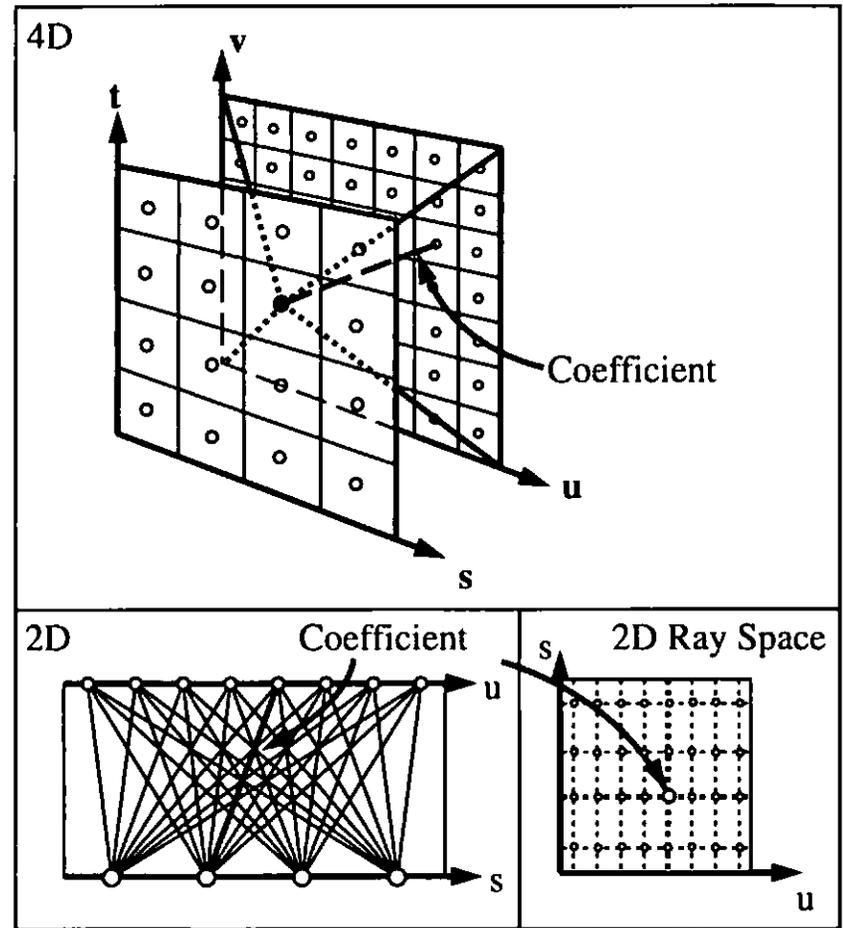
▷ Parameterize the Lumigraph function



▷ Relationship between a Lumigraph and an image



- ▷ Discretize the Parameter space
 one 4D (s, t, u, v) array of coefficients
 --or--
 one (u, v) image for each (s, t) grid point

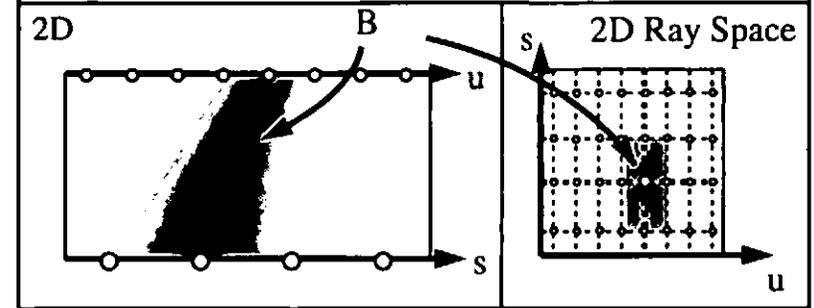
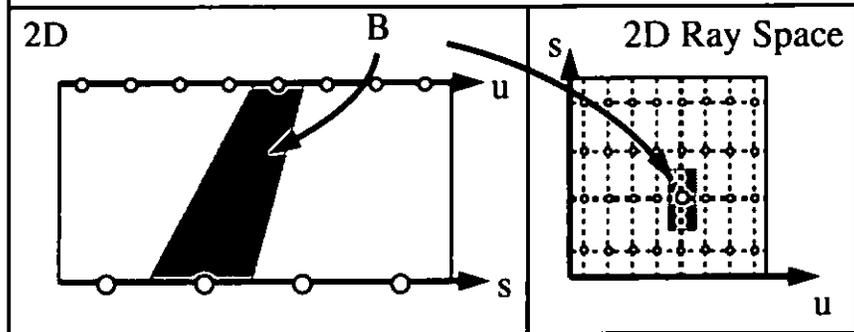
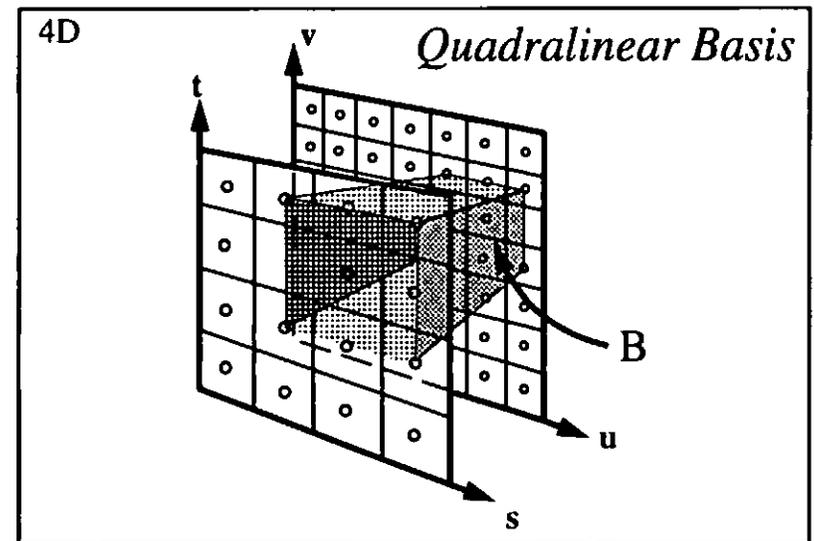
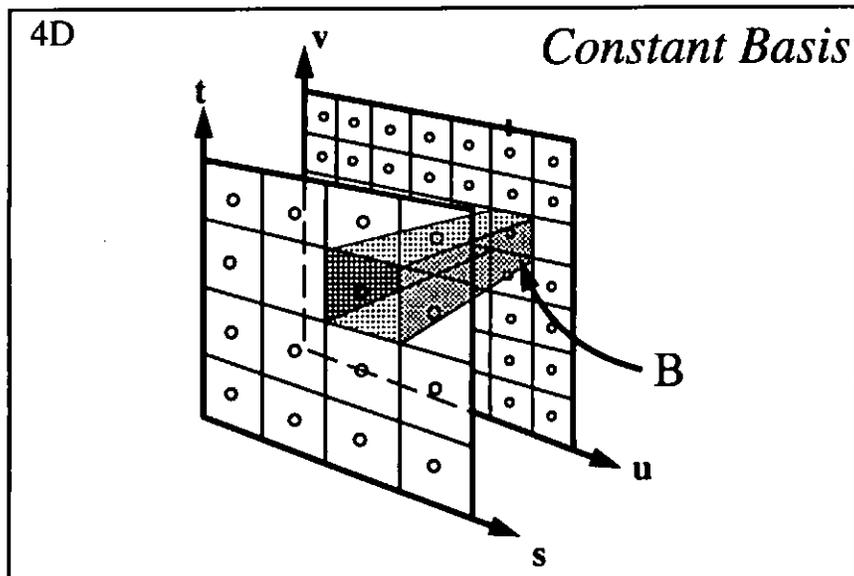


▷ Approximating the Lumigraph

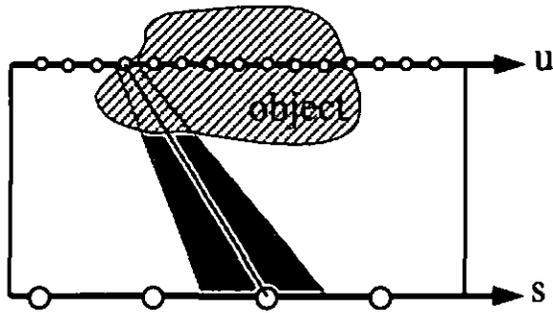
- Reconstruct the function using basis functions

$$\tilde{L}(s,t,u,v) = \sum x B(s,t,u,v)$$

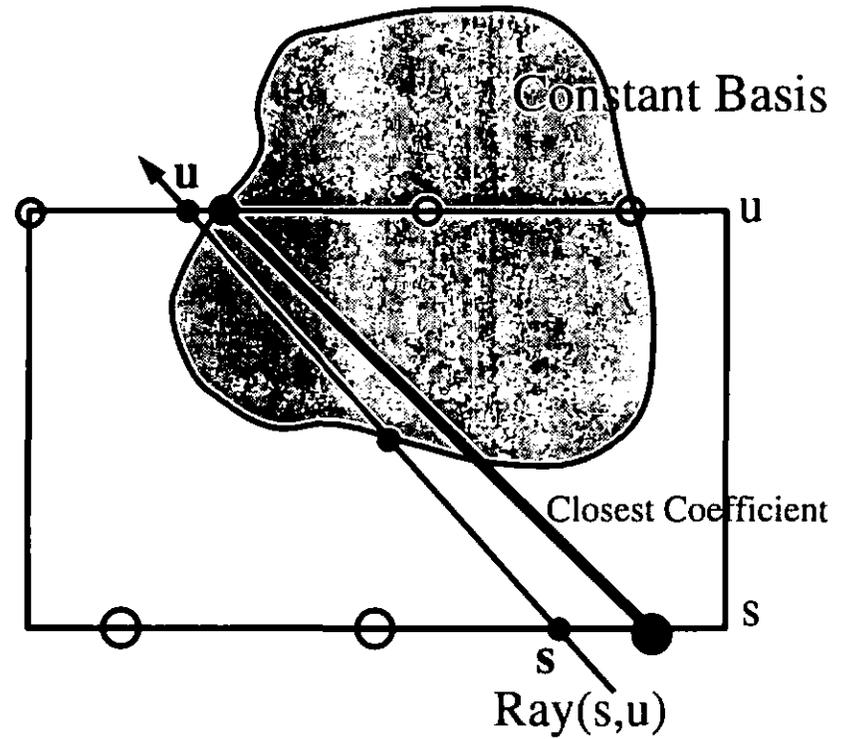
$$\tilde{L}(s,t,u,v) = \sum x B(s,t,u,v)$$



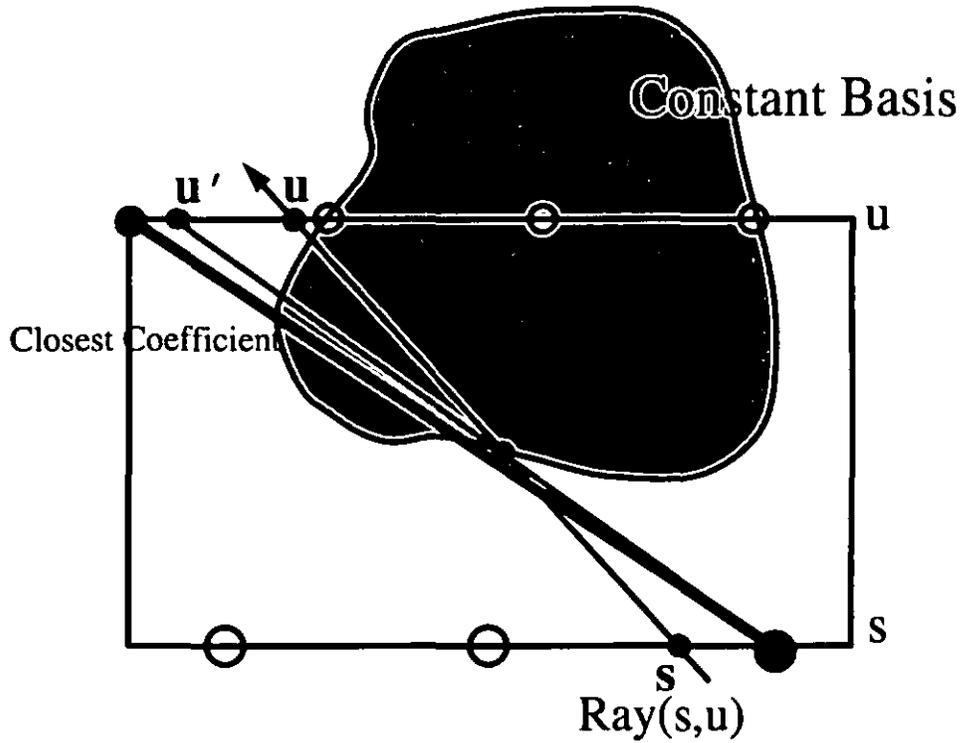
- ▷ Calculate coefficients
 - integral/average over a bundle of rays



- ▷ Without Geometry

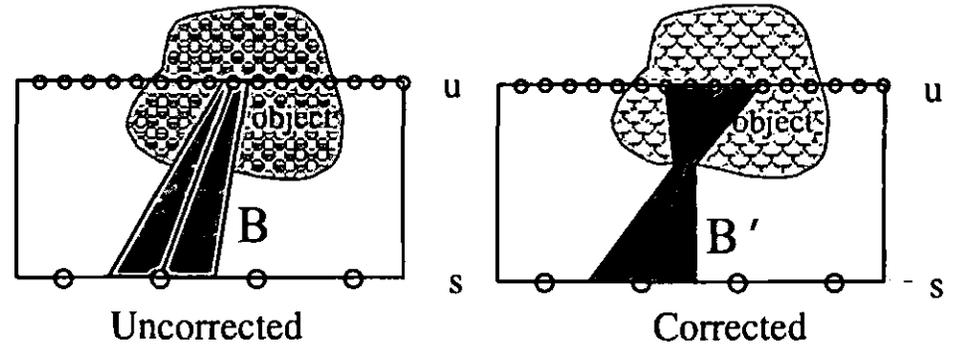


▷ With Geometric Correction

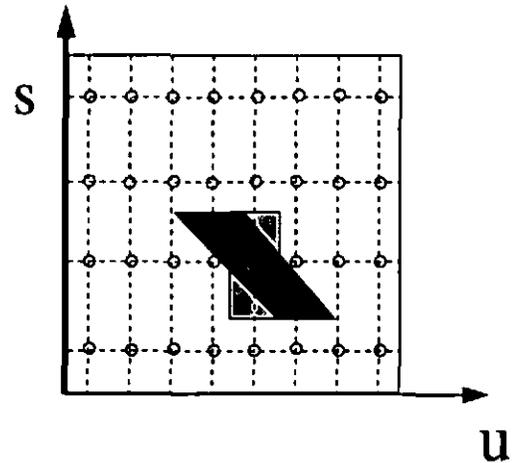


▷ Geometric Correction

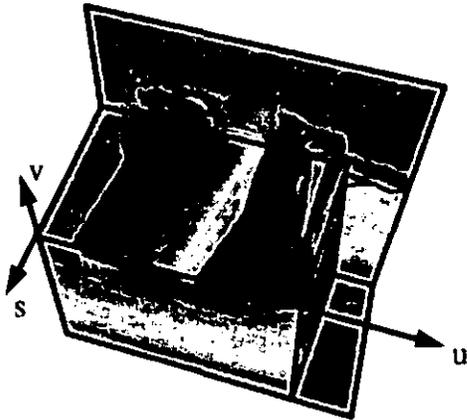
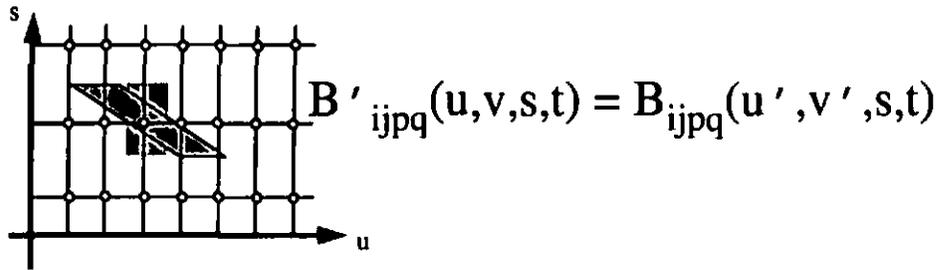
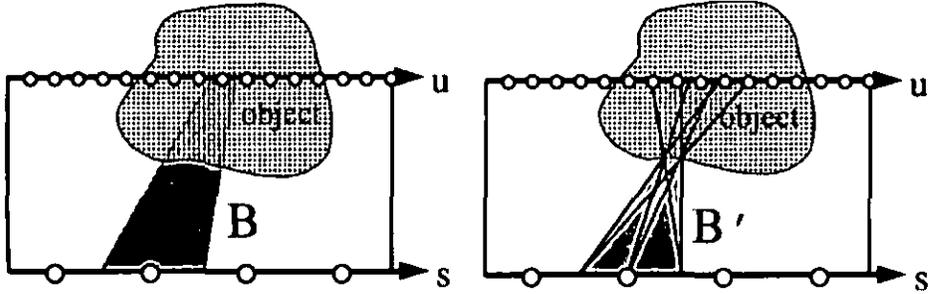
Geometric corrected basis functions



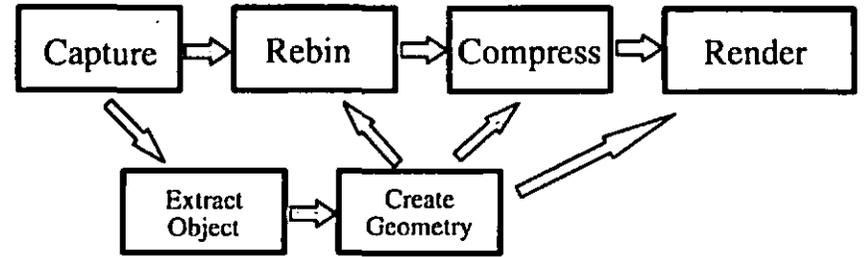
$$B'(u,v,s,t) = B(u',v',s,t)$$

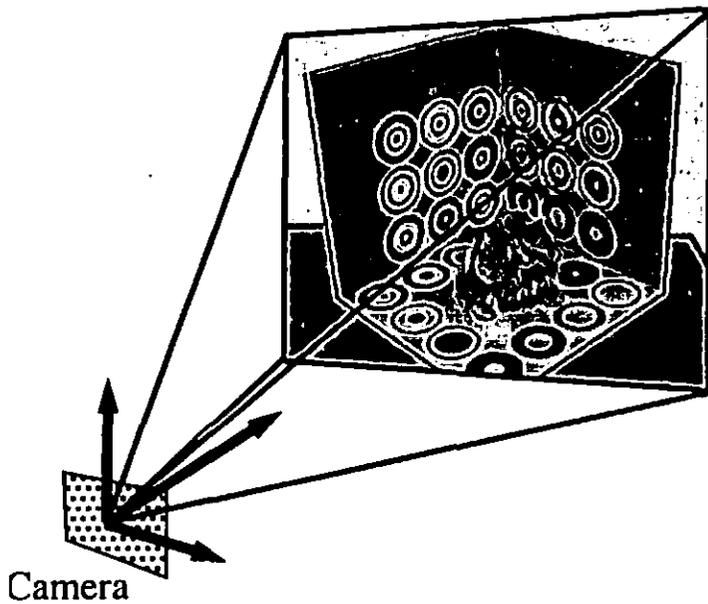
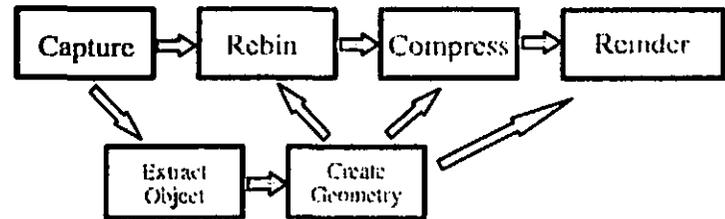
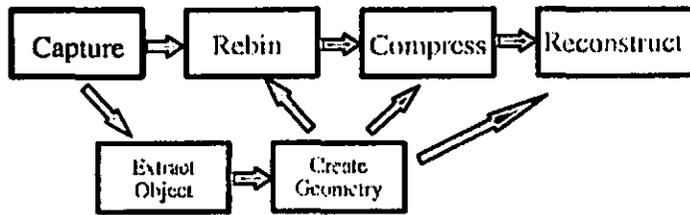


Geometric correction



► Lumigraph System

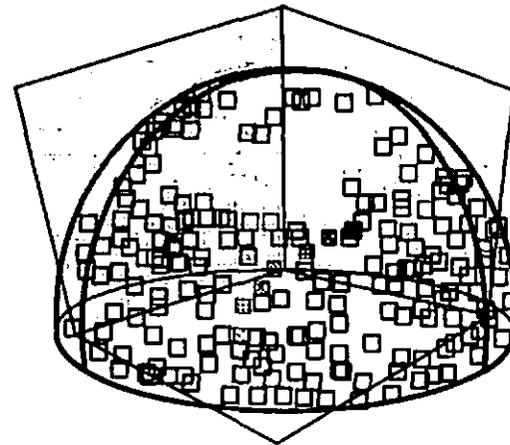


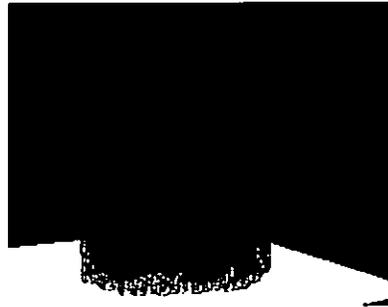
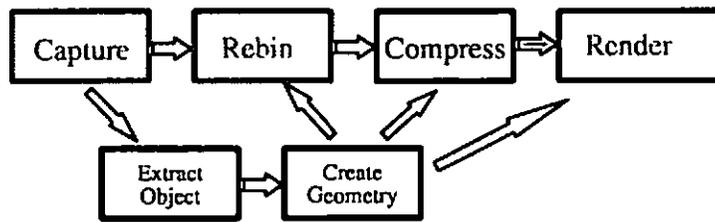


calibrate camera
for each image

- find markers
- solve for camera pose

▷ Provide the user with graphical feedback about camera position to aid in "painting" the sphere.





▷ Extract Object (from images): chromakey (blue screen)

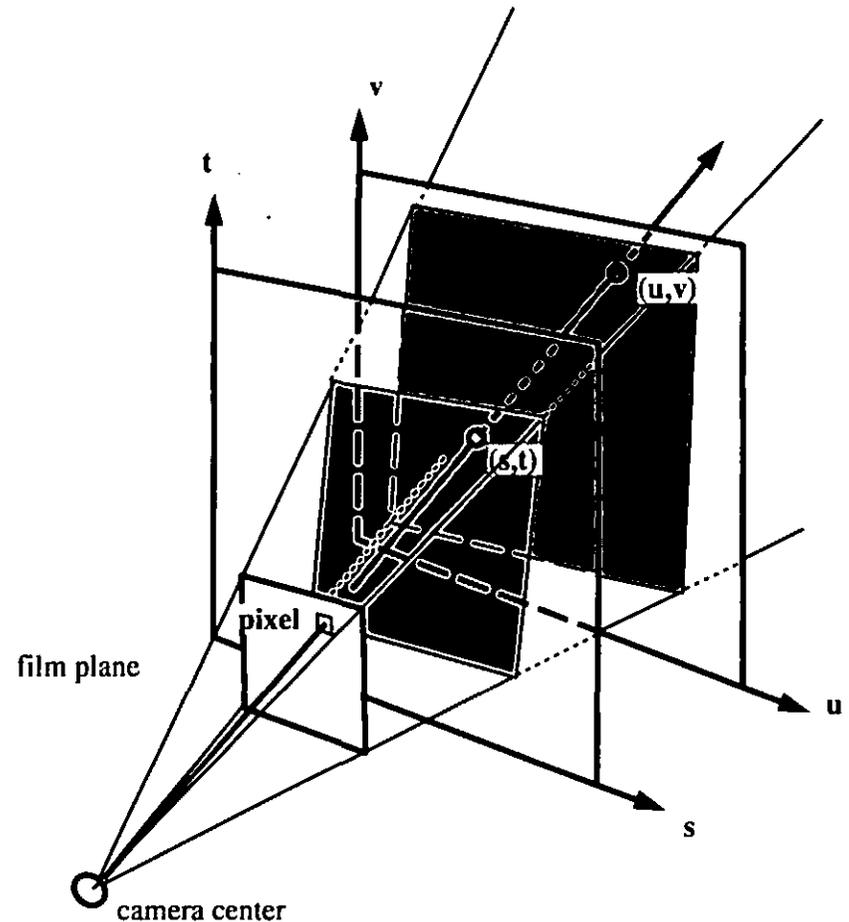
▷ Create Geometry:

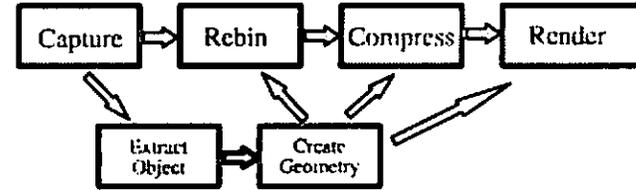
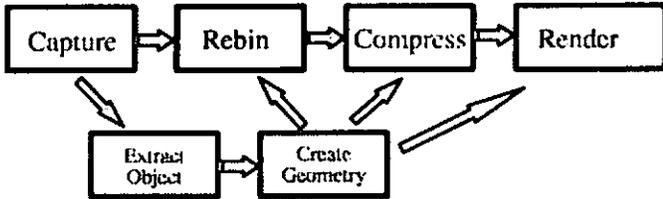
- use silhouettes to create "line hull" (Szeliski)
- compute optical flow, equivalent to depth given camera poses

▷ Compress:

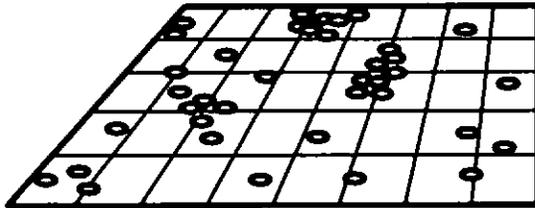
- use geometry and all other coherence available
- we hope for 200–1000 to 1 compression
- BIG TOPIC for research

▷ Each pixel in each image represents a "sample" of the 4D lumigraph.

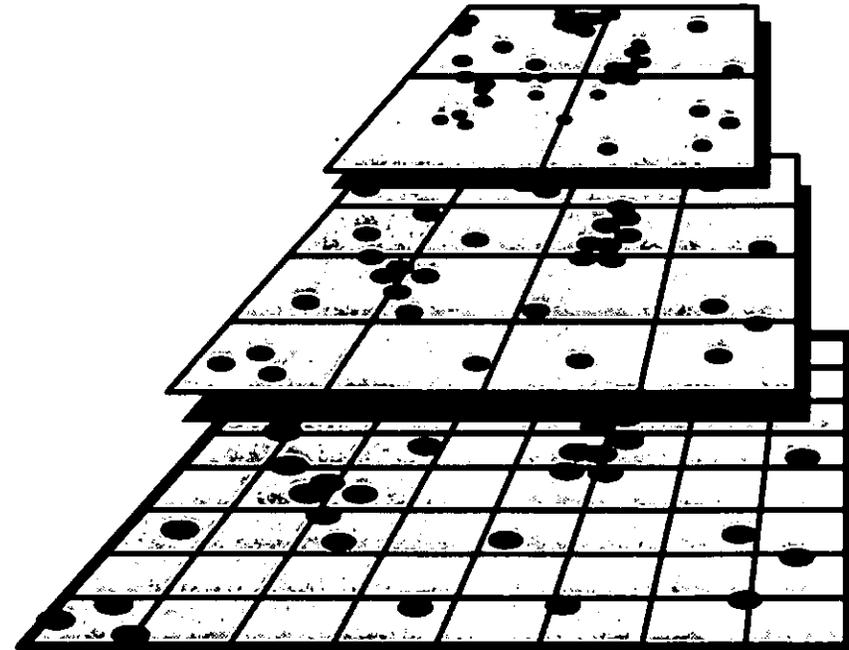




- ▶ Each pixel in each image represents a "sample" of the 4D lumigraph.

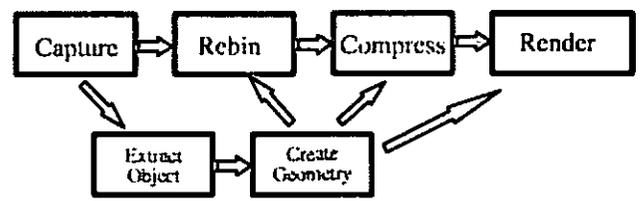
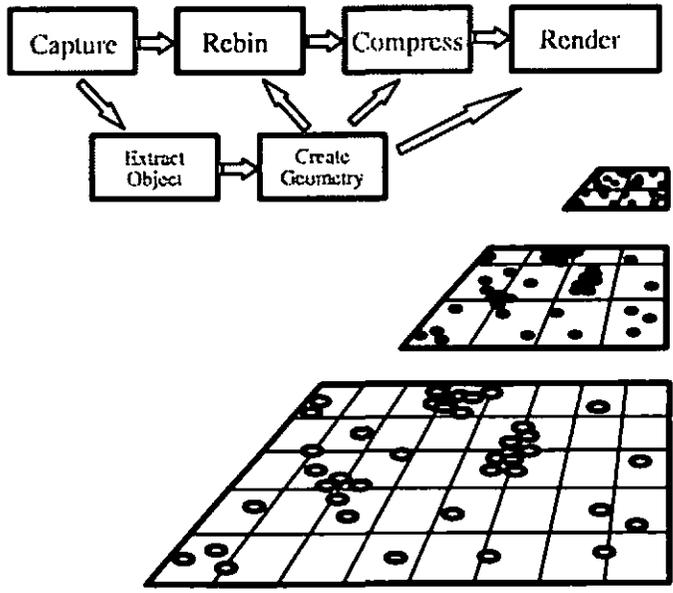


- ▶ Samples may be scattered in clumps at arbitrary locations.
- ▶ Want values at regular lattice points.

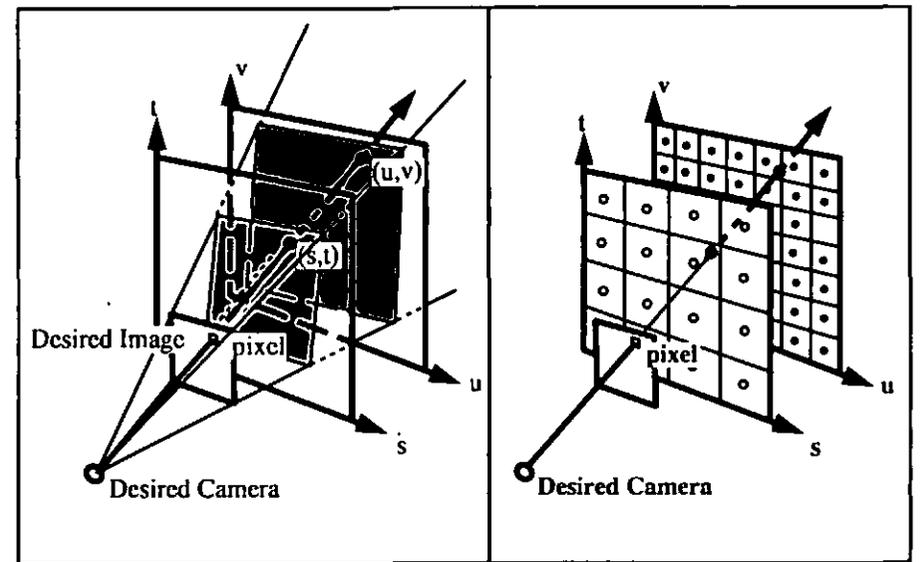


- ▶ Solution:
 - *splat* samples into lumigraph
 - *pull* to create a hierarchy of lower res Lumigraphs
 - *push* averages down to regions w/o sufficient samples

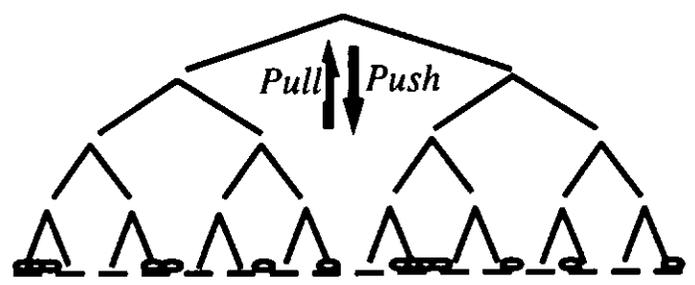
Related work:
 Burt '88
 Mitchell '87

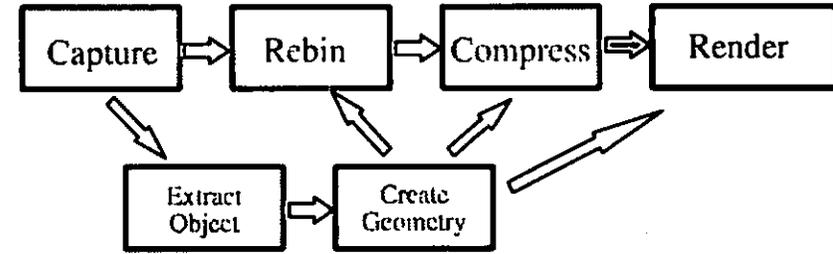
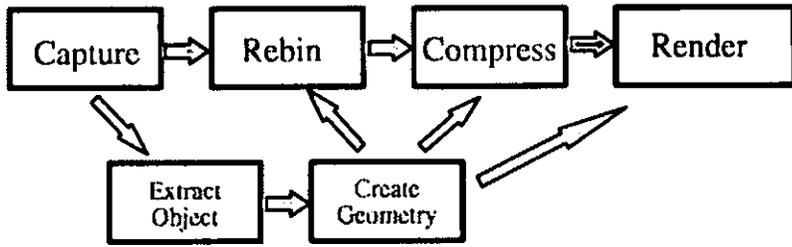


► Ray Tracing



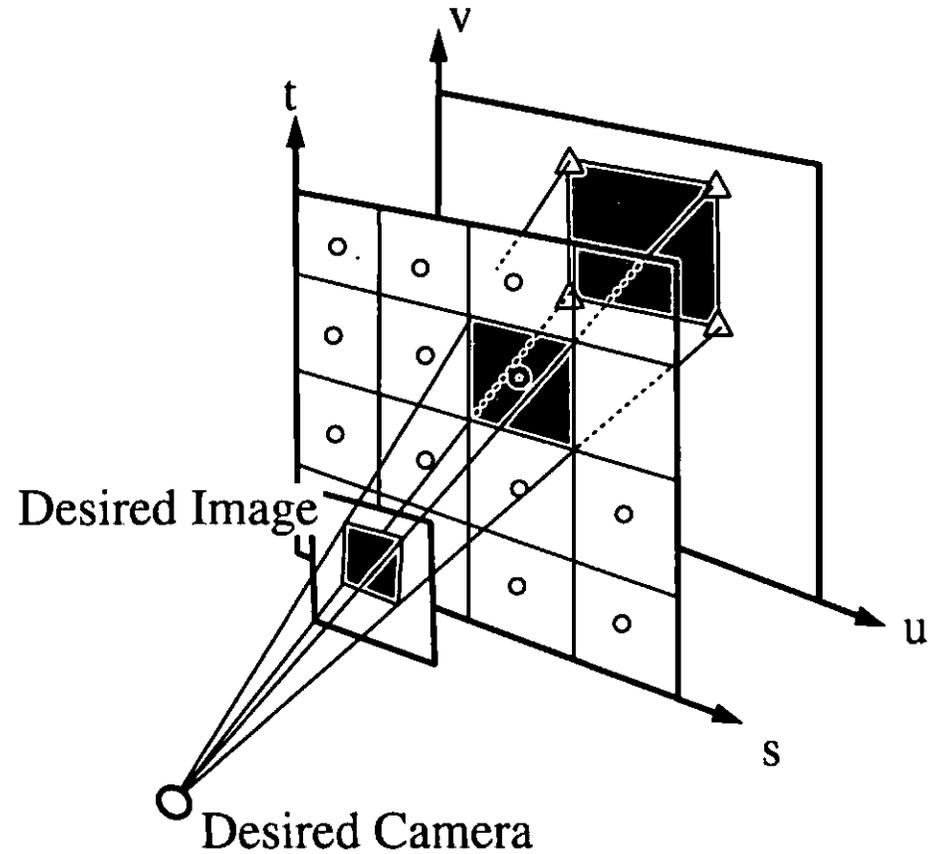
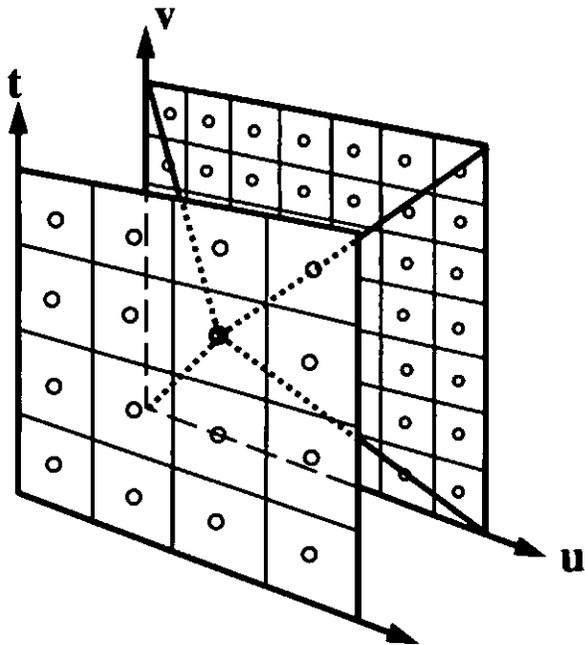
- Solution:
- *splat* samples into lumigraph
 - *pull* to create a hierarchy of lower res Lumigraphs
 - *push* averages down to regions w/o sufficient samples



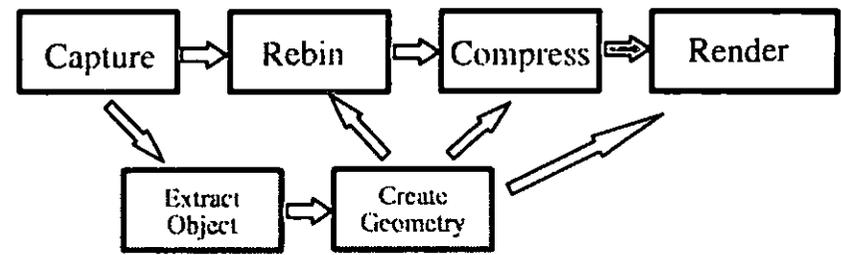
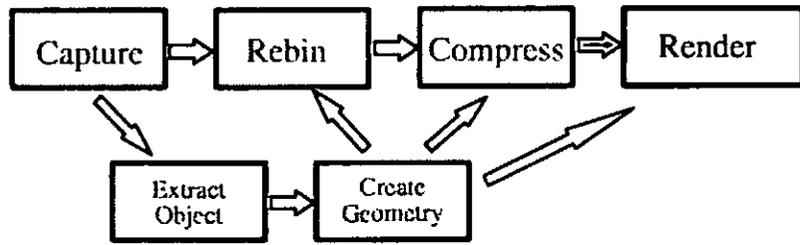


▷ UV image as texture

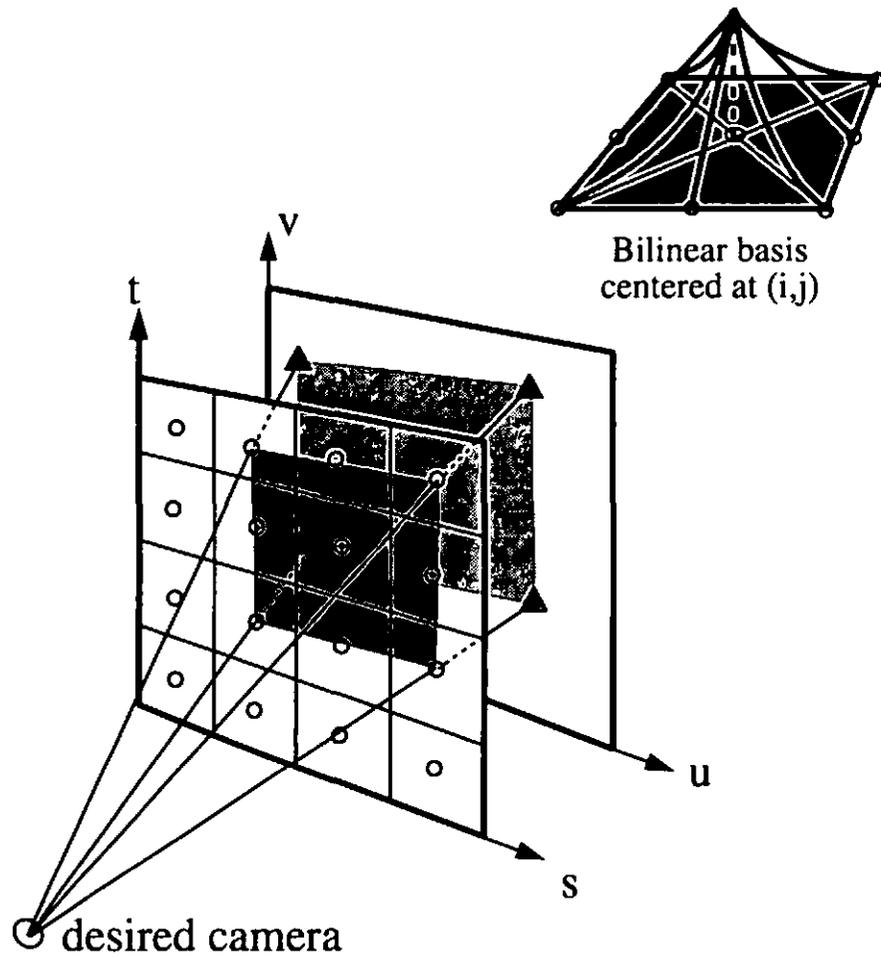
▷ Texture Mapping (Constant Basis)



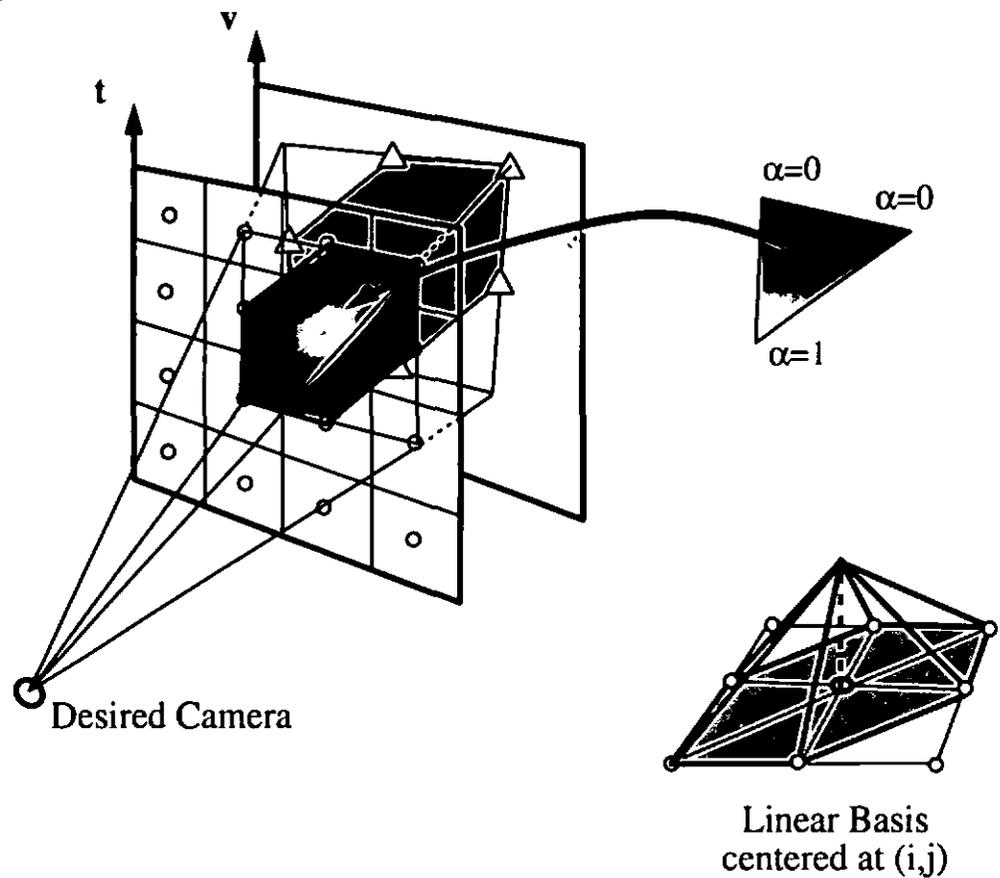
one (u,v) image for each (s,t) grid point



▷ Texture Mapping (Quadralinear Basis)



▷ Texture Mapping (linear Basis)



Timings

Capture	1 hour
Segment, Geometry, Rebin	12 hours
Render (512x512)	10–30 fps
behavior depends on texture memory swapping rendered on SGI Maximum Impact, 150 Mhz proc., 4 Mb texture memory	
Resolution	
16x16x256x256	–or– 32x32x256x256

Conclusion

Light based representation

Geometric correction

Hand held camera capture + Rebinning

Fast rendering using texture mapping
Independent of scene complexity

Polyhedral Geometry and the Two-Plane Parameterization

Xianfeng Gu Steven J. Gortler Michael F. Cohen
Harvard University Harvard University Microsoft Research

Abstract

Recently the light-field and lumigraph systems have been proposed as general methods of representing the visual information present in a scene. These methods represent this information as a 4D function of light over the domain of directed lines. These systems use the intersection points of the lines on two planes to parameterize the lines in space. This paper explores the structure of the two-plane parameterization in detail. In particular we analyze the association between the geometry of the scene and subsets of the 4D data. The answers to these questions are essential to understanding the relationship between a lumigraph, and the geometry that it attempts to represent. This knowledge is potentially important for a variety of applications such as extracting shape from lumigraph data, and lumigraph compression.

1 Introduction

Recently the light-field and lumigraph systems have been proposed as general methods of representing the visual information present in a scene [7, 9]. These methods represent this information as a 4D function of light over the domain of directed lines. This information can be captured using video cameras, without requiring the solution to a stereo vision problem. Given this representation one can generate novel views of the scene quickly by extracting the relevant data from the representation.

Generally, when representing the visual information about a scene, one needs to store the five dimensional radiance function (3 degrees of freedom for position and 2 for direction). Since in “free space”, radiance is constant along a line, the complete appearance of an object from outside its convex hull, or conversely, the appearance of scene viewed from within an empty convex region, is fully represented by the radiance along all directed lines intersection the convex region. The space of lines has only 4 degrees of freedom, which makes a lumigraph a 4D representation. There are many possible ways to parameterize lines in 3 space. The light-field/lumigraph systems use the lines’ intersection with two planes to parameterize the lines. The motivation for this choice is the simplicity of the representation, and the speed at which the data needed to generate a novel image can be extracted. In particular, the new image can be constructed using texture mapping operations [7].

Typically, the light leaving some geometric point is a smooth function over directions, e.g. constant in the case of a diffuse surface. Therefore the lumigraph function will be correspondingly smooth over some associated lumigraph domain region. Thus we may wish to understand the association between the geometry of the scene and subsets of the 4D data. This paper explores the structure of the two-plane parameterization in detail. This knowledge is both interesting theoretically and is potentially important for a variety of applications, for example, lumigraph compression, deriving geometry from lumigraphs, and creating more accurate renderings from lumigraphs. We do not address the specific issues surrounding each application here, but rather discuss the more basic theoretical issues.

There are many questions we seek to answer. The simplest such question is to determine what “the set of all lines that pass through some fixed point in space” corresponds to in a lumigraph. This set of lines has two continuous degrees of freedom, and so the associated lumigraph subset is a two dimensional subset. Moreover we will show that this 2D subset is an affine manifold; one could call this a 2D plane in a 4D space. We are also interested in “converse” questions; suppose we choose some arbitrary 2D affine subset of a lumigraph, does this correspond to the set of lines through some fixed point? The answer to this question is necessarily negative. There are only three degrees of freedom in choosing a point in space, while there are six degrees of freedom in choosing a 2D affine subset of a 4D space.¹ Therefore there must be 2D affine subsets that do not correspond to the set of lines through a fixed point. In this paper, we will explore this question fully, and characterize all 2D affine lumigraph subsets.

In this paper, we will also address questions such as: what is the algebraic structure of the subset of the lumigraph that corresponds to all lines that pass through some triangle in space? Because a triangle is bounded by line segments, it becomes relevant to ask: what is the subset corresponding to all lines through some line segment in space?

The knowledge gained by answering the above questions may prove useful for a variety of practical applications. One such application is extracting geometric information from real world lumigraph data. Bolles et al. use a 3D data structure comprised of images taken as a camera follows a linear path [3]. They then analyze the structure of the “epipolar plane image” (EPI) slices of the 3D data, in order to extract geometric information about the scene. In particular, a feature point in scene corresponds to a 1D affine subset of an EPI. Instead of solving the standard stereo correspondence problem, Bolles et al. simply search for lines in the EPI’s. This allows them to robustly deduce the depth of scene features. Unfortunately, in order to use a 3D data structure, the camera must follow a linear path; in practice this is a very restrictive assumption. On the other hand, as described in Gortler et al. [7], even when one allows the camera to move freely, one can create a 4D lumigraph data structure using a process called rebinning. Thus by looking for certain features in the lumigraph data, one may be able to robustly deduce geometric information about the scene, much like can be done from an EPI. For example, by identifying certain 2D affine subsets of lumigraph data one can extract the 3D scene locations of the associated points in space. Pursuing such a program requires us to clearly understand the the relationship between scene features and lumigraph subsets.

Another important application is lumigraph compression since these data sets are

¹ One can specify an affine 2D frame with 3 points in 4D. This is described by $3 \cdot 4 = 12$ numbers. Since we are only interested in the affine subset, and not the particular frame, we are free to apply any affine transform and still obtain the same subset. There are 6 degrees of freedom in a 2D affine transform, which we have overspecified. Thus, the subset is described by $12 - 6 = 6$ degrees of freedom. More generally, in an n dimensional linear space, there are $nk + n - k^2 - k$ degrees of freedom in choosing a k dimensional affine subset [11].

large, but highly redundant. Levoy and Hanrahan discuss a low dimensional vector quantization method for compressing the 4D datasets that arise from a lumigraph setting, achieving 120:1 lossy compression rates [9]. Their method uses no knowledge about the structure of the data. It seems plausible that much higher compression ratios may be achieved if structural knowledge is taken into account. By knowing what to expect in a lumigraph, we hope to gain insight as to how to most efficiently represent them.

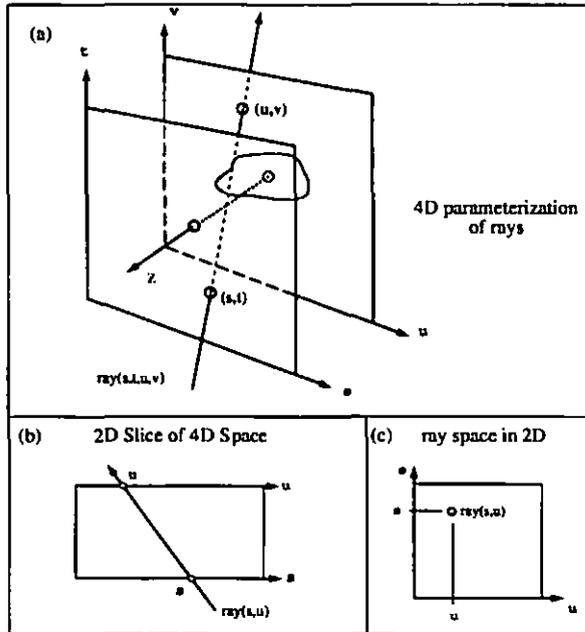


Fig. 1. 2D and 4D line parameterization

In this paper, we will use two parallel planes to parameterize lines in 3 space, and will refer to this parameterization as 2PP. Without loss of generality, the “front plane” will lie at $z = 0$ with points of the form $(s, t, 0)$. The “back plane” is $z = -1$, with points of the form $(u, v, -1)$. A (non-parallel) line will intersect both planes exactly once, and is identified with the 4D parameters (s, t, u, v) . Words such as point, line, segment, and plane will describe sets in 3D geometric space. Words such as 1D, 2D and 3D affine subsets will be used to refer to “lines”, “planes” and “hyperplanes” in the 2PP parameterization.

2 Flatland

In order to build up some intuition for the problem, we will start by reviewing a lumigraph in flatland. In this case all geometry resides in the (x, z) plane. A line is parameterized by where it crosses two canonical parallel lines. Thus a line in (x, z) space is parameterized as (s, u) in the lumigraph (see Figure 1). Because lines and points are

projective duals in 2 space, flatland is easy to understand. This kind of analysis is found in [3].

2.1 Lines Through a Point

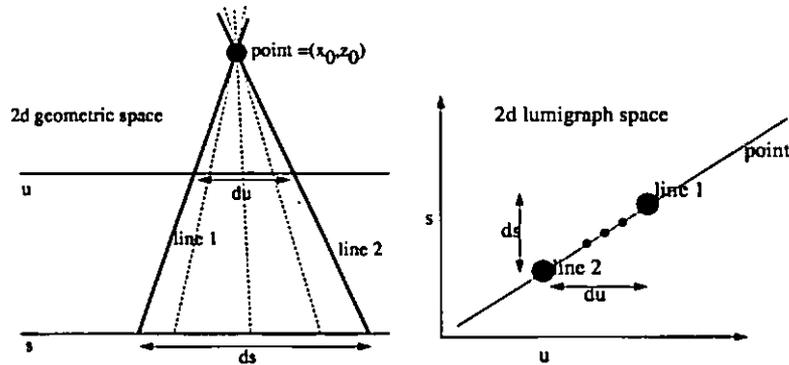


Fig. 2. Lines through a point are associated with a line

We first discuss the set of all lines that pass through some point (x_0, z_0) . This pencil corresponds to a 1D subset of the (s, u) domain. Using a simple similar triangles argument, it is clear that as one moves a constant distance ds , in order for the line to still pass through (x_0, z_0) , one must move some distance du which is linear in ds (see Figure 2). Thus, this set of lines is a one dimensional affine subset of the 2D (s, u) domain. For example, if the scene geometry consists of a single light emitting point, the 2D lumigraph function will only be non zero over the support of a single 1D affine subset.

The same analysis is true for all lines that pass through a single camera's pin-hole. When synthesizing an image with a pin hole camera, one measures the radiance along all lines that pass through the pin-hole. Thus, the lumigraph data needed to synthesize an image lies along a 1D affine subset in (s, u) space. The 1D subset is entirely determined by the (x, z) position of the camera pin hole. The orientation, and intrinsic parameters of the camera have no effect the choice of the subset. They just determine the projective mapping between the data on that 1D subset, and the camera's "film line".

The converse statement is equally as simple. Any affine 1D subset of the (s, u) domain corresponds to the set of lines that pass through a single point (x_0, z_0) in space.

2.2 Lines Through a Segment

In a flatland environment, one may approximate all of the visible "surfaces" using line segments. Thus it is important to consider the set of all lines that pass through some line segment in the (x, z) domain. Let us identify the line l_0 on which this segment lies; this line has some particular parameter value (s_0, u_0) . Let us specify points on the segment with the single parameter λ ; each λ fixes some point (x_0, z_0) on l_0 . All lines through that point are associated with a single 1D affine subset in (s, u) . This 1D subset must include (s_0, u_0) , because the line l_0 passes through all points on the segment. As

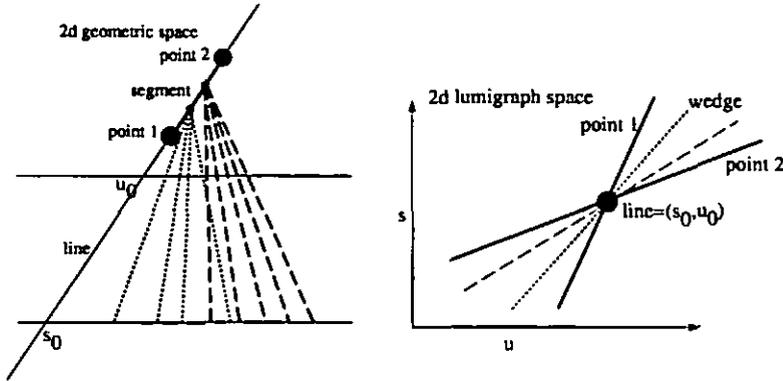


Fig. 3. Lines through a segment are associated with a wedge

a result, all lines passing through a segment, correspond in the lumigraph to a pencil of 1D subsets through (s_0, u_0) . The entire pencil spans all of the 2D (s, u) space. When approximating geometry, we are considering a segment with a boundary. The corresponding (s, u) subset consists of a "wedge" of the complete 360 degree pencil through (s_0, u_0) (see Figure 3).

3 3D Geometry

In 3 dimensions, lines and points are not projective duals, and the association becomes more complicated. It is our goal to understand this case as well as the flatland case.

3.1 Lines Through a Point

The most basic question to ask is what 2PP subset corresponds to all lines through a point (p_x, p_y, p_z) in space? Points along the line passing through some point $(s, t, 0)$ and the point (p_x, p_y, p_z) can be parameterized by w and expressed as:

$$\begin{bmatrix} s \\ t \\ 0 \end{bmatrix} + w \begin{bmatrix} p_x - s \\ p_y - t \\ p_z \end{bmatrix}$$

This line crosses the $z = -1$ plane when $w = -1/p_z$ at the point

$$\begin{bmatrix} s + s/p_z - p_x/p_z \\ t + t/p_z - p_y/p_z \\ -1 \end{bmatrix}$$

making the 2PP parameterization for this line to be

$$(s, t, (1 + 1/p_z)s - p_x/p_z, (1 + 1/p_z)t - p_y/p_z)$$

Thus, the set of lines passing through the point describes a 2 dimensional affine subset of the 2PP. If one parameterizes the 2 dimensional subset with parameters (a, b) , then the corresponding (s, t, u, v) value is

$$subset(a, b) = (a, b, (1 + 1/p_z)a - p_x/p_z, (1 + 1/p_z)b - p_y/p_z) \quad (1)$$

3.2 Lines Through a Segment

In general it is common to approximate the 3D geometry of a surface using a set of triangles. A triangle is defined by 3 segments. Thus our next question will be what is the 2PP subset corresponding to a segment in space. Clearly, we have added a new degree of freedom and the corresponding subset is a three dimensional subset.

Call the line of the segment l_0 , it is defined by a point (p_x, p_y, p_z) and a direction (d_x, d_y, d_z) . l_0 corresponds to some (s_0, t_0, u_0, v_0) of the 2PP domain. Define one degree of freedom λ to move along the line l_0 which specifies some point (x_0, y_0, z_0) . The lines through each point (x_0, y_0, z_0) on l_0 make up one 2D affine 2PP subset. Also note that the set of lines that passes through (x_0, y_0, z_0) includes l_0 itself. Thus, the 2D affine subset contains (s_0, t_0, u_0, v_0) . In this sense the three dimensional subset corresponding to all lines through l_0 is a "pencil" of 2D affine subsets that all share the single point (s_0, t_0, u_0, v_0) . If we consider a bounded line segment, then the associated subset is a wedge of the "pencil".

The geometry of this subset can be determined by fixing s , t , and λ . Points along the resulting line is are written as

$$\begin{bmatrix} s \\ t \\ 0 \end{bmatrix} + w \begin{bmatrix} p_x + \lambda d_x - s \\ p_y + \lambda d_y - t \\ p_z + \lambda d_z \end{bmatrix}$$

The line crosses the $z = -1$ plane when $w = -1/(p_z + \lambda d_z)$ at the point

$$\begin{bmatrix} s + s/(p_z + \lambda d_z) - p_x/(p_z + \lambda d_z) - \lambda d_x/(p_z + \lambda d_z) \\ t + t/(p_z + \lambda d_z) - p_y/(p_z + \lambda d_z) - \lambda d_y/(p_z + \lambda d_z) \\ -1 \end{bmatrix}$$

Thus the corresponding 3D subset can be parameterized and defined as

$$\text{subset}(a, b, \lambda) = (a, b, a + a/(p_z + \lambda d_z) - p_x/(p_z + \lambda d_z) - \lambda d_x/(p_z + \lambda d_z), (2) \\ b + b/(p_z + \lambda d_z) - p_y/(p_z + \lambda d_z) - \lambda d_y/(p_z + \lambda d_z))$$

If the line is not parallel to the $z = 0$ plane, we can specify l_0 with the direction $(d_x, d_y, 1)$ and the point along the line that crosses the $z = 0$ plane, $(p_x, p_y, 0)$. Thus the coordinates (s_0, t_0, u_0, v_0) of l_0 are $(p_x, p_y, p_x - d_x, p_y - d_y)$. Defining $\lambda = 1 + 1/\lambda$ equation 2 becomes

$$\text{subset}(a, b, \lambda) = (a, b, \lambda(a - s_0) + u_0, \lambda(b - t_0) + v_0)$$

This surface is bilinear in the variables (a, b, λ) . Because the λ parameter is not directly measurable from lumigraph data, it is best to eliminate this hidden parameter. When we do so, the subset is expressed as the solution to the following implicit bilinear equation

$$0 = (s - s_0)(v - v_0) - (t - t_0)(u - u_0) \quad (3)$$

Another natural representation is to eliminate λ and express the (s, t, u, v) subset parametrically as

$$\text{subset}(a, b, c) = (a, b, c, \frac{(b - t_0)(c - u_0) + v_0}{a - s_0})$$

If we hold s, t constant, then v is linear in u . This is simply the image of the line as seen from a camera at s, t . If we hold s, u constant, then v is linear in t . This is exactly what is observed in a single EPI slice. See figure 4 for such a lumigraph decomposition of the fruitbowl lumigraph used in [7]. If we hold t, u constant, then v is linear rational in s . See figure 5 for such a lumigraph decomposition.

3.3 Lines Through a Triangle

The set of all lines through a triangle is a 4D manifold of the 2PP domain with a boundary. The boundary of this manifold is defined by the 3D subsets associated with the set of all lines that pass through the three line segments defining the triangle. Thus the subset of lines passing through a single triangle can be fully described by the equations above.

4 Occlusion

In an actual scene, a triangle may be obscured by other geometry. As such, not all of the light rays emanating from some triangle are represented in the lumigraph of the scene. If a triangle is partially obscured by another triangle, then only the rays that emanate from the first triangle and do not pass through the blocking triangle will be represented in the lumigraph. In this case, the set of lines seen from the first triangle will make up a more complicated 2PP subset. The three dimensional geometry of these "critical" sets of lines is well understood [6]. The relationship between critical manifolds of various dimensions is also well understood [4]. We wish to understand the algebraic nature of these critical sets when expressed in the 2PP parameterization.

4.1 Apparent vertices: Lines Through 2 Lines

When a triangle is obscured, then the observed set of rays can terminate at apparent vertices. These are created when edges from two polygons appear to cross from the receiver's point of view [1]. The set of rays that pass through apparent vertices can be expressed as "the set of lines that pass through 2 lines". Equation 3 gives us the implicit equation for all lines that pass through one line. Thus the sets of lines passing through two lines is the solution to the two equations

$$\begin{aligned}0 &= (s - s_0)(v - v_0) - (t - t_0)(u - u_0) \\0 &= (s - s_1)(v - v_1) - (t - t_1)(u - u_1)\end{aligned}$$

By subtracting the first equation from the second, we can rewrite the equations as

$$\begin{aligned}0 &= (s - s_0)(v - v_0) - (t - t_0)(u - u_0) \\0 &= (s_1 - s_0)v + (v_1 - v_0)s + (s_0v_0 - s_1v_1) - (t_1 - t_0)u + (u_1 - u_0)t + (u_0t_0 - u_1t_1)\end{aligned}$$

This is the intersection of the solution set of a quadratic equation, and a 3D affine subset. The result is a 2D quadric subset of the 2PP domain.

Thus in a partially obscured triangle, the corresponding 2PP subset is a 4D manifold bounded by a 3D piecewise quadric manifold (corresponding to the apparent edges). The 3D quadratic pieces meet at 2D affine manifolds, (corresponding to actual vertices), and at 2D quadric manifolds (corresponding to apparent vertices).

4.2 Lines Through Three Lines

In a scene with multiple triangles, there exist so-called "critical" locations; these are places where the number of apparent vertices changes. These locations make up the contours of the aspect graph of the scene [6, 13, 8]. Such critical locations occur at "eee" events, when three edges of the scene appear to intersect from the point of view of the receiver. A degenerate "eee" case occurs when one polygon edge and one polygon

vertex appear from the point of view of the receiver to coincide. Such cases are called “vc” events. At these “ccc” and “ev” locations, the irradiance undergoes a discontinuity in its second derivatives [1, 8].

We would like to understand the structure in the 2PP of sets of rays on “ccc” events. This problem can be expressed as “what are the set of lines that pass through 3 lines”. This can be expressed as the solution to

$$\begin{aligned} 0 &= (s - s_0)(v - v_0) - (t - t_0)(u - u_0) \\ 0 &= (s - s_1)(v - v_1) - (t - t_1)(u - u_1) \\ 0 &= (s - s_2)(v - v_2) - (t - t_2)(u - u_2) \end{aligned}$$

This set is reducible to the following three equations

$$\begin{aligned} 0 &= (s - s_0)(v - v_0) - (t - t_0)(u - u_0) \\ 0 &= (s_1 - s_0)v + (v_1 - v_0)s + (s_0v_0 - s_1v_1) - (t_1 - t_0)u + (u_1 - u_0)t + (u_0t_0 - u_1t_1) \\ 0 &= (s_2 - s_0)v + (v_2 - v_0)s + (s_0v_0 - s_2v_2) - (t_2 - t_0)u + (u_2 - u_0)t + (u_0t_0 - u_2t_2) \end{aligned}$$

This is the intersection of a quadratic equation, and a 2D affine subset. The result is a 1D conic subset of the 2PP domain.

4.3 Lines Through One Point and One Line

A “ve” event is a special case of an “ccc” event. In this case, it can be shown that the quadratic elements of the equations can be eliminated leaving three linear constraints. Thus the corresponding subset is a 1D affine subset.

Thus for a partially obscured triangle, three of the 3D quadratic manifolds comprising its boundary can meet either in a 1D conic section, or degenerately in a 1D affine subset.

5 Converse Questions

In order to better understand the lumigraph structure, it is useful to ask converse questions such as: what is the set of lines corresponding to a 2D affine 2PP subset. This will help us geometrically interpret observed lumigraph structure.

We said above that the subset associated with all lines that pass through a point is a 2D affine subset. As mentioned in the introduction, a simple counting argument shows that there must be 2D affine subsets that do not correspond to lines through a single point. In particular it takes only 3 numbers to specify a point in space, whereas there are 6 degrees of freedom in choosing a 2D affine subset of 4D space [11]. What are the other 2D affine subsets? In order to understand these questions we first must introduce the concept of a shallow segment.

5.1 Lines Through a Shallow Segment

We define a shallow line to be a line that lies in a plane of constant z . A shallow segment is a segment of a shallow line. As we will see, shallow lines are important in characterizing affine 2PP subsets.

What 2PP subset corresponds to all lines through a shallow line l defined by a point $(0, p_y, p_z)$ and a direction $(1, d_y, 0)$ in space? ² This subset can be computed starting from equation 2, and setting d_z to zero.

$$\text{subset}(a, b, \lambda) = (a, b, a + a/p_z - \lambda/p_z, b + b/p_z - p_y/p_z - \lambda d_y/p_z)$$

Once again, we eliminate λ resulting in

$$\text{subset}(a, b, c) = (a, b, c, (1 + 1/p_z)b - d_y(1 + 1/p_z)a + d_y c - p_y/p_z)$$

Thus we conclude that for a shallow line, the corresponding 3D lumigraph subset is affine.

5.2 2D Affine Lumigraph Subsets

In general a 2D affine subset of a 4D space results from the intersection of two 3D affine subsets. As a result, the set of lines that pass through two shallow lines l_1, l_2 , must be a 2D affine subset. There are 3 degrees of freedom in specifying each shallow line, giving us a total of 6 degrees of freedom. It can be shown that these 6 dofs are independent and can thus specify any 2D affine lumigraph subset.

Thus we conclude that any 2D affine subset corresponds to the set of lines through two shallow lines. The set of lines passing through a single point, is simply a degenerate case where two shallow lines intersect in a point.

6 Relationship to Plücker Coordinates

One classic way to represent lines in 3D space is using 6 coordinates called Plücker coordinates. Plücker coordinates express elements of the space $P(G_2(R^4))$ from the Grassman-Cayley algebra [10, 5, 2, 13, 11, 12]. This 5D projective space consists of lines in 3D and “linear combinations of lines in 3D” up to an arbitrary scale. Given the 6 Plücker coordinates of an element of this space, one can test if it represents a single line (and not a non-decomposable linear combination of multiple lines) by seeing if the coordinates satisfy a certain quadratic constraint. This leaves 5 degrees of freedom. Because the scale is arbitrary, we are left with 4 independent degrees of freedom to describe a line.

For Lumigraph purposes, the 2PP parameterization has certain advantages over Plücker coordinates. The 2PP domain is a four dimensional linear space, while lines in Plücker coordinates live on a quadric four dimensional manifold in a five dimensional projective space. This can have a dramatic impact on the space complexity of the representation as well as the time required to compute the coordinates of the lines needed to extract an image. But because the algebra and geometry of Plücker coordinates is well understood, we wish to understand the relationship between 2PP coordinates and Plücker coordinates.

Given a line (s_0, t_0, u_0, v_0) in the two plane parameterization, one identifies two points on the line, $(s_0, t_0, 0)$ and $(u_0, v_0, -1)$ and computes the normalized Plücker coordinates as

$$(s_0 v_0 - t_0 u_0, -s_0, s_0 - u_0, -t_0, 1, v_0 - t_0)$$

² If the line is “vertical”, then we can define it by a point $(p_x, 0, p_z)$ and a direction $(d_x, 1, 0)$.

The formula for the first Plücker coordinate is non-linear and so in general this mapping is non-linear. Interestingly, the Plücker subsets corresponding to many of the geometric features discussed in this paper still have the same algebraic complexity as they do in the two plane parameterization. For example, to compute the Plücker coordinates of the lines that pass through a single point (p_x, p_y, p_z) we start with equation 1 and map them to the Plücker coordinates:

$$\text{subset}(a, b) = (ap_y/p_z - bp_x/p_z, -a, -a/p_z + p_x/p_z, -b, 1, b/p_z - p_y/p_z)$$

The non-linear parts of the mapping cancel out. As a result we see that the set of lines passing through a single geometric point is a 2D affine subset of normalized Plücker coordinates. The same result is obtained in a coordinate free setting in [5, Prop 2.4].

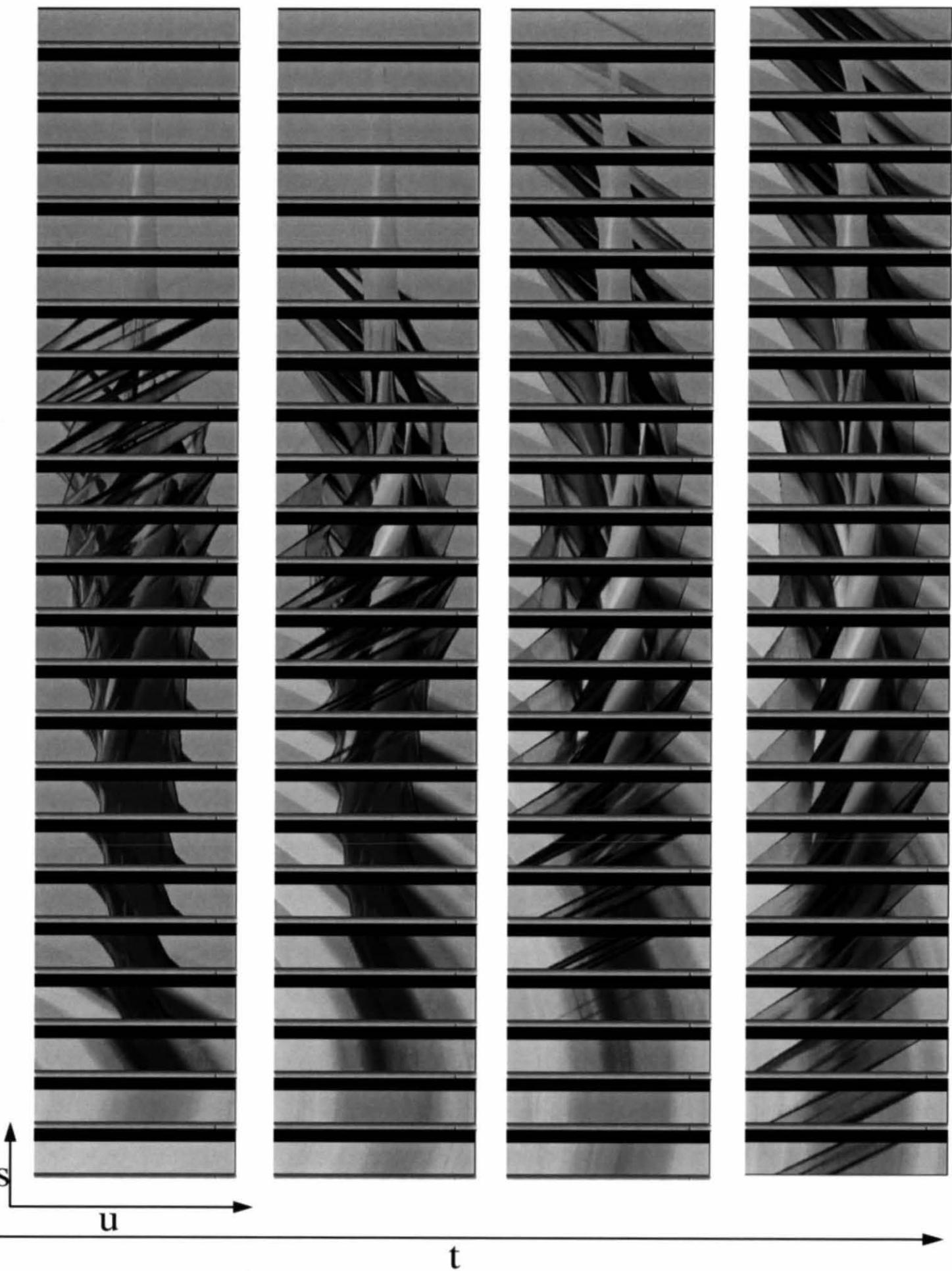
The set of lines that pass through two lines is associated with a quadric in (s, t, u, v) , and the set of lines that pass through three lines is associated with a conic in (s, t, u, v) . Once again, the same is true of the associated subsets of Plücker coordinates [13].

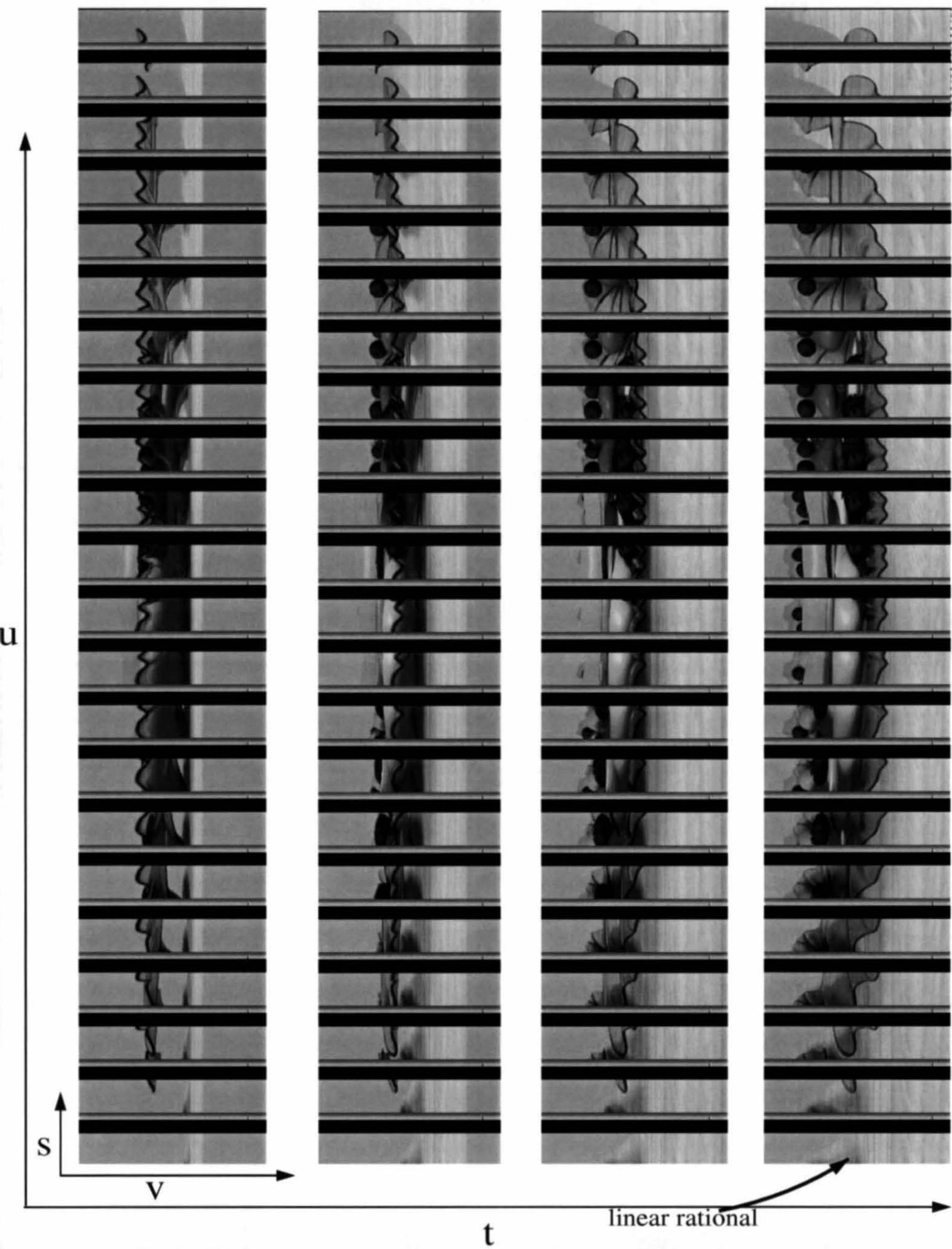
7 Conclusion

In this paper we have investigated the association between sets of lines through certain features in 3D geometric space, and subsets of the two plane parametric domain. This gives us insight into the structure of lumigraph functions, and may prove very useful in extracting geometric information from lumigraphs as well as lumigraph compression. We plan to pursue these avenues as future work.

References

1. Jim Arvo. The irradiance jacobian for partially occluded polyhedral sources. In *Computer Graphics, Annual Conference Series, 1994*, pages 343–350.
2. Marilena Barnabei, Andrea Brini, and Gian-Carlo Rota. On the exterior calculus of invariant theory. *Journal of Algebra*, 96:120–160, 1985.
3. R. C. Bolles, H. H. Baker, and D. H. Marimont. Epipolar-plane image analysis: An approach to determining structure from motion. *IJCV*, 1:7–55, 1987.
4. Fredo Durand, George Drettakis, and Claude Puech. The 3d visibility complex, a new approach to the problems of accurate visibility. In *7th Eurographics Workshop on Rendering*, 1996.
5. O. Faugeras and Q. T. Luong. A projective geometric approach to multiple image analysis. *unpublished book in progress*.
6. Ziv Gigus and Jitendra Malik. Computing the aspect graph for line drawings of polyhedral objects. *IEEE Trans. PAMI*, 12(2):113–122, 1990.
7. Steven Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The lumigraph. In *Computer Graphics, Annual Conference Series, 1996*, pages 43–54.
8. P. Heckbert. *Simulating global illumination using adaptive meshing*. PhD thesis, The University of California at Berkeley, June 1991.
9. Mark Levoy and Pat Hanrahan. Light-field rendering. In *Computer Graphics, Annual Conference Series, 1996*, pages 31–42.
10. D. Sommerville. *Analytical Geometry of Three Dimensions*. Cambridge University Press, 1959.
11. Jorge Stolfi. *Oriented Projective Geometry - A Framework for Geometric Computations*. Academic Press, 1991.
12. Abraham Dean Stone. personal communication, ‘the graßman bunch’. 1997.
13. Seth J. Teller. Computing the antipenumbra of an area light source. *Computer Graphics*, 26(2):139–148, 1992.





Light Field Rendering

Marc Levoy and Pat Hanrahan
Computer Science Department
Stanford University

Abstract

A number of techniques have been proposed for flying through scenes by redisplaying previously rendered or digitized views. Techniques have also been proposed for interpolating between views by warping input images, using depth information or correspondences between multiple images. In this paper, we describe a simple and robust method for generating new views from arbitrary camera positions without depth information or feature matching, simply by combining and resampling the available images. The key to this technique lies in interpreting the input images as 2D slices of a 4D function - the light field. This function completely characterizes the flow of light through unobstructed space in a static scene with fixed illumination.

We describe a sampled representation for light fields that allows for both efficient creation and display of inward and outward looking views. We have created light fields from large arrays of both rendered and digitized images. The latter are acquired using a video camera mounted on a computer-controlled gantry. Once a light field has been created, new views may be constructed in real time by extracting slices in appropriate directions. Since the success of the method depends on having a high sample rate, we describe a compression system that is able to compress the light fields we have generated by more than a factor of 100:1 with very little loss of fidelity. We also address the issues of antialiasing during creation, and resampling during slice extraction.

CR Categories: 1.3.2 [Computer Graphics]: Picture/Image Generation — *Digitizing and scanning, Viewing algorithms*; 1.4.2 [Computer Graphics]: Compression — *Approximate methods*

Additional keywords: image-based rendering, light field, holographic stereogram, vector quantization, epipolar analysis

1. Introduction

Traditionally the input to a 3D graphics system is a scene consisting of geometric primitives composed of different materials and a set of lights. Based on this input specification, the rendering system computes and outputs an image. Recently a new approach to rendering has emerged: *image-based rendering*. Image-based rendering systems generate different views of an environment from a set of pre-acquired imagery. There are several advantages to this approach:

- The display algorithms for image-based rendering require modest computational resources and are thus suitable for real-time implementation on workstations and personal computers.
- The cost of interactively viewing the scene is independent of scene complexity.
- The source of the pre-acquired images can be from a real or virtual environment, i.e. from digitized photographs or from rendered models. In fact, the two can be mixed together.

The forerunner to these techniques is the use of environment maps to capture the incoming light in a texture map [Blinn76, Greene86]. An environment map records the incident light arriving from all directions at a point. The original use of environment maps was to efficiently approximate reflections of the environment on a surface. However, environment maps also may be used to quickly display any outward looking view of the environment from a fixed location but at a variable orientation. This is the basis of the Apple QuickTimeVR system [Chen95]. In this system environment maps are created at key locations in the scene. The user is able to navigate discretely from location to location, and while at each location continuously change the viewing direction.

The major limitation of rendering systems based on environment maps is that the viewpoint is fixed. One way to relax this fixed position constraint is to use view interpolation [Chen93, Greene94, Fuchs94, McMillan95a, McMillan95b, Narayanan95]. Most of these methods require a depth value for each pixel in the environment map, which is easily provided if the environment maps are synthetic images. Given the depth value it is possible to reproject points in the environment map from different vantage points to warp between multiple images. The key challenge in this warping approach is to "fill in the gaps" when previously occluded areas become visible.

Another approach to interpolating between acquired images is to find corresponding points in the two [Laveau94, McMillan95b, Seitz95]. If the positions of the cameras are known, this is equivalent to finding the depth values of the corresponding points. Automatically finding correspondences between pairs of images is the classic problem of stereo vision, and unfortunately although many algorithms exist, these algorithms are fairly fragile and may not always find the correct correspondences.

In this paper we propose a new technique that is robust and allows much more freedom in the range of possible views. The major idea behind the technique is a representation of the *light field*, the radiance as a function of position and direction, in regions of space free of occluders (free space). In free space, the light field is a 4D, not a 5D function. An image is a two dimensional slice of the 4D light field. Creating a light field from a set of images corresponds to inserting each 2D slice into the 4D light field representation. Similarly, generating new views corresponds to extracting and resampling a slice.

Address: Gates Computer Science Building 3B
Stanford University
Stanford, CA 94305

levoy@cs.stanford.edu
hanrahan@cs.stanford.edu
<http://www-graphics.stanford.edu>

Generating a new image from a light field is quite different than previous view interpolation approaches. First, the new image is generally formed from many different pieces of the original input images, and need not look like any of them. Second, no model information, such as depth values or image correspondences, is needed to extract the image values. Third, image generation involves only resampling, a simple linear process.

This representation of the light field is similar to the epipolar volumes used in computer vision [Bolles87] and to horizontal-parallax-only holographic stereograms [Benton83]. An epipolar volume is formed from an array of images created by translating a camera in equal increments in a single direction. Such a representation has recently been used to perform view interpolation [Katayama95]. A holographic stereogram is formed by exposing a piece of film to an array of images captured by a camera moving sideways. Halle has discussed how to set the camera aperture to properly acquire images for holographic stereograms [Halle94], and that theory is applicable to this work. Gavin Miller has also recognized the potential synergy between true 3D display technologies and computer graphics algorithms [Miller95].

There are several major challenges to using the light field approach to view 3D scenes on a graphics workstation. First, there is the choice of parameterization and representation of the light field. Related to this is the choice of sampling pattern for the field. Second, there is the issue of how to generate or acquire the light field. Third, there is the problem of fast generation of different views. This requires that the slice representing rays through a point be easily extracted, and that the slice be properly resampled to avoid artifacts in the final image. Fourth, the obvious disadvantage of this approach is the large amount of data that may be required. Intuitively one suspects that the light field is coherent and that it may be compressed greatly. In the remaining sections we discuss these issues and our proposed solutions.

2. Representation

We define the light field as the radiance at a point in a given direction. Note that our definition is equivalent to the *plenoptic function* introduced by Adelson and Bergen [Adelson91]. The phrase light field was coined by A. Gershun in his classic paper describing the radiometric properties of light in a space [Gershun36].¹ McMillan and Bishop [McMillan95b] discuss the representation of 5D light fields as a set of panoramic images at different 3D locations.

However, the 5D representation may be reduced to 4D in free space (regions free of occluders). This is a consequence of the fact that the radiance does not change along a line unless blocked. 4D light fields may be interpreted as functions on the space of oriented lines. The redundancy of the 5D representation is undesirable for two reasons: first, redundancy increases the size of the total dataset, and second, redundancy complicates the reconstruction of the radiance function from its samples. This reduction in dimension has been used to simplify the representation of radiance emitted by luminaires [Levin71, Ashdown93]. For the remainder of this paper we will be only concerned with 4D light fields.

¹ For those familiar with Gershun's paper, he actually uses the term light field to mean the irradiance vector as a function of position. For this reason P. Moon in a later book [Moon81] uses the term photic field to denote what we call the light field.

Although restricting the validity of the representation to free space may seem like a limitation, there are two common situations where this assumption is useful. First, most geometric models are bounded. In this case free space is the region outside the convex hull of the object, and hence all views of an object from outside its convex hull may be generated from a 4D light field. Second, if we are moving through an architectural model or an outdoor scene we are usually moving through a region of free space; therefore, any view from inside this region, of objects outside the region, may be generated.

The major issue in choosing a representation of the 4D light field is how to parameterize the space of oriented lines. There are several issues in choosing the parameterization:

Efficient calculation. The computation of the position of a line from its parameters should be fast. More importantly, for the purposes of calculating new views, it should be easy to compute the line parameters given the viewing transformation and a pixel location.

Control over the set of lines. The space of all lines is infinite, but only a finite subset of line space is ever needed. For example, in the case of viewing an object we need only lines intersecting the convex hull of the object. Thus, there should be an intuitive connection between the actual lines in 3-space and line parameters.

Uniform sampling. Given equally spaced samples in line parameter space, the pattern of lines in 3-space should also be uniform. In this sense, a uniform sampling pattern is one where the *number of lines* in intervals between samples is constant everywhere. Note that the correct measure for number of lines is related to the form factor kernel [Sbert93].

The solution we propose is to parameterize lines by their intersections with two planes in arbitrary position (see figure 1). By convention, the coordinate system on the first plane is (u, v) and on the second plane is (s, t) . An oriented line is defined by connecting a point on the uv plane to a point on the st plane. In practice we restrict u, v, s , and t to lie between 0 and 1, and thus points on each plane are restricted to lie within a convex quadrilateral. We call this representation a *light slab*. Intuitively, a light slab represents the beam of light entering one quadrilateral and exiting another quadrilateral.

A nice feature of this representation is that one of the planes may be placed at infinity. This is convenient since then lines may be parameterized by a point and a direction. The latter will prove useful for constructing light fields either from orthographic images or images with a fixed field of view. Furthermore, if all calculations are performed using homogeneous coordinates, the two cases may be handled at no additional cost.

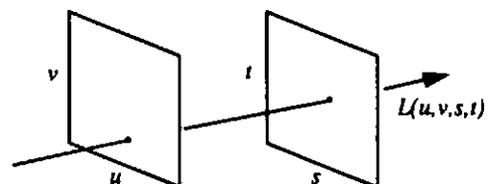


Figure 1: The light slab representation.

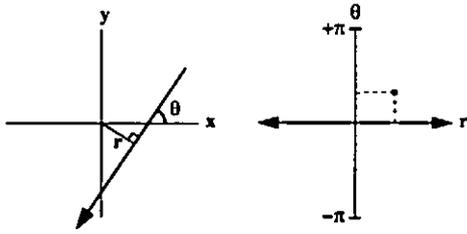


Figure 2: Definition of the line space we use to visualize sets of light rays. Each oriented line in Cartesian space (at left) is represented in line space (at right) by a point. To simplify the visualizations, we show only lines in 2D; the extension to 3D is straightforward.

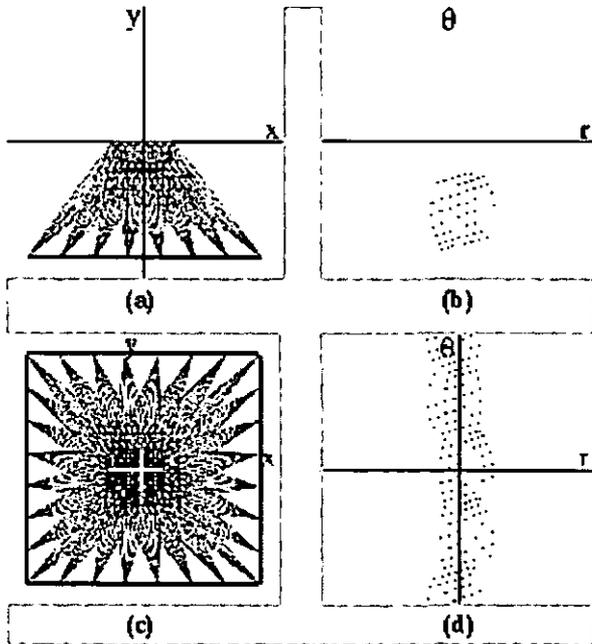


Figure 3: Using line space to visualize ray coverage. (a) shows a single light slab. Light rays (drawn in gray) connect points on two defining lines (drawn in red and green). (c) shows an arrangement of four rotated copies of (a). (b) and (d) show the corresponding line space visualizations. For any set of lines in Cartesian space, the envelope formed by the corresponding points in line space indicates our coverage of position and direction; ideally the coverage should be complete in θ and as wide as possible in r . As these figures show, the single slab in (a) does not provide full coverage in θ , but the four-slab arrangement in (c) does. (c) is, however, narrow in r . Such an arrangement is suitable for inward-looking views of a small object placed at the origin. It was used to generate the lion light field in figure 14d.

A big advantage of this representation is the efficiency of geometric calculations. Mapping from (u, v) to points on the plane is a projective map and involves only linear algebra (multiplying by a 3×3 matrix). More importantly, as will be discussed in section 5, the inverse mapping from an image pixel (x, y) to (u, v, s, t) is also a projective map. Methods using spherical or cylindrical coordinates require substantially more computation.

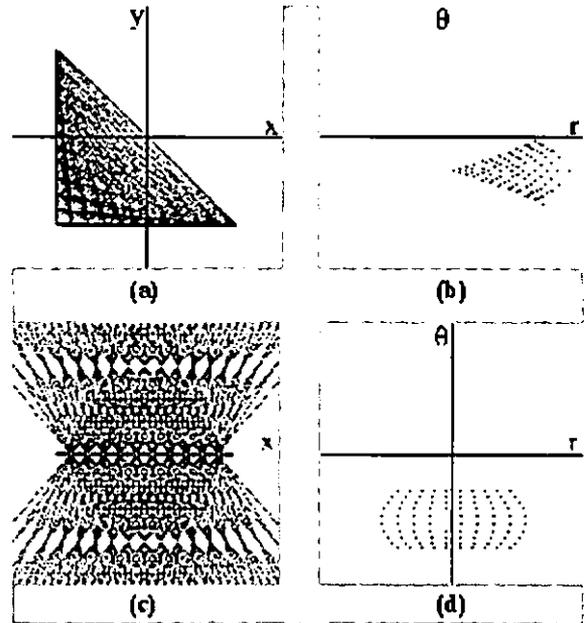


Figure 4: Using line space to visualize sampling uniformity. (a) shows a light slab defined by two lines at right angles. (c) shows a light slab where one defining line is at infinity. This arrangement generates rays passing through the other defining line with an angle between -45° and $+45^\circ$. (b) and (d) show the corresponding line space visualizations. Our use of (r, θ) to parameterize line space has the property that equal areas in line space correspond to equally dense sampling of position and orientation in Cartesian space; ideally the density of points in line space should be uniform. As these figures show, the singularity at the corner in (a) leads to a highly nonuniform and therefore inefficient sampling pattern, indicated by dark areas in (b) at angles of 0 and $-\pi/2$. (c) generates a more uniform set of lines. Although (c) does not provide full coverage of θ , four rotated copies do. Such an arrangement is suitable for outward-looking views by an observer standing near the origin. It was used to generate the hallway light field in figure 14c.

Many properties of light fields are easier to understand in line space (figures 2 through 4). In line space, each oriented line is represented by a point and each set of lines by a region. In particular, the set of lines represented by a light slab and the set of lines intersecting the convex hull of an object are both regions in line space. All views of an object could be generated from one light slab if its set of lines include all lines intersecting the convex hull of the object. Unfortunately, this is not possible. Therefore, it takes multiple light slabs to represent all possible views of an object. We therefore tile line space with a collection of light slabs, as shown in figure 3.

An important issue related to the parameterization is the sampling pattern. Assuming that all views are equally likely to be generated, then any line is equally likely to be needed. Thus all regions of line space should have an equal density of samples. Figure 4 shows the density of samples in line space for different arrangements of slabs. Note that no slab arrangement is perfect: arrangements with a singularity such as two polygons joined at a corner (4a) are bad and should be avoided, whereas slabs formed

from parallel planes (3a) generate fairly uniform patterns. In addition, arrangements where one plane is at infinity (4c) are better than those with two finite planes (3a). Finally, because of symmetry the spacing of samples in uv should in general be the same as st . However, if the observer is likely to stand near the uv plane, then it may be acceptable to sample uv less frequently than st .

3. Creation of light fields

In this section we discuss the creation of both virtual light fields (from rendered images) and real light fields (from digitized images). One method to create a light field would be to choose a 4D sampling pattern, and for each line sample, find the radiance. This is easily done directly for virtual environments by a ray tracer. This could also be done in a real environment with a spot radiometer, but it would be very tedious. A more practical way to generate light fields is to assemble a collection of images.

3.1. From rendered images

For a virtual environment, a light slab is easily generated simply by rendering a 2D array of images. Each image represents a slice of the 4D light slab at a fixed uv value and is formed by placing the center of projection of the virtual camera at the sample

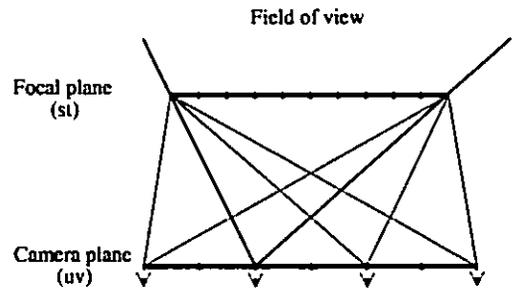


Figure 5: The viewing geometry used to create a light slab from an array of perspective images.

location on the uv plane. The only issue is that the xy samples of each image must correspond exactly with the st samples. This is easily done by performing a sheared perspective projection (figure 5) similar to that used to generate a stereo pair of images. Figure 6 shows the resulting 4D light field, which can be visualized either as a uv array of st images or as an st array of uv images.

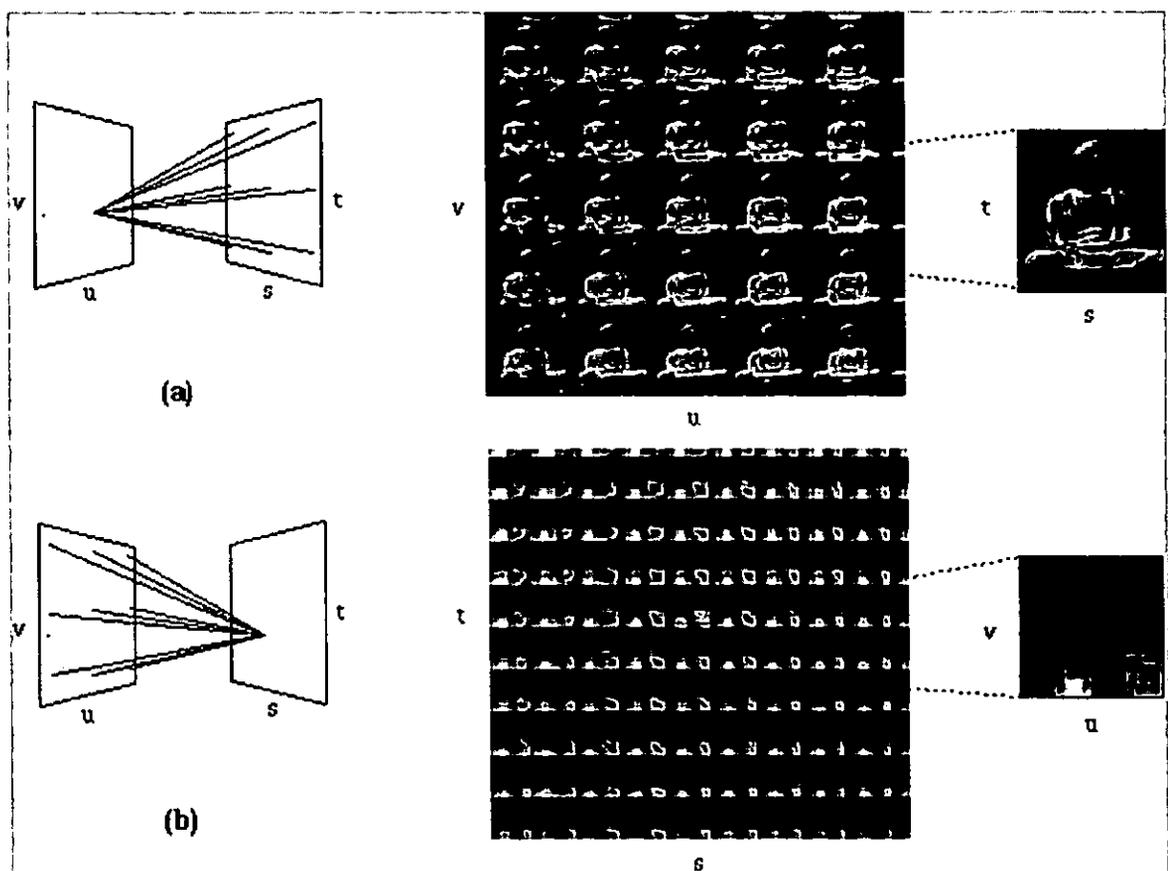


Figure 6: Two visualizations of a light field. (a) Each image in the array represents the rays arriving at one point on the uv plane from all points on the st plane, as shown at left. (b) Each image represents the rays leaving one point on the st plane bound for all points on the uv plane. The images in (a) are off-axis (i.e. sheared) perspective views of the scene, while the images in (b) look like reflectance maps. The latter occurs because the object has been placed astride the focal plane, making sets of rays leaving points on the focal plane similar in character to sets of rays leaving points on the object.

Two other viewing geometries are useful. A light slab may be formed from a 2D array of orthographic views. This can be modeled by placing the uv plane at infinity, as shown in figure 4c. In this case, each uv sample corresponds to the direction of a parallel projection. Again, the only issue is to align the xy and st samples of the image with the st quadrilateral. The other useful geometry consists of a 2D array of outward looking (non-sheared) perspective views with fixed field of view. In this case, each image is a slice of the light slab with the st plane at infinity. The fact that all these cases are equally easy to handle with light slabs attests to the elegance of projective geometry. Light fields using each arrangement are presented in section 6 and illustrated in figure 14.

As with any sampling process, sampling a light field may lead to aliasing since typical light fields contain high frequencies. Fortunately, the effects of aliasing may be alleviated by filtering before sampling. In the case of a light field, a 4D filter in the space of lines must be employed (see figure 7). Assuming a box filter, a weighted average of the radiances on all lines connecting sample squares in the uv and st planes must be computed. If a camera is placed on the uv plane and focussed on the st plane, then the filtering process corresponds to integrating both over a pixel corresponding to an st sample, and an aperture equal in size to a uv sample, as shown in figure 8. The theory behind this filtering process has been discussed in the context of holographic stereograms by Halle [Halle94].

Note that although prefiltering has the desired effect of antialiasing the light field, it has what at first seems like an undesirable side effect — introducing blurriness due to depth of field. However, this blurriness is precisely correct for the situation. Recall what happens when creating a pair of images from two adjacent camera locations on the uv plane: a given object point will project to different locations, potentially several pixels apart, in these two images. The distance between the two projected locations is called the stereo disparity. Extending this idea to multiple camera locations produces a sequence of images in which the object appears to jump by a distance equal to the disparity. This jumping is aliasing. Recall now that taking an image with a finite aperture causes points out of focus to be blurred on the film plane by a circle of confusion. Setting the diameter of the aperture to the spacing between camera locations causes the circle of confusion for each object point to be equal in size to its stereo disparity. This replaces the jumping with a sequence of blurred images. Thus, we are removing aliasing by employing finite depth of field.

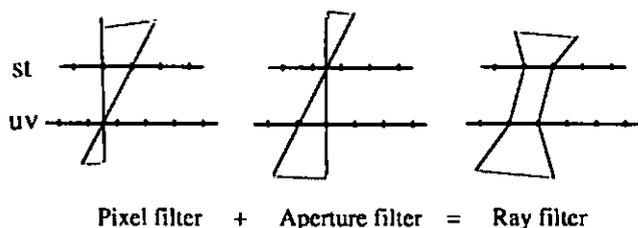


Figure 7: Prefiltering a light field. To avoid aliasing, a 4D low pass filter must be applied to the radiance function.

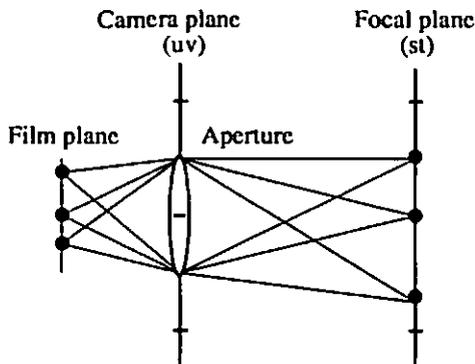


Figure 8: Prefiltering using an aperture. This figure shows a camera focused on the st plane with an aperture on the uv plane whose size is equal to the uv sample spacing. A hypothetical film plane is drawn behind the aperture. Ignore the aperture for a moment (consider a pinhole camera that precisely images the st plane onto the film plane). Then integrating over a pixel on the film plane is equivalent to integrating over an st region bounded by the pixel. Now consider fixing a point on the film plane while using a finite sized aperture (recall that all rays from a point on the film through the aperture are focussed on a single point on the focal plane). Then integrating over the aperture corresponds to integrating all rays through the uv region bounded by the aperture. Therefore, by simultaneously integrating over both the pixel and the aperture, the proper 4D integral is computed.

The necessity for prefiltering can also be understood in line space. Recall from our earlier discussion that samples of the light field correspond to points in line space. Having a finite depth of field with an aperture equal in size to the uv sample spacing insures that each sample adequately covers the interval between these line space points. Too small or too large an aperture yields gaps or overlaps in line space coverage, resulting in views that are either aliased or excessively blurry, respectively.

3.2. From digitized images

Digitizing the imagery required to build a light field of a physical scene is a formidable engineering problem. The number of images required is large (hundreds or thousands), so the process must be automated or at least computer-assisted. Moreover, the lighting must be controlled to insure a static light field, yet flexible enough to properly illuminate the scene, all the while staying clear of the camera to avoid unwanted shadows. Finally, real optical systems impose constraints on angle of view, focal distance, depth of field, and aperture, all of which must be managed. Similar issues have been faced in the construction of devices for performing near-field photometric measurements of luminaires [Ash-down93]. In the following paragraphs, we enumerate the major design decisions we faced in this endeavor and the solutions we adopted.

Inward versus outward looking. The first decision to be made was between a flyaround of a small object and a flythrough of a large-scale scene. We judged flyarounds to be the simpler case, so we attacked them first.



Figure 9: Our prototype camera gantry. A modified Cyberware MS motion platform with additional stepping motors from Lin-Tech and Parker provide four degrees of freedom: horizontal and vertical translation, pan, and tilt. The camera is a Panasonic WV-F300 3-CCD video camera with a Canon f/1.7 10-120mm zoom lens. We keep it locked off at its widest setting (10mm) and mounted so that the pitch and yaw axes pass through the center of projection. While digitizing, the camera is kept pointed at the center of the focal plane. Calibrations and alignments are verified with the aid of a Faro digitizing arm, which is accurate to 0.3 mm.

Human versus computer-controlled. An inexpensive approach to digitizing light fields is to move a handheld camera through the scene, populating the field from the resulting images [Gortler96]. This approach necessitates estimating camera pose at each frame and interpolating the light field from scattered data - two challenging problems. To simplify the situation, we chose instead to build a computer-controlled camera gantry and to digitize images on a regular grid.

Spherical versus planar camera motion. For flyarounds of small objects, an obvious gantry design consists of two concentric hemicycles, similar to a gyroscope mounting. The camera in such a gantry moves along a spherical surface, always pointing at the center of the sphere. Apple Computer has constructed such a gantry to acquire imagery for Quick-Time VR flyarounds [Chen95]. Unfortunately, the lighting in their system is attached to the moving camera, so it is unsuitable for acquiring static light fields. In general, a spherical gantry has three advantages over a planar gantry: (a) it is easier to cover the entire range of viewing directions, (b) the sampling rate in direction space is more uniform, and (c) the distance between the camera and the object is fixed, providing sharper focus throughout the range of camera motion. A planar gantry has two advantages over a spherical gantry: (a) it is easier to build; the entire structure can be assembled from linear motion stages, and (b) it is closer to our light slab representation. For our first prototype gantry, we chose to build a planar gantry, as shown in figure 9.

Field of view. Our goal was to build a light field that allowed 360 degrees of azimuthal viewing. To accomplish this using a planar gantry meant acquiring four slabs each providing 90

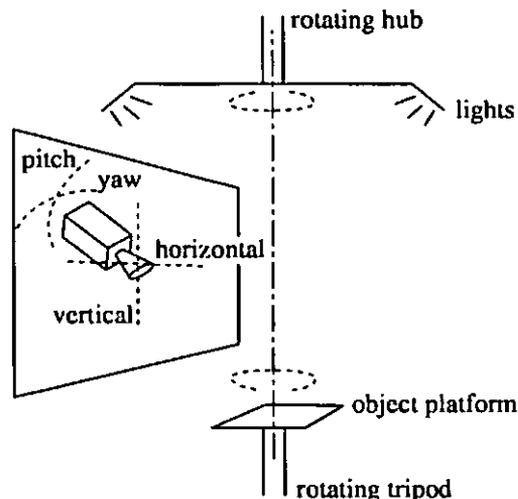


Figure 10: Object and lighting support. Objects are mounted on a Bogen fluid-head tripod, which we manually route to four orientations spaced 90 degrees apart. Illumination is provided by two 600W Lowell Omni spotlights attached to a ceiling-mounted rotating hub that is aligned with the rotation axis of the tripod. A stationary 6' x 6' diffuser panel is hung between the spotlights and the gantry, and the entire apparatus is enclosed in black velvet to eliminate stray light.

degrees. This can be achieved with a camera that translates but does not pan or tilt by employing a wide-angle lens. This solution has two disadvantages: (a) wide-angle lenses exhibit significant distortion, which must be corrected after acquisition, and (b) this solution trades off angle of view against sensor resolution. Another solution is to employ a view camera in which the sensor and optical system translate in parallel planes, the former moving faster than the latter. Horizontal parallax holographic stereograms are constructed using such a camera [Halle94]. Incorporating this solution into a gantry that moves both horizontally and vertically is difficult. We instead chose to equip our camera with pan and tilt motors, enabling us to use a narrow-angle lens. The use of a rotating camera means that, in order to transfer the acquired image to the light slab representation, it must be reprojected to lie on a common plane. This reprojection is equivalent to keystone correction in architectural photography.

Standoff distance. A disadvantage of planar gantries is that the distance from the camera to the object changes as the camera translates across the plane, making it difficult to keep the object in focus. The view camera described above does not suffer from this problem, because the ratio of object distance to image distance stays constant as the camera translates. For a rotating camera, servo-controlled focusing is an option, but changing the focus of a camera shifts its center of projection and changes the image magnification, complicating acquisition. We instead mitigate this problem by using strong lighting and a small aperture to maximize depth of field.

Sensor rotation. Each sample in a light slab should ideally represent the integral over a pixel, and these pixels should lie on a common focal plane. A view camera satisfies this constraint because its sensor translates in a plane. Our use of a rotating camera means that the focal plane also rotates. Assuming that

we resample the images carefully during reprojection, the presence of a rotated focal plane will introduce no additional error into the light field. In practice, we have not seen artifacts due to this resampling process.

Aperture size. Each sample in a light slab should also represent the integral over an aperture equal in size to a uv sample. Our use of a small aperture produces a light field with little or no uv antialiasing. Even fully open, the apertures of commercial video cameras are small. We can approximate the required antialiasing by averaging together some number of adjacent views, thereby creating a *synthetic aperture*. However, this technique requires a very dense spacing of views, which in turn requires rapid acquisition. We do not currently do this.

Object support. In order to acquire a 360-degree light field in four 90-degree segments using a planar gantry, either the gantry or the object must be rotated to each of four orientations spaced 90 degrees apart. Given the massiveness of our gantry, the latter was clearly easier. For these experiments, we mounted our objects on a tripod, which we manually rotate to the four positions as shown in figure 10.

Lighting. Given our decision to rotate the object, satisfying the requirement for fixed illumination means that either the lighting must exhibit fourfold symmetry or it must rotate with the object. We chose the latter solution, attaching a lighting system to a rotating hub as shown in figure 10. Designing a lighting system that stays clear of the gantry, yet provides enough light to evenly illuminate an object, is a challenging problem.

Using this gantry, our procedure for acquiring a light field is as follows. For each of the four orientations, the camera is translated through a regular grid of camera positions. At each position, the camera is panned and tilted to point at the center of the object, which lies along the axis of rotation of the tripod. We then acquire an image, and, using standard texture mapping algorithms, reproject it to lie on a common plane as described earlier. Table II gives a typical set of acquisition parameters. Note that the distance between camera positions (3.125 cm) exceeds the diameter of the aperture (1.25 mm), underscoring the need for denser spacing and a synthetic aperture.

4. Compression

Light field arrays are large — the largest example in this paper is 1.6 GB. To make creation, transmission, and display of light fields practical, they must be compressed. In choosing from among many available compression techniques, we were guided by several unique characteristics of light fields:

Data redundancy. A good compression technique removes redundancy from a signal without affecting its content. Light fields exhibit redundancy in all four dimensions. For example, the smooth regions in figure 6a tell us that this light field contains redundancy in s and t, and the smooth regions in figure 6b tell us that the light field contains redundancy in u and v. The former corresponds to our usual notion of interpixel coherence in a perspective view. The latter can be interpreted either as the interframe coherence one expects in a motion sequence or as the smoothness one expects in the bidirectional reflectance distribution function (BRDF) for a diffuse or moderately specular surface. Occlusions introduce discontinuities in both cases, of course.

Random access. Most compression techniques place some constraint on random access to data. For example, variable-bitrate coders may require scanlines, tiles, or frames to be decoded at once. Examples in this class are variable-bitrate vector quantization and the Huffman or arithmetic coders used in JPEG or MPEG. Predictive coding schemes further complicate random-access because pixels depend on previously decoded pixels, scanlines, or frames. This poses a problem for light fields since the set of samples referenced when extracting an image from a light field are dispersed in memory. As the observer moves, the access patterns change in complex ways. We therefore seek a compression technique that supports low-cost random access to individual samples.

Asymmetry. Applications of compression can be classified as symmetric or asymmetric depending on the relative time spent encoding versus decoding. We assume that light fields are assembled and compressed ahead of time, making this an asymmetric application.

Computational expense. We seek a compression scheme that can be decoded without hardware assistance. Although software decoders have been demonstrated for standards like JPEG and MPEG, these implementations consume the full power of a modern microprocessor. In addition to decompression, the display algorithm has additional work to perform, as will be described in section 5. We therefore seek a compression scheme that can be decoded quickly.

The compression scheme we chose was a two-stage pipeline consisting of fixed-rate vector quantization followed by entropy coding (Lempel-Ziv), as shown in figure 11. Following similar motivations, Beers et al. use vector quantization to compress textures for use in rendering pipelines [Beers96].

4.1. Vector quantization

The first stage of our compression pipeline is vector quantization (VQ) [Gersho92], a lossy compression technique wherein a vector of samples is quantized to one of a number of predetermined reproduction vectors. A reproduction vector is called a codeword, and the set of codewords available to encode a source is called the codebook. Codebooks are constructed during a training phase in which the quantizer is asked to find a set of codewords that best approximates a set of sample vectors, called the training set. The quality of a codeword is typically characterized

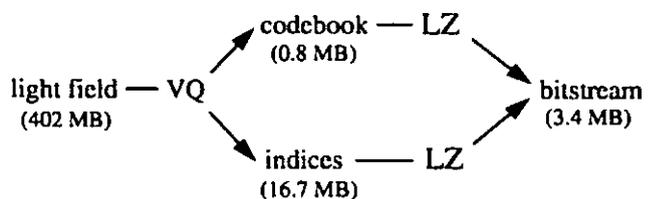


Figure 11 Two-stage compression pipeline. The light field is partitioned into tiles, which are encoded using vector quantization to form an array of codebook indices. The codebook and the array of indices are further compressed using Lempel-Ziv coding. Decompression also occurs in two stages: entropy decoding as the file is loaded into memory, and dequantization on demand during interactive viewing. Typical file sizes are shown beside each stage.

using mean-squared error (MSE), i.e. the sum over all samples in the vector of the squared difference between the source sample and the codeword sample. Once a codebook has been constructed, encoding consists of partitioning the source into vectors and finding for each vector the closest approximating codeword from the codebook. Decoding consists of looking up indices in the codebook and outputting the codewords found there — a very fast operation. Indeed, decoding speed is one of the primary advantages of vector quantization.

In our application, we typically use 2D or 4D tiles of the light field, yielding 12-dimensional or 48-dimensional vectors, respectively. The former takes advantage of coherence in s and t only, while the latter takes advantage of coherence in all four dimensions. To maximize image quality, we train on a representative subset of each light field to be compressed, then transmit the resulting codebook along with the codeword index array. Since light fields are large, even after compression, the additional overhead of transmitting a codebook is small, typically less than 20%. We train on a subset rather than the entire light field to reduce the expense of training.

The output of vector quantization is a sequence of fixed-rate codebook indices. Each index is $\log N$ bits where N is the number of codewords in the codebook, so the compression rate of the quantizer is $(kl) / (\log N)$ where k is the number of elements per vector (i.e. the dimension), and l is the number of bits per element, usually 8. In our application, we typically use 16384-word codebooks, leading to a compression rate for this stage of the pipeline of $(48 \times 8) / (\log 16384) = 384 \text{ bits} / 14 \text{ bits} = 27:1$. To simplify decoding, we represent each index using an integral number of bytes, 2 in our case, which reduces our compression slightly, to 24:1.

4.2. Entropy coding

The second stage of our compression pipeline is an entropy coder designed to decrease the cost of representing high-probability code indices. Since our objects are typically rendered or photographed against a constant-color background, the array contains many tiles that occur with high probability. For the examples in this paper, we employed gzip, an implementation of Lempel-Ziv coding [Ziv77]. In this algorithm, the input stream is partitioned into nonoverlapping blocks while constructing a dictionary of blocks seen thus far. Applying gzip to our array of code indices typically gives us an additional 5:1 compression. Huffman coding would probably yield slightly higher compression, but encoding and decoding would be more expensive. Our total compression is therefore $24 \times 5 = 120:1$. See section 6 and table III for more detail on our compression results.

4.3. Decompression

Decompression occurs in two stages. The first stage — gzip decoding — is performed as the file is loaded into memory. The output of this stage is a codebook and an array of code indices packed in 16-bit words. Although some efficiency has been lost by this decoding, the light field is still compressed 24:1, and it is now represented in a way that supports random access.

The second stage — dequantization — proceeds as follows. As the observer moves through the scene, the display engine requests samples of the light field. Each request consists of a (u, v, s, t) coordinate tuple. For each request, a subscripting calculation is performed to determine which sample tile is being

addressed. Each tile corresponds to one quantization vector and is thus represented in the index array by a single entry. Looking this index up in the codebook, we find a vector of sample values. A second subscripting calculation is then performed, giving us the offset of the requested sample within the vector. With the aid of precomputed subscripting tables, dequantization can be implemented very efficiently. In our tests, decompression consumes about 25% of the CPU cycles.

5. Display

The final part of the system is a real time viewer that constructs and displays an image from the light slab given the imaging geometry. The viewer must resample a 2D slice of lines from the 4D light field; each line represents a ray through the eye point and a pixel center as shown in figure 12. There are two steps to this process: step 1 consists of computing the (u, v, s, t) line parameters for each image ray, and step 2 consists of resampling the radiance at those line parameters.

As mentioned previously, a big advantage of the light slab representation is the efficiency of the inverse calculation of the line parameters. Conceptually the (u, v) and (s, t) parameters may be calculated by determining the point of intersection of an image ray with each plane. Thus, any ray tracer could easily be adapted to use light slabs. However, a polygonal rendering system also may be used to view a light slab. The transformation from image coordinates (x, y) to both the (u, v) and the (s, t) coordinates is a projective map. Therefore, computing the line coordinates can be done using texture mapping. The uv quadrilateral is drawn using the current viewing transformation, and during scan conversion the (uw, vw, w) coordinates at the corners of the quadrilateral are interpolated. The resulting $u = uw/w$ and $v = vw/w$ coordinates at each pixel represent the ray intersection with the uv quadrilateral. A similar procedure can be used to generate the (s, t) coordinates by drawing the st quadrilateral. Thus, the inverse transformation from (x, y) to (u, v, s, t) reduces essentially to two texture coordinate calculations per ray. This is cheap and can be done in real time, and is supported in many rendering systems, both hardware and software.

Only lines with (u, v) and (s, t) coordinates inside both quadrilaterals are represented in the light slab. Thus, if the texture coordinates for each plane are computed by drawing each quadrilateral one after the other, then only those pixels that have both valid uv and st coordinates should be looked up in the light slab array. Alternatively, the two quadrilaterals may be simultaneously scan converted in their region of overlap to cut down on unnecessary calculations; this is the technique that we use in our software implementation.

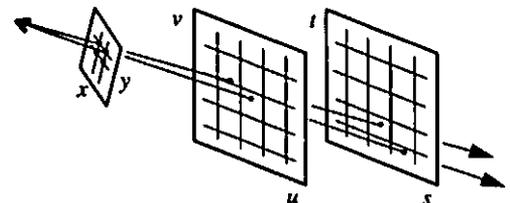


Figure 12: The process of resampling a light slab during display.

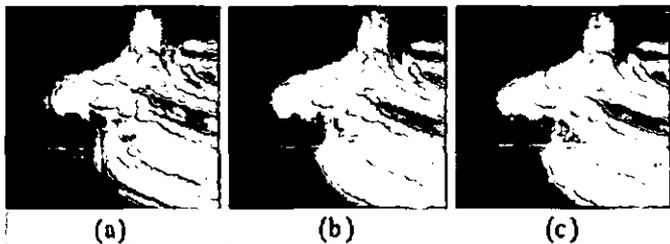


Figure 13: The effects of interpolation during slice extraction. (a) No interpolation. (b) Linear interpolation in uv only. (c) Quadrilinear interpolation in uvst.

To draw an image of a collection of light slabs, we draw them sequentially. If the sets of lines in the collection of light slabs do not overlap, then each pixel is drawn only once and so this is quite efficient. To further increase efficiency, "back-facing" light slabs may be culled.

The second step involves resampling the radiance. The ideal resampling process first reconstructs the function from the original samples, and then applies a bandpass filter to the reconstructed function to remove high frequencies that may cause aliasing. In our system, we approximate the resampling process by simply interpolating the 4D function from the nearest samples. This is correct only if the new sampling rate is greater than the original sampling rate, which is usually the case when displaying light fields. However, if the image of the light field is very small, then some form of prefiltering should be applied. This could easily be done with a 4D variation of the standard mipmapping algorithm [Williams83].

Figure 13 shows the effect of nearest neighbor versus bilinear interpolation on the uv plane versus quadrilinear interpolation of the full 4D function. Quadrilinear interpolation coupled with the proper prefiltering generates images with few aliasing artifacts. The improvement is particularly dramatic when the object or camera is moving. However, quadrilinear filtering is more expensive and can sometimes be avoided. For example, if the sampling rates in the uv and st planes are different, and then the benefits of filtering one plane may be greater than the other plane.

6. Results

Figure 14 shows images extracted from four light fields. The first is a buddha constructed from rendered images. The model is an irregular polygon mesh constructed from range data. The input images were generated using RenderMan, which also provided the machinery for computing pixel and aperture

	buddha	kidney	hallway	lion
Number of slabs	1	1	4	4
Images per slab	16x16	64x64	64x32	32x16
Total images	256	4096	8192	2048
Pixels per image	256 ²	128 ²	256 ²	256 ²
Raw size (MB)	50	201	1608	402
Prefiltering	uvst	st only	uvst	st only

Table I: Statistics of the light fields shown in figure 14.

antialiasing. The light field configuration was a single slab similar to that shown in figure 3a.

Our second light field is a human abdomen constructed from volume renderings. The two tan-colored organs on either side of the spine are the kidneys. In this case, the input images were orthographic views, so we employed a slab with one plane at infinity as shown in figure 4c. Because an orthographic image contains rays of constant direction, we generated more input images than in the first example in order to provide the angular range needed for creating perspective views. The images include pixel antialiasing but no aperture antialiasing. However, the dense spacing of input images reduces aperture aliasing artifacts to a minimum.

Our third example is an outward-looking light field depicting a hallway in Berkeley's Soda Hall, rendered using a radiosity program. To allow a full range of observer motion while optimizing sampling uniformity, we used four slabs with one plane at infinity, a four-slab version of figure 4c. The input images were rendered on an SGI RealityEngine, using the accumulation buffer to provide both pixel and aperture antialiasing.

Our last example is a light field constructed from digitized images. The scene is of a toy lion, and the light field consists of four slabs as shown in figure 3c, allowing the observer to walk completely around the object. The sensor and optical system provide pixel antialiasing, but the aperture diameter was too small to provide correct aperture antialiasing. As a result, the light field exhibits some aliasing, which appears as double images. These artifacts are worst near the head and tail of the lion because of their greater distance from the axis around which the camera rotated.

Table I summarizes the statistics of each light field. Table II gives additional information on the lion dataset. Table III gives the performance of our compression pipeline on two representative datasets. The buddha was compressed using a 2D tiling of the

Camera motion	
translation per slab	100 cm x 50 cm
pan and tilt per slab	90° x 45°
number of slabs	4 slabs 90° apart
total pan and tilt	360° x 45°
Sampling density	
distance to object	50 cm
camera pan per sample	3.6°
camera translation per sample	3.125 cm
Aperture	
focal distance of lens	10mm
F-number	f/8
aperture diameter	1.25 mm
Acquisition time	
time per image	3 seconds
total acquisition time	4 hours

Table II: Acquisition parameters for the lion light field. Distance to object and camera pan per sample are given at the center of the plane of camera motion. Total acquisition time includes longer gantry movements at the end of each row and manual setup time for each of the four orientations. The aperture diameter is the focal length divided by the F-number.

	buddha	lion
Vector quantization		
raw size (MB)	50.3	402.7
fraction in training set	5%	3%
samples per tile	2x2x1x1	2x2x2x2
bytes per sample	3	3
vector dimension	12	48
number of codewords	8192	16384
codebook size (MB)	0.1	0.8
bytes per codeword index	2	2
index array size (MB)	8.4	16.8
total size (MB)	8.5	17.6
compression rate	6:1	23:1
Entropy coding		
gzipped codebook (MB)	0.1	0.6
gzipped index array (MB)	1.0	2.8
total size (MB)	1.1	3.4
compression due to gzip	8:1	5:1
total compression	45:1	118:1
Compression performance		
training time	15 mins	4 hrs
encoding time	1 mins	8 mins
original entropy (bits/pixel)	4.2	2.9
image quality (PSNR)	36	27

Table III: Compression statistics for two light fields. The buddha was compressed using 2D tiles of RGB pixels, forming 12-dimensional vectors, and the lion was compressed using 4D tiles (2D tiles of RGB pixels from each of 2 x 2 adjacent camera positions), forming 48-dimensional vectors. Bytes per codeword index include padding as described in section 4. Peak signal-to-noise ratio (PSNR) is computed as $10 \log_{10}(255^2/MSE)$.

light field, yielding a total compression rate of 45:1. The lion was compressed using a 4D tiling, yielding a higher compression rate of 118:1. During interactive viewing, the compressed buddha is indistinguishable from the original; the compressed lion exhibits some artifacts, but only at high magnifications. Representative images are shown in figure 15. We have also experimented with higher rates. As a general rule, the artifacts become objectionable only above 200:1.

Finally, table IV summarizes the performance of our interactive viewer operating on the lion light field. As the table shows, we achieve interactive playback rates for reasonable image sizes. Note that the size of the light field has no effect on playback rate; only the image size matters. Memory size is not an issue because the compressed fields are small.

7. Discussion and future work

We have described a new light field representation, the light slab, for storing all the radiance values in free space. Both inserting images into the field and extracting new views from the field involve resampling, a simple and robust procedure. The resulting system is easily implemented on workstations and personal computers, requiring modest amounts of memory and cycles. Thus, this technique is useful for many applications requiring interaction with 3D scenes.

Display times (ms)	no bilerp	uv lerp	uvst lerp
coordinate calculation	13	13	13
sample extraction	14	59	214
overhead	3	3	3
total	30	75	230

Table IV: Display performance for the lion light field. Displayed images are 192 x 192 pixels. Sample extraction includes VQ decoding and sample interpolation. Display overhead includes reading the mouse, computing the observer position, and copying the image to the frame buffer. Timings are for a software-only implementation on a 250 MHz MIPS 4400 processor.

There are three major limitation of our method. First, the sampling density must be high to avoid excessive blurriness. This requires rendering or acquiring a large number of images, which may take a long time and consume a lot of memory. However, denser sample spacing leads to greater inter-sample coherence, so the size of the light field is usually manageable after compression. Second, the observer is restricted to regions of space free of occluders. This limitation can be addressed by stitching together multiple light fields based on a partition of the scene geometry into convex regions. If we augment light fields to include Z-depth, the regions need not even be convex. Third, the illumination must be fixed. If we ignore interreflections, this limitation can be addressed by augmenting light fields to include surface normals and optical properties. To handle interreflections, we might try representing illumination as a superposition of basis functions [Nimeroff94]. This would correspond in our case to computing a sum of light fields each lit with a different illumination function.

It is useful to compare this approach with depth-based or correspondence-based view interpolation. In these systems, a 3D model is created to improve quality of the interpolation and hence decrease the number of pre-acquired images. In our approach, a much larger number of images is acquired, and at first this seems like a disadvantage. However, because of the 3D structure of the light field, simple compression schemes are able to find and exploit this same 3D structure. In our case, simple 4D block coding leads to compression rates of over 100:1. Given the success of the compression, a high density compressed light field has an advantage over other approaches because the resampling process is simpler, and no explicit 3D structure must be found or stored.

There are many representations for light used in computer graphics and computer vision, for example, images, shadow and environment maps, light sources, radiosity and radiance basis functions, and ray tracing procedures. However, abstract light representations have not been systematically studied in the same way as modeling and display primitives. A fruitful line of future research would be to reexamine these representations from first principles. Such reexaminations may in turn lead to new methods for the central problems in these fields.

Another area of future research is the design of instrumentation for acquisition. A large parallel array of cameras connected to a parallel computer could be built to acquire and compress a light field in real time. In the short term, there are many interesting engineering issues in designing and building gantries to move

a small number of cameras and lights to sequentially acquire both inward- and outward-looking light fields. This same instrumentation could lead to breakthroughs in both 3D shape acquisition and reflection measurements. In fact, the interaction of light with any object can be represented as a higher-dimensional interaction matrix; acquiring, compressing, and manipulating such representations are a fruitful area for investigation.

8. Acknowledgements

We would like to thank David Addleman and George Dabrowski of Cyberware for helping us design and build the camera gantry, Craig Kolb and James Davis for helping us calibrate it, Brian Curless for scanning the buddha, Julie Dorsey for shading it and allowing us to use it, Carlo Sequin for the Soda Hall model, Seth Teller, Celeste Fowler, and Thomas Funkhouser for its radiosity solution, Lucas Pereira for rendering it, Benjamin Zhu for reimplementing our hardware-accelerated viewer in software, and Navin Chaddha for his vector quantization code. We also wish to thank Eric Chen and Michael Chen for allowing us to examine the Apple ObjectMaker, and Alain Fournier and Bob Lewis for showing us their wavelet light field work. Finally, we wish to thank Nina Amenta for sparking our interest in two-plane parameterizations of lines, Michael Cohen for reinforcing our interest in image-based representations, and Gavin Miller for inspiring us with his grail of volumetric hyperreality. This work was supported by the NSF under contracts CCR-9157767 and CCR-9508579.

9. References

- [Adelson91] Adelson, E.H., Bergen, J.R., "The Plenoptic Function and the Elements of Early Vision," In *Computation Models of Visual Processing*, M. Landy and J.A. Movshon, eds., MIT Press, Cambridge, 1991.
- [Ashdown93] Ashdown, I., "Near-Field Photometry: A New Approach," *Journal of the Illuminating Engineering Society*, Vol. 22, No. 1, Winter, 1993, pp. 163-180.
- [Beers96] Beers, A., Agrawala, M., Chaddha, N., "Rendering from Compressed Textures." In these proceedings.
- [Benton83] Benton, S., "Survey of Holographic Stereograms," *Processing and Display of Three-Dimensional Data*, Proc. SPIE, Vol. 367, 1983.
- [Blinn76] Blinn, J.F., Newell, M.E., "Texture and Reflection in Computer Generated Images," *CACM*, Vol. 19, No. 10, October, 1976, pp. 542-547.
- [Bolles87] Bolles, R., Baker, H., Marimont, D., "Epipolar-Plane Image Analysis: An Approach to Determining Structure from Motion," *International Journal of Computer Vision*, Vol. 1, No. 1, 1987, pp. 7-55.
- [Chen93] Chen, S.E., Williams, L., "View Interpolation for Image Synthesis," Proc. SIGGRAPH '93 (Anaheim, California, August 1-6, 1993). In *Computer Graphics Proceedings, Annual Conference Series, 1993*, ACM SIGGRAPH, pp. 279-288.
- [Chen95] Chen, S.E., "QuickTime VR — An Image-Based Approach to Virtual Environment Navigation," Proc. SIGGRAPH '95 (Los Angeles, CA, August 6-11, 1995). In *Computer Graphics Proceedings, Annual Conference Series, 1995*, ACM SIGGRAPH, pp. 29-38.
- [Fuchs94] Fuchs, H., Bishop, G., Arthur, K., McMillan, L., Bajcsy, R., Lee, S.W., Farid, H., Kanade, T., "Virtual Space Teleconferencing Using a Sea of Cameras," *Proc. First International Conference on Medical Robotics and Computer Assisted Surgery*, 1994, pp. 161-167.
- [Gersho92] Gersho, A., Gray, R.M., *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, 1992.
- [Gershun36] Gershun, A., "The Light Field," Moscow, 1936. Translated by P. Moon and G. Timoshenko in *Journal of Mathematics and Physics*, Vol. XVIII, MIT, 1939, pp. 51-151.
- [Gortler96] Gortler, S.J., Grzeszczuk, R., Szeliski, R., Cohen, M., "The Lumigraph." In these proceedings.
- [Greene86] Greene, N., "Environment Mapping and Other Applications of World Projections," *IEEE Computer Graphics and Applications*, Vol. 6, No. 11, November, 1986, pp. 21-29.
- [Greene94] Greene, N. and Kass, M., "Approximating Visibility with Environment Maps," Apple Technical Report No. 41, November, 1994.
- [Halle94] Halle, M., "Holographic Stereograms as Discrete Imaging Systems," *Practical Holography*, Proc. SPIE, Vol. 2176, February, 1994.
- [Katayama95] Katayama, A., Tanaka, K., Oshino, T., Tamura, H., "Viewpoint-Dependent Stereoscopic Display Using Interpolation of Multi-viewpoint Images," *Stereoscopic Displays and Virtual Reality Systems II*, Proc. SPIE, Vol. 2409, S. Fisher, J. Merritt, B. Bolas eds. 1995, pp. 11-20.
- [Laveau94] Laveau, S., Faugeras, O.D., "3-D Scene Representation as a Collection of Images and Fundamental Matrices," INRIA Technical Report No. 2205, 1994.
- [Levin71] Levin, R., "Photometric Characteristics of Light Controlling Apparatus," *Illuminating Engineering*, Vol. 66, No. 4, 1971, pp. 205-215.
- [McMillan95a] McMillan, L., Bishop, G., "Head-Trackable Stereoscopic Display Using Image Warping," *Stereoscopic Displays and Virtual Reality Systems II*, Proc. SPIE, Vol. 2409, S. Fisher, J. Merritt, B. Bolas eds. 1995, pp. 21-30.
- [McMillan95b] McMillan, L., Bishop, G., "Plenoptic Modeling: An Image-Based Rendering System," Proc. SIGGRAPH '95 (Los Angeles, CA, August 6-11, 1995). In *Computer Graphics Proceedings, Annual Conference Series, 1995*, ACM SIGGRAPH, pp. 39-46.
- [Miller95] Miller, G., "Volumetric Hyper-Reality: A Computer Graphics Holy Grail for the 21st Century?," *Proc. Graphics Interface '95*, W. Davis and P. Prusinkiewicz eds., Canadian Information Processing Society, 1995, pp. 56-64.
- [Moon81] Moon, P., Spencer, D.E., *The Photoc Field*, MIT Press, 1981.
- [Narayanan95] Narayanan, P.J., "Virtualized Reality: Concepts and Early Results," *Proc. IEEE Workshop on the Representation of Visual Scenes*, IEEE, 1995.
- [Nimeroff94] Nimeroff, J., Simoncelli, E., Dorsey, J., "Efficient Rendering of Naturally Illuminated Scenes," *Proc. Fifth Eurographics Rendering Workshop*, 1994, pp. 359-373.
- [Sbert93] Sbert, A.M., "An Integral Geometry Based Method for Form-Factor Computation," *Computer Graphics Forum*, Vol. 13, No. 3, 1993, pp. 409-420.
- [Seitz95] Seitz, S., Dyer, C., "Physically-Valid View Synthesis by Image Interpolation," *Proc. IEEE Workshop on the Representation of Visual Scenes*, IEEE, 1995.
- [Williams83] Williams, L., "Pyramidal Parametrics," *Computer Graphics (Proc. Siggraph '83)*, Vol. 17, No. 3, July, 1983, pp. 1-11.
- [Ziv77] Ziv, J., Lempel, A., "A universal algorithm for sequential data compression," *IEEE Transactions on Information Theory*, IT-23:337-343, 1977.

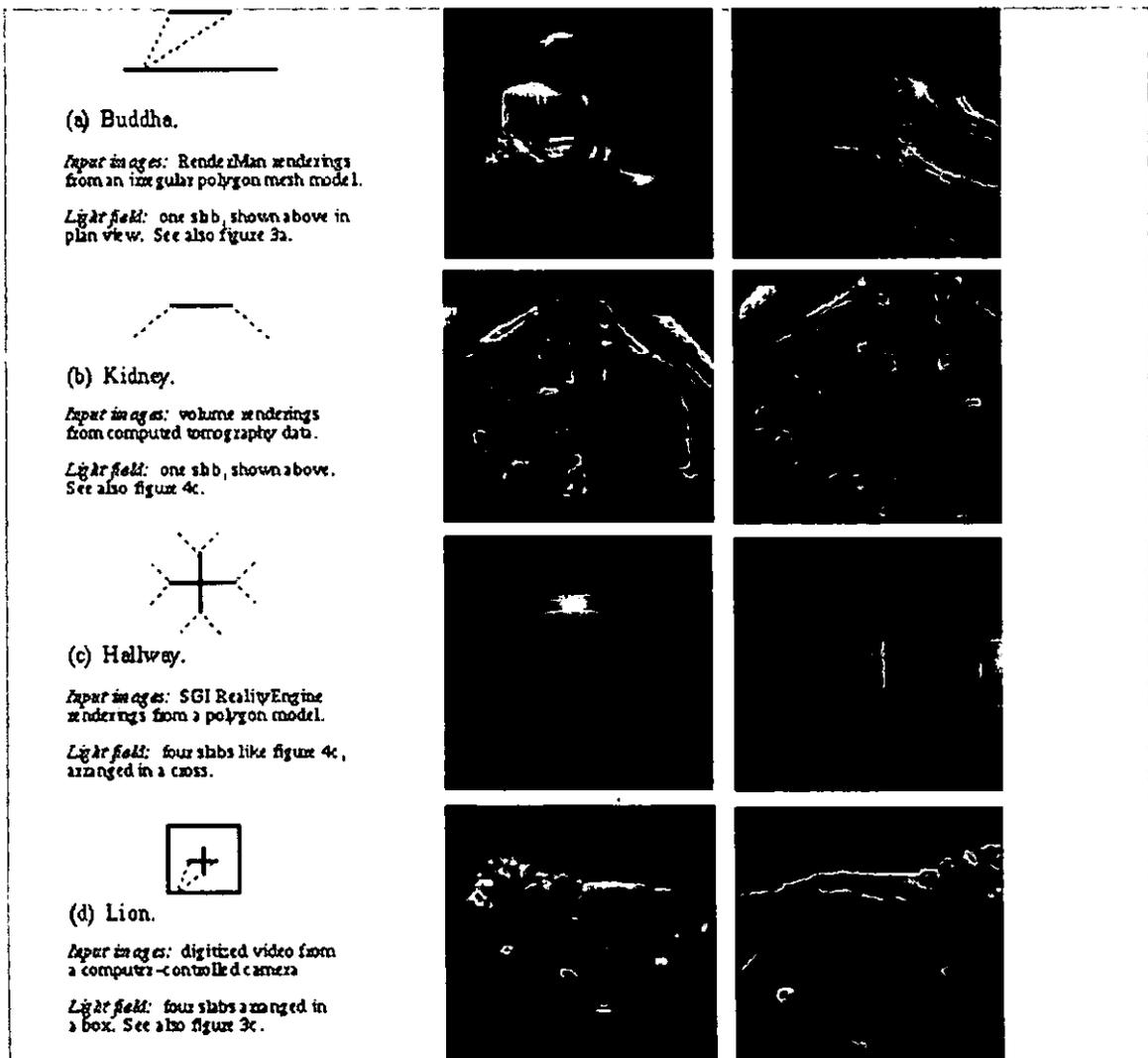


Figure 14: Example images from four light fields, extracted during a typical interactive viewing session.

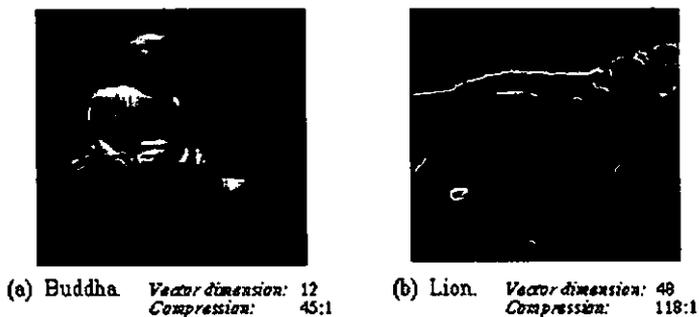


Figure 15: Images extracted from compressed light fields.

The Lumigraph

Steven J. Gortler
Microsoft Research

Radek Grzeszczuk
University of Toronto*

Richard Szeliski
Microsoft Research

Michael F. Cohen
Microsoft Research

Abstract

This paper discusses a new method for capturing the complete appearance of both synthetic and real world objects and scenes, representing this information, and then using this representation to render images of the object from new camera positions. Unlike the shape capture process traditionally used in computer vision and the rendering process traditionally used in computer graphics, our approach does not rely on geometric representations. Instead we sample and reconstruct a 4D function, which we call a Lumigraph. The Lumigraph is a subset of the complete plenoptic function that describes the flow of light at all positions in all directions. With the Lumigraph, new images of the object can be generated very quickly, independent of the geometric or illumination complexity of the scene or object. The paper discusses a complete working system including the capture of samples, the construction of the Lumigraph, and the subsequent rendering of images from this new representation.

1 Introduction

The process of creating a virtual environment or object in computer graphics begins with modeling the geometric and surface attributes of the objects in the environment along with any lights. An image of the environment is subsequently rendered from the vantage point of a virtual camera. Great effort has been expended to develop computer aided design systems that allow the specification of complex geometry and material attributes. Similarly, a great deal of work has been undertaken to produce systems that simulate the propagation of light through virtual environments to create realistic images.

Despite these efforts, it has remained difficult or impossible to recreate much of the complex geometry and subtle lighting effects found in the real world. The modeling problem can potentially be bypassed by capturing the geometry and material properties of objects directly from the real world. This approach typically involves some combination of cameras, structured light, range finders, and mechanical sensing devices such as 3D digitizers. When successful, the results can be fed into a rendering program to create images of real objects and scenes. Unfortunately, these systems are still unable to completely capture small details in geometry and material properties. Existing rendering methods also continue to be limited in their capability to faithfully reproduce real world illumination, even if given accurate geometric models.

Quicktime VR [6] was one of the first systems to suggest that the traditional modeling/rendering process can be skipped. Instead, a series of captured environment maps allow a user to *look around* a scene from fixed points in space. One can also flip through different views of an object to create the illusion of a 3D model. Chen and Williams [7] and Werner et al [30] have investigated smooth interpolation between images by modeling the motion of pixels (i.e., the *optical flow*) as one moves from one camera position to another. In Plenoptic Modeling [19], McMillan and Bishop discuss finding the disparity of each pixel in stereo pairs of cylindrical images. Given the disparity (roughly equivalent to depth information), they can then move pixels to create images from new vantage points. Similar work using stereo pairs of planar images is discussed in [14].

This paper extends the work begun with Quicktime VR and Plenoptic Modeling by further developing the idea of capturing the complete flow of light in a region of the environment. Such a flow is described by a *plenoptic function*[1]. The plenoptic function is a five dimensional quantity describing the flow of light at every 3D spatial position (x, y, z) for every 2D direction (θ, ϕ) . In this paper, we discuss computational methods for capturing and representing a plenoptic function, and for using such a representation to render images of the environment from any arbitrary viewpoint.

Unlike Chen and Williams' view interpolation [7] and McMillan and Bishop's plenoptic modeling [19], our approach does not rely explicitly on any optical flow information. Such information is often difficult to obtain in practice, particularly in environments with complex visibility relationships or specular surfaces. We do, however, use approximate geometric information to improve the quality of the reconstruction at lower sampling densities. Previous flow based methods implicitly rely on diffuse surface reflectance, allowing them to use a pixel from a single image to represent the appearance of a single geometric location from a variety of viewpoints. In contrast, our approach regularly samples the full plenoptic function and thus makes no assumptions about reflectance properties.

If we consider only the subset of light leaving a bounded object (or equivalently entering a bounded empty region of space), the fact that radiance along any ray remains constant¹ allows us to reduce the domain of interest of the plenoptic function to four dimensions. This paper first discusses the representation of this 4D function which we call a Lumigraph. We then discuss a system for sampling the plenoptic function with an inexpensive hand-held camera, and "developing" the captured light into a Lumigraph. Finally this paper describes how to use texture mapping hardware to quickly reconstruct images from any viewpoint with a virtual camera model. The Lumigraph representation is applicable to synthetic objects as well, allowing us to encode the complete appearance of a complex model and to re-render the object at speeds independent of the model complexity. We provide results on synthetic and real sequences and discuss work that is currently underway to make the system more efficient.

*Work performed while visiting Microsoft Research.

¹ We are assuming the medium (i.e., the air) to be transparent.

2 Representation

2.1 From 5D to 4D

The plenoptic function is a function of 5 variables representing position and direction². If we assume the air to be transparent then the radiance along a ray through empty space remains constant. If we furthermore limit our interest to the light leaving the convex hull of a bounded object, then we only need to represent the value of the plenoptic function along some surface that surrounds the object. A cube was chosen for its computational simplicity (see Figure 1). At any point in space, one can determine the radiance along any ray in any direction, by tracing backwards along that ray through empty space to the surface of the cube. Thus, the plenoptic function due to the object can be reduced to 4 dimensions³.

The idea of restricting the plenoptic function to some surrounding surface has been used before. In full-parallax holographic stereograms [3], the appearance of an object is captured by moving a camera along some surface (usually a plane) capturing a 2D array of photographs. This array is then transferred to a single holographic image, which can display the appearance of the 3D object. The work reported in this paper takes many of its concepts from holographic stereograms.

Global illumination researchers have used the "surface restricted plenoptic function" to efficiently simulate light-transfer between regions of an environment containing complicated geometric objects. The plenoptic function is represented on the surface of a cube surrounding some region; that information is all that is needed to simulate the light transfer from that region of space to all other regions [17]. In the context of illumination engineering, this idea has been used to model and represent the illumination due to physical luminaires. Ashdown [2] describes a gantry for moving a camera along a sphere surrounding a luminaire of interest. The captured information can then be used to represent the light source in global illumination simulations. Ashdown traces this idea of the surface-restricted plenoptic function back to Levin [15].

A limited version of the work reported here has been described by Katayama et al. [11]. In their system, a camera is moved along a track, capturing a 1D array of images of some object. This information is then used to generate new images of the object from other points in space. Because they only capture the plenoptic function along a line, they only obtain horizontal parallax, and distortion is introduced as soon as the new virtual camera leaves the line. Finally, in work concurrent to our own, Levoy and Hanrahan [16] represent a 4D function that allows for undistorted, full parallax views of the object from anywhere in space.

2.2 Parameterization of the 4D Lumigraph

There are many potential ways to parameterize the four dimensions of the Lumigraph. We adopt a parameterization similar to that used in digital holographic stereograms [9] and also used by Levoy and Hanrahan [16]. We begin with a cube to organize a Lumigraph and, without loss of generality, only consider for discussion a single square face of the cube (the full Lumigraph is constructed from six such faces).

²We only consider a snapshot of the function, thus time is eliminated. Without loss of generality, we also consider only a monochromatic function (in practice 3 discrete color channels), eliminating the need to consider wavelength. We furthermore ignore issues of dynamic range and thus limit ourselves to scalar values lying in some finite range.

³In an analogous fashion one can reconstruct the complete plenoptic function inside an empty convex region by representing it only on the surface bounding the empty region. At any point inside the region, one can find the light entering from any direction by finding that direction's intersection with the region boundary.

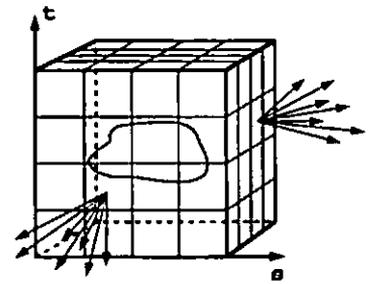


Figure 1: The surface of a cube holds all the radiance information due to the enclosed object.

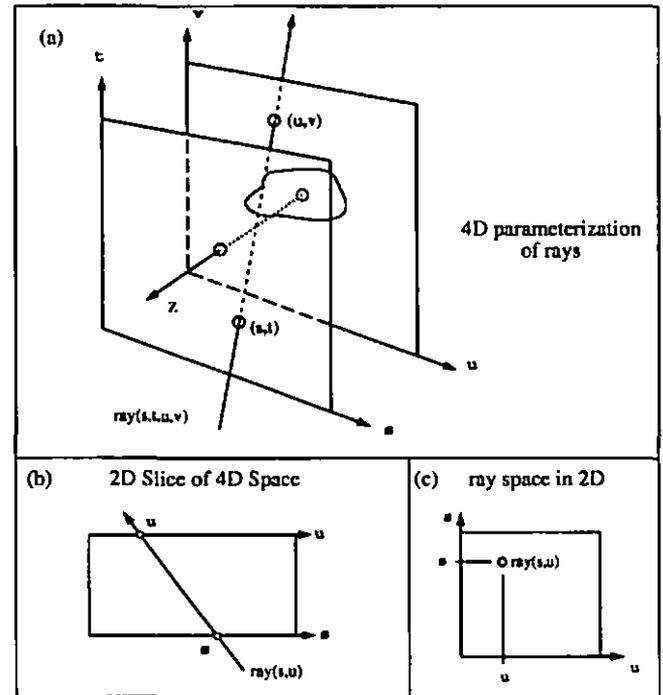


Figure 2: Parameterization of the Lumigraph

We choose a simple parameterization of the cube face with orthogonal axes running parallel to the sides labeled s and t (see Figure 1). Direction is parameterized using a second plane parallel to the st plane with axes labeled u and v (Figure 2). Any point in the 4D Lumigraph is thus identified by its four coordinates (s, t, u, v) , the coordinates of a ray piercing the first plane at (s, t) and intersecting the second plane at (u, v) (see Ray (s, t, u, v) in Figure 2). We place the origin at the center of the uv plane, with the z axis normal to the plane. The st plane is located at $z = 1$. The full Lumigraph consists of six such pairs of planes with normals along the x , $-x$, y , $-y$, z , and $-z$ directions.

It will be instructive at times to consider two 2D analogs to the 4D Lumigraph. Figure 2(b) shows a 2D slice of the 4D Lumigraph that indicates the u and s axes. Figure 2(c) shows the same arrangement in 2D ray coordinates in which rays are mapped to points (e.g., ray (s, u)) and points are mapped to lines.⁴

Figure 3 shows the relationship between this parameterization of the Lumigraph and a pixel in some arbitrary image. Given a Lu-

⁴More precisely, a line in ray space represents the set of rays through a point in space.

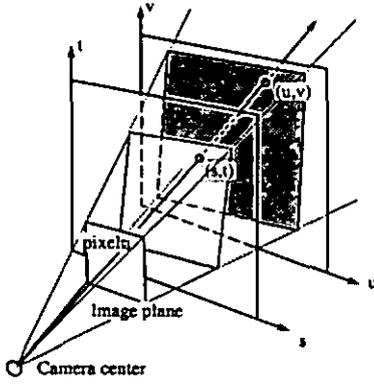


Figure 3: Relationship between Lumigraph and a pixel in an arbitrary image

migraph, L , one can generate an arbitrary new image coloring each pixel with the appropriate value $L(s, t, u, v)$. Conversely given some arbitrary image and the position and orientation of the camera, each pixel can be considered a sample of the Lumigraph value at (s, t, u, v) to be used to construct the Lumigraph.

There are many advantages of the two parallel plane parameterization. Given the geometric description of a ray, it is computationally simple to compute its coordinates; one merely finds its intersection with two planes. Moreover, reconstruction from this parameterization can be done rapidly using the texture mapping operations built into hardware on modern workstations (see section 3.6.2). Finally, in this parameterization, as one moves an eyepoint along the st plane in a straight line, the projection on the uv plane of points on the geometric object track along parallel straight lines. This makes it computationally efficient to compute the apparent motion of a geometric point (i.e., the *optical flow*), and to apply depth correction to the Lumigraph.

2.3 Discretization of the 4D Parameterization

So far, the Lumigraph has been discussed as an unknown, continuous, four dimensional function within a hypercubical domain in s, t, u, v and scalar range. To map such an object into a computational framework requires a discrete representation. In other words, we must choose some finite dimensional function space within which the function resides. To do so, we choose a discrete subdivision in each of the (s, t, u, v) dimensions and associate a coefficient and a basis function (reconstruction kernel) with each 4D grid point.

Choosing M subdivisions in the s and t dimensions and N subdivisions in u and v results in a grid of points on the st and uv planes (Figure 4). An st grid point is indexed with (i, j) and is located at (s_i, t_j) . A uv grid point is indexed with (p, q) and is located at (u_p, v_q) . A 4D grid point is indexed (i, j, p, q) . The data value (in fact an RGB triple) at this grid point is referred to as $x_{i,j,p,q}$.

2.3.1 Choice of Basis

We associate with each grid point a basis function $B_{i,j,p,q}$ so that the continuous Lumigraph is reconstructed as the linear sum

$$\tilde{L}(s, t, u, v) = \sum_{i=0}^M \sum_{j=0}^M \sum_{p=0}^N \sum_{q=0}^N x_{i,j,p,q} B_{i,j,p,q}(s, t, u, v)$$

where \tilde{L} is a finite dimensional Lumigraph that exists in the space defined by the choice of basis.

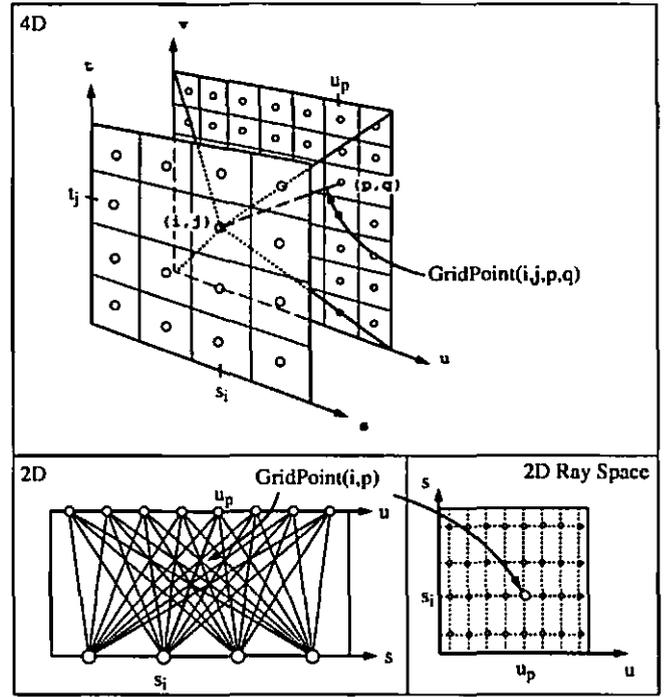


Figure 4: Discretization of the Lumigraph

For example, if we select constant basis functions (i.e., a 4D box with value 1 in the 4D region closest to the associated grid point and zero elsewhere), then the Lumigraph is piecewise constant, and takes on the value of the coefficient of the nearest grid point.

Similarly, a quadrilinear basis function has a value of 1 at the grid point and drops off to 0 at all neighboring grid points. The value of $\tilde{L}(s, t, u, v)$ is thus interpolated from the 16 grid points forming the hypercube in which the point resides.

We have chosen to use the quadrilinear basis for its computational simplicity and the C^0 continuity it imposes on \tilde{L} . However, because this basis is not band limited by the Nyquist frequency, and thus the corresponding finite dimensional function space is not shift invariant [24], the grid structure will be slightly noticeable in our results.

2.3.2 Projection into the Chosen Basis

Given a continuous Lumigraph, L , and a choice of basis for the finite dimensional Lumigraph, \tilde{L} , we still need to define a *projection* of L into \tilde{L} (i.e., we need to find the coefficients x that result in an \tilde{L} which is by some metric *closest* to L). If we choose the L^2 distance metric, then the projection is defined by integrating L against the *duals* of the basis functions [8], given by the inner products,

$$x_{i,j,p,q} = \langle L, \tilde{B}_{i,j,p,q} \rangle \quad (1)$$

In the case of the box basis, $B = \tilde{B}$. The duals of the quadrilinear basis functions are more complex, but these basis functions sufficiently approximate their own duals for our purposes.

One can interpret this projection as point sampling L after it has been low pass filtered with the kernel \tilde{B} . This interpretation is pursued in the context of holographic stereograms by Halle [9]. One can also interpret this projection as the result of placing a physical or synthetic "skewed" camera at grid point (s_i, t_j) with an aperture corresponding to the bilinear basis and with a pixel centered at

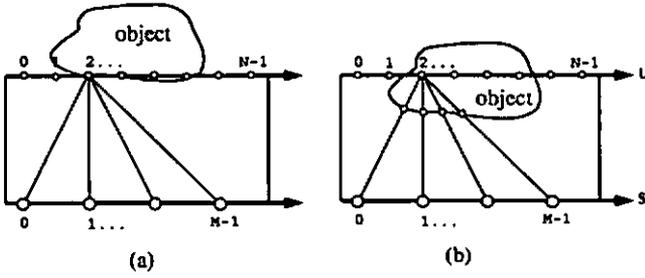


Figure 5: Choice of resolution on the uv plane

(u_p, v_q) antialiased with a bilinear filter. This analogy is pursued in [16].

In Figure 16 we show images generated from Lumigraphs. The geometric scene consisted of a partial cube with the pink face in front, yellow face in back, and the brown face on the floor. These Lumigraphs were generated using two different quadrature methods to approximate equation 1, and using two different sets of basis functions, constant and quadrilinear. In (a) and (c) only one sample was used to compute each Lumigraph coefficient. In these examples severe ghosting artifacts can be seen. In (b) and (d) numerical integration over the support of \hat{B} in st was computed for each coefficient. It is clear that best results are obtained using quadrilinear basis function, with a full quadrature method.

2.3.3 Resolution

An important decision is how to set the resolutions, M and N , that best balance efficiency and the quality of the images reconstructed from the Lumigraph. The choices for M and N are influenced by the fact that we expect the visible surfaces of the object to lie closer to the uv plane than the st plane. In this case, N , the resolution of the uv plane, is closely related to the final image resolution and thus a choice for N close to final image resolution works best (we consider a range of resolutions from 128 to 512).

One can gain some intuition for the choice of M by observing the 2D subset of the Lumigraph from a single grid point on the uv plane (see $u = 2$ in Figure 5(a)). If the surface of the object lies exactly on the uv plane at a gridpoint, then all rays leaving that point represent samples of the radiance function at a single position on the object's surface. Even when the object's surface deviates from the uv plane as in Figure 5(b), we can still expect the function across the st plane to remain smooth and thus a low resolution is sufficient. Thus a significantly lower resolution for M than N can be expected to yield good results. In our implementation we use values of M ranging from 16 to 64.

2.3.4 Use of Geometric Information

Assuming the radiance function of the object is well behaved, knowledge about the geometry of the object gives us information about the coherence of the associated Lumigraph function, and can be used to help define the shape of our basis functions.

Consider the ray (s, u) in a two-dimensional Lumigraph (Figure 6). The closest grid point to this ray is (s_{i+1}, u_p) . However, gridpoints (s_{i+1}, u_{p-1}) and (s_i, u_{p+1}) are likely to contain values closer to the true value at (s, u) since these grid points represent rays that intersect the object nearby the intersection with (s, u) . This suggests adapting the shape of the basis functions.

Suppose we know the depth value z at which ray (s, u) first intersects a surface of the object. Then for a given s_i , one can compute a corresponding u' for a ray (s_i, u') that intersects the same geomet-

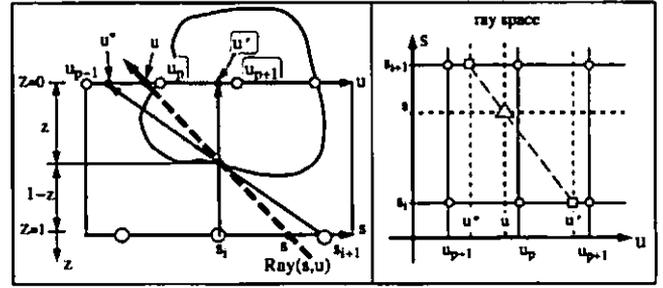


Figure 6: Depth correction of rays

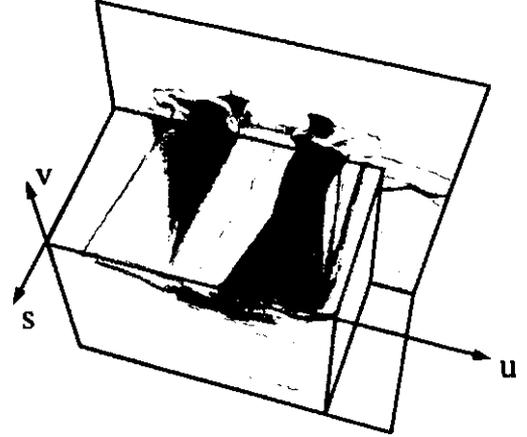


Figure 7: An (s, u, v) slice of a Lumigraph

ric location on the object as the original ray (s, u) ⁵. Let the depth z be 0 at the uv plane and 1 at the st plane. The intersections can then be found by examining the similar triangles in Figure 6,

$$u' = u + (s - s_i) \frac{u - u_p}{1 - z} \quad (2)$$

It is instructive to view the same situation as in Figure 6(a), plotted in *ray space* (Figure 6(b)). In this figure, the triangle is the ray (s, u) , and the circles indicate the nearby gridpoints in the discrete Lumigraph. The diagonal line passing through (s, u) indicates the *optical flow* (in this case, horizontal motion in 2D) of the intersection point on the object as one moves back and forth in s . The intersection of this line with s_i and s_{i+1} occurs at u' and u'' respectively.

Figure 7 shows an (s, u) slice through a three-dimensional (s, u, v) subspace of the Lumigraph for the ray-traced fruitbowl used in Figure 19. The flow of pixel motion is along straight lines in this space, but more than one motion may be present if the scene includes transparency. The slope of the flow lines corresponds to the depth of the point on the object tracing out the line. Notice how the function is coherent along these flow lines [4].

We expect the Lumigraph to be smooth along the optical flow lines, and thus it would be beneficial to have the basis functions adapt their shape correspondingly. The remapping of u and v values to u' and v' performs this reshaping. The idea of shaping the support of basis functions to closely match the structure of the function being approximated is used extensively in finite element methods. For example, in the Radiosity method for image synthesis, the mesh of elements is adapted to fit knowledge about the illumination function.

⁵ Assuming there has been no change in visibility.

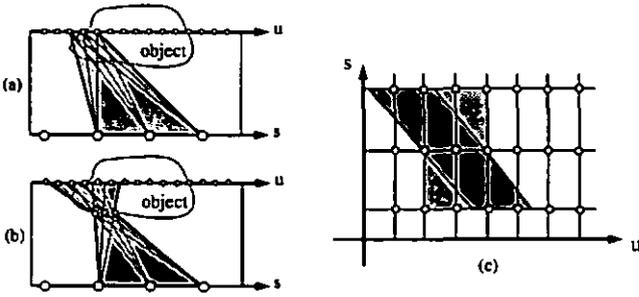


Figure 8: (a) Support of an uncorrected basis function. (b) Support of a depth corrected basis function. (c) Support of both basis functions in ray space.

The new basis function $B'_{i,j,p,q}(s, t, u, v)$ is defined by first finding u' and v' using equation 2 and then evaluating B , that is

$$B'_{i,j,p,q}(s, t, u, v) = B_{i,j,p,q}(s, t, u', v')$$

Although the shape of the new *depth corrected* basis is complicated, $\tilde{L}(s, t, u, v)$ is still a linear sum of coefficients and the weights of the contributing basis functions still sum to unity. However, the basis is no longer representable as a tensor product of simple boxes or hats as before. Figure 8 shows the support of an uncorrected (light gray) and a depth corrected (dark gray) basis function in 2D geometric space and in 2D ray space. Notice how the support of the depth corrected basis intersects the surface of the object across a narrower area compared to the uncorrected basis.

We use depth corrected quadrilinear basis functions in our system. The value of $\tilde{L}(s, t, u, v)$ in the corrected quadrilinear basis is computed using the following calculation:

```

QuadrilinearDepthCorrect(s,t,u,v,z)
Result = 0
hst = s1 - s0 /* grid spacing */
huv = u1 - u0
for each of the four (si, tj) surrounding (s, t)
    u' = u + (s - si) * z / (1 - z)
    v' = v + (t - tj) * z / (1 - z)
    temp = 0
    for each of the four (up, vq) surrounding (u', v')
        interpWeightuv =
            (huv - |up - u'|) * (huv - |vq - v'|) / huv2
        temp += interpWeightuv * L(si, tj, up, vq)
    interpWeightst =
        (hst - |si - s|) * (hst - |tj - t|) / hst2
    Result += interpWeightst * temp
return Result

```

Figure 17 shows images generated from a Lumigraph using uncorrected and depth corrected basis functions. The depth correction was done using a 162 polygon model to approximate the original 70,000 polygons. The approximation was generated using a mesh simplification program [10]. These images show how depth correction reduces the artifacts present in the images.

3 The Lumigraph System

This section discusses many of the practical implementation issues related to creating a Lumigraph and generating images from it. Figure 9 shows a block diagram of the system. The process begins with capturing images with a hand-held camera. From known markers

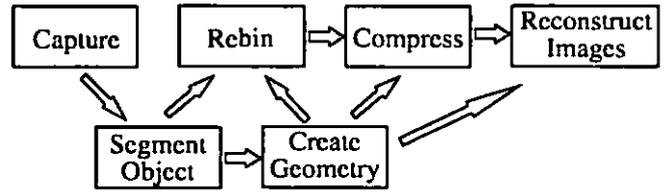


Figure 9: The Lumigraph system

in the image, the camera's position and orientation (its *pose*) is estimated. This provides enough information to create an approximate geometric object for use in the depth correction of (u, v) values. More importantly, each pixel in each image acts as a sample of the plenoptic function and is used to estimate the coefficients of the discrete Lumigraph (i.e., to *develop* the Lumigraph). Alternatively, the Lumigraph of a synthetic object can be generated directly by integrating a set of rays cast in a rendering system. We only briefly touch on compression issues. Finally, given an arbitrary virtual camera, new images of the object are quickly rendered.

3.1 Capture for Synthetic Scenes

Creating a Lumigraph of a synthetic scene is straightforward. A single sample per Lumigraph coefficient can be captured for each gridpoint (i, j) by placing the center of a virtual pin hole camera at (s_i, t_j) looking down the z axis, and defining the imaging frustum using the uv square as the film location. Rendering an image using this skewed perspective camera produces the Lumigraph coefficients. The pixel values in this image, indexed (p, q) , are used as the Lumigraph coefficients $x_{i,j,p,q}$. To perform the integration against the kernel \tilde{B} , multiple rays per coefficient can be averaged by jittering the camera and pixel locations, weighting each image using \tilde{B} . For ray traced renderings, we have used the ray tracing program provided with the Generative Modeling package[25].

3.2 Capture for Real Scenes

Computing the Lumigraph for a real object requires the acquisition of object images from a large number of viewpoints. One way in which this can be accomplished is to use a special motion control platform to place the real camera at positions and orientations coincident with the (s_i, t_j) gridpoints [16]. While this is a reasonable solution, we are interested in acquiring the images with a regular hand-held camera. This results in a simpler and cheaper system, and may extend the range of applicability to larger scenes and objects.

To achieve this goal, we must first calibrate the camera to determine the mapping between directions and image coordinates. Next, we must identify special calibration markers in each image and compute the camera's pose from these markers. To enable depth-corrected interpolation of the Lumigraph, we also wish to recover a rough geometric model of the object. To do this, we convert each input image into a silhouette using a blue-screen technique, and then build a volumetric model from these binary images.

3.2.1 Camera Calibration and Pose Estimation

Camera calibration and pose estimation can be thought of as two parts of a single process: determining a mapping between screen pixels and rays in the world. The parameters associated with this process naturally divide into two sets: extrinsic parameters, which define the camera's pose (a rigid rotation and translation), and intrinsic parameters, which define a mapping of 3D camera coordinates onto the screen. This latter mapping not only includes a perspective (pinhole) projection from the 3D coordinates to undistorted

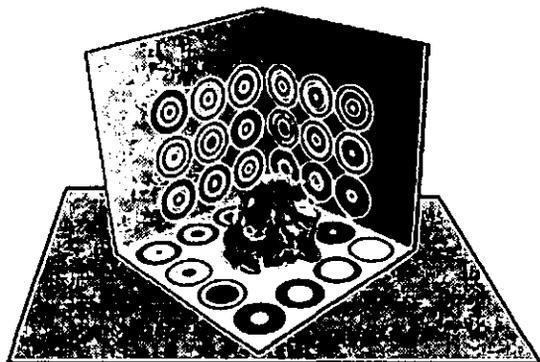


Figure 10: The capture stage

image coordinates, but also a radial distortion transformation and a final translation and scaling into screen coordinates [29, 31].

We use a camera with a fixed lens, thus the intrinsic parameters remain constant throughout the process and need to be estimated only once, before the data acquisition begins. Extrinsic parameters, however, change constantly and need to be recomputed for each new video frame. Fortunately, given the intrinsic parameters, this can be done efficiently and accurately with many fewer calibration points. To compute the intrinsic and extrinsic parameters, we employ an algorithm originally developed by Tsai [29] and extended by Willson [31].

A specially designed stage provides the source of calibration data (see Figure 10). The stage has two walls fixed together at a right angle and a base that can be detached from the walls and rotated in 90 degree increments. An object placed on such a movable base can be viewed from all directions in the upper hemisphere. The stage background is painted cyan for later blue-screen processing. Thirty markers, each of which consists of several concentric rings in a darker shade of cyan, are distributed along the sides and base. This number is sufficiently high to allow for a very precise intrinsic camera calibration. During the extrinsic camera calibration, only 8 or more markers need be visible to reliably compute a pose.

Locating markers in each image is accomplished by first converting the image into a binary (i.e., black or white) image. A *double thresholding* operator divides all image pixels into three groups separated by intensity thresholds T_1 and T_2 . Pixels with an intensity below T_1 are considered black, pixels with an intensity above T_2 are considered white. Pixels with an intensity between T_1 and T_2 are considered black only if they have a black neighbor, otherwise they are considered white. The binary thresholded image is then searched for *connected components* [23]. Sets of connected components with similar centers of gravity are the likely candidates for the markers. Finally, the ratio of radii in each marker is used to uniquely identify the marker. To help the user correctly sample the viewing space, a real-time visual feedback displays the current and past locations of the camera in the view space (Figure 11). Marker tracking, pose estimation, feedback display, and frame recording takes approximately 1/2 second per frame on an SGI Indy.

3.3 3D Shape Approximation

The recovery of 3D shape information from natural imagery has long been a focus of computer vision research. Many of these techniques assume a particularly simple shape model, for example, a polyhedral scene where all edges are visible. Other techniques, such as stereo matching, produce sparse or incomplete depth estimates. To produce complete, closed 3D models, several approaches have been tried. One family of techniques builds 3D volumetric models

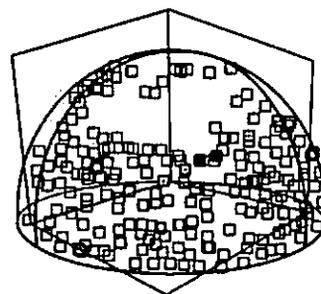


Figure 11: The user interface for the image capture stage displays the current and previous camera positions on a viewing sphere. The goal of the user is to "paint" the sphere.



Figure 12: Segmented image plus volume construction

directly from silhouettes of the object being viewed [21]. Another approach is to fit a deformable 3D model to sparse stereo data. Despite over 20 years of research, the reliable extraction of accurate 3D geometric information from imagery (without the use of active illumination and positioning hardware) remains elusive.

Fortunately, a rough estimate of the shape of the object is enough to greatly aid in the capture and reconstruction of images from a Lumigraph. We employ the octree construction algorithm described in [26] for this process. Each input image is first segmented into a binary object/background image using a blue-screen technique [12] (Figure 12). An octree representation of a cube that completely encloses the object is initialized. Then for each segmented image, each voxel at a coarse level of the octree is projected onto the image plane and tested against the silhouette of the object. If a voxel falls outside of the silhouette, it is removed from the tree. If it falls on the boundary, it is marked for subdivision into eight smaller cubes. After a small number of images are processed, all marked cubes subdivide. The algorithm proceeds for a preset number of subdivisions, typically 4. The resulting 3D model consists of a collection of voxels describing a volume which is known to contain the object⁶ (Figure 12). The external polygons are collected and the resulting polyhedron is then smoothed using Taubin's polyhedral smoothing algorithm [27].

3.4 Rebinning

As described in Equation 1, the coefficient associated with the basis function $B_{i,j,p,q}$ is defined as the integral of the continuous Lumigraph function multiplied by some kernel function \tilde{B} . This can be written as

$$x_{i,j,p,q} = \int L(s, t, u, v) \tilde{B}_{i,j,p,q}(s, t, u, v) ds dt du dv \quad (3)$$

In practice this integral must be evaluated using a finite number of samples of the function L . Each pixel in the input video stream coming from the hand-held camera represents a single sample

⁶Technically, the volume is a superset of the *visual hull* of the object [13].

$L(s_k, t_k, u_k, v_k)$, of the Lumigraph function. As a result, the sample points in the domain cannot be pre-specified or controlled. In addition, there is no guarantee that the incoming samples are evenly spaced.

Constructing a Lumigraph from these samples is similar to the problem of multidimensional scattered data approximation. In the Lumigraph setting, the problem is difficult for many reasons. Because the samples are not evenly spaced, one cannot apply standard Fourier-based sampling theory. Because the number of sample points may be large ($\approx 10^8$) and because we are working in a 4 dimensional space, it is too expensive to solve systems of equations (as is done when solving thin-plate problems [28, 18]) or to build spatial data structures (such as Delauny triangulations).

In addition to the number of sample points, the distribution of the data samples have two qualities that make the problem particularly difficult. First, the sampling density can be quite sparse, with large gaps in many regions. Second, the sampling density is typically very non-uniform.

The first of these problems has been addressed in a two dimensional scattered data approximation algorithm described by Burt [5]. In his algorithm, a hierarchical set of lower resolution data sets is created using an image pyramid. Each of these lower resolutions represents a "blurred" version of the input data; at lower resolutions, the gaps in the data become smaller. This low resolution data is then used to fill in the gaps at higher resolutions.

The second of these problems, the non-uniformity of the sampling density, has been addressed by Mitchell [20]. He solves the problem of obtaining the value of a pixel that has been super-sampled with a non-uniform density. In this problem, when averaging the sample values, one does not want the result to be overly influenced by the regions sampled most densely. His algorithm avoids this by computing average values in a number of smaller regions. The final value of the pixel is then computed by averaging together the values of these strata. This average is not weighted by the number of samples falling in each of the strata. Thus, the non-uniformity of the samples does not bias the answer.

For our problem, we have developed a new hierarchical algorithm that combines concepts from both of these algorithms. Like Burt, our method uses a pyramid algorithm to fill in gaps, and like Mitchell, we ensure that the non-uniformity of the data does not bias the "blurring" step.

For ease of notation, the algorithm is described in 1D, and will use only one index i . A hierarchical set of basis functions is used, with the highest resolution labeled 0 and with lower resolutions having higher indices. Associated with each coefficient x_i^r at resolution r is a weight w_i^r . These weights determine how the coefficients at different resolution levels are eventually combined. The use of these weights is the distinguishing feature of our algorithm.

The algorithm proceeds in three phases. In the first phase, called *splat*, the sample data is used to approximate the integral of Equation 3, obtaining coefficients x_i^0 and weights w_i^0 . In regions where there is little or no nearby sample data, the weights are small or zero. In the second phase, called *pull*, coefficients are computed for basis functions at a hierarchical set of lower resolution grids by combining the coefficient values from the higher resolution grids. In the lower resolution grids, the gaps (regions where the weights are low) become smaller (see figure 13). In the third phase, called *push*, information from the each lower resolution grid is combined with the next higher resolution grid, filling in the gaps while not unduly blurring the higher resolution information already computed.

3.4.1 Splatting

In the splatting phase, coefficients are computed by performing Monte-Carlo integration using the following weighted average es-

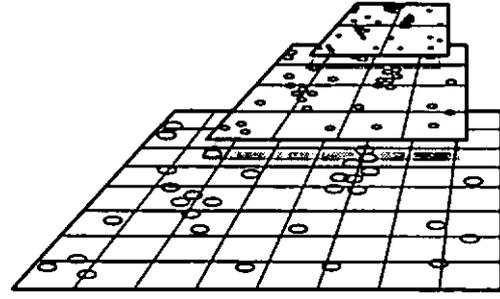


Figure 13: 2D pull-push. At lower resolutions the gaps are smaller.

imator:

$$\begin{aligned} w_i^0 &= \sum_k \tilde{B}_i(s_k) \\ x_i^0 &= \frac{1}{w_i^0} \sum_k \tilde{B}_i(s_k) L(s_k) \end{aligned} \quad (4)$$

where s_k denotes the domain location of sample k . If w_i^0 is 0, then the x_i^0 is undefined. If the \tilde{B}_i have compact support, then each sample influences only a constant number of coefficients. Therefore, this step runs in time linear in the number of samples.

If the sample points s_k are chosen from a uniform distribution, this estimator converges to the correct value of the integral in Equation (3), and for n sample points has a variance of approximately $\frac{1}{n} \int (\tilde{B}_i(s) L(s) - x_i \tilde{B}_i(s))^2 ds$. This variance is similar to that obtained using importance sampling, which is often much smaller than the crude Monte Carlo estimator. For a full analysis of this estimator, see [22].

3.4.2 Pull

In the *pull* phase, lower resolution approximations of the function are derived using a set of wider kernels. These wider kernels are defined by linearly summing together the higher resolution kernels ($\tilde{B}_i^{r+1} = \sum_k \tilde{h}_{k-2i} \tilde{B}_k^r$) using some discrete sequence \tilde{h} . For linear "hat" functions, $\tilde{h}[-1..1]$ is $\{\frac{1}{2}, 1, \frac{1}{2}\}$

The lower resolution coefficients are computed by combining the higher resolution coefficients using \tilde{h} . One way to do this would be to compute

$$\begin{aligned} w_i^{r+1} &= \sum_k \tilde{h}_{k-2i} w_k^r \\ x_i^{r+1} &= \frac{1}{w_i^{r+1}} \sum_k \tilde{h}_{k-2i} w_k^r x_k^r \end{aligned} \quad (5)$$

It is easy to see that this formula, which corresponds to the method used by Burt, computes the same result as would the original estimator (Equation (4)) applied to the wider kernels. Once again, this estimator works if the sampling density is uniform. Unfortunately, when looking on a gross scale, it is imprudent to assume that the data is sampled uniformly. For example, the user may have held the camera in some particular region for a long time. This non-uniformity can greatly bias the estimator.

Our solution to this problem is to apply Mitchell's reasoning to this context, replacing Equation (5) with:

$$\begin{aligned} w_i^{r+1} &= \sum_k \tilde{h}_{k-2i} \min(w_k^r, 1) \\ x_i^{r+1} &= \frac{1}{w_i^{r+1}} \sum_k \tilde{h}_{k-2i} \min(w_k^r, 1) x_k^r \end{aligned}$$

The value 1 represents full saturation⁷, and the min operator is used to place an upper bound on the degree that one coefficient in a highly

⁷Using the value 1 introduces no loss of generality if the normalization of \tilde{h} is not fixed.

sampled region, can influence the total sum ⁸.

The pull stage runs in time linear in the number of basis function summed over all of the resolutions. Because each lower resolution has half the density of basis functions, this stage runs in time linear in the number of basis functions at resolution 0.

3.4.3 Push

During the push stage, the lower resolution approximation is used to fill in the regions in the higher resolution that have low weight ⁹. If a higher resolution coefficient has a high associated confidence (i.e., has weight greater than one), we fully disregard the lower resolution information there. If the higher resolution coefficient does not have sufficient weight, we blend in the information from the lower resolution.

To blend this information, the low resolution approximation of the function must be expressed in the higher resolution basis. This is done by upsampling and convolving with a sequence h , that satisfies $B_i^{r+1} = \sum_k h_{k-2i} B_k^r$.

We first compute temporary values

$$\begin{aligned} t w_i^r &= \sum_k h_{i-2k} \min(w_k^{r+1}, 1) \\ t x_i^r &= \frac{1}{t w_i^r} \sum_k h_{i-2k} \min(w_k^{r+1}, 1) x_k^{r+1} \end{aligned}$$

These temporary values are now ready to be blended with the values x and w values already at level r .

$$\begin{aligned} x_i^r &= t x_i^r (1 - w_i^r) + w_i^r x_i^r \\ w_i^r &= t w_i^r (1 - w_i^r) + w_i^r \end{aligned}$$

This is analogous to the blending performed in image compositing.

3.4.4 Use of Geometric Information

This three phase algorithm must be adapted slightly when using the depth corrected basis functions B' . During the splat phase, each sample ray $L(s_k, t_k, u_k, v_k)$ must have its u and v values remapped as explained in Section 2.3.4. Also, during the push and pull phases, instead of simply combining coefficients using basis functions with neighboring indices, depth corrected indices are used.

3.4.5 2D Results

The validity of the algorithm was tested by first applying it to a 2D image. Figure 18 (a) shows a set of scattered samples from the well known mandrill image. The samples were chosen by picking 256 random line segments and sampling the mandrill very densely along these lines ¹⁰. Image (b) shows the resulting image after the *pull/push* algorithm has been applied. Image (c) and (d) show the same process but with only 100 sample lines. The success of our algorithm on both 2D image functions and 4D Lumigraph functions leads us to believe that it may have many other uses.

3.5 Compression

A straightforward sampling of the Lumigraph requires a large amount of storage. For the examples shown in section 4, we use, for a single face, a 32×32 sampling in (s, t) space and 256×256

⁸This is actually less extreme than Mitchell's original algorithm. In this context, his algorithm would set all non-zero weights to 1.

⁹Variance measures could be used instead of weight as a measure of confidence in this phase.

¹⁰We chose this type of sampling pattern because it mimics in many ways the structure of the Lumigraph samples taken from a hand-held camera. In that case each input video image is a dense sampling of the 4D Lumigraph along a 2D plane.

(u, v) images. To store the six faces of our viewing cube with 24-bits per pixel requires $32^2 \cdot 256^2 \cdot 6 \cdot 3 = 1.125\text{GB}$ of storage.

Fortunately, there is a large amount of coherence between (s, t, u, v) samples. One could apply a transform code to the 4D array, such as a wavelet transform or block DCT. Given geometric information, we can expect to do even better by considering the 4D array as a 2D array of images. We can then *predict* new (u, v) images from adjacent images, (i.e., images at adjacent (s, t) locations). Intraframe compression issues are identical to compressing single images (a simple JPEG compression yields about a 20:1 savings). Interframe compression can take advantage of increased information over other compression methods such as MPEG. Since we know that the object is static and know the camera motion between adjacent images, we can predict the motion of pixels. In addition, we can leverage the fact that we have a 2D array of images rather than a single linear video stream.

Although we have not completed a full analysis of compression issues, our preliminary experiments suggest that a 200:1 compression ratio should be achievable with almost no degradation. This reduces the storage requirements to under 6MB. Obviously, further improvements can be expected using a more sophisticated prediction and encoding scheme.

3.6 Reconstruction of Images

Given a desired camera (position, orientation, resolution), the reconstruction phase colors each pixel of the output image with the color that this camera would create if it were pointed at the real object.

3.6.1 Ray Tracing

Given a Lumigraph, one may generate a new image from an arbitrary camera pixel by pixel, ray by ray. For each ray, the corresponding (s, t, u, v) coordinates are computed, the nearby grid points are located, and their values are properly interpolated using the chosen basis functions (see Figure 3).

In order to use the depth corrected basis functions given an approximate object, we transform the (u, v) coordinates to the depth corrected (u', v') before interpolation. This depth correction of the (u, v) values can be carried out with the aid of graphics hardware. The polygonal approximation of the object is drawn from the point of view and with the same resolution as the desired image. Each vertex is assigned a *red, green, blue* value corresponding to its (x, y, z) coordinate resulting in a "depth" image. The corrected depth value is found by examining the blue value in the corresponding pixel of the depth image for the $\pm z$ -faces of the Lumigraph cube (or the red or green values for other faces). This information is used to find u' and v' with Equation 2.

3.6.2 Texture mapping

The expense of tracing a ray for each pixel can be avoided by reconstructing images using texture mapping operations. The st plane itself is tiled with texture mapped polygons with the textures defined by slices of the Lumigraph: $\text{tex}_{i,j}(u_p, v_q) = x_{i,j,p,q}$. In other words, we have one texture associated with each st gridpoint.

Constant Basis

Consider the case of constant basis functions. Suppose we wish to render an image from the desired camera shown in Figure 14. The set of rays passing through the shaded square on the st plane have (s, t) coordinates closest to the grid point (i, j) . Suppose that the uv plane is filled with $\text{tex}_{i,j}$. Then, when using constant basis functions, the shaded region in the desired camera's film plane should be filled with the corresponding pixels in the shaded region of the uv plane. This computation can be accomplished by placing a virtual camera at the desired location, drawing a square polygon on the

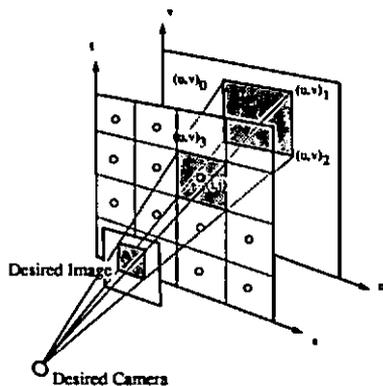


Figure 14: Texture mapping a portion of the st plane

st plane, and texture mapping it using the four texture coordinates $(u, v)_0$, $(u, v)_1$, $(u, v)_2$, and $(u, v)_3$ to index into $\text{tex}_{i,j}$.

Repeating this process for each grid point on the st plane and viewing the result from the desired camera results in a complete reconstruction of the desired image. Thus, if one has an $M \times M$ resolution for the st plane, one needs to draw at most M^2 texture mapped squares, requiring on average, only one ray intersection for each square since the vertices are shared. Since many of the M^2 squares on the st plane are invisible from the desired camera, typically only a small fraction of these squares need to be rendered. The rendering cost is independent of the resolution of the final image.

Intuitively, you can think of the st plane as a piece of holographic film. As your eye moves back and forth you see different things at the same point in st since each point holds a complete image.

Quadralinear Basis

The reconstruction of images from a quadralinear basis Lumigraph can also be performed using a combination of texture mapping and alpha blending. In the quadralinear basis, the support of the basis function at i, j covers a larger square on the st plane than does the box basis (see Figure 15(a)). Although the regions do not overlap in the constant basis, they do in the quadralinear basis. For a given pixel in the desired image, values from 16 4D grid points contribute to the final value.

The quadralinear interpolation of these 16 values can be carried out as a sequence of bilinear interpolations, first in uv and then in st . A bilinear basis function is shown in Figure 15(b) centered at grid point (i, j) . A similar basis would lie over each grid point in uv and every grid point in st .

Texture mapping hardware on an SGI workstation can automatically carry out the bilinear interpolation of the texture in uv . Unfortunately, there is no hardware support for the st bilinear interpolation. We could approximate the bilinear pyramid with a linear pyramid by drawing the four triangles shown on the floor of the basis function in Figure 15(b). By assigning α values to each vertex ($\alpha = 1$ at the center, and $\alpha = 0$ at the outer four vertices) and using alpha blending, the final image approximates the full quadralinear interpolation with a linear-bilinear one. Unfortunately, such a set of basis functions do not sum to unity which causes serious artifacts.

A different pyramid of triangles can be built that does sum to unity and thus avoids these artifacts. Figure 15(c) shows a hexagonal region associated with grid point (i, j) and an associated linear basis function. We draw the six triangles of the hexagon with $\alpha = 1$ at the center and $\alpha = 0$ at the outside six vertices¹¹. The linear interpolation of α values together with the bilinear interpolation of the texture map results in a linear-bilinear interpolation. In practice we have found it to be indistinguishable from the full quad-

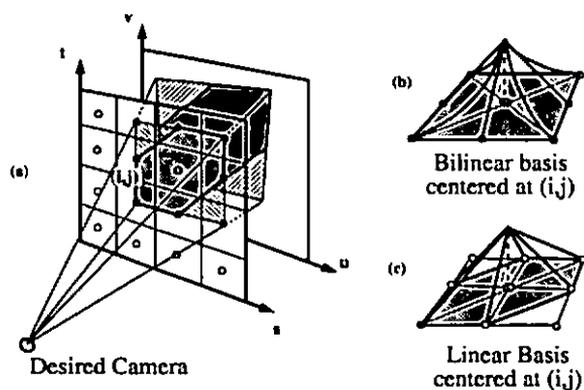


Figure 15: Quadralinear vs. linear-bilinear

ralinear interpolation. This process requires at most $6M^2$ texture mapped, α -blended triangles to be drawn.

Depth Correction

As before, the (u, v) coordinates of the vertices of the texture mapped triangles can be depth corrected. At interior pixels, the depth correction is only approximate. This is not valid when there are large depth changes within the bounds of the triangle. Therefore, we adaptively subdivide the triangles into four smaller ones by connecting the midpoints of the sides until they are (a) smaller than a minimum screen size or (b) have a sufficiently small variation in depth at the three corners and center. The α values at intermediate vertices are the average of the vertices of the parent triangles.

4 Results

We have implemented the complete system described in this paper and have created Lumigraphs of both synthetic and actual objects. For synthetic objects, Lumigraphs can be created either from polygon rendered or ray traced images. Computing all of the necessary images is a lengthy process often taking weeks of processing time.

For real objects, the capture is performed with an inexpensive, single chip Panasonic analog video camera. The capture phase takes less than one hour. The captured data is then "developed" into a Lumigraph. This off-line processing, which includes segmenting the image from its background, creating an approximate volumetric representation, and rebinning the samples, takes less than one day of processing on an SGI Indy workstation.

Once the Lumigraph has been created, arbitrary new images of the object or scene can be generated. One may generate these new images on a ray by ray basis, which takes a few seconds per frame at 450×450 resolution. If one has hardware texture mapping available, then one may use the acceleration algorithm described in Section 3.6.2. This texture mapping algorithm is able to create multiple frames per second from the Lumigraph on an SGI Reality Engine. The rendering speed is almost independent of the desired resolution of the output images. The computational bottleneck is moving the data from main memory to the smaller texture cache.

Figure 19 shows images of a synthetic fruit bowl, an actual fruit bowl, and a stuffed lion, generated from Lumigraphs. No geometric information was used in the Lumigraph of the synthetic fruit bowl. For the actual fruit bowl and the stuffed lion, we have used the approximate geometry that was computed using the silhouette information. These images can be generated in a fraction of a second, independent of scene complexity. The complexity of both the geometry and the lighting effects present in these images would be difficult to achieve using traditional computer graphics techniques.

¹¹ The alpha blending mode is set to perform a simple summation.

5 Conclusion

In this paper we have described a rendering framework based on the plenoptic function emanating from a static object or scene. Our method makes no assumptions about the reflective properties of the surfaces in the scene. Moreover, this representation does not require us to derive any geometric knowledge about the scene such as depth. However, this method does allow us to include any geometric knowledge we may compute, to improve the efficiency of the representation and improve the quality of the results. We compute the approximate geometry using silhouette information.

We have developed a system for capturing plenoptic data using a hand-held camera, and converting this data into a Lumigraph using a novel rebinning algorithm. Finally, we have developed an algorithm for generating new images from the Lumigraph quickly using the power of texture mapping hardware.

In the examples shown in this paper, we have not captured the complete plenoptic function surrounding an object. We have limited ourselves to only one face of a surrounding cube. There should be no conceptual obstacles to extending this work to complete captures using all six cube faces.

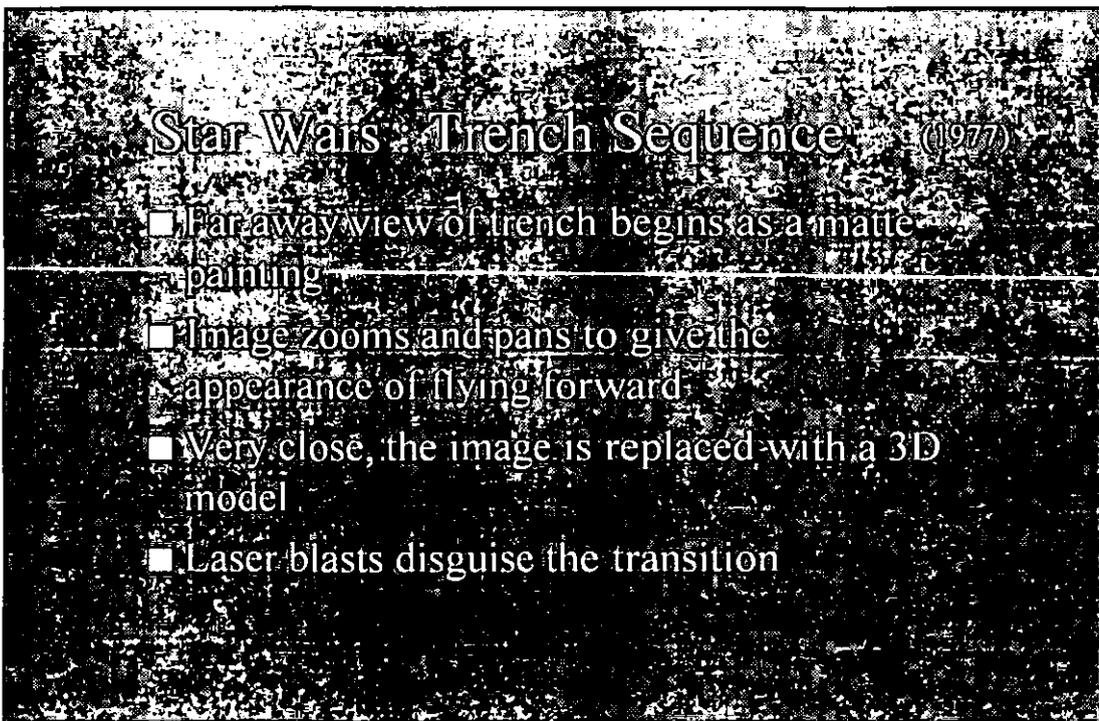
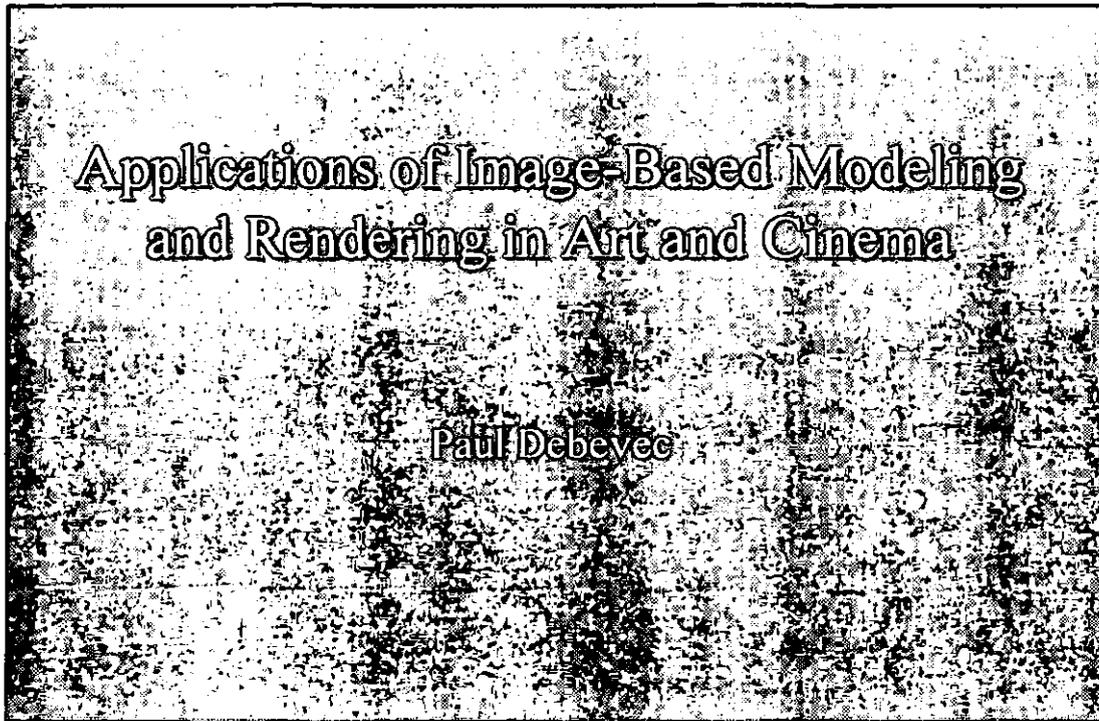
There is much future work to be done on this topic. It will be important to develop powerful compression methods so that Lumigraphs can be efficiently stored and transmitted. We believe that the large degree of coherence in the Lumigraph will make a high rate of compression achievable. Future research also includes improving the accuracy of our system to reduce the amount of artifacts in the images created by the Lumigraph. With these extensions we believe the Lumigraph will be an attractive alternative to traditional methods for efficiently storing and rendering realistic 3D objects and scenes.

Acknowledgments

The authors would like to acknowledge the help and advice we received in the conception, implementation and writing of this paper. Thanks to Marc Levoy and Pat Hanrahan for discussions on issues related to the 5D to 4D simplification, the two-plane parameterization and the camera based aperture analog. Jim Kajiya and Tony DeRose provided a terrific sounding board throughout this project. The ray tracer used for the synthetic fruit bowl was written by John Snyder. The mesh simplification code used for the bunny was written by Hugues Hoppe. Portions of the camera capture code were implemented by Matthew Turk. Jim Blinn, Hugues Hoppe, Andrew Glassner and Jutta Joesch provided excellent editing suggestions. Erynn Ryan is deeply thanked for her creative crisis management. Finally, we wish to thank the anonymous reviewers who pointed us toward a number of significant references we had missed.

References

- [1] ADLSON, E. H., AND BERGEN, J. R. The plenoptic function and the elements of early vision. In *Computational Models of Visual Processing*, Landy and Movshon, Eds. MIT Press, Cambridge, Massachusetts, 1991, ch. 1.
- [2] ASHDOWN, I. Near-field photometry: A new approach. *Journal of the Illumination Engineering Society* 22, 1 (1993), 163-180.
- [3] BENTON, S. A. Survey of holographic stereograms. *Proceedings of the SPIE* 391 (1982), 15-22.
- [4] BOLLES, R. C., BAKER, H. H., AND MARIMONT, D. H. Epipolar-plane image analysis: An approach to determining structure from motion. *International Journal of Computer Vision* 1 (1987), 7-55.
- [5] BURT, P. J. Moment images, polynomial fit filters, and the problem of surface interpolation. In *Proceedings of Computer Vision and Pattern Recognition* (June 1988), IEEE Computer Society Press, pp. 144-152.
- [6] CHEN, S. E. Quicktime VR - an image-based approach to virtual environment navigation. In *Computer Graphics, Annual Conference Series, 1995*, pp. 29-38.
- [7] CHEN, S. E., AND WILLIAMS, L. View interpolation for image synthesis. In *Computer Graphics, Annual Conference Series, 1993*, pp. 279-288.
- [8] CIHUI, C. K. *An Introduction to Wavelets*. Academic Press Inc., 1992.
- [9] HALLE, M. W. Holographic stereograms as discrete imaging systems. *Practical Holography VIII (SPIE) 2176* (1994), 73-84.
- [10] HOPPE, H. Progressive meshes. In *Computer Graphics, Annual Conference Series, 1996*.
- [11] KATAYAMA, A., TANAKA, K., OSHINO, T., AND TAMURA, H. A viewpoint independent stereoscopic display using interpolation of multi-viewpoint images. *Stereoscopic displays and virtual reality systems II (SPIE) 2409* (1995), 11-20.
- [12] KLINKER, G. J. *A Physical Approach to Color Image Understanding*. A K Peters, Wellesley, Massachusetts, 1993.
- [13] LAURENTINI, A. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16, 2 (February 1994), 150-162.
- [14] LAVBAU, S., AND FAUGERAS, O. 3-D scene representation as a collection of images and fundamental matrices. Tech. Rep. 2205, INRIA-Sophia Antipolis, February 1994.
- [15] LEVIN, R. E. Photometric characteristics of light-controlling apparatus. *Illuminating Engineering* 66, 4 (1971), 205-215.
- [16] LEVOY, M., AND HANRAHAN, P. Light-field rendering. In *Computer Graphics, Annual Conference Series, 1996*.
- [17] LEWIS, R. R., AND FOURNIER, A. Light-driven global illumination with a wavelet representation of light transport. UBC CS Technical Reports 95-28, University of British Columbia, 1995.
- [18] LITWINOWICZ, P., AND WILLIAMS, L. Animating images with drawings. In *Computer Graphics, Annual Conference Series, 1994*, pp. 409-412.
- [19] MCMILLAN, L., AND BISHOP, G. Plenoptic modeling: An image-based rendering system. In *Computer Graphics, Annual Conference Series, 1995*, pp. 39-46.
- [20] MITCHELL, D. P. Generating antialiased images at low sampling densities. *Computer Graphics* 21, 4 (1987), 65-72.
- [21] POTMESIL, M. Generating octree models of 3D objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing* 40 (1987), 1-29.
- [22] POWELL, M. J. D., AND SWANN, J. Weighted uniform sampling - a monte carlo technique for reducing variance. *J. Inst. Maths Applies* 2 (1966), 228-236.
- [23] ROSENFELD, A., AND KAK, A. C. *Digital Picture Processing*. Academic Press, New York, New York, 1976.
- [24] SIMONCELLI, E. P., FREEMAN, W. T., ADELSON, E. H., AND HEBBER, D. J. Shiftable multiscale transforms. *IEEE Transactions on Information Theory* 38 (1992), 587-607.
- [25] SNYDER, J. M., AND KAJIYA, J. T. Generative modeling: A symbolic system for geometric modeling. *Computer Graphics* 26, 2 (1992), 369-379.
- [26] SZELISKI, R. Rapid octree construction from image sequences. *CVGIP: Image Understanding* 58, 1 (July 1993), 23-32.
- [27] TAUBIN, G. A signal processing approach to fair surface design. In *Computer Graphics, Annual Conference Series, 1995*, pp. 351-358.
- [28] TERZOPOULOS, D. Regularization of inverse visual problems involving discontinuities. *IEEE PAMI* 8, 4 (July 1986), 413-424.
- [29] TSAI, R. Y. A versatile camera calibration technique for high-accuracy 3D machine vision metrology using off-the-shelf TV cameras and lenses. *IEEE Journal of Robotics and Automation* RA-3, 4 (August 1987), 323-344.
- [30] WERNER, T., HIRSCH, R. D., AND HLAVAC, V. Rendering real-world objects using view interpolation. In *Fifth International Conference on Computer Vision (ICCV'95)* (Cambridge, Massachusetts, June 1995), pp. 957-962.
- [31] WILLSON, R. G. *Modeling and Calibration of Automated Zoom Lenses*. PhD thesis, Carnegie Mellon University, 1994.



Aspen and Golden Gate (MIT, 1978-1980)

Moviemaps

(Naimark, Exploratorium,
1987)

- 2D array of video taken of environment
- Video transferred to random-access videodisc
- Computer-based user interface allows real-time photorealistic exploration of the space

Displacements

(Naimark, San Francisco
Museum of Modern Art,
1984)

- Image-based modeling and rendering with real geometry and real light
- Living room filmed with rotating movie camera
- Room painted white
- Film reprojected with rotating movie projector

Rouen Revisited

(Levin and Debevec,
SIGGRAPH 96 Art Show)

- Time-series of photographs taken from 3 positions outside the cathedral
- 3D model built using Façade
- Monet paintings and historic images aligned to model
- Interactive user interface allows user to independently vary viewpoint, time of day, and level of fog, and to see as a painting, as it stands today, or as it appeared to Monet



Synthetic View:
1996



Synthetic View:
1896



Synthetic View:
Monet Painting

Like a Rolling Stone (SIGGRAPH 96 ET)

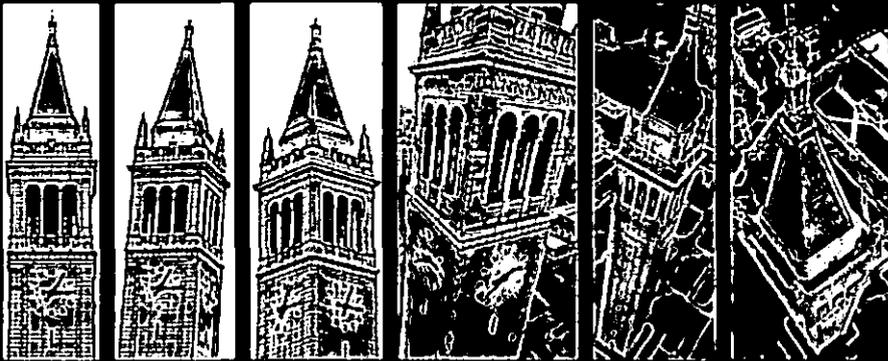
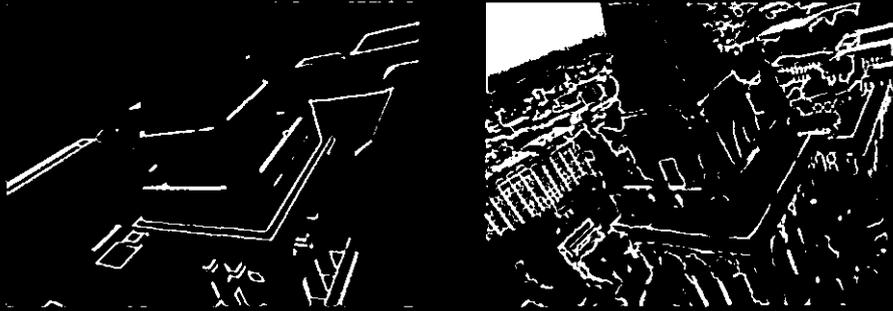
- Uses two techniques to achieve its look:
 - Morphing between sparsely sampled frames
 - ◆ Morphing not done with “correct” view interpolation (e.g. Seitz and Dyer ‘96)
 - ◆ Problems arising from scene motion, occlusions, and incorrect mathematics and are used for artistic effect
 - Playback of a series of images taken at the same time but from different viewpoints
 - ◆ See Dayton Taylor’s Virtual Camera work <http://www.virtualcamera.com>

Tour into the Picture (Anjyo et. al. SIGGRAPH 97)

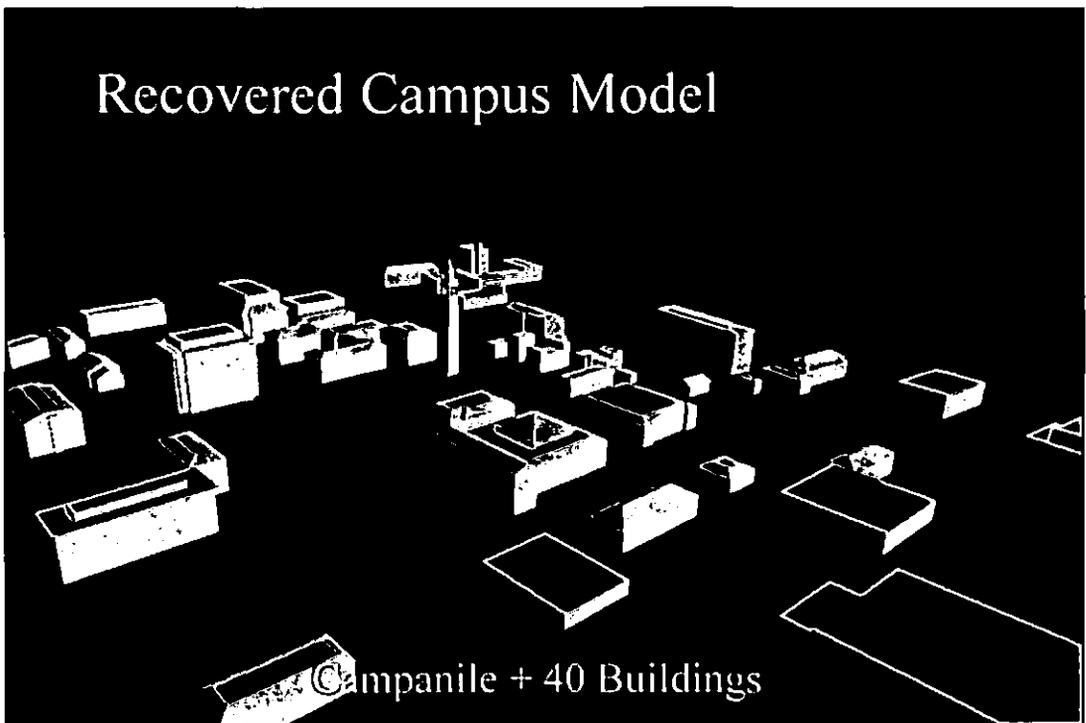
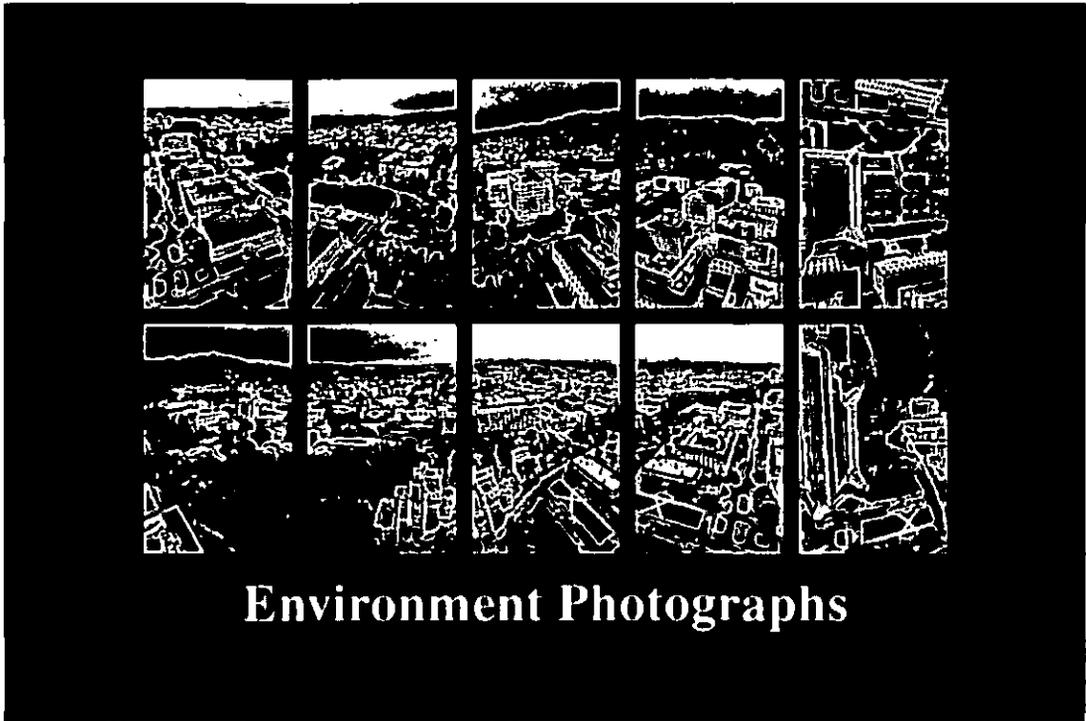
- An approximate depth map is constructed interactively using simple tools
- Foreground objects are masked out
- The user fills in the occluded areas
- Animations from the single photograph reveal depth as well as visual detail

The Campanile Movie (SIGGRAPH 97 ET)

- 20 photographs from ground, tower, kite
- 1-2 weeks modeling time
- Real time rendering with projective texture mapping



Tower Photographs



Elements of Realspace Imaging: A Proposed Taxonomy

Michael Naimark

independent media artist
San Francisco

Originally published in:
SPIE Proceedings Vol. 1457, p. 169-179,
Stereoscopic Displays and Applications II,
John O. Merritt, Scott S. Fisher, Eds. 1991

ABSTRACT

Along with the marriage of motion pictures and computers has come an increasing interest in making images appear to have a greater degree of realness or presence, which I call "realspace imaging." Such topics as high definition television, 3D, fisheye lenses, surrogate travel, and "cyberspace" reflect such interest. These topics are usually piled together and are unparsable, with the implicit assumptions that "the more resolution, the more presence" and "the more presence, the better." This paper proposes a taxonomy of the elements of realspace imaging. The taxonomy is organized around six sections: 1) monoscopic imaging, 2) stereoscopic imaging, 3) multiscopic imaging, 4) panoramics, 5) surrogate travel, and 6) realtime imaging.

INTRODUCTION: REALSPACE IMAGING

Realspace imaging is the process of recording and displaying sensory information indistinguishable from unmediated reality. Imagine looking at a framed image as if it were a window. Fooling the eye into believing the image is real is a difficult task. Fooling two eyes is even more difficult. Fooling two eyes while allowing freedom of head motion is yet more difficult. Now imagine removing the frame and having the freedom to look around. Now imagine having the freedom to move around. Now add time-based phenomena such as motion and sound. These are the elements of realspace imaging proposed.

1. MONOSCOPIC IMAGING

Monoscopic images represent one single point of view.

1.1. Orthoscopy (Scale)

Orthoscopic images, images viewed in proper scale, require the viewer see from the same angle of view as that of the camera lens. Orthoscopically correct images must be viewed from only one single point of view, hence virtually every image we see is non-orthoscopic. In addition to scale change, off-axis viewing results in trapezoidal distortion. We are rarely cognizant that we are looking at a trapezoid when we sit to the side of a movie theater, though it has been shown that additional cognitive processing is required to "straighten it out."¹

1.2. Spatial Resolution and Color

Spatial resolution is possibly the most-discussed aspect of realspace imaging. Today, the images we see on television and in movie theaters are recorded on a wide variety of formats whose principal difference is spatial resolution (and color, partly a subset of spatial resolution and partly a subset of recording and display technology). The current dilemma over "how much is enough?" has been additionally fueled by standardization arguments for high definition television (HDTV). To some, the research is disheartening: a display with a 45° by 90° field of view would ideally require a 3000 by 6000 pixel display, many times the resolution of even present HDTV standards.²

1.3. Dynamic Range and Brightness

Dynamic range is the span from the whitest whites to the blackest blacks in visual recording and display. The eye has a broad dynamic range allowing us to see bright outdoor scenes and shadow detail simultaneously. Film has less dynamic range than the eye. Scenes shot in film must be carefully lit to "squeeze" the dynamic range into films' limits, such as "fill lighting" the shaded areas more and "key lighting" the bright areas less. Video has even less dynamic range than film, requiring more complicated lighting to achieve the same effect as film. Film shot and transferred to video results in a greater dynamic range than video-originated material, which is one reason why many cinematographers prefer the "film look" (another reason is frame rate, see below).

1.4. Spatial Consistency and Spatialization

Many kinds of lower resolution images appear acceptable, particularly if the noise or artifacts have realworld analogs, such as looking at the world through a veil (or a fan blade). But when the styles of resolution are inconsistent with each other in the same image, it looks "wrong." Steve Yelick once referred to this as the "Gumby in Manhattan" problem.

Similarly an overlay can be "spatialized" by making it appear as a contiguous part of the image, where lighting and shadow, scale, and synchronous movement must correspond with the background. The motion picture special effects industry has long been aware of such importance, and the so-called "virtual reality" field is currently realizing the need for spatializing data. A videodisc-based example of spatializing data was recently produced by the Apple Multimedia Lab, where electronic graphics were inserted along a filmed desert highway to teach scale to schoolchildren.³

1.5. Monoscopic Depth Cues

1.5.1. Content Cues

There are several monoscopic cues to depth perception which are based on image content, such as perspective, overlapping or occlusion, aerial perspective or atmospheric, light and shade, and textural gradient.⁴ It is noteworthy that these cues are automatically captured with a camera but must be addressed explicitly with computer-generated imagery, where these factors have historically been of great concern.

1.5.2. Accommodation (Focus)

With one eye open and a fixed head position, a prominent depth cue is accommodation, the focussing of the eye's lense by the surrounding muscles. It is similar to focussing the lense of a camera. There are two ways to determine depth from focussing a camera. One is to focus on an object and read the calibrated focus setting. We sense our eye muscles in a similar way when we change focus. The other way to determine depth is by the amount of blur that exists for objects out-of-focus, which is partly a function of distance and partly a function of brightness. Obtaining depth data in an image by comparing two samples of blurriness has been demonstrated.⁵

Accommodation discrepancies are often prominent while viewing landscape images, where the eyes should be focussed on infinity (and where parallax diminishes to near zero). One method of achieving "infinity focus" is simply to project farfield imagery far away on a large screen, but a large space is required. Another technique, common in flight simulators, uses a large concave mirror which magnifies and refocusses the image from a small monitor. But even with this large mirror, the effective viewing area is small, enough for only one person. Smaller and less expensive optics can be used instead of a mirror if the viewer is further restricted to an even smaller viewing area, like a peephole.

Similarly, a concave mirror may be used for nearfield imaging by projecting a real image in front of the mirror's surface. The "floating nickel" illusion and video "genies" hovering in space are popular examples.

2. STEREOSCOPIC IMAGING

Stereoscopic images represent two single points of view, one for each eye, separated to give a noticeable lateral displacement, or parallax. Parallax is often erroneously pitched as all that is necessary for depth in

imagery (the stereoscopic movies of the 1950s were simply labelled "3D"). There is no easy way to record and reproduce parallax. First, two simultaneous views must be recorded, with care taken for proper convergence and disparity. Then, each view must be seen exclusively by each eye.

Stereoscopic photography has a lively history dating back at least to Wheatstone's invention of the stereoscope in 1833.⁶ The most popular techniques require glasses to be worn (such as anaglyphic, polarized, or shutter). Methods not requiring glasses usually require the head to be held in a particular position (using mirrors, pcepholes, or lenticular screens, for example).

3. MULTISCOPIC IMAGING

Multiscopic imaging represents multiple points of view: lateral head movement while the body is more-or-less still. The result is local motion parallax (global motion parallax equals travel). Local motion parallax is a stronger depth cue than stereoscopic parallax because more than two points of view can be seen in a relatively short period of time.

That local motion parallax occurs when we rotate our head is a wonderful evolutionary feature of humans (and most other animals) because our eyes are displaced from our neck's axis of rotation. If our eyes were *on* the neck's axis of rotation (like a camera mounted on a tripod), there would be no lateral displacement when we turn our head and therefore no parallax.

3.1. Mirrors

A mirror is multiscopic by nature. When we view ourselves in a mirror, each eye is seeing a different point of view, so we see stereoscopically. But also when we move our head, we see correspondingly different points of view.

An early technique for achieving multiscopic images required a giant half-silvered mirror with which to reflect hidden images and props over an actual stage set. Like normal mirrors, both parallax and accomodation are preserved, but since it is half-silvered, the reflected imagery appears transparent, making it great for ghosts but not much else. Such giant half-silvered mirrors date back to the Phantasmagoria shows of the 18th century and have been popularized by the ballroom of Disney's Haunted Mansion. Similar examples, but where the 3D floating images were of 2D film and video screens (and if done cleverly appears 3D), were popular attractions at the last three Worlds' Fairs: the GM "Spirit Lodge" (EXPO '86, Vancouver), the Australian Pavillion (EXPO '88, Brisbane), and the Ginko, Gas, and Mitsui Toshiba pavillions (EXPO '90, Osaka).

A technique for producing small multiscopic images employs a flexible vibrating mirror to rapidly change focal lengths. This varifocal mirror reflects a video display whose image is in sync with the vibration, resulting in a relatively small volumetric display.⁷ Since the video must be from a computer-generated 3D model, direct display of camera-originated images is not possible. Indeed, no such camera exists.

3.2. Relief Projection

Another multiscopic technique is sometimes called relief projection, where an image is projected onto a screen whose shape physically matches the image itself. Historically, the most popular application of relief projection are "talking heads," where a mask is made of a person's face to be used as the projection screen. The person is filmed with their head totally motionless but their mouth and eyes moving. The film is projected onto the facemask screen, with careful alignment such that the eyes fall in the eye sockets, the mouth along the mouth line, etc. The illusion is very powerful, and the fact that the image of the eyes and mouth move and the screen does not is barely noticable. The talking head in Disney's Haunted Mansion uses this technique. A more advanced version, where the mask screen moved in sync with the image, was produced at MIT in 1980.⁸ The author has produced room-sized relief projection by painting entire stage sets white after filming them and projecting the original image back on the white-painted surfaces.⁹ The limit of relief projection, of course, is that the shape of the screen cannot easily change.

One method of making a more flexible relief projection display is to rapidly spin a disc or corkscrew-shaped screen while projecting on it with synchronized lasers.¹⁰ The result is a volumetric display (usually inside a clear housing for safety) whose size, detail, and flicker rate are related to the computational horsepower and the mechanics of the system. And like varifocal mirror displays, all the imagery must come from a 3D computer model rather than directly from a camera.

3.3. Holography

Holography achieves both parallax and accommodation, but is a filmic medium (with the extremely significant exception of Benton's most recent work at the MIT Media Lab). Being film-based has its implications. It cannot be transmitted live like video. Also, it is near-impossible for any kind of computer control and interactivity.

Another popular misconception about holography is its "projection." The holographic image can only be seen while viewing through the film: it may appear behind the film, in front of the film, or both, but one must always be looking *through* the film. The concept of both the audience and the hologram being on the ground and a holographic image "projected" in the sky is simply inaccurate.

"Stereograms" (or integrams) are holograms made from filmed or computer-generated material, where images are recorded from multiple points of view along a single straight or curved track. If the material is shot with a single moving camera, any motion counteracts the stereoscopy. Real holographic movies, though demonstrated, require massive amounts of holographic film and even still offer very limited viewing.

3.4. Viewpoint-Dependent Imaging

Local motion parallax is possible with a conventional display if driven by the user's head position. An example was produced at MIT, where an outdoor scene was shot laterally and mastered on videodisc. The videodisc speed and direction was controlled by a single user wearing a position-tracking device on his head. As the user sways back and forth, the video image changes correspondingly.¹¹ Because such a display is interactive, it is limited to one single user. Though trivial with a virtual camera, recording with a real camera becomes increasingly difficult when more than one dimension is shot (allowing the user to sway back and forth, in and out, and up and down simultaneously). And like other single camera applications for multiple points of view, time artifacts (motion) counteracts the multiscope effect.

4. PANORAMICS

Panoramics is the ability to look around. An image is considered panoramic as it approaches framelessness, when the image is larger than the viewer's field of vision. When this occurs, there is a sense of *immersion*, of being inside rather than outside looking in. Panoramic imagery allows freedom of angular movement.

4.1. Rectilinear Perspective

Rectilinear perspective is shot on flat film and displayed on a flat surface. When viewing rectilinear perspective images off-axis, the frame will appear trapezoidal, but straight lines will always appear straight. Practically every camera we have ever seen or used and practically every image we've ever seen or made, has been of rectilinear perspective.

Because of their flat nature, all rectilinear perspective images must have less than a 180° angle of view, and therefore full panoramic construction requires multiple images. For example, MIT's Aspen Moviemap was shot with four 16mm cameras with slightly less than 90° lenses, pointing front, back, left, and right. For these images to be viewed together properly, one must stand in the center of a four-walled projection space, otherwise trapezoidal distortion will occur. The four images, when laid flat, will exhibit discontinuities which can be computer-corrected by "undistorting" them linearly.¹²

4.2. Cylindrical Perspective

Cylindrical perspective is shot on cylindrically-positioned film and displayed on a cylindrical surface. Unlike rectilinear perspective, only one cylindrical perspective image is required for a 360° panorama, and since it is a single image, there are no discontinuities. Cameras with rotating slits and lenses (such as the Widelux, Hulcher, Globus, and Roundshot cameras) can shoot a single image over a relatively short period of time.

Optimal viewing is from the center of the cylinder, like being inside a large lampshade. When the viewer is off-axis, straight horizontal lines appear curved, while straight vertical lines remain straight. The distortion can be "dewarped" with a computer by non-linear correction in one dimension and linear correction in the other dimension for a flat display.

4.3. Spherical Perspective

Spherical perspective is shot with spherical optics (such as fisheye lenses) and displayed on a spherical surface. Optimal viewing is from the center of the sphere, like being inside a large dome or an Omnimax theater. When the viewer is off-axis, straight lines in both dimensions will appear curved.

Spherical recording is most often associated with fisheye lenses, but other such specialty lenses exist. For example, the Peri-Appolar lens made by Volpi was used for the Aspen Moviemap. It produces a donut-shaped image representing a 360° azimuth by $\pm 30^\circ$ elevation, centered on the horizon rather than on the zenith when pointing upward. Shooting off convex mirrors also produces spherical perspective (a Legg's pantyhose package is a favorite), but the camera will be visible in the middle of the frame.

Spherical perspective images can capture the most in a single shot, but flat viewing results in distortion. The distortion can be "dewarped" with a computer by non-linear correction in both dimensions.¹³

4.4. Substituting Interactivity for Wholeness

For each type of perspective, it is possible to store the entire panoramic image in such a way that the user may access a subset of it. The obvious advantages are that it eliminates the need for a 360° projection space and requires less display bandwidth.

A reasonable method of viewing panoramic imagery is through a small flat rectilinear window such as a video display if the user has control of the point of view, using a joystick, for example. Intel's DVI technology has such a method for "dewarping" and displaying imagery shot with a fisheye lens.¹⁴

A less reasonable method (but one that kept this author obsessed for several years) is where a projected image physically moves around the playback space in order to retain the spatial correspondence.¹⁵ A "moving movie" requires neither the power nor bandwidth to fill the entire playback space, but it nevertheless requires a special playback space.

It is possible to combine "interactive small window" viewing with spatial correspondence by wearing the display on one's head and tracking head position. These head-mounted displays (HMDs) can also offer properly accommodated, stereoscopic, wide angle optics¹⁶ and have received a great deal of recent attention under such labels as "virtual realities," "virtual environments," and "cyberspace."

Virtually all imagery shown in HMDs today are either computer-generated or from live telerobotic cameras. Realworld recording and storage for HMDs presents novel challenges. For example, shooting for both stereoscopy and panoramics has no simple solution, since two panoramic cameras separated for stereoscopy results in variable parallax as the "interactive small windows" rotate.

5. SURROGATE TRAVEL

Surrogate travel, or "moviemaps," is the ability to move around, allowing the user to laterally move through a recorded or created place. Moving around any virtual space presents some problems not present when looking around a panoramic scene. Looking around need not be explicitly interactive: the entire view

can be displayed. But moving around under one's own control *must* be explicitly interactive: one must tell the system to change lateral position. Thus, while an audience in a panoramic theater can all look in different directions, an audience in a surrogate travel theater must somehow to come to grips with who is navigating.

Problems arise when surrogate travel is shot with real cameras, rather than generated from 3D databases. Though it is possible to record an entire panorama from a given point in a single instant, the only way to record surrogate travel in a single instant is with one camera at each location. The more realistic alternative is to move a single camera from one location to another, but time artifacts caused by moving clouds, shadows, cars, and people may result.

Another major difference between panoramic recording and surrogate travel recording is in continuousness. Once a panoramic scene is recorded and stored as a 2D single image, the user may have continuous access to any portion of it. But surrogate travel requires recording many 2D images at spatial intervals (like one frame every ten feet) and creating in-between images from these is a state-of-the-art computing problem. Hence surrogate travel material made from realworld recording is currently stored as many 2D images, on fast-access lookup media such as optical videodiscs.

5.1. One-Dimensional Movement

One-dimensional surrogate travel is along one particular path. The user may go forward and backward, at any speed, but cannot stray from this path.

5.1.1. Distance-Dependent Recording

In order to give the user a predictable sense of speed control, realworld images along a route are best shot at regular spatial intervals. Motion picture cameras, both film and video, are time-triggered instruments, for example recording one frame every 1/24 or 1/30 of a second. If the camera tracking speed can be held constant, then time triggering is equivalent to distance triggering. Otherwise explicit distance triggering is necessary like from an odometer or an external "fifth wheel."

The triggering distance affects visual continuity on the one hand and frame storage "real estate" on the other. The more images, the smoother the apparent movement, but the more storage space required. Smoothness is also related to angle of view of the camera, height and distance to the nearest objects, and camera stability.

5.1.2. Image Stabilization

Camera stability is a realworld problem, not relevant for virtual cameras or for model cameras on motion control systems. Instability results from any variance of the lateral path or the angular position of the camera during shooting. High frequency instabilities such as vibrations will produce blur or smear and affect individual frames. They can be minimized by using a short exposure time, a wide angle lens, and staying away from close or fast-moving objects.

Low frequency instabilities will produce a "wobble" from frame to frame. Since moviemaps are often shot at frame rates less than normal motion pictures' (one frame per second may be an average recording speed, for example), such instabilities are exaggerated. Consequently, closed-loop gyroscopic stabilizers (such as Wescams, Gyrospheres, or Tyler "Sea Mounts") perform better than either passive gyroscopic stabilizers (such as some helicopter mounts) or passive inertial stabilizers (such as Steadicams). In-camera and in-lens stabilizers (such as the 1962 "Dynamens," Arriflex's Image Stabilizer, and Schwem's "Gyrozoom") can only correct for pan and tilt but not for rotation.

5.2. "1.1" Dimensional Movement

Moving along a path with occasional choice-points is a far cry from being able to "travel anywhere." One might call this class of surrogate travel "1.1 dimensional" because only some of the points along the path have a two dimensional choice and most have only a one dimensional choice.

5.2.1. Match-Cuts

At nodes (points with a two-dimensional choice), the better the match-cut between two intersecting routes, the greater the sense of seamlessness. Several factors contribute to matching cuts. First, the camera has to be in the same position and pointing in the same direction for both routes as it passes through the node. One may use lines on the street or one may use compass coordinates, but there is no easy way to do this in the real world. Also, temporal artifacts are inevitable, since the matching shots must be recorded at different times. Lighting and shadow discrepancies can be minimized by shooting during a narrow window of time, like from 10 am to 2 pm, or shooting on cloudy days. For 3D database recording, as well as for motion control model shooting, these problems don't exist.

5.2.2. Camera Angle

Since panoramic recording is often impractical, the camera's angular position becomes an issue, since a less-than-360° lense must be explicitly pointed. The simplest technique is to fix the camera angle to the lateral direction of motion, either pointing straight ahead or pointing sideways. At each node, every possible turn must be separately recorded in order to match-cut the intersecting routes. MIT's Aspen Moviemap was shot in this fashion.¹⁷

Another, more complicated way to point the camera is in an absolute direction independent of lateral position. For example, a camera could always point north regardless of whether it is facing forward, sideways, or backward. An advantage is that shooting turns is not required since the camera is pointing in the same direction at any given point.

An even more complex way is to point the camera at an absolute location, such as tracking a central object. For example, the "Golden Gate Videodisc" produced by Advanced Interaction Inc. and directed by the author for the Exploratorium is an aerial moviemap over the Bay Area where the camera always pointed at the center of the Golden Gate Bridge. Like absolute position, the payoff is that separate turn sequences are not necessary to record since the camera always points in the same direction at any given point.

5.3. 2-D and 3-D Movement

Recording and storing two-dimensional grids and three-dimensional lattices, where the user has freedom of movement, is problematic because the numbers grow quickly. Consider that a 15 by 20 foot space with a 10 foot high ceiling requires 3,000 frames if shot at intervals of one feet and over 5 million frames at intervals of one inch!

In the future, the very idea of discrete frame storage will be obsolete. Computers will store information in spatial databases based on whatever data has been collected (and will interpolate what is missing). It has been demonstrated that significant bandwidth compression occurs when the visual information from separate (highly redundant) movie frames are stored as a single computer model.¹⁸ But data will still need to be collected, and visual data will be collected with cameras.

6. REALTIME IMAGING

Realtime imaging is the process of recording and displaying *temporal* sensory information indistinguishable from unmediated reality.

6.1. Dynamic Visual Cues

6.1.1. Frame Rate

At least 15 updates per second are necessary for motion to appear on a screen. The upper level is arguable. Modern American film runs at 24 frames per second (fps), American video updates at 60 fps, but 80 or 90 fps may be necessary.¹⁹

Apparently, part of our association with the "film look" is film's *lack of* a sufficient frame rate. When video is "defluttered" (every other field removed reducing the effective update rate from 60 to 30 fps), the result takes on a film look.²⁰ Similarly, the Showscan film format, which records and projects at 60 fps, has a video look.

6.1.2. Temporal Continuity

Temporal continuity is the opposite of cuts, or montage. The real world exhibits temporal continuity always, regardless if it is seen looking out from a train or from a racecar or sitting still. There are no cuts in the real world. (Believing that you really are instantly somewhere else, as opposed to imagining it, is the definition of psychosis.) Temporal continuity is the temporal equivalent of spatial consistency.

Cinema, on the other hand, consists of adjacent frames which are either continuous (those within a shot) or not (those between shots, the "cuts"). Cinema is the counterpoint between "respect for spatial unity"²¹ and its "first and foremost" characteristic, montage.²² Noteworthy is Alfred Hitchcock's *Rope*, a feature film shot with a carefully orchestrated camera, which has virtually no cuts.

6.2. Dynamic Non-Visual Cues

6.2.1. Audio

Audio in synchronization with image is part of our association with cinema's ability to convey presence, and audio has its own resolution specifications. Of particular relevance here is the spatialization of sound. Sound can be spatialized one of two ways: by using multiple speakers each positioned in the point of origin of the sound source or by using binaural sound.²³

6.2.2. Inertial Motion

In addition to visual and auditory cues, we receive temporal cues by how we physically feel. This feeling of motion is based primarily in the vestibular system in the inner ear and is sensitive to linear and angular acceleration.²⁴ Flight simulators (as well as Disney's *Star Tours* and *Body Tours*) move the viewers on a motion platform synchronized with the image and sound to enhance their effect.

6.2.3. Force Feedback

Force feedback is the ability to "touch" a virtual object inside an image. For example, a force-feedback joystick has been used to simulate textures.²⁵ Similarly, a hand grip made of a four inch bar with three computer-actuated springs on each end can simulate angular and lateral force, and has been used successfully to augment visual display for spatial tasks.²⁶

7. AFTERWORD

Each of these elements of realspace imaging can either be respected or violated. An image either is orthoscopic, stereoscopic, or panoramic or it's not. Sometimes violations of these elements are by default: it's more convenient to carry around non-orthoscopic images of your family than "actual size," stereoscopic cameras are expensive, and panoramic movies require special theaters.

But sometimes violations of these elements are intentional: a cut in a film, slow frame rate in a rock video, a simple line drawing rather than a high resolution image, silence rather than sound. The very idea of respecting all elements of realspace imaging is ultimately a losing battle. Giving the user everything is rarely possible. There is never enough bandwidth. There will always be artifacts.

The trick is to give the sense of everything without actually giving everything. The question, then, is how to choose what is most important. And what is most important is always context-dependent. This report is an attempt to lay out the choices. Choose wisely: that is where the art lies.

8. ACKNOWLEDGEMENTS

This paper is a much-condensed version of a forthcoming Apple Computer Technical Report entitled "Elements of Realspace Imaging" written for the multimedia community and supported by the Apple Multimedia Lab in San Francisco. The author wishes to thank Phil Agre, Doug Crockford, Scott Fisher, Brenda Laurel, Robert Mohl, and Rachel Strickland as well as the members of the Apple Multimedia Lab, particularly its Director, Kristina Hooper, all for their lively discussions and criticisms throughout the course of that report.

9. REFERENCES

1. Hochberg, J. and Gellman, I. "Feature Saliency, 'Mental Rotation' Times and the Integration of Successive Views," Memory and Cognition, No. 5, 1977.
2. Schrieber, W.F. "Psychophysics and the Improvement of Television Image Quality"; SMPTE Journal, pp. 717-725, August, 1984.
3. (report on the "Visual Almanac" forthcoming from the Apple Multimedia Lab).
4. Lipton, L. Foundations of the Stereoscopic Cinema, New York: Van Nostrand Reinhold Company, 1982.
5. Bove V. M. Synthetic Movies Derived from Multi-Dimensional Image Sensors, Cambridge, MA: Ph.D. dissertation, Media Laboratory, M.I.T., 1989.
6. Lipton, Stereoscopic Cinema.
7. "3D Display - Without Glasses"; Design News, p. 13, December 1988.
8. Negroponte, N. "Media Room"; SID 22, no. 2, 1981.
9. Reveaux, A. "Displacing Time and Image"; Artweek, June 30 1984.
10. Williams, R. D. and Garcia, F. "Volume Visualization Displays"; Information Display 5, no. 4, pp. 8-10, April 1989.
11. Fisher, S. S. "Viewpoint Dependent Imaging: An Interactive Stereoscopic Display"; SPIE, Volume 367, Processing and Display of Three-Dimensional Data, 1982.
12. Yelick, S. Anamorphic Image Processing, B.S. thesis, Architecture Machine Group, M.I.T., 1980.
13. Ibid.
14. Wilson, K.S. The Palenque Design: Children's Discovery Learning Experiences in an Interactive Multimedia Environment, PhD dissertation, Graduate School of Education, Harvard University, 1988.
15. Naimark, M. "Spatial Correspondence in Motion Picture Display"; SPIE, Volume 462, Optics and Entertainment, 1984.
16. Fisher, S.S. et al. "Virtual Environment Display System"; ACM, Workshop on Interactive 3D Graphics, 1986.
17. Mohl, R. Cognitive Space in the Interactive Movie Map: An Investigation of Spatial Learning in Virtual Environments, PhD dissertation, Education and Media Technology, M.I.T., 1981.
18. Bove, Synthetic Movies.
19. Schrieber, "Psychophysics."
20. Naimark, M. "Videodisc Production of the 'Visual Almanac'"; Technical Report #14, The Multimedia Lab, Apple Computer, Inc., 1988.
21. Bazin, A. What Is Cinema?, Volume 1, Berkeley: University of California Press, 1967.
22. Eisenstein, S. Film Form, New York: Harcourt Brace Jovanovich, Inc., 1949.
23. Wenzel, E.M. et al. "A Virtual Display System for Conveying Three-Dimensional Acoustic Information"; Proceedings of the Human Factors Society, 32nd Annual Meeting, 1988.
24. Deyo, R. and Ingebreisen, D., "Notes on Real-Time Vehicle Simulation"; ACM Siggraph, Course Notes 29, Implementing and Interacting with Real-Time Microworlds, 1989.
25. Minsky, M. "Texture Identification Experiments"; ACM Siggraph, Course Notes 29, Implementing and Interacting with Real-Time Microworlds, 1989.
26. Ming, O. et al. "Force Display Performs Better than Visual Display in a Simple 6-D Docking Task"; ACM Siggraph, Course Notes 29, Implementing and Interacting with Real-Time Microworlds, 1989.

Rouen Revisited

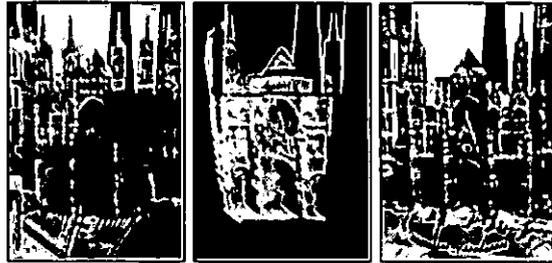
An interactive art installation by
Golan Levin (*Interval Research Corporation*) and
Paul Debevec (*University of California at Berkeley*)

Française • Deutsche

Presented at:
The Siegraph '96 Art Show
New Orleans, USA, 4 - 9 August 1996

Fleshfactor: Ars Electronica Festival '97
Linz, Austria, 8 - 13 September 1997

Eighth International Symposium on Electronic Art (ISEA '97)
Chicago, USA, 22 - 27 September 1997



Synthetic views of the Rouen Cathedral derived (respectively) from:
photographs taken in January 1996; a painting by Claude Monet, made in 1894-95;
and photographs taken in the mid-1890's. All are shown from the same point of view.

Table of Contents

- [Overview](#)
 - [The Technology](#)
 - [The Presentation](#)
 - [The Experience](#)
 - [Renderings from Rouen Revisited](#)
 - [Artist Biographies](#)
 - [Exhibition Information](#)
 - [Credits](#)
-

Overview

Between 1892 and 1894, the French Impressionist Claude Monet produced nearly 30 oil paintings of the main façade of the Rouen Cathedral in Normandy. Fascinated by the play of light and atmosphere over the Gothic church, Monet systematically painted the cathedral at different times of day, from slightly different angles, and in varied weather conditions. Each painting, quickly executed, offers a glimpse into a narrow slice of time and mood.

We are interested in widening these slices, extending and connecting the dots occupied by Monet's paintings in the multidimensional space of turn-of-the-century Rouen. In *Rouen Revisited*, we present an interactive kiosk in which users are invited to explore the façade of the Rouen Cathedral, as Monet might have painted it, from any angle, time of day, and degree of atmospheric haze. Users can contrast these re-rendered paintings with similar views synthesized from century-old archival photographs, as well as from recent photographs that reveal the scars of a century of weathering and war.

Rouen Revisited is our homage to the hundredth anniversary of Monet's cathedral paintings. Like Monet's series, our installation is a constellation of impressions, a document of moments and percepts played out over space and time. In our homage, we extend the scope of Monet's study to where he could not go, bringing forth his object of fascination from a hundred feet in the air and across a hundred years of history.

The Technology

To produce renderings of the cathedral's façade from arbitrary angles, we needed an accurate, three-dimensional model of the cathedral. For this purpose, we made use of new modeling and rendering techniques, developed by Paul Debevec at the University of California at Berkeley, that allow three-dimensional models of architectural scenes to be constructed from a small number of ordinary photographs. We traveled to Rouen in January 1996, where, in addition to taking a set of photographs from which we could generate the model, we obtained reproductions of Monet's paintings as well as antique photographs of the cathedral as it would have been seen by Monet.

Once the 3D model was built, the photographs and Monet paintings were registered with and projected onto the 3D model. Re-renderings of each of the projected paintings and photographs were then generated from hundreds of points of view; renderings of the cathedral in different atmospheric conditions and at arbitrary times of day were derived from our own time-lapse photographs of the cathedral and by interpolating between the textures of Monet's original paintings. The model recovery and image rendering was accomplished with custom graphics software on a Silicon Graphics Indigo2. The *Rouen Revisited* interface runs in Macromedia Director on a 166-MHz Pentium PC, and allows unencumbered exploration of more than 12,000 synthesized renderings.

The execution of *Rouen Revisited* entailed more than half a dozen novel technical achievements. The most basic of these were the techniques of view-dependent texture-mapping, photogrammetric modeling, and model-based stereo that Paul Debevec developed in his Berkeley Ph.D. thesis. Other achievements, however, were more specific to the *Rouen Revisited* installation itself. A brief survey of the technological accomplishments in *Rouen Revisited* can be found here.

Further information about the modeling and rendering algorithms used in *Rouen Revisited* can be found in:

- Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik, Modeling and Rendering Architecture from Photographs. In SIGGRAPH '96 Proceedings, August 1996.
-

The Presentation

Rouen Revisited is presented in an arch-shaped maple cabinet, seven feet three inches tall. Its front face is articulated by three features: Near the top, a backlit stained-glass rosette (whose design is based on the rosette of the Rouen Cathedral) acts as a beacon for passers-by. Below that, a 17-inch color monitor, configured on its side, provides users with a view onto the cathedral's surface. Finally, a projecting wedge-shaped block at waist-level provides the interface controls for operating the kiosk.

Users explore the surface of the Rouen Cathedral by touching one of three force-sensitive regions exposed within a brass plate mounted on the interface wedge. Each region affords the user with control of a different dimension of the façade:

- Touching the corners of the upper, triangular region of the brass plate allows users to select between renderings of Monet paintings, archival photographs from the 1890's, or new photographs from 1996. Dragging one's finger along this groove creates a blend between these modes.
- Moving one's finger left and right inside the central, upside-down-T-shaped region of the brass control plate allows users to change the time of day. Moving one's finger up and down the vertical groove of this control changes the level of fog. This control is disabled for the archival photographs, for which time-series and fog-series source stills were unavailable. Nevertheless, this control is active for the new photographs and Monet paintings, and permits users to draw comparisons between the actual appearance of the cathedral (given certain lighting conditions) and Monet's interpretation of the cathedral so lit.
- Dragging one's finger across the rectangular, bottom region of the brass plate allows users to change their point of view around the Rouen Cathedral.



The Experience

The *Rouen Revisited* interactive kiosk allows its users to explore eight dimensions of the façade of the Rouen Cathedral in Normandy. We can examine the cathedral in various levels of fog; at different times of day; from different points of view on a three-dimensional viewing surface—and lastly, along three dimensions of interpretation and media: namely, as the cathedral appeared to photographers a hundred years ago, as it appears today, and as it would appear if Monet's impressionist paintings of it were aligned with and projected onto its surface.

Many more ways of exploring and understanding the Gothic cathedral are afforded by moving between and around these modes. We can observe the many ways in which the cathedral has changed in the past century, for example, by moving between the re-renderings of the old photographs and those of the new photographs. We can come to an understanding of which *geometric* details Monet chose to focus on, by moving between views of his paintings and the historic photographs from the same time period. We can come to understand how the play of light at a given time of day may have inspired Monet to paint the colors and textures he did—by moving between a painting and the new photograph which shares the same time of day. And, when we scrub through the time-series of Monet paintings, we have a unique opportunity to access the entire set of Monet's Cathedral paintings, and gain an appreciation for the both the range of Monet's exploration as well as the constraints within which he chose to work. Finally, by changing the time of day and our point of view around the cathedral, we may derive a *sense of place*—a feeling for the Rouen Cathedral as a real physical artifact, and a sense of the passage of a day in Rouen.

Rouen Revisited is an artifact about artifacts about an artifact—an interactive and open-ended interpretation of paintings and photographs, which are themselves interpretations of an ancient Gothic artwork. Ultimately, the interpretation which *Rouen Revisited* affords is a *dynamic* one, forged in the mind of the user when she creates, using the multidimensional interface, her *own* Rouen Cathedral composition.

Renderings from Rouen Revisited

The unwieldy size of the *Rouen Revisited* image database (nearly 3 gigabytes of renderings) prohibits us from creating an interactive, web-based version of the project at this time. In this section of the Rouen site, we instead attempt to convey the experience made possible by *Rouen Revisited* in a compact, easily-transmissible and (for now) non-interactive form. In addition to presenting still images, we have also converted select paths through the multidimensional space of possible renderings into short animations and digital videos. The animations are presented as animated gifs, reduced to one-ninth of their original size and dithered from 24-bit color down to a browser-safe 8-bit palette. Viewing these animations requires a Netscape 2.0 or better browser. The Quicktime and AVI videos are similarly reduced in size, and additionally compressed with the Apple Video or Microsoft Video compression formats.

[To the animations and stills.](#)

Artist Biographies

Golan Levin is an artist and designer of artifacts and experiences. Before he joined Interval Research in 1994, Golan completed his self-made undergraduate degree in Media Arts and Sciences at the Massachusetts Institute of Technology. Since then, he has focused on the design of expressive instruments, tools and toys for producing and playing with media.

Paul Debevec received degrees in Math and Computer Engineering from the University of Michigan in 1992, and recently completed his Ph.D. in Computer Science at the University of California at Berkeley. For his thesis, Paul developed a method of modeling and rendering architectural scenes photorealistically from ordinary photographs by synthesizing techniques from computer vision with those of computer graphics. With no current commitments after graduate school, Paul is interested in continuing to capture, visualize, and interpret the world in new and creative ways through novel photographic techniques.

A photograph of the authors with the *Rouen Revisited* kiosk, August 1996.

A photograph of the authors in front of the Rouen Cathedral, January 1996.

Exhibition Information

Rouen Revisited is available for public exhibition on an expenses-only basis. When it is not on loan to outside exhibitors, *Rouen Revisited* may be seen by appointment at the Interval Research Corporation Gallery. For further information about the *Rouen Revisited* installation, Please email gallery@interval.com or contact the Interval Gallery at (650) 842-6222 [*phone*], (650) 354-0872 [*fax*].

Credits

Rouen Revisited was conceived and developed by Golan Levin and Paul Debevec. The kiosk's maple cabinet was constructed by Warren H. Shaw, furniture-maker, of South San Francisco. The stained-glass rosette was hand-made by David Kaczor, glass artist, of Mountain View, California. The brass interface gate was machined by Shane Levin, president of the HAP Engraving Company, New York City. Invaluable suggestions and assistance with the presentation design were provided by Joe Ansel of Ansel Associates, and Charles "Bud" Lassiter of Interval Research Corporation. Scott Snibbe and Geoff Smith provided key software assistance; additional hardware, electronics and construction support were lent by Scott Wallters, Chris \$eguine and Bernie Lubell.

Rouen Revisited would not have been possible without the generous support of Paul Allen, David Liddle, and Noel Hirst; Michael Naimark, Bud Lassiter, Sally Rosenthal, Carol Moran, Marc Davis, Frank Crow, Stephan Gehring, Marie-Dominique Baudot, Laurence Shelvin, and Mark Keen; Jitendra Malik and Camillo J. Taylor; Shane Levin; and Joe Ansel.

Golan Levin
Interval Research Corporation
1801-C Page Mill Road
Palo Alto, CA 94304
+1.650.424.0722
levin@interval.com

Paul Debevec
Computer Science Division
University of California at Berkeley
545 Soda Hall
Berkeley, CA 94720-1776
debevec@cs.berkeley.edu

Back to the [top](#) of this page.
Last updated 12 February 1998, webmaster@interval.com

Video Based Animation Techniques for Human Motion

Chris Bregler, U.C. Berkeley

Check <http://www.cs.berkeley.edu/~bregler/sig98ibr.htm> for updated course materials.

Most image based rendering techniques are applied to rigid domains: Static environment maps, indoor scenes, or architectural scenes. Explicit geometric structures are combined with image data. Texture mapping and view morphing are simple examples. We can generate new images from a collection of recorded images. Simple geometry dictates coarse transformations of fine grained image texture. New views of a scene can be generated in blending between the transformed example textures. This is a trade-off between explicit structure (collection of views and geometric model) and implicit example data (the image texture).

Such trade-offs are applied to other domains as well. The most successful speech production systems (text-to-speech, concatenative speech) follow a similar philosophy. A collection of annotated example sounds are used to create new sounds. A sentence is build from phonemes (explicit structure). To blend the phonemes together, the sound examples are pitch and time warped (implicit data). We will show how this extends to video data and human motion animation.

Structure vs Data for Animation:

So far most graphical animation techniques do not exploit such trade-offs between explicit structure and implicit data. Many facial and body animations are generated by 3D volumetric models and physical simulations. Some facial animation systems texture map images onto the geometric model, or morph between a few example images. The appearance and motion become increasingly realistic. We will survey some of these systems.

In contrast to physical simulations, motion capture based animation techniques become increasingly popular. An actor performs the desired motion, and devices record body joint configurations or facial configurations. This data is mapped onto graphical computer models. Motion editing techniques allow to modify the motion data and create new motions. This has similarities to image morphing techniques. Instead of warping image texture, spatio-temporal configurations are warped.

Some systems allow to blend between different motion-capture data sets of different actions. New animations are assembled using existing examples.

Video Based Animation of People:

In order to create animations, that have natural motion **AND** have photo-realistic appearance, we need to combine motion-capture and image based (or video based) techniques. The goal is to build video based representations of annotated example motions.

Unlike standard motion capture techniques that are based on markers or other devices, we need to

annotate body and facial configurations directly in unconstrained video. In static scenes the user could supply annotations by hand, but for video sequences, automatic techniques are crucial (10 min of video has 18,000 images, no-one has the budget, patience, and consistency to do this by hand). We will survey several visual tracking and annotation techniques that are tailored for full body movements and facial movements. We demonstrate these visual annotation techniques on lab recordings of people walking and talking. We also demonstrate how to process historic footage. Examples are the famous Edward Muybridge Plates from over 100 years ago of walking people, and stock-footage of John F. Kennedy giving a public speech.

To build libraries of example motions, we also need techniques that annotate coarse motion categories automatically. Again, this has to be done automatically. For example a 10 minute video of someone talking could be transformed into a video-based library of more than 2,000 phonetic lip motions (phonemes or visemes).

Once they are annotated, we can re-animate the data or create new data. We will present work-in-progress of photo-realistic animations of kinematic chain models, and we will cover in more detail work presented at last years SIGGRAPH conference on photo-realistic animation of talking heads: Video Rewrite.

More technical details of such techniques can be found in following papers:

- Video Rewrite: C. Bregler, M. Covell, M. Slaney

Video Rewrite uses existing footage to create automatically new video of a person mouthing words that she did not speak in the original footage. This technique is useful in movie dubbing, for example, where the movie sequence can be modified to sync the actors' lip motions to the new soundtrack.

Video Rewrite automatically labels the phonemes in the training data and in the new audio track. Video Rewrite reorders the mouth images in the training footage to match the phoneme sequence of the new audio track. When particular phonemes are unavailable in the training footage, Video Rewrite selects the closest approximations. The resulting sequence of mouth images is stitched into the background footage. This stitching process automatically corrects for differences in head position and orientation between the mouth images and the background footage.

Video Rewrite uses computer-vision techniques to track points on the speaker's mouth in the training footage, and morphing techniques to combine these mouth gestures into the final video sequence. The new video combines the dynamics of the original actor's articulations with the mannerisms and setting dictated by the background footage.

Video Rewrite is the first facial-animation system to automate all the labeling and assembly tasks required to resync existing footage to a new soundtrack.

- Video Motion Capture: C. Bregler, J. Malik

This paper demonstrates a new vision based motion capture technique that is able to recover high degree-of-freedom articulated human body configurations in complex video sequences. It does not require any markers, body suits, or other devices attached to the subject. The only input needed is a video recording of the person whose motion is to be captured. For visual tracking we introduce the use of a novel mathematical technique, the product of exponential maps and twist motions, and

its integration into a differential motion estimation. This results in solving simple linear systems, and enables us to recover robustly the kinematic degrees-of-freedom in noise and complex self occluded configurations. We demonstrate this on several image sequences of people doing articulated full body movements, and visualize the results in re-animating an artificial 3D human model. We are also able to recover and re-animate the famous movements of Eadweard Muybridge's motion studies from the last century. To the best of our knowledge, this is the first computer vision based system that is able to process such challenging footage and recover complex motions with such high accuracy.

Preliminary slides in PDF

References

- [Beier92] T. Beier, S. Neely. Feature-based image metamorphosis. SIGGRAPH 92, 26(2):35-42,1992.
- [Ezzat98] T. Ezzat, T. Poggio. MikeTalk: A Talking Facial Display Based on Morphing Visemes. Proc. Computer Animation Conference, Philadelphia, Pennsylvania, 1998. (CD-version)
- [Litwinowicz94] P. Litwinowicz, L. Williams. Animating images with drawings. SIGGRAPH 94, Orlando, FL, pp. 409-412,1994
- [Moulines90] E. Moulines, P. Emerard, D. Larreur, J.L. Le Saint Milon, L. Le Faucheur, F. Marty, F. Charpentier, C. Sorin. A real-time French text-to-speech system generating high-quality synthetic speech. Proc. Int. Conf. Acoustics, Speech, and Signal Processing, Albuquerque, MN, pp. 309-312, 1990.
- [Scott94] K.C. Scott, D.S. Kagels, S.H. Watson, H. Rom, J.R. Wright, M. Lee, K.J. Hussey. Synthesis of speaker facial movement to match selected speech sequences. Proc. Australian Conf. Speech Science and Technology, Perth Australia, pp. 620-625, 1994. (CD-version)
- [Water95] K. Waters, T.Levergood. DECface: A System for Synthetic Face Applications. J. Multimedia Tools and Applications, 1(4):349-366, 1995.(web-pointer)
- [Williams90] L. Williams. Performance-Driven Facial Animation. SIGGRAPH 90, 24(4):235-242, 1990.

IBR Techniques for Non-Rigid Domains

Video-Based Animation of Human Motion

Christoph Bregler

Computer Science Division
University of California, Berkeley

Includes Work done with
Jitendra Malik, Jerome A. Feldman, Charles H. Ying, UC Berkeley
Michele Covell, Malcolm Slaney, Interval.

Problem Domains

- Motion Capture / Animation



Body Suits, Markers



Video Motion Capture

Overview

- Survey Technology

Measurement / Learn Models / Recognition

- New Animation Paradigm



Movie

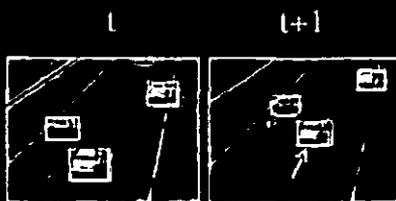


Actor Model



Video-Based Animation

Visual Tracking



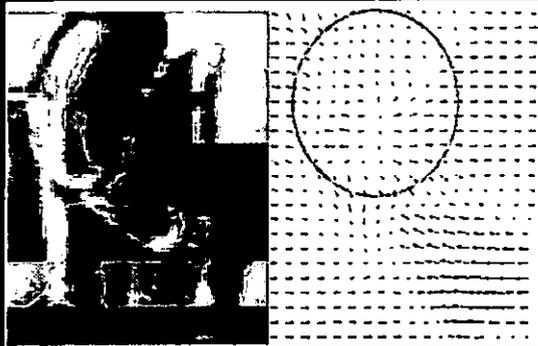
Standard Techniques:

- Template Matching
- Edges / Shape / Color
- Background Subtraction
- Optical Flow

New Challenges:

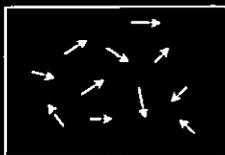
- Complex Variation
- Self Occlusion
- Noise (Folds, Low Contrast)

Optical Flow



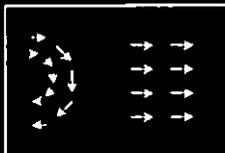
local ambiguities

Motion Constraints



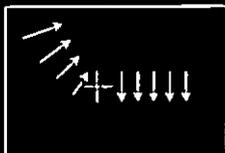
N Pixel

• Optical Flow: $2 \cdot N$ Parameters



L Areas

• Layered Motion: $6 \cdot L$ Parameters

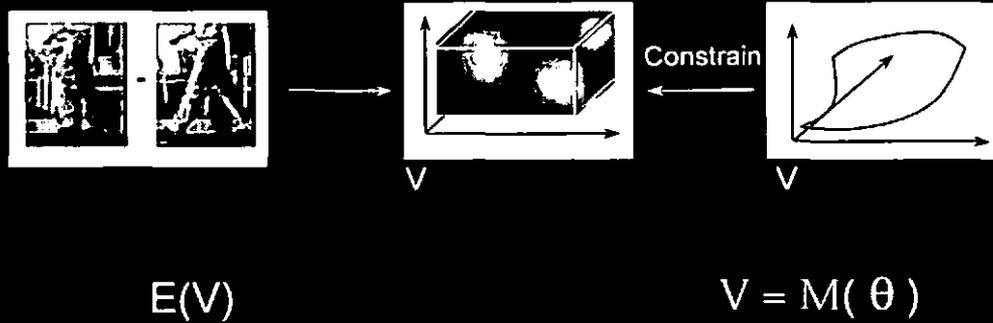


K DOFs

• Articulated Chains: $6 + K$ Parameters

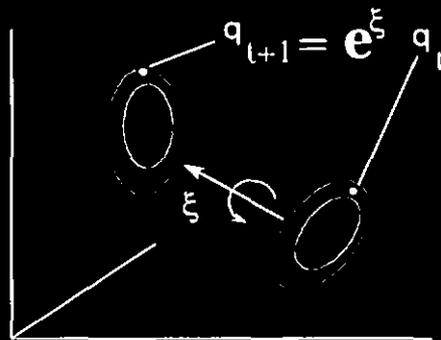
Video Motion Capture

Bregler, Malik

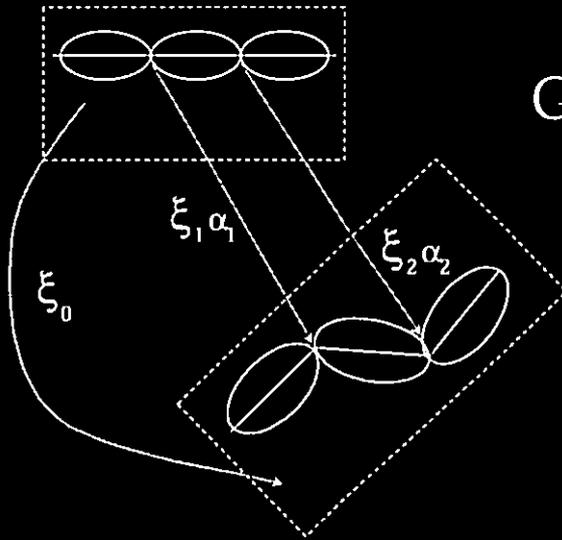


Twist Motion / Exponential Map

Murray, Li, Sastry

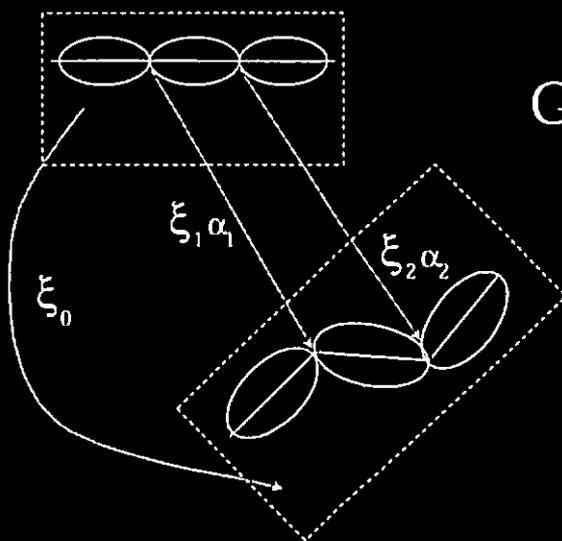


Product of Exponential Map



$$G = e^{\hat{\xi}_0} e^{\hat{\xi}_1 \alpha_1} e^{\hat{\xi}_2 \alpha_2}$$

Product of Exponential Map

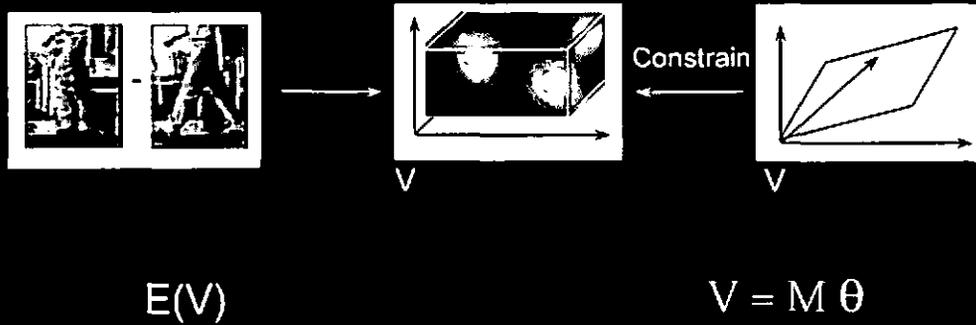


$$G = e^{\hat{\xi}_0} e^{\hat{\xi}_1 \alpha_1} e^{\hat{\xi}_2 \alpha_2}$$

$$\downarrow \frac{\delta}{\delta t}$$

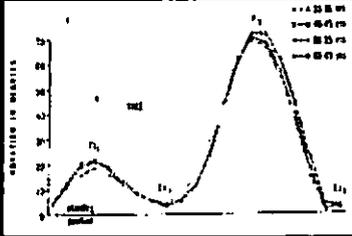
$$v_i = M_i \begin{bmatrix} \dot{\xi} \\ \alpha \end{bmatrix}$$

Video Motion Capture

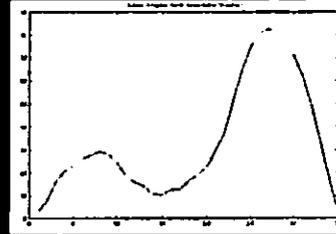


Video

Comparison to Biometric Data

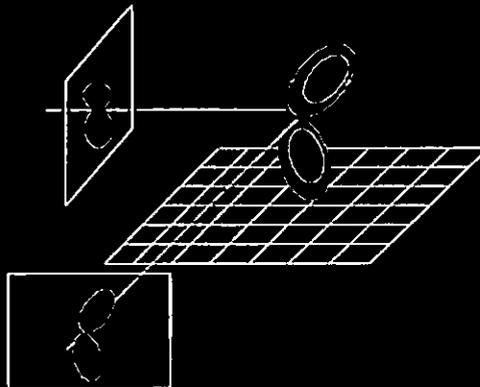


Murray, Drought, Kory, 1964



kinematic tracker

Multiple Views



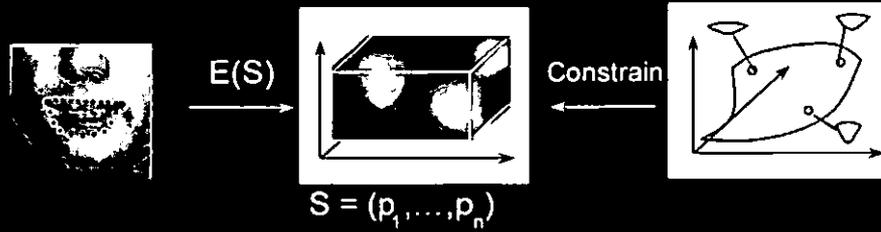
Eadweard Muybridge



Video

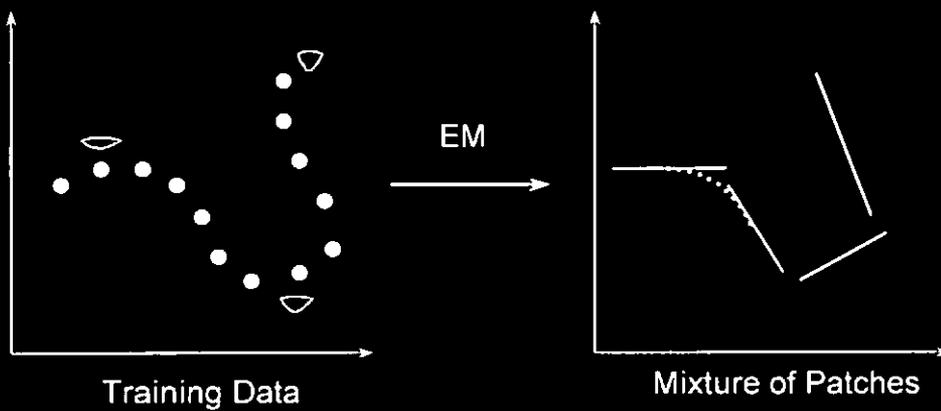
Graphics by Charles Ying

Non-Rigid Measurement

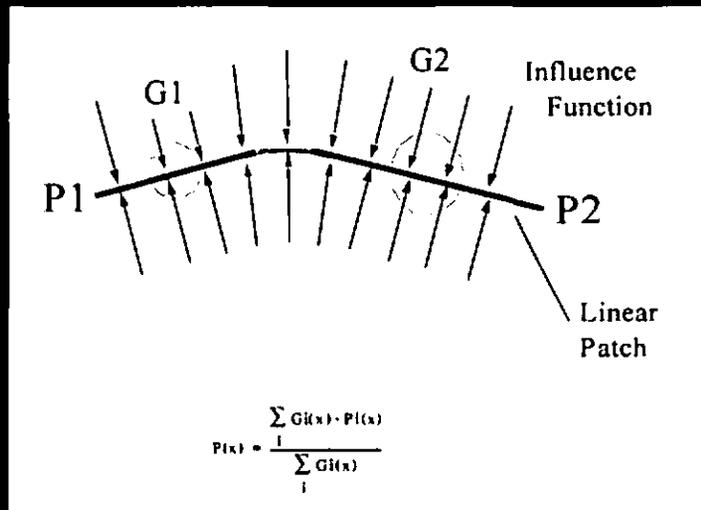


Manifold Learning for Tracking

Chris Bregler, Steve Omohundro



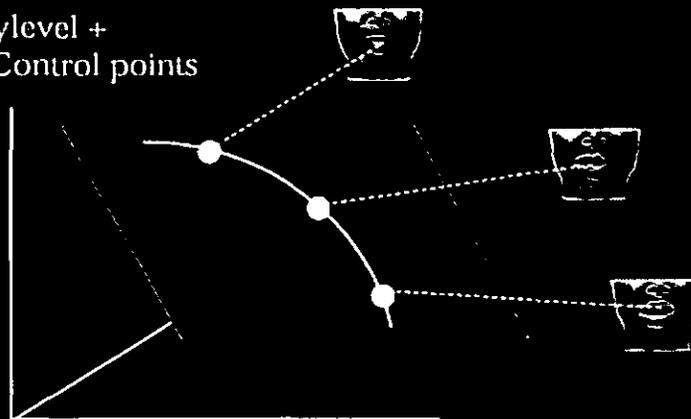
Mixture of Projections



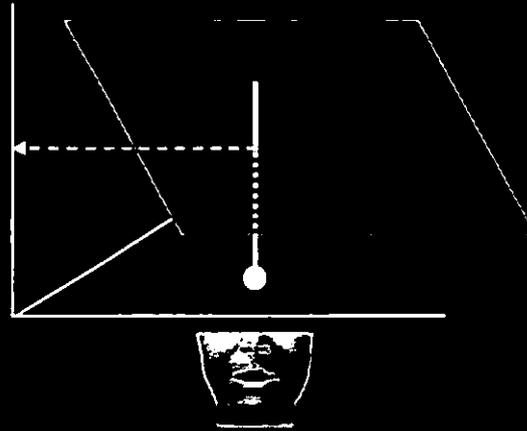
8

Eigenpoints - Training -

Graylevel +
XY Control points

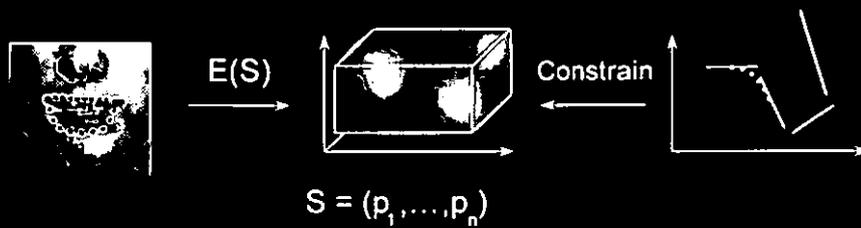


Eigenpoints - Mapping -



Graylevel +
XY Control point
Space

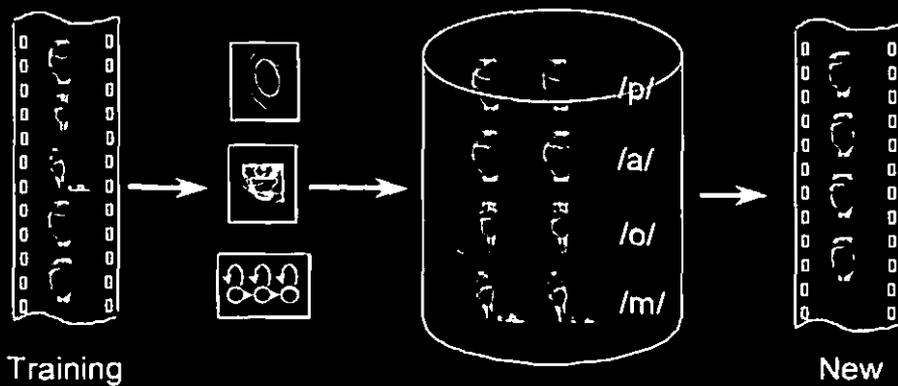
Constrained Tracking



Video

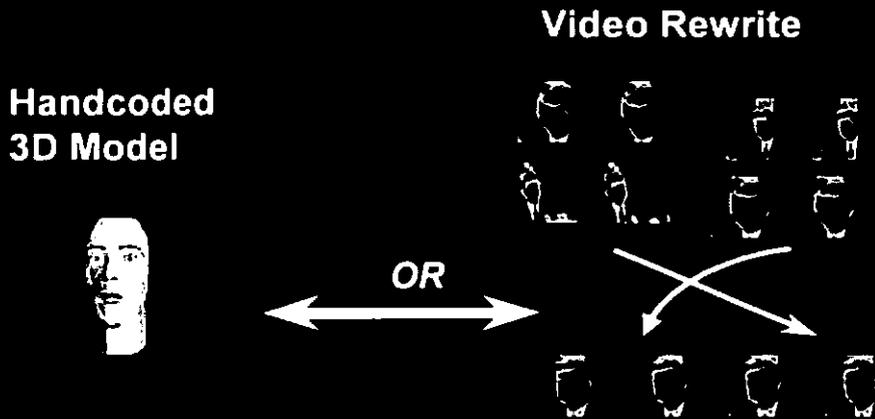
Video Rewrite

C. Bregler, M. Covell, M. Slaney
Interval Research Corp.



Goal: *Photo-realistic Talking Face*

2



Building Video Model

5

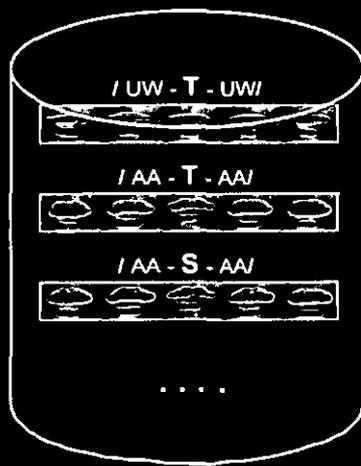
- *Phonetic*
- *Head Pose*
- *Mouth Shape*



video

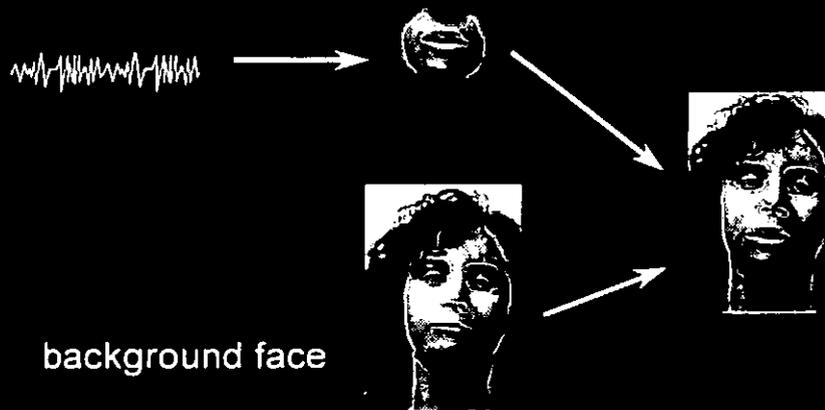
Video Model:

8 min =
1,700 triphones



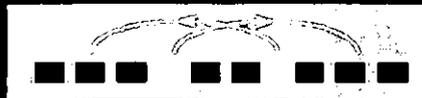
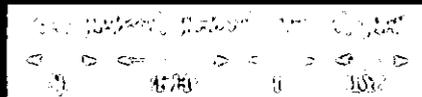
"Ellen Model"

Synthesis - Overview -

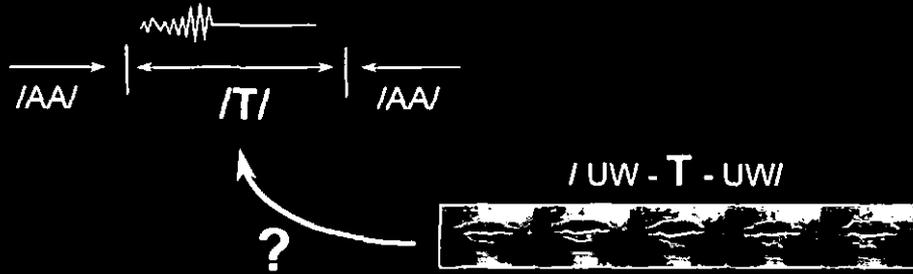


Synthesis:

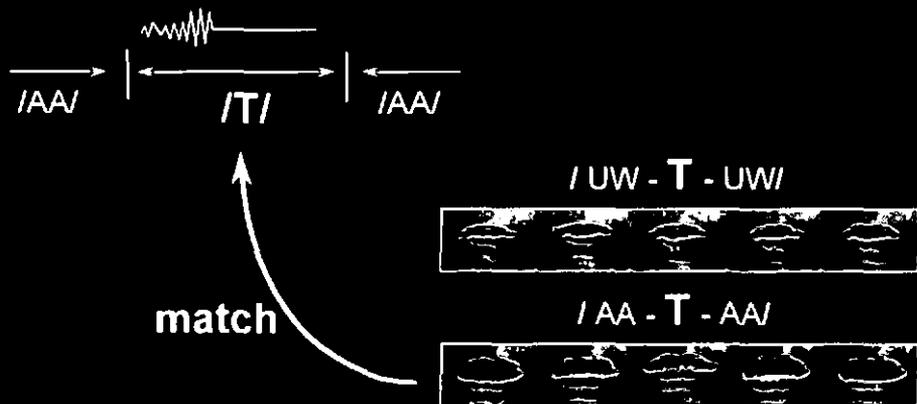
- *Transcribe*
- *Find Lip Clips*
- *Stitch Together*



Matching: *Co-Articulation*



Matching: *Co-Articulation*



Co-Articulation: *Tri-Phones*

More than
20,000 Tri-Phones
in English

/ UW - T - UW /



/ AA - T - AA /



/ AA - S - AA /



...

Matching: *Viseme-Distance*



approximate
match

/ UW - T - UW /

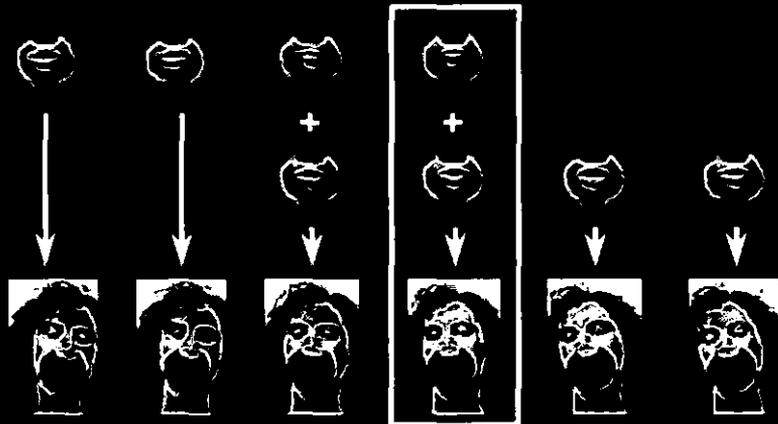


/ AA - S - AA /

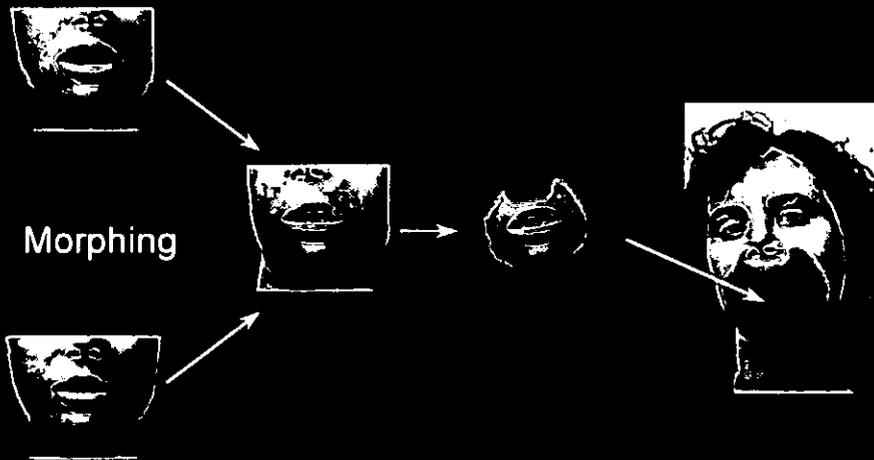


video

Stitching



Stitching



Video Rewrite Results



Ellen - Video Model

8 minutes data



JFK - Video Model

2 minutes data

Future: Virtual Actors



Challenges



Key Technologies

Learning



Video-Based
Rendering

Video Rewrite: Driving Visual Speech with Audio

Christoph Bregler, Michele Covell, Malcolm Slaney
Interval Research Corporation

ABSTRACT

Video Rewrite uses existing footage to create automatically new video of a person mouthing words that she did not speak in the original footage. This technique is useful in movie dubbing, for example, where the movie sequence can be modified to sync the actors' lip motions to the new soundtrack.

Video Rewrite automatically labels the phonemes in the training data and in the new audio track. Video Rewrite reorders the mouth images in the training footage to match the phoneme sequence of the new audio track. When particular phonemes are unavailable in the training footage, Video Rewrite selects the closest approximations. The resulting sequence of mouth images is stitched into the background footage. This stitching process automatically corrects for differences in head position and orientation between the mouth images and the background footage.

Video Rewrite uses computer-vision techniques to track points on the speaker's mouth in the training footage, and morphing techniques to combine these mouth gestures into the final video sequence. The new video combines the dynamics of the original actor's articulations with the mannerisms and setting dictated by the background footage. Video Rewrite is the first facial-animation system to automate all the labeling and assembly tasks required to resync existing footage to a new soundtrack.

CR Categories: 1.3.3 [Computer Graphics]: Picture/Image Generation—Morphing; 1.4.6 [Image Processing]: Segmentation—Feature Detection; 1.3.8 [Computer Graphics]: Applications—Facial Synthesis; 1.4.10 [Image Processing]: Applications—Feature Transformations.

Additional Keywords: Facial Animation, Lip Sync.

1 WHY AND HOW WE REWRITE VIDEO

We are very sensitive to the synchronization between speech and lip motions. For example, the special effects in *Forest Gump* are compelling because the Kennedy and Nixon footage is lip synced to the movie's new soundtrack. In contrast, close-ups in dubbed movies are often disturbing due to the lack of lip sync. Video Rewrite is a system for automatically synthesizing faces with proper lip sync. It can be used for dubbing movies, teleconferencing, and special effects.

1801 Page Mill Road, Building C, Palo Alto, CA, 94304. E-mail: bregler@cs.berkeley.edu, covell@interval.com, malcolm@interval.com. See the SIGGRAPH Video Proceedings or <http://www.interval.com/papers/1997-012/> for the latest animations.

Permission to make digital/hard copy of all or part of this material for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

© 1997 ACM.

Video Rewrite automatically pieces together from old footage a new video that shows an actor mouthing a new utterance. The results are similar to labor-intensive special effects in *Forest Gump*. These effects are successful because they start from actual film footage and modify it to match the new speech. Modifying and reassembling such footage in a smart way and synchronizing it to the new sound track leads to final footage of realistic quality. Video Rewrite uses a similar approach but does not require labor-intensive interaction.

Our approach allows Video Rewrite to learn from example footage how a person's face changes during speech. We learn what a person's mouth looks like from a video of that person speaking normally. We capture the dynamics and idiosyncrasies of her articulation by creating a database of video clips. For example, if a woman speaks out of one side of her mouth, this detail is recreated accurately. In contrast, most current facial-animation systems rely on generic head models that do not capture the idiosyncrasies of an individual speaker.

To model a new person, Video Rewrite requires a small number (26 in this work) of hand-labeled images. This is the only human intervention that is required in the whole process. Even this level of human interaction is not a fundamental requirement: We could use face-independent models instead [Kirby90, Covell96].

Video Rewrite shares its philosophy with concatenative speech synthesis [Moulines90]. Instead of modeling the vocal tract, concatenative speech synthesis analyzes a corpus of speech, selects examples of phonemes, and normalizes those examples. Phonemes are the distinct sounds within a language, such as the /Y/ and /P/ in "teapot." Concatenative speech synthesizes new sounds by concatenating the proper sequence of phonemes. After the appropriate warping of pitch and duration, the resulting speech is natural sounding. This approach to synthesis is data driven: The algorithms analyze and resynthesize sounds using little hand-coded knowledge of speech. Yet they are effective at implicitly capturing the nuances of human speech.

Video Rewrite uses a similar approach to create new sequences of visemes. Visemes are the visual counterpart to phonemes. Visemes are visually distinct mouth, teeth, and tongue articulations for a language. For example, the phonemes /B/ and /P/ are visually indistinguishable and are grouped into a single viseme class.

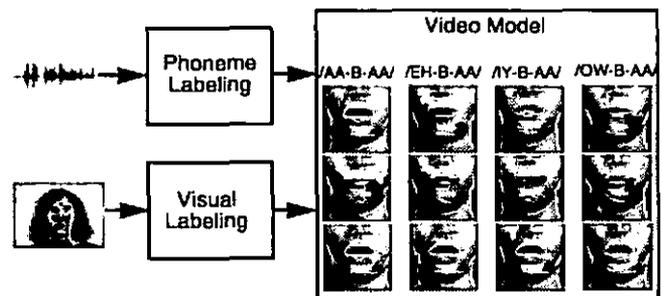


Figure 1: Overview of analysis stage. Video Rewrite uses the audio track to segment the video into triphones. Vision techniques find the orientation of the head, and the shape and position of the mouth and chin in each image. In the synthesis stage, Video Rewrite selects from this video model to synchronize new lip videos to any given audio.

Video Rewrite creates new videos using two steps: analysis of a training database and synthesis of new footage. In the *analysis* stage, Video Rewrite automatically segments into phonemes the audio track of the training database. We use these labels to segment the video track as well. We automatically track facial features in this segmented footage. The phoneme and facial labels together completely describe the visemes in the training database. In the *synthesis* stage, our system uses this video database, along with a new utterance. It automatically retrieves the appropriate viseme sequences, and blends them into a background scene using morphing techniques. The result is a new video with lip and jaw movements that synchronize to the new audio. The steps used in the analysis stage are shown in Figure 1; those of the synthesis stage are shown in Figure 2.

In the remainder of this paper, we first review other approaches to synthesizing talking faces (Section 2). We then describe the analysis and synthesis stages of Video Rewrite. In the analysis stage (Section 3), a collection of video is analyzed and stored in a database that matches sounds to video sequences. In the synthesis stage (Section 4), new speech is labeled, and the appropriate sequences are retrieved from the database. The final sections of this paper describe our results (Section 5), future work (Section 6), and contributions (Section 7).

2 SYNTHETIC VISUAL SPEECH

Facial-animation systems build a model of what a person's speech sounds and looks like. They use this model to generate a new output sequence, which matches the (new) target utterance. On the model-building side (analysis), there are typically three distinguishing choices: how the facial appearance is learned or described, how the facial appearance is controlled or labeled, and how the viseme labels are learned or described. For output-sequence generation (synthesis), the distinguishing choice is how the target utterance is characterized. This section reviews a representative sample of past research in these areas.

2.1 Source of Facial Appearance

Many facial-animation systems use a generic 3D mesh model of a face [Parke72, Lewis91, Guiard-Marigny94], sometimes adding texture mapping to improve realism [Morshima91, Cohen93, Waters95]. Another synthetic source of face data is hand-drawn images [Litwinowicz94]. Other systems use real faces for their source examples, including approaches that use 3D scans [Williams90] and still images [Scott94]. We use video footage to train Video Rewrite's models.

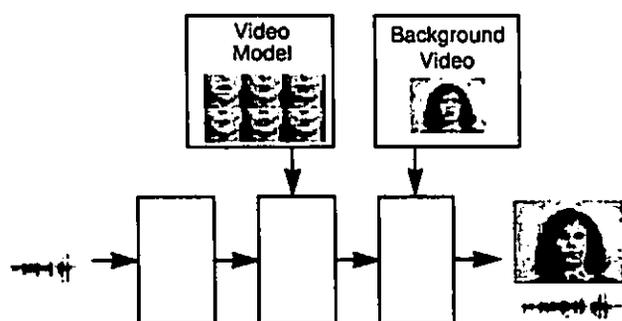


Figure 2: Overview of synthesis stage. Video Rewrite segments new audio and uses it to select triphones from the video model. Based on labels from the analysis stage, the new mouth images are morphed into a new background face.

2.2 Facial Appearance Control

Once a facial model is captured or created, the control parameters that exercise that model must be defined. In systems that rely on a 3D mesh model for appearance, the control parameters are the allowed 3D mesh deformations. Most of the image-based systems label the positions of specific facial locations as their control parameters. Of the systems that use facial-location labels, most rely on manual labeling of each example image [Scott94, Litwinowicz94]. Video Rewrite creates its video model by automatically labeling specific facial locations.

2.3 Viseme Labels

Many facial-animation systems label different visual configurations with an associated *phoneme*. These systems then match these phoneme labels with their corresponding labels in the target utterance. With synthetic images, the phoneme labels are artificial or are learned by analogy [Morshima91]. For natural images, taken from a video of someone speaking, the phonemic labels can be generated manually [Scott94] or automatically. Video Rewrite determines the phoneme labels automatically (Section 3.1).

2.4 Output-Sequence Generation

The goal of facial animation is to generate an image sequence that matches a target utterance. When phoneme labels are used, those for the target utterance can be entered manually [Scott94] or computed automatically [Lewis91, Morshima91]. Another option for phoneme labeling is to create the new utterance with synthetic speech [Parke72, Cohen93, Waters95]. Approaches that do not use phoneme labels include motion capture of facial locations that are artificially highlighted [Williams90, Guiard-Marigny94] and manual control by an animator [Litwinowicz94]. Video Rewrite uses a combination of phoneme labels (from the target utterance) and facial-location labels (from the video-model segments). Video Rewrite derives all these labels automatically.

Video Rewrite is the first facial-animation system to automate all these steps and to generate realistic lip-synched video from natural speech and natural images.

3 ANALYSIS FOR VIDEO MODELING

As shown in Figure 1, the analysis stage creates an annotated database of example video clips, derived from unconstrained footage. We refer to this collection of annotated examples as a video model. This model captures how the subject's mouth and jaw move during speech. These training videos are labeled automatically with the phoneme sequence uttered during the video, and with the locations of fiducial points that outline the lips, teeth, and jaw.

As we shall describe, the phonemic labels are from a time-aligned transcript of the speech, generated by a hidden Markov model (HMM). Video Rewrite uses the phonemic labels from the HMM to segment the input footage into short video clips, each showing three phonemes or a triphone. These triphone videos, with the fiducial-point locations and the phoneme labels, are stored in the video model.

In Sections 3.1 and 3.2, we describe the visual and acoustic analyses of the video footage. In Section 4, we explain how to use this model to synthesize new video.

3.1 Annotation Using Image Analysis

Video Rewrite uses any footage of the subject speaking. As her face moves within the frame, we need to know the mouth position and the lip shapes at all times. In the synthesis stage, we use this information to warp overlapping videos such that they have the same lip shapes, and to align the lips with the background face.

Manual labeling of the fiduciary points around the mouth and jaw is error prone and tedious. Instead, we use computer-vision techniques to label the face and to identify the mouth and its shape. A major hurdle to automatic annotation is the low resolution of the images. In a typical scene, the lip region has a width of only 40 pixels. Conventional contour-tracking algorithms [Kass87, Yuille89] work well on high-contrast outer lip boundaries with some user interaction, but fail on inner lip boundaries at this resolution, due to the low signal-to-noise ratios. Grayscale-based algorithms, such as eigenimages [Kirby90, Turk91], work well at low resolutions, but estimate only the location of the lips or jaw, rather than estimating the desired fiduciary points. Eigenpoints [Covell96], and other extensions of eigenimages [Lanitis95], estimate control points reliably and automatically, even in such low-resolution images. As shown in Figure 3, eigenpoints learns how fiduciary points move as a function of the image appearance, and then uses this model to label new footage.

Video Rewrite labels each image in the training video using a total of 54 eigenpoints: 34 on the mouth (20 on the outer boundary, 12 on the inner boundary, 1 at the bottom of the upper teeth, and 1 at the top of the lower teeth) and 20 on the chin and jaw line. There are two separate eigenpoint analyses. The first eigenspace controls the placement of the 34 fiduciary points on the mouth, using 50×40 pixels around the nominal mouth location, a region that covers the mouth completely. The second eigenspace controls the placement of the 20 fiduciary points on the chin and jaw line, using 100×75 pixels around the nominal chin-location, a region that covers the upper neck and the lower part of the face.

We created the two eigenpoint models for locating the fiduciary points from a small number of images. We hand annotated only 26 images (of 14,218 images total; about 0.2%). We extended the hand-annotated dataset by morphing pairs of annotated images to form intermediate images, expanding the original 26 to 351 annotated images without any additional manual work. We then derived eigenpoints models using this extended data set.

We use eigenpoints to find the mouth and jaw and to label their contours. The derived eigenpoint models locate the facial features using six basis vectors for the mouth and six different vectors for the jaw. Eigenpoints then places the fiduciary points around the feature locations: 32 basis vectors place points around the lips and 64 basis vectors place points around the jaw.

Eigenpoints assumes that the features (the mouth or the jaw) are undergoing pure translational motion. It does a comparatively poor job at modeling rotations and scale changes. Yet, Video Rewrite is designed to use unconstrained footage. We expect rotations and scale changes. Subjects may lean toward the camera or turn away from it, tilt their heads to the side, or look up from under their eyelashes.

To allow for a variety of motions, we warp each face image into a standard reference plane, prior to eigenpoints labeling. We

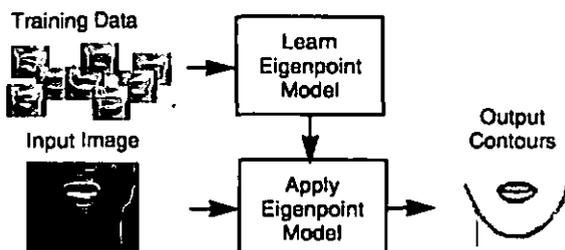


Figure 3: Overview of eigenpoints. A small set of hand-labeled facial images is used to train subspace models. Given a new image, the eigenpoint models tell us the positions of points on the lips and jaw.

find the global transform that minimizes the mean-squared error between a large portion of the face image and a facial template. We currently use an affine transform [Black95]. The mask shown in Figure 4 defines the support of the minimization integral. Once the best global mapping is found, it is inverted and applied to the image, putting that face into the standard coordinate frame. We then perform eigenpoints analysis on this pre-warped image to find the fiduciary points. Finally, we back-project the fiduciary points through the global warp to place them on the original face image.

The labels provided by eigenpoints allow us automatically to (1) build the database of example lip configurations, and (2) track the features in a background scene that we intend to modify. Section 4.2 describes how we match the points we find in step 1 to each other and to the points found in step 2.

3.2 Annotation Using Audio Analysis

All the speech data in Video Rewrite (and their associated video) are segmented into sequences of phonemes. Although single phonemes are a convenient representation for linguistic analysis, they are not appropriate for Video Rewrite. We want to capture the visual dynamics of speech. To do so correctly, we must consider *coarticulation*, which causes the lip shapes for many phonemes to be modified based on the phoneme's context. For example, the /T/ in "beet" looks different from the /T/ in "boot."

Therefore, Video Rewrite segments speech and video into triphones: collections of three sequential phonemes. The word "tea-pot" is split into the sequence of triphones /SIL-T-IY/, ¹/T-IY-P/, /IY-P-AA/, /P-AA-T/, and /AA-T-SIL/. When we synthesize a video, we emphasize the middle of each triphone. We cross-fade the overlapping regions of neighboring triphones. We thus ensure that the precise transition points are not critical, and that we can capture effectively many of the dynamics of both forward and backward coarticulation.

Video Rewrite uses HMMs [Rabiner89] to label the training footage with phonemes. We trained the HMMs using the TIMIT speech database [Lamel86], a collection of 4200 utterances with phonemic transcriptions that gives the uttered phonemes and their timing. Each of the 61 phoneme categories in TIMIT is modeled with a separate three-state HMM. The emission probabilities of each state are modeled with mixtures of eight Gaussians with diagonal covariances. For robustness, we split the available data by gender and train two speaker-independent, gender-specific systems, one based on 1300 female utterances, and one based on 2900 male utterances.

We used these gender-specific HMMs to create a fine-grained phonemic transcription of our input footage, using forced Viterbi

1. /SIL/ indicates silence. Two /SIL/ in a row are used at the beginnings and ends of utterances to allow all segments—including the beginning and end—to be treated as triphones.



Figure 4: Mask used to estimate the global warp. Each image is warped to account for changes in the head's position, size, and rotation. The transform minimizes the difference between the transformed images and the face template. The mask (left) forces the minimization to consider only the upper face (right).

search [Viterbi67]. Forced Viterbi uses unaligned sentence-level transcriptions and a phoneme-level pronunciation dictionary to create a time-aligned phoneme-level transcript of the speech. From this transcript, Video Rewrite segments the video automatically into triphone videos, labels them, and includes them in the video model.

4 SYNTHESIS USING A VIDEO MODEL

As shown in Figure 2, Video Rewrite synthesizes the final lip-synched video by labeling the new speech track, selecting a sequence of triphone videos that most accurately matches the new speech utterance, and stitching these images into a background video.

The background video sets the scene and provides the desired head position and movement. The background sequence in Video Rewrite includes most of the subject's face as well as the scene behind the subject. The frames of the background video are taken from the source footage in the same order as they were shot. The head tilts and the eyes blink, based on the background frames.

In contrast, the different triphone videos are used in whatever order is needed. They simply show the motions associated with articulation. For all the animations in this paper, the triphone images include the mouth, chin, and part of the cheeks, so that the chin and jaw move and the cheeks dimple appropriately as the mouth articulates. We use illumination-matching techniques [Burt83] to avoid visible seams between the triphone and background images.

The first step in synthesis (Figure 2) is labeling the new soundtrack. We label the new utterance with the same HMM that we used to create the video-model phoneme labels. In Sections 4.1 and 4.2, we describe the remaining steps: selecting triphone videos and stitching them into the background.

4.1 Selection of Triphone Videos

The new speech utterance determines the target sequence of speech sounds, marked with phoneme labels. We would like to find a sequence of triphone videos from our database that matches this new speech utterance. For each triphone in the new utterance, our goal is to find a video example with exactly the transition we need, and with lip shapes that match the lip shapes in neighboring triphone videos. Since this goal often is not reachable, we compromise by choosing a sequence of clips that approximates the desired transitions and shape continuity.

Given a triphone in the new speech utterance, we compute a matching distance to each triphone in the video database. The matching metric has two terms: the *phoneme-context distance*, D_p , and the *distance between lip shapes* in overlapping visual triphones, D_s . The total error is

$$\text{error} = \alpha D_p + (1 - \alpha) D_s,$$

where the weight, α , is a constant that trades off the two factors.

The phoneme-context distance, D_p , is based on categorical distances between phoneme categories and between viseme classes. Since Video Rewrite does not need to create a new soundtrack (it needs only a new video track), we can cluster phonemes into viseme classes, based on their visual appearance.

We use 26 viseme classes. Ten are consonant classes: (1) /CH/, /JH/, /SH/, /ZH/; (2) /K/, /G/, /N/, /L/; (3) /T/, /D/, /S/, /Z/; (4) /P/, /B/, /M/; (5) /F/, /V/; (6) /TH/, /DH/; (7) /W/, /R/; (8) /HH/; (9) /Y/; and (10) /NG/. Fifteen are vowel classes: one each for /EH/, /EY/, /ER/, /UH/, /AA/, /AO/, /AW/, /AY/, /UW/, /OW/, /OY/, /Y/, /IH/, /AE/, /AH/. One class is for silence, /SIL/.

The phoneme-context distance, D_p , is the weighted sum of phoneme distances between the target phonemes and the video-model phonemes within the context of the triphone. If the phonemic categories are the same (for example, /P/ and /P/), then this distance is 0. If they are in different viseme classes (/P/ and /Y/), then the distance is 1. If they are in different phonemic categories but are in the same viseme class (/P/ and /B/), then the distance is a value between 0 and 1. The intraclass distances are derived from published confusion matrices [Owens85].

In D_p , the center phoneme of the triphone has the largest weight, and the weights drop smoothly from there. Although the video model stores only triphone images, we consider the triphone's original context when picking the best-fitting sequence. In current animations, this context covers the triphone itself, plus one phoneme on either side.

The second term, D_s , measures how closely the mouth contours match in overlapping segments of adjacent triphone videos. In synthesizing the mouth shapes for "teapot" we want the contours for the /Y/ and /P/ in the lip sequence used for /T-Y-P/ to match the contours for the /Y/ and /P/ in the sequence used for /Y-P-AA/. We measure this similarity by computing the Euclidean distance, frame by frame, between four-element feature vectors containing the overall lip width, overall lip height, inner lip height, and height of visible teeth.

The lip-shape distance (D_s) between two triphone videos is minimized with the correct time alignment. For example, consider the overlapping contours for the /P/ in /T-Y-P/ and /Y-P-AA/. The /P/ phoneme includes both a silence, when the lips are pressed together, and an audible release, when the lips move rapidly apart. The durations of the initial silence within the /P/ phoneme may be different. The phoneme labels do not provide us with this level of detailed timing. Yet, if the silence durations are different, the lip-shape distance for two otherwise-well-matched videos will be large. This problem is exacerbated by imprecision in the HMM phonemic labels.

We want to find the temporal overlap between neighboring triphones that maximizes the similarity between the two lip shapes. We shift the two triphones relative to each other to find the best temporal offset and duration. We then use this optimal overlap both in computing the lip-shape distance, D_s , and in cross-fading the triphone videos during the stitching step. The optimal overlap is the one that minimizes D_s while still maintaining a minimum-allowed overlap.

Since the fitness measure for each triphone segment depends on that segment's neighbors in both directions, we select the sequence of triphone segments using dynamic programming over the entire utterance. This procedure ensures the selection of the optimal segments.

4.2 Stitching It Together

Video Rewrite produces the final video by stitching together the appropriate entries from the video database. At this point, we have already selected a sequence of triphone videos that most closely matches the target audio. We need to align the overlapping lip images temporally. This internally time-aligned sequence of videos is then time aligned to the new speech utterance. Finally, the resulting sequences of lip images are spatially aligned and are stitched into the background face. We describe each step in turn.

4.2.1 Time Alignment of Triphone Videos

We have a sequence of triphone videos that we must combine to form a new mouth movie. In combining the videos, we want to maintain the dynamics of the phonemes and their transitions. We need to time align the triphone videos carefully before blending

them. If we are not careful in this step, the mouth will appear to flutter open and closed inappropriately.

We align the triphone videos by choosing a portion of the overlapping triphones where the two lips shapes are as similar as possible. We make this choice when we evaluate D_s to choose the sequence of triphone videos (Section 4.1). We use the overlap duration and shift that provide the minimum value of D_s for the given videos.

4.2.2 Time Alignment of the Lips to the Utterance

We now have a self-consistent temporal alignment for the triphone videos. We have the correct articulatory motions, in the correct order to match the target utterance, but these articulations are not yet time aligned with the target utterance.

We align the lip motions with the target utterance by comparing the corresponding phoneme transcripts. The starting time of the center phone in the triphone sequence is aligned with the corresponding label in the target transcript. The triphone videos are then stretched or compressed such that they fit the time needed between the phoneme boundaries in the target utterance.

4.2.3 Combining of the Lips and the Background

The remaining task is to stitch the triphone videos into the background sequence. The correctness of the facial alignment is critical to the success of the recombination. The lips and head are constantly moving in the triphone and background footage. Yet, we need to align them all so that the new mouth is firmly planted on the face. Any error in spatial alignment causes the mouth to jitter relative to the face—an extremely disturbing effect.

We again use the mask from Figure 4 to help us find the optimal global transform to register the faces from the triphone videos with the background face. The combined transforms from the mouth and background images to the template face (Section 3.1) give our starting estimate in this search. Re-estimating the global transform by directly matching the triphone images to the background improves the accuracy of the mapping.

We use a replacement mask to specify which portions of the final video come from the triphone images and which come from the background video. This replacement mask warps to fit the new mouth shape in the triphone image and to fit the jaw shape in the background image. Figure 5 shows an example replacement mask, applied to triphone and background images.

Local deformations are required to stitch the shape of the mouth and jaw line correctly. These two shapes are handled differently. The mouth's shape is completely determined by the triphone images. The only changes made to these mouth shapes are imposed to align the mouths within the overlapping triphone images: The lip shapes are linearly cross-faded between the shapes in the overlapping segments of the triphone videos.



Figure 5: Facial fading mask. This mask determines which portions of the final movie frames come from the background frame, and which come from the triphone database. The mask should be large enough to include the mouth and chin. These images show the replacement mask applied to a triphone image, and its inverse applied to a background image. The mask warps according to the mouth and chin motions.

The jaw's shape, on the other hand, is a combination of the background jaw line and the two triphone jaw lines. Near the ears, we want to preserve the background video's jaw line. At the center of the jaw line (the chin), the shape and position are determined completely by what the mouth is doing. The final image of the jaw must join smoothly together the motion of the chin with the motion near the ears. To do this, we smoothly vary the weighting of the background and triphone shapes as we move along the jawline from the chin towards the ears.

The final stitching process is a three-way tradeoff in shape and texture among the fade-out lip image, the fade-in lip image, and the background image. As we move from phoneme to phoneme, the relative weights of the mouth shapes associated with the overlapping triphone-video images are changed. Within each frame, the relative weighting of the jaw shapes contributed by the background image and of the triphone-video images are varied spatially.

The derived fiduciary positions are used as control points in morphing. All morphs are done with the Beier-Neely algorithm [Beier92]. For each frame of the output image we need to warp four images: the two triphones, the replacement mask, and the background face. The warping is straightforward since we automatically generate high-quality control points using the eigenpoints algorithm.

5 RESULTS

We have applied Video Rewrite to several different training databases. We recorded one video dataset specifically for our evaluations. Section 5.1 describes our methods to collect this data and create lip-sync videos. Section 5.2 evaluates the resulting videos.

We also trained video models using truncated versions of our evaluation database. Finally, we used old footage of John F. Kennedy. We present the results from these experiments in Section 5.3.

5.1 Methods

We recorded about 8 minutes of video, containing 109 sentences, of a subject narrating a fairy tale. During the reading, the subject was asked to directly face the camera for some parts (still-head video) and to move and glance around naturally for others (moving-head video). We use these different segments to study the errors in local deformations separately from the errors in global spatial registration. The subject was also asked to wear a hat during the filming. We use this landmark to provide a quantitative evaluation of our global alignment. The hat is strictly outside all our alignment masks and our eigenpoints models. Thus, having the subject wear the hat does not effect the magnitude or type of errors that we expect to see in the animations—it simply provides us with a reference marker for the position and movement of her head.

To create a video model, we trained the system on all the still-head footage. Video Rewrite constructed and annotated the video model with just under 3500 triphone videos automatically, using HMM labeling of triphones and eigenpoint labeling of facial contours.

Video Rewrite was then given the target sentence, and was asked to construct the corresponding image sequence. To avoid unduly optimistic results, we removed from the database the triphone videos from training sentences similar to the target. A training sentence was considered similar to the target if the two shared a phrase two or more words long. Note that Video Rewrite would not normally pare the database in this manner: Instead, it would take advantage of these coincidences. We remove the similar sentences to avoid biasing our results.

We evaluated our output footage both qualitatively and quantitatively. Our qualitative evaluation was done informally, by a panel

of observers. There are no accepted metrics for evaluating lip-synced footage. Instead, we were forced to rely on the qualitative judgements listed in Section 5.2.

Only the (global) spatial registration is evaluated quantitatively. Since our subject wore a hat that moved rigidly with her upper head, we were able to measure quantitatively our global-registration error on this footage. We did so by first warping the full frame (instead of just the mouth region) of the triphone image into the coordinate frame of the background image. If this global transformation is correct, it should overlay the two images of the hat exactly on top of one another. We measured the error by finding the offset of the correlation peak for the image regions corresponding to the front of the hat. The offset of the peak is the registration error (in pixels).

5.2 Evaluation

Examples of our output footage can be seen at <http://www.interval.com/papers/1997-012/>. The top row of Figure 6 shows example frames, extracted from these videos. This section describes our evaluation criteria and the results.

5.2.1 Lip and Utterance Synchronization

How well are the lip motions synchronized with the audio? We evaluate this measure on the still-head videos. There occasionally are visible timing errors in plosives and stops.

5.2.2 Triphone-Video Synchronization

Do the lips flutter open and closed inappropriately? This artifact usually is due to synchronization error in overlapping triphone videos. We evaluated this measure on the still-head videos. We do not see any artifacts of this type.

5.2.3 Natural Articulation

Assuming that neither of the artifacts from Sections 5.2.1 or 5.2.2 appear, do the lip and teeth articulations look natural? Unnatural-looking articulation can result if the desired sequence of phonemes is not available in the database, and thus another sequence is used in its place. In our experiments, this replacement occurred on 31 percent of the triphone videos. We evaluated this measure on the

still-head videos. We do not see this type of error when we use the full video model. Additional experiments in this area are described in Section 5.3.1.

5.2.4 Fading-Mask Visibility and Extent

Does the fading mask show? Does the animation have believable texture and motion around the lips and chin? Do the dimples move in sync with the mouth? We evaluated this measure on all the output videos. The still-head videos better show errors associated with the extent of the fading mask, whereas the moving-head videos better show errors due to interactions between the fading mask and the global transformation. Without illumination correction, we see artifacts in some of the moving-head videos, when the subject looked down so that the lighting on her face changed significantly. These artifacts disappear with adaptive illumination correction [Burt83].

5.2.5 Background Warping

Do the outer edges of the jaw line and neck, and the upper portions of the cheeks look realistic? Artifacts in these areas are due to incorrect warping of the background image or to a mismatch between the texture and the warped shape of the background image. We evaluated this measure on all the output videos. In some segments, we found minor artifacts near the outer edges of the jaw.

5.2.6 Spatial Registration

Does the mouth seem to float around on the face? Are the teeth rigidly attached to the skull? We evaluated this measure on the moving-head videos. No registration errors are visible.

We evaluated this error quantitatively as well, using the hat-registration metric described in Section 5.1. The mean, median, and maximum errors in the still-head videos were 0.6, 0.5, and 1.2 pixels (standard deviation 0.3); those in the moving-head videos were 1.0, 1.0, and 2.0 pixels (standard deviation 0.4). For comparison, the face covers approximately 85×120 pixels.

5.2.7 Overall Quality

Is the lip-sync believable? We evaluated this measure on all the output videos. We judged the overall quality as excellent.

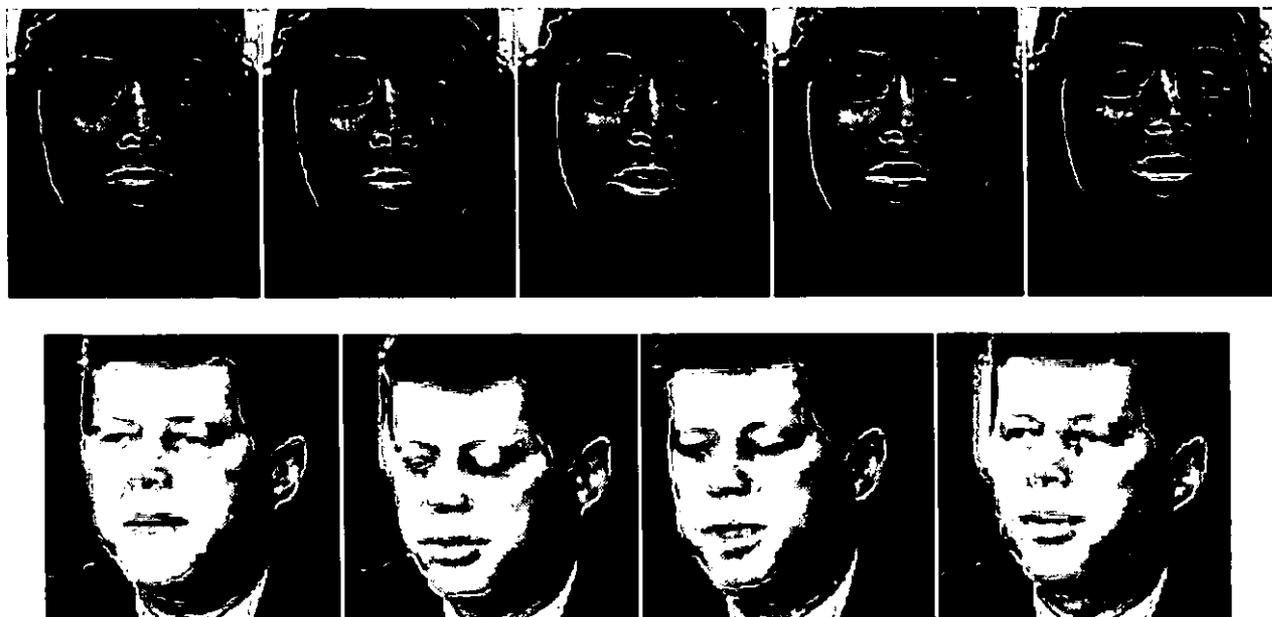


Figure 6: Examples of synthesized output frames. These frames show the quality of our output after triphone segments have been stitched into different background video frames.

5.3 Other Experiments

In this section, we examine our performance using steadily smaller training databases (Section 5.3.1) and using historic footage (Section 5.3.2).

5.3.1 Reduction of Video Model Size

We wanted to see how the quality fell off as the number of data available in the video model were reduced. With the 8 minutes of video, we have examples of approximately 1700 different triphones (of around 19,000 naturally occurring triphones); our animations used triphones other than the target triphones 31 percent of the time. What happens when we have only 1 or 2 minutes of data? We truncated our video database to one-half, one-quarter, and one-eighth of its original size, and then reanimated our target sentences. The percent of mismatched triphones increased by about 15 percent with each halving of the database (that is, 46, 58, and 74 percent of the triphones were replaced in the reduced datasets). The perceptual quality also degraded smoothly as the database size was reduced. The video from the reduced datasets are shown on our web site.

5.3.2 Reanimation of Historic Footage

We also applied Video Rewrite to public-domain footage of John F. Kennedy. For this application, we digitized 2 minutes (1157 triphones) of Kennedy speaking during the Cuban missile crisis. Forty-five seconds of this footage are from a close-up camera, about 30 degrees to Kennedy's left. The remaining images are medium shots from the same side. The size ratio is approximately 5:3 between the close-up and medium shots. During the footage, Kennedy moves his head about 30 degrees vertically, reading his speech from notes on the desk and making eye contact with a center camera (which we do not have).

We used this video model to synthesize new animations of Kennedy saying, for example, "Read my lips" and "I never met Forrest Gump." These animations combine the footage from both camera shots and from all head poses. The resulting videos are shown on our web site. The bottom row of Figure 6 shows example frames, extracted from these videos.

In our preliminary experiments, we were able to find the correct triphone sequences just 6% of the time. The lips are reliably synchronized to the utterance. The fading mask is not visible, nor is the background warping. However, the overall animation quality is not as good as our earlier results. The animations include some lip fluttering, because of the mismatched triphone sequences.

Our quality is limited for two reasons. The available viseme footage is distributed over a wide range of vertical head rotations. If we choose triphones that match the desired pose, then we cannot find good matches for the desired phoneme sequence. If we choose triphones that are well matched to the desired phoneme sequence, then we need to dramatically change the pose of the lip images. A large change in pose is difficult to model with our global (affine) transform. The lip shapes are distorted because we assumed, implicitly in the global transform, that the lips lie on a flat plane. Both the limited-triphone and pose problems can be avoided with additional data.

6 FUTURE WORK

There are many ways in which Video Rewrite could be extended and improved. The phonemic labeling of the triphone and background footage could consider the mouth- and jaw-shape information, as well as acoustic data [Bregler95]. Additional lip-image data and multiple eigenpoints models could be added, allowing larger out-of-plane head rotations. The acoustic data could be used in selecting the triphone videos, because facial expressions affect

voice qualities (you can hear a smile). The synthesis could be made real-time, with low-latency.

In Sections 6.1 through 6.3, we explore extensions that we think are most promising and interesting.

6.1 Alignment Between Lips and Target

We currently use the simplest approach to time aligning the lip sequences with the target utterance: We rely on the phoneme boundaries. This approach provides a rough alignment between the motions in the lip sequence and the sounds in the target utterance. As we mentioned in Section 4.1, however, the phoneme boundaries are both imprecise (the HMM alignment is not perfect) and coarse (significant visual and auditory landmarks occur within single phonemes).

A more accurate way to time align the lip motions with the target utterance uses dynamic time warping of the audio associated with each triphone video to the corresponding segment of the target utterance. This technique would allow us to time align the auditory landmarks from the triphone videos with those of the target utterance, even if the landmarks occur at subphoneme resolution. This time alignment, when applied to the triphone image sequence, would then align the visual landmarks of the lip sequence with the auditory landmarks of the target utterance.

The overlapping triphone videos would provide overlapping and conflicting time warpings. Yet we want to keep fixed the time alignment of the overlapping triphone videos, as dictated by the visual distances (Section 4.1 and 4.2). Research is needed in how best to trade off these potentially conflicting time-alignment maps.

6.2 Animation of Facial Features

Another promising extension is animation of other facial parts, based on simple acoustic features or other criteria. The simplest version of this extension would change the position of the eyebrows with pitch [Ohalo94]. A second extension would index the video model by both triphone and expression labels. Using such labels, we would select smiling or frowning lips, as desired. Alternatively, we could impose the desired expression on a neutral mouth shape, for those times when the appropriate combinations of triphones and expression are not available. To do this imposition correctly, we must separate which deformations are associated with articulations, and which are associated with expressions, and how the two interact. This type of factorization must be learned from examples [Tenenbaum97].

6.3 Perception of Lip Shapes

In doing this work, we solved many problems—automatic labeling, matching, and stitching—yet we found many situations where we did not have sufficient knowledge of how people perceive speaking faces. We would like to know more about how important the correct lip shapes and motions are in lip synching. For example, one study [Owens85] describes the confusibility of consonants in vowel-consonant-vowel clusters. The clustering of consonants into viseme class depends on the surrounding vowel context. Clearly, we need more sophisticated distance metrics within and between viseme classes.

7 CONTRIBUTIONS

Video Rewrite is a facial animation system that is driven by audio input. The output sequence is created from real video footage. It combines background video footage, including natural facial movements (such as eye blinks and head motions) with natural footage of mouth and chin motions. Video Rewrite is the first facial-animation system to automate all the audio- and video-labeling tasks required for this type of reanimation.

Video Rewrite can use images from unconstrained footage both to create the video model of the mouth and chin motions and to provide a background sequence for the final output footage. It preserves the individual characteristics of the subject in the original footage, even while the subject appears to mouth a completely new utterance. For example, the temporal dynamics of John F. Kennedy's articulatory motions can be preserved, reorganized, and reimposed on Kennedy's face.

Since Video Rewrite retains most of the background frame, modifying only the mouth area, it is well suited to applications such as movie dubbing. The setting and action are provided by the background video. Video Rewrite maintains an actor's visual mannerisms, using the dynamics of the actor's lips and chin from the video model for articulatory mannerisms, and using the background video for all other mannerisms. It maintains the correct timing, using the action as paced by the background video and speech as paced by the new soundtrack. It undertakes the entire process without manual intervention. The actor convincingly mouths something completely new.

ACKNOWLEDGMENTS

Many colleagues helped us. Ellen Tauber and Marc Davis graciously submitted to our experimental manipulation. Trevor Darrell and Subutai Ahmad contributed many good ideas to the algorithm development. Trevor, Subutai, John Lewis, Bud Lassiter, Gaile Gordon, Kris Rahardja, Michael Bajura, Frank Crow, Bill Verplank, and John Woodfill helped us to evaluate our results and the description. Bud Lassiter and Chris Seguin helped us with the video production. We offer many thanks to all.

REFERENCES

- [Beier92] T. Beier, S. Neely. Feature-based image metamorphosis. *Computer Graphics*, 26(2):35-42, 1992. ISSN 0097-8930.
- [Black95] M.J. Black, Y. Yacoob. Tracking and recognizing rigid and non-rigid facial motions using local parametric models of image motion. *Proc. IEEE Int. Conf. Computer Vision*, Cambridge, MA, pp. 374-381, 1995. ISBN 0-8186-7042-8.
- [Bregler95] C. Bregler, S. Omohundro. Nonlinear manifold learning for visual speech recognition. *Proc. IEEE Int. Conf. Computer Vision*, Cambridge, MA, pp. 494-499, 1995. ISBN 0-8186-7042-8.
- [Burt83] P.J. Burt, E.H. Adelson. A multiresolution spline with application to image mosaics. *ACM Trans. Graphics*, 2(4):217-236, 1983. ISSN 0730-0301.
- [Cohen93] M.M. Cohen, D.W. Massaro. Modeling coarticulation in synthetic visual speech. In *Models and Techniques in Computer Animation*, ed. N.M. Thalmann, D. Thalmann, pp. 139-156, Tokyo: Springer-Verlag, 1993. ISBN 0-3877-0124-9.
- [Covell96] M. Covell, C. Bregler. Eigenpoints. *Proc. Int. Conf. Image Processing*, Lausanne, Switzerland, Vol. 3, pp. 471-474, 1996. ISBN 0-7803-3258-x.
- [Guiard-Marigny94] T. Guiard-Marigny, A. Adjoudani, C. Benoit. A 3-D model of the lips for visual speech synthesis. *Proc. ESCA/IEEE Workshop on Speech Synthesis*, New Paltz, NY, pp. 49-52, 1994.
- [Kass87] M. Kass, A. Witkin, D. Terzopoulos. Snakes: Active contour models. *Int. J. Computer Vision*, 1(4):321-331, 1987. ISSN 0920-5691.
- [Kirby90] M. Kirby, L. Sirovich. Application of the Karhunen-Loeve procedure for the characterization of human faces. *IEEE PAMI*, 12(1):103-108, Jan. 1990. ISSN 0162-8828.
- [Lamcl86] L. F. Lamel, R. H. Kessel, S. Seneff. Speech database development: Design and analysis of the acoustic-phonetic corpus. *Proc. Speech Recognition Workshop (DARPA)*, Report #SAIC-86/1546, pp. 100-109, McLean VA: Science Applications International Corp., 1986.
- [Lanitis95] A. Lanitis, C.J. Taylor, T.F. Cootes. A unified approach for coding and interpreting face images. *Proc. Int. Conf. Computer Vision*, Cambridge, MA, pp. 368-373, 1995. ISBN 0-8186-7042-8.
- [Lewis91] J.Lewis. Automated lip-sync: Background and techniques. *J. Visualization and Computer Animation*, 2(4):118-122, 1991. ISSN 1049-8907.
- [Litwinowicz94] P. Litwinowicz, L. Williams. Animating images with drawings. *SIGGRAPH 94*, Orlando, FL, pp. 409-412, 1994. ISBN 0-89791-667-0.
- [Morishima91] S. Morishima, H. Harashima. A media conversion from speech to facial image for intelligent man-machine interface. *IEEE J Selected Areas Communications*, 9 (4):594-600, 1991. ISSN 0733-8716.
- [Moulines90] E. Moulines, P. Emerard, D. Larreur, J. L. Le Saint Milon, L. Le Faucheur, F. Marty, F. Charpentier, C. Sorin. A real-time French text-to-speech system generating high-quality synthetic speech. *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, Albuquerque, NM, pp. 309-312, 1990.
- [Ohala94] J.J. Ohala. The frequency code underlies the sound symbolic use of voice pitch. In *Sound Symbolism*, ed. L. Hinton, J. Nichols, J. J. Ohala, pp. 325-347, Cambridge UK: Cambridge Univ. Press, 1994. ISBN 0-5214-5219-8.
- [Owens85] E. Owens, B. Blazek. Visemes observed by hearing-impaired and normal-hearing adult viewers. *J. Speech and Hearing Research*, 28:381-393, 1985. ISSN 0022-4685.
- [Parke72] F. Parke. Computer generated animation of faces. *Proc. ACM National Conf.*, pp. 451-457, 1972.
- [Rabiner89] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Readings in Speech Recognition*, ed. A. Waibel, K. F. Lee, pp. 267-296, San Mateo, CA: Morgan Kaufmann Publishers, 1989. ISBN 1-5586-0124-4.
- [Scout94] K.C. Scott, D.S. Kagels, S.H. Watson, H. Rom, J.R. Wright, M. Lee, K.J. Hussey. Synthesis of speaker facial movement to match selected speech sequences. *Proc. Australian Conf. Speech Science and Technology*, Perth Australia, pp. 620-625, 1994. ISBN 0-8642-2372-2.
- [Tenenbaum97] J. Tenenbaum, W. Freeman. Separable mixture models: Separating style and content. In *Advances in Neural Information Processing 9*, ed. M. Jordan, M. Mozer, T. Patsche, Cambridge, MA: MIT Press, (in press).
- [Turk91] M. Turk, A. Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3(1):71-86, 1991. ISSN 0898-929X
- [Viterbi67] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Informat. Theory*, IT-13:260-269, 1967. ISSN 0018-9448.
- [Waters95] K. Waters, T. Levergood. DECface: A System for Synthetic Face Applications. *J. Multimedia Tools and Applications*, 1 (4):349-366, 1995. ISSN 1380-7501.
- [Williams90] L. Williams. Performance-Driven Facial Animation. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24(4):235-242, 1990. ISSN 0097-8930.
- [Yuille89] A.L. Yuille, D.S. Cohen, P.W. Hallinan. Feature extraction from faces using deformable templates. *Proc. IEEE Computer Vision and Pattern Recognition*, San Diego, CA, pp. 104-109, 1989. ISBN 0-8186-1952-x.

Video Motion Capture

Christoph Bregler

Jitendra Malik

Computer Science Division

University of California, Berkeley

Berkeley, CA 94720-1776

bregler@cs.berkeley.edu, malik@cs.berkeley.edu

Abstract

This paper demonstrates a new vision based motion capture technique that is able to recover high degree-of-freedom articulated human body configurations in complex video sequences. It does not require any markers, body suits, or other devices attached to the subject. The only input needed is a video recording of the person whose motion is to be captured. For visual tracking we introduce the use of a novel mathematical technique, the product of exponential maps and twist motions, and its integration into a differential motion estimation. This results in solving simple linear systems, and enables us to recover robustly the kinematic degrees-of-freedom in noise and complex self occluded configurations. We demonstrate this on several image sequences of people doing articulated full body movements, and visualize the results in re-animating an artificial 3D human model. We are also able to recover and re-animate the famous movements of Eadweard Muybridge's motion studies from the last century. To the best of our knowledge, this is the first computer vision based system that is able to process such challenging footage and recover complex motions with such high accuracy.

CR Categories:

Keywords: Computer Vision, Animation, Motion Capture, Visual Tracking, Twist Kinematics, Exponential Maps, Muybridge

1 Introduction

In this paper, we offer a new approach to motion capture based just on ordinary video recording of the actor performing naturally. The approach does not require any markers, body suits or any other devices attached to the body of the actor. The actor can move about wearing his or her regular clothes. This implies that one can use historical footage—motion capture Charlie Chaplin's inimitable walk, for instance. Indeed in this paper we shall go even further back historically and show motion capture results from Muybridge sequences—the first examples of photographically recorded motion [15].

Motion capture occupies an important role in the creation of special effects. Its application to CG character animation has been much more controversial; SIGGRAPH 97 featured a lively panel debate[4] between its proponents and opponents. Our goal in this paper is not to address that debate. Rather we take it as a given that motion capture, like any other technology, can be correctly or incorrectly applied and we are merely extending its possibilities.

Our approach, from a user's point of view, is rather straightforward. The user marks limb segments in an initial frame; if multiple video streams are available from synchronized cameras, then the limb segments are marked in the corresponding initial frames in all of them. The computer program does the rest—tracking the multiple degrees of freedom of the human body configuration from frame to frame.

Attempts to track the human body without special markers go back quite a few years – we review past work in Sec. 2. However in spite of many years of work in computer vision on this problem, it is fair to describe it as not yet solved. There are many reasons why human body tracking is very challenging, compared to tracking other objects such as footballs, robots or cars. These include

1. *High Accuracy Requirements.* Especially in the context of motion capture applications, one desires to record all the degrees of freedom of the configuration of arms, legs, torso, head etc accurately from frame to frame. At playback time, any error will be instantly noticed by a human observer.
2. *Frequent inter-part occlusion* During normal motion, from any camera angle some parts of the body are occluded by other parts of the body
3. *Lack of contrast* Distinguishing the edge of a limb from, say the torso underneath, is made difficult by the fact that typically the texture or color of the shirt is usually the same in both regions.

Our contribution to this problem is the introduction of a novel mathematical technique, the product of exponential maps and twist motions, and its integration into a differential motion estimation scheme. This formalism will be explained fully in Section 3. The advantage of this particular formulation is that it results in the equations that need to be solved to update the kinematic chain parameters from frame to frame being *linear*. Also the only parameters that need to be solved for are the true degrees of freedom and pose parameters—there are no intermediate stages which may be unnecessarily hard. For instance recovering the local affine motion parameters of each and every limb segment separately is harder than the final goal of knowing the configuration of all the joints from frame to frame—the fact that the joints are constrained to move together reduces considerably the number of degrees of freedom. This in turn provides robustness to self-occlusions, loss of contrast, large motions etc.

We applied this technique to several video recordings of walking people and to the famous photo plates of Eadweard Muybridge. We achieved accurate tracking results with high degree-of-freedom full body models and could successfully re-animate the data. The accompanying video shows the tracking results and the naturalness of the animated motion capture data.

Section 2 reviews previous video tracking techniques, section 3 introduces the new motion tracking framework and its mathematical formulation, section 4 details our experiments, and we discuss the results and future directions in section 5.

2 Review

The earliest computer vision attempt to recognize human movements was reported by O'Rourke and Badler [16] working on syn-

thetic images using a 3D structure of rigid segments, joints, and constraints between them.

Marker-free visual tracking on video recordings of human bodies goes back to work by Hogg and by Rohr [8, 18]. Both systems are specialized to one degree-of-freedom walking models. Edge and line features are extracted from images and matched to a cylindrical 3D human body model. Higher degree-of-freedom articulated hand configurations are tracked by Regh and Kanade [17], full body configurations by Gravira and Davis [7], and arm configurations by Kakadiaris and Metaxas [11] and by Goncalves and Perona [5]. All these approaches are demonstrated in constrained environments with high contrast edge boundaries. In most cases this is achieved by uniform backgrounds, and skintight clothing of uniform color. Also, in order to estimate 3D configurations, a camera calibration is needed. Alternatively, Weng et. al demonstrated how to track full bodies with color features [20], and Ju et. al showed motion based tracking of leg configurations [10]. No 3D kinematic chain models were used in the last two cases.

To the best of our knowledge, there is no system reported so far, which would be able to successfully track accurate high-degree-of-freedom human body configurations in the challenging footage that we will demonstrate here.

3 Articulated Tracking

There exist a wide range of visual tracking techniques in the literature ranging from edge feature based to region based tracking, and brute-force search methods to differential approaches.

Edge feature based tracking techniques usually require clean data with high contrast object boundaries. Unfortunately on human bodies such features are very noisy. Clothes have many folds. Also if the left and right leg have the same color and they overlap, they are separated only by low contrast boundaries.

Region based techniques can track objects with arbitrary texture. Such techniques attempt to match areas between consecutive frames. For example if the area describes a rigid planar object, a 2D affine deformation of this area has to be found. This requires the estimation of 6 free parameters that describe this deformation (x/y translation, x/y scaling, rotation, and shear). Instead of exhaustively searching over these parameters, differential methods link local intensity changes to parameter changes, and allow for Newton-step like optimizations.

In the following we will introduce a new region based differential technique that is tailored to articulated objects modeled by kinematic chains. We will first review a commonly used motion estimation framework [2, 19], and then show how this can be extended for our task, using the twist and product of exponential formulation [14].

3.1 Preliminaries

Assuming that changes in image intensity are only due to translation of local image intensity, a parametric image motion between consecutive time frames t and $t + 1$ can be described by the following equation:

$$I(x + u_x(x, y, \phi), y + u_y(x, y, \phi), t + 1) = I(x, y, t) \quad (1)$$

$I(x, y, t)$ is the image intensity. The motion model $u(x, y, \phi) = [u_x(x, y, \phi), u_y(x, y, \phi)]^T$ describes the pixel displacement dependent on location (x, y) and model parameters ϕ . For example, a 2D affine motion model with parameters $\phi = [a_1, a_2, a_3, a_4, d_x, d_y]^T$ is defined as

$$u(x, y, \phi) = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix} \quad (2)$$

The first-order Taylor series expansion of (1) leads to the commonly used gradient formulation [12]:

$$I_t(x, y) + [I_x(x, y), I_y(x, y)] \cdot u(x, y, \phi) = 0 \quad (3)$$

$I_t(x, y)$ is the temporal image gradient and $[I_x(x, y), I_y(x, y)]$ is the spatial image gradient at location (x, y) . Assuming a motion model of K degrees of freedom (in case of the affine model $K = 6$) and a region of $N > K$ pixels, we can write an over-constrained set of N equations. For the case that the motion model is linear (as in the affine case), we can write the set of equations in matrix form (see [2] for details):

$$\mathbf{H} \cdot \phi + \vec{z} = \vec{0} \quad (4)$$

where $\mathbf{H} \in \mathbb{R}^{N \times K}$, and $\vec{z} \in \mathbb{R}^N$. The least squares solution to (3) is:

$$\phi = -(\mathbf{H}^T \cdot \mathbf{H})^{-1} \cdot \mathbf{H}^T \vec{z} \quad (5)$$

Because (4) is the first-order Taylor series linearization of (1), we linearize around the new solution and iterate. This is done by warping the image $I(t + 1)$ using the motion model parameters ϕ found by (5). Based on the re-warped image we compute the new image gradients (3). Repeating this process is equivalent to a Newton-Raphson style minimization.

A convenient representation of the shape of an image region is a probability mask $w(x, y) \in [0, 1]$. $w(x, y) = 1$ declares that pixel (x, y) is part of the region. Equation (5) can be modified, such that it weights the contribution of pixel location (x, y) according to $w(x, y)$:

$$\phi = -((\mathbf{W} \cdot \mathbf{H})^T \cdot \mathbf{H})^{-1} \cdot (\mathbf{W} \cdot \mathbf{H})^T \vec{z} \quad (6)$$

\mathbf{W} is an $N \times N$ diagonal matrix, with $\mathbf{W}(i, i) = w(x_i, y_i)$. We assume for now that we know the exact shape of the region. For example, if we want to estimate the motion parameters for a human body part, we supply a weight matrix \mathbf{W} that defines the image support map of that specific body part, and run this estimation technique for several iterations. Section 3.4 describes how we can estimate the shape of the support maps as well.

Tracking over multiple frames can be achieved by applying this optimization technique successively over the complete image sequence.

3.2 Twists and the Product of Exponential Formula

In the following we develop a motion model $u(x, y, \phi)$ for a 3D kinematic chain under scaled orthographic projection and show how these domain constraints can be incorporated into one linear system similar to (6). ϕ will represent the 3D pose and angle configuration of such a kinematic chain and can be tracked in the same fashion as already outlined for simpler motion models.

3.2.1 3D pose

The pose of an object relative to the camera frame can be represented as a rigid body transformation in \mathbb{R}^3 using homogeneous coordinates (we will use the notation from [14]):

$$q_c = \mathbf{G} \cdot q_o \quad \text{with } \mathbf{G} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & d_x \\ r_{2,1} & r_{2,2} & r_{2,3} & d_y \\ r_{3,1} & r_{3,2} & r_{3,3} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

$q_o = [x_o, y_o, z_o, 1]^T$ is a point in the object frame and $q_c = [x_c, y_c, z_c, 1]^T$ is the corresponding point in the camera frame. Using scaled orthographic projection with scale s , the point q_c in the camera frame gets projected into the image point $[x_{im}, y_{im}]^T = s \cdot [x_c, y_c]^T$.

The 3D translation $[d_x, d_y, d_z]^T$ can be arbitrary, but the rotation matrix:

$$\mathbf{R} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{bmatrix} \in SO(3) \quad (8)$$

has only 3 degrees of freedom. Therefore the rigid body transformation $\mathbf{G} \in SE(3)$ has a total of 6 degrees of freedom.

Our goal is to find a model of the image motion that is parameterized by 6 degrees of freedom for the 3D rigid motion and the scale factor s for scaled orthographic projection. *Euler angles* are commonly used to constrain the rotation matrix to $SO(3)$, but they suffer from singularities and don't lead to a simple formulation in the optimization procedure (for example [1] propose a 3D ellipsoidal tracker based on Euler angles). In contrast, the *twist* representation provides a more elegant solution [14] and leads to a very simple linear representation of the motion model. It is based on the observation that every rigid motion can be represented as a rotation around a 3D axis and a translation along this axis. A twist ξ has two representations: (a) a 6D vector, or (b) a 4×4 matrix with the upper 3×3 component as a skew-symmetric matrix:

$$\xi = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \quad \text{or} \quad \hat{\xi} = \begin{bmatrix} 0 & -\omega_x & \omega_y & v_1 \\ \omega_x & 0 & -\omega_z & v_2 \\ -\omega_y & \omega_z & 0 & v_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (9)$$

ω is a 3D unit vector that points in the direction of the rotation axis. The amount of rotation is specified with a scalar angle θ that is multiplied by the twist: $\xi\theta$. The v component determines the location of the rotation axis and the amount of translation along this axis. See [14] for a detailed geometric interpretation. For simplicity, we drop the constraint that ω is unit, and discard the θ coefficient. Therefore $\xi \in \mathbb{R}^6$.

It can be shown [14] that for any arbitrary $\mathbf{G} \in SE(3)$ there exists a $\xi \in \mathbb{R}^6$ twist representation.

A twist can be converted into the \mathbf{G} representation with following exponential map:

$$\mathbf{G} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & d_x \\ r_{2,1} & r_{2,2} & r_{2,3} & d_y \\ r_{3,1} & r_{3,2} & r_{3,3} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ = e^{\hat{\xi}} = \mathbf{I} + \hat{\xi} + \frac{(\hat{\xi})^2}{2!} + \frac{(\hat{\xi})^3}{3!} + \dots \quad (10)$$

3.2.2 Twist motion model

At this point we would like to track the 3D pose of a rigid object under scaled orthographic projection. We will extend this formulation in the next section to a kinematic chain representation. The pose of an object is defined as $[s, \xi^T]^T = [s, v_1, v_2, v_3, \omega_x, \omega_y, \omega_z]^T$. A point q_o in the object frame is projected to the image location (x_{im}, y_{im}) with:

$$\begin{bmatrix} x_{im} \\ y_{im} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot s \cdot e^{\hat{\xi}} \cdot q_o \quad (11)$$

The image motion of point (x_{im}, y_{im}) from time t to time $t+1$ is:

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} x_{im}(t+1) - x_{im}(t) \\ y_{im}(t+1) - y_{im}(t) \end{bmatrix} \\ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \left(s(t+1) \cdot e^{\hat{\xi}(t+1)} \cdot q_o - s(t) \cdot e^{\hat{\xi}(t)} \cdot q_o \right) \\ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \left((1+s') \cdot e^{\hat{\xi}'} - \mathbf{I} \right) \cdot s(t) q_c \quad (12)$$

$$\text{with } \hat{\xi}(t+1) = \hat{\xi}(t) + \hat{\xi}' \\ s(t+1) = s(t) \cdot (1+s')$$

Using the first order Taylor expansion from (10) we can approximate:

$$(1+s') \cdot e^{\hat{\xi}'} \approx (1+s') \cdot \mathbf{I} + (1+s') \cdot \hat{\xi}' \quad (13)$$

and can rewrite (12) as:

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} s' & -\omega'_x & \omega'_y & v'_1 \\ \omega'_x & s' & -\omega'_z & v'_2 \end{bmatrix} \cdot q_c \quad (14)$$

with

$$\omega(t+1) = \omega(t) + \frac{1}{1+s'} \cdot \omega' \\ v(t+1) = v(t) + \frac{1}{1+s'} \cdot v'$$

$\phi = [s', v'_1, v'_2, \omega'_x, \omega'_y, \omega'_z]^T$ codes the relative scale and twist motion from time t to $t+1$. Note that (14) does not include v'_3 . Translation in the Z direction of the camera frame is not measurable under scaled orthographic projection.

Equation (14) describes the image motion of a point (x_i, y_i) in terms of the motion parameters ϕ and the corresponding 3D point $q_c(i)$ in the camera frame. The 3D point $q_c(i)$ is computed by intersecting the camera ray of the image point (x_i, y_i) with the 3D model. In this paper we assume that the body segments can be approximated by ellipsoidal 3D blobs. Therefore q_c is the solution of a quadratic equation. This computation has to be done only once for each new image. It is outside the Newton-Raphson iterations. It could be replaced by more complex models and rendering algorithms.

Inserting (14) into (3) leads to:

$$I_t + I_x \cdot [s', -\omega'_x, \omega'_y, v'_1] \cdot q_c + I_y \cdot [\omega'_x, s', -\omega'_z, v'_2] \cdot q_c = 0 \\ \Leftrightarrow I_t(i) + H_i \cdot [s, v'_1, v'_2, \omega'_x, \omega'_y, \omega'_z]^T = 0 \quad (15)$$

with $I_t := I_t(x_i, y_i)$, $I_x := I_x(x_i, y_i)$, $I_y := I_y(x_i, y_i)$

For N pixel positions we have N equations of the form (15). This can be written in matrix form:

$$\mathbf{H} \cdot \phi + \bar{z} = 0 \quad (16)$$

with

$$\mathbf{H} = \begin{bmatrix} H_1 \\ H_2 \\ \dots \\ H_N \end{bmatrix} \quad \text{and} \quad \bar{z} = \begin{bmatrix} I_t(x_1, y_1) \\ I_t(x_2, y_2) \\ \dots \\ I_t(x_N, y_N) \end{bmatrix}$$

Finding the least-squares solution (3D twist motion ϕ) for this equation is done using (6).

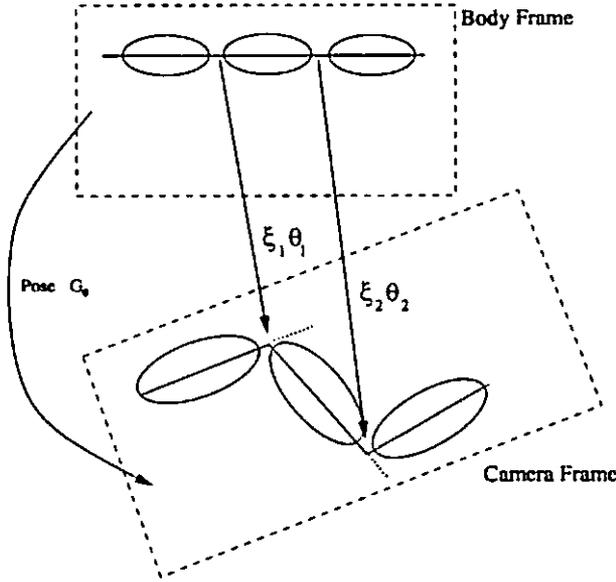


Figure 1: Kinematic chain defined by twists

3.2.3 Kinematic chain as a Product of Exponentials

So far we have parameterized the 3D pose and motion of a body segment by the 6 parameters of a twist ξ . Points on this body segment in a canonical object frame are transformed into a camera frame by the mapping $G_0 = e^{\xi}$. Assume that a second body segment is attached to the first segment with a joint. The joint can be defined by an axis of rotation in the object frame. We define this rotation axis in the object frame by a 3D unit vector ω_1 along the axis, and a point q_1 on the axis (figure 1). This is a so called revolute joint, and can be modeled by a twist ([14]):

$$\xi_1 = \begin{bmatrix} -\omega_1 \times q_1 \\ \omega_1 \end{bmatrix} \quad (17)$$

A rotation of angle θ_1 around this axis can be written as:

$$g_1 = e^{\xi_1 \cdot \theta_1} \quad (18)$$

$$(19)$$

The global mapping from object frame points on the first body segment into the camera frame is described by the following product:

$$\begin{aligned} g(\theta_1) &= G_0 \cdot e^{\xi_1 \cdot \theta_1} \\ q_c &= g(\theta_1) \cdot q_o \end{aligned} \quad (20)$$

If we have a chain of $K+1$ segments linked with K joints (kinematic chain) and describe each joint by a twist ξ_k , a point on segment k is mapped from the object frame into the camera frame dependent on G_0 and angles $\theta_1, \theta_2, \dots, \theta_k$:

$$g_k(\theta_1, \theta_2, \dots, \theta_k) = G_0 \cdot e^{\xi_1 \cdot \theta_1} \cdot e^{\xi_2 \cdot \theta_2} \cdot \dots \cdot e^{\xi_k \cdot \theta_k} \quad (21)$$

This is called the **product of exponential maps** for kinematic chains.

The velocity of a segment k can be described with a twist V_k that is a linear combination of twists $\xi'_1, \xi'_2, \dots, \xi'_k$ and the angular velocities $\dot{\theta}_1, \dot{\theta}_2, \dots, \dot{\theta}_k$ (see [14] for the derivations):

$$\begin{aligned} V_k &= \xi'_1 \cdot \dot{\theta}_1 + \xi'_2 \cdot \dot{\theta}_2 + \dots + \xi'_k \cdot \dot{\theta}_k \\ \xi'_k &= \text{Ad}_{e^{\xi_1 \cdot \theta_1} \cdot \dots \cdot e^{\xi_{k-1} \cdot \theta_{k-1}}} \xi_k \end{aligned} \quad (22)$$

Ad_g is the adjoint transformation associated with g .¹

Given a point q_c on the k 'th segment of a kinematic chain, its motion vector in the image is related to the angular velocities by:

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot [\hat{\xi}'_1 \cdot \dot{\theta}_1 + \hat{\xi}'_2 \cdot \dot{\theta}_2 + \dots + \hat{\xi}'_k \cdot \dot{\theta}_k] \cdot q_c \quad (23)$$

Recall (15) relates the image motion of a point q_c to changes in pose G_0 . We combine (15) and (23) to relate the image motion to the combined vector of pose change and angular change $\Phi = [s', v'_1, v'_2, \omega'_x, \omega'_y, \omega'_z, \phi_1, \phi_2, \dots, \phi_K]^T$:

$$I_t + H_i \cdot [s, v'_1, v'_2, \omega'_x, \omega'_y, \omega'_z]^T + J_i \cdot [\dot{\theta}_1, \dot{\theta}_2, \dots, \dot{\theta}_K]^T = 0 \quad (24)$$

$$[H, J] \cdot \Phi + \bar{z} = 0 \quad (25)$$

with

$$J = \begin{bmatrix} J_1 \\ J_2 \\ \dots \\ J_N \end{bmatrix} \quad \text{and } H, \bar{z} \text{ as before}$$

$$J_i = [J_{i1}, J_{i2}, \dots, J_{iK}]$$

$$J_{ik} = \begin{cases} [I_x, I_y] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \hat{\xi}_k \cdot q_c \\ 0 & \text{if pixel } i \text{ is on a segment that} \\ & \text{is not affected by joint } \xi_k \end{cases}$$

The least squares solution to (25) is:

$$\Phi = -([H, J]^T \cdot [H, J])^{-1} \cdot [H, J]^T \cdot \bar{z} \quad (26)$$

Φ is the new estimate of the pose and angular change between two consecutive images. As outlined earlier, this solution is based on the assumption that the local image intensity variations can be approximated by the first-order Taylor expansion (3). We linearize around this new solution and iterate. This is done in warping the image $I(t+1)$ using the solution Φ . Based on the re-warped image we compute the new image gradients. Repeating this process of warping and solving (26) is equivalent to a Newton-Raphson style minimization.

3.3 Multiple Camera Views

In cases where we have access to multiple synchronized cameras, we can couple the different views in one equation system. Let's assume we have C different camera views at the same time. View c corresponds to following equation system (from (25)):

$$[H_c, J_c] \cdot \begin{bmatrix} \Omega_c \\ \phi_1 \\ \phi_2 \\ \dots \\ \phi_K \end{bmatrix} + \bar{z}_c = 0 \quad (27)$$

$\Omega_c = [s'_c, v'_{1,c}, v'_{2,c}, \omega'_{x,c}, \omega'_{y,c}, \omega'_{z,c}]^T$ describes the pose seen from view c . All views share the same angular parameters, because

$${}^1 \text{Ad}_g = \begin{bmatrix} R & \hat{p} \cdot R \\ 0 & R \end{bmatrix}, \text{ and } g = \begin{bmatrix} R & p \\ 000 & 1 \end{bmatrix}$$

the cameras are triggered at the same time. We can simply combine all C equation systems into one large equation system:

$$\begin{bmatrix} \mathbf{H}_1 & \mathbf{0} & \dots & \mathbf{0} & \mathbf{J}_1 \\ \mathbf{0} & \mathbf{H}_2 & \dots & \mathbf{0} & \mathbf{J}_2 \\ \dots & \dots & \dots & \dots & \dots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{H}_C & \mathbf{J}_C \end{bmatrix} \cdot \begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \dots \\ \Omega_C \\ \phi_1 \\ \phi_2 \\ \dots \\ \phi_K \end{bmatrix} + \begin{bmatrix} \tilde{z}_1 \\ \tilde{z}_2 \\ \dots \\ \tilde{z}_C \end{bmatrix} = \mathbf{0} \quad (28)$$

Operating with multiple views has three main advantages. The estimation of the angular parameters is more robust: (1) the number of measurements and therefore the number of equations increases with the number of views, (2) some angular configurations might be close to a singular pose in one view, whereas they can be estimated in a orthogonal view much better. (3) With more camera views, the chance decreases that one body part is occluded in all views.

3.4 Adaptive Support Maps using EM

As in (3), the update can be constrained to estimate the motion only in a weighted support map W_k for each segment k using:

$$\Phi = - ((W_k \cdot [H, J])^T \cdot [H, J])^{-1} \cdot (W_k \cdot [H, J])^T \tilde{z} \quad (29)$$

We approximate the shape of the body segments as ellipsoids, and can compute the support map as the projection of the ellipsoids into the image. Such a support map usually covers a larger region, including pixels from the environment. That distracts the exact motion measurement. Robust statistics would be one solution to this problem [3]. Another solution is an EM-based layered representation [6, 9]. It is beyond the scope of this paper to describe this method in detail, but we would like to outline the method briefly: We start with an initial guess of the support map (ellipsoidal projection in this case). Given the initial W_k , we compute the motion estimate Φ (M-step). Given such a Φ we can compute for each pixel location the probability that it complies with the motion model defined by Φ . We do this for each blob and the background (dominant motion) and normalize the sum of all probabilities per pixel location to 1. This results in new W_k maps that are better "tuned" to the real shape of the body segment. In this paper we repeat the EM iteration only once.

3.5 Tracking Recipe

We summarize the algorithm for tracking the pose and angles of a kinematic chain in an image sequence:

- **Input:** $I(t)$, $I(t+1)$, $G_0(t)$, $\theta_1(t)$, $\theta_2(t)$, ..., $\theta_K(t)$
(Two images and the pose and angles for the first image).
 - **Output:** $G_0(t+1)$, $\theta_1(t+1)$, $\theta_2(t+1)$, ..., $\theta_K(t+1)$.
(Pose and angles for second image).
1. Compute for each image location (x_i, y_i) in $I(t)$ the 3D point $q_c(i)$ (using ellipsoids or more complex models and rendering algorithm).
 2. Compute for each body segment the support map W_k .
 3. Set $G_0(t+1) := G_0(t)$, $\forall k: \theta_k(t+1) := \theta_k(t)$.

4. Iterate:

- (a) Compute spatiotemporal image gradients: I_t, I_x, I_y .
- (b) Estimate Φ using (29)
- (c) Update $G_0(t+1) := G_0(t+1) \cdot (1 + s') \cdot e^{\frac{\tilde{z}'}{1+s'}}$
- (d) $\forall k$ Update $\theta_k(t+1) := \theta_k(t+1) + \hat{\theta}_k$.
- (e) $\forall k$ Warp the region inside W_k of $I(t+1)$ by $G_0(t+1) \cdot g_k(t+1) \cdot (G^t) \cdot g_k(t)^{-1}$.

3.6 Initialization

The visual tracking is based on an initialized first frame. We have to know the initial pose and the initial angular configuration. If more than one view is available, all views for the first time step have to be known. A user clicks on the 2D joint locations in all views at the first time step. Given that, the 3D pose and the image projection of the matching angular configuration is found in minimizing the sum of squared differences between the projected model joint locations and the user supplied model joint locations. The optimization is done over the poses, angles, and body dimensions. Example body dimensions are "upper-leg-length", "lower-leg-length", or "shoulder-width". The dimensions and angles have to be the same in all views, but the pose can be different. Symmetry constraints, that the left and right body lengths are the same, are enforced as well. Minimizing only over angles, or only over model dimensions results in linear equations similar to what we have shown so far. Unfortunately the global minimization criteria over all parameters is a tri-linear equation system, that cannot be easily solved by simple matrix inversions. There are several possible techniques for minimizing such functions. We achieved good results with a Quasi-Newton method and a mixed quadratic and cubic line search procedure.

4 Results

We applied this technique to video recordings in our lab and to photo-plate sequence of Eadweard Muybridge's motion studies.

4.1 Single camera recordings

Our lab video recordings were done with a single camera. Therefore the 3D pose and some parts of the body can not be estimated completely. Figure 2 shows one example sequences of a person walking in a frontoparallel plane. We defined a 6 DOF kinematic structure: One blob for the body trunk, three blobs for the frontal leg and foot, connected with a hip joint, knee joint, and ankle joint, and two blobs for the arm connected with a shoulder and elbow joint. All joints have an axis orientation parallel to the Z -axis in the camera frame. The head blob was connected with one joint to the body trunk. The first image in figure 2 shows the initial blob support maps.

After the hand-initialization we applied the motion tracker to a sequence of 53 image frames. We could successfully track all body parts in this video sequence (see video). The video shows that the appearance of the upper leg changes significantly due to moving folds on the subject's jeans. The lower leg appearance does not change to the same extent. The constraints were able to enforce compatible motion vectors for the upper leg, based on more reliable measurements on the lower leg.

We can compare the estimated angular configurations with motion capture data reported in the literature. Murray, Brought, and

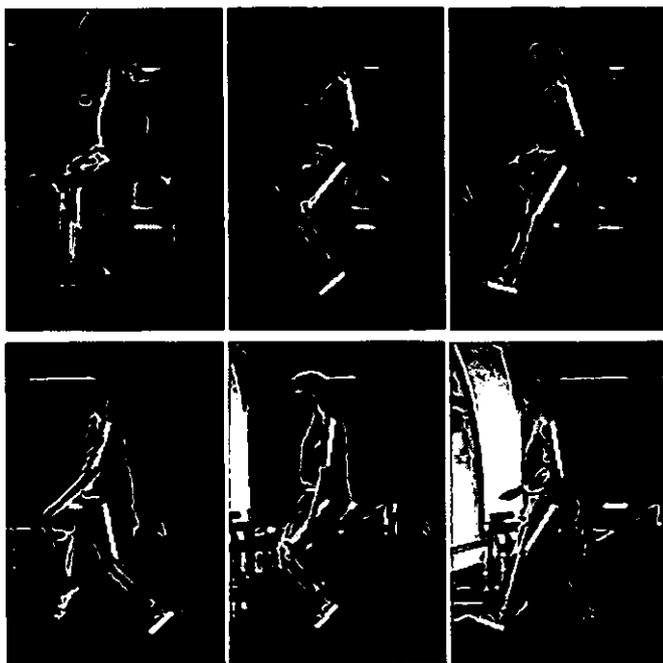


Figure 2: Example configurations of the estimated kinematic structure. First image shows the support maps of the initial configuration. In subsequent images the white lines show blob axes. The joint is the position on the intersection of two axes.

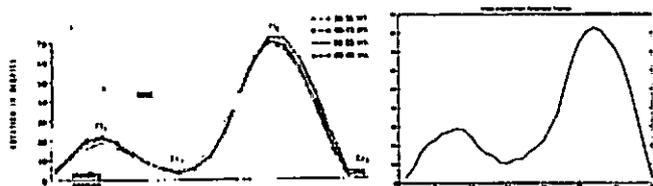


Figure 3: Comparison of a) data from [Murray et al] (left) and b) our motion tracker (right).

Kory published [13] such measurements for the hip, knee, and ankle joints. We compared our motion tracker measurements with the published curves and found good agreement. Figure 4.1a shows the curves for the knee and ankle reported in [13], and figure 4.1b shows our measurements.

We also experimented with a walking sequence of a subject seen from an oblique view with a similar kinematic model. As seen in figure 4, we tracked the angular configurations and the pose successfully over the complete sequence of 45 image frames. Because we use a scaled orthographic projection model, the perspective effects of the person walking closer to the camera had to be compensated by different scales. The tracking algorithm could successfully estimate the scale changes.

4.2 Digital Muybridge

The final set of experiments was done on historic footage recorded by Eadweard Muybridge in 1884. His methods are of independent interest, as they predate motion pictures. Muybridge had his models walk in an open shed. Parallel to the shed was a fixed battery of 24 cameras. Two portable batteries of 12 cameras each were positioned at both ends of the shed, either at an angle of 90 deg relative to the shed or an angle of 60 deg. Three photographs were take

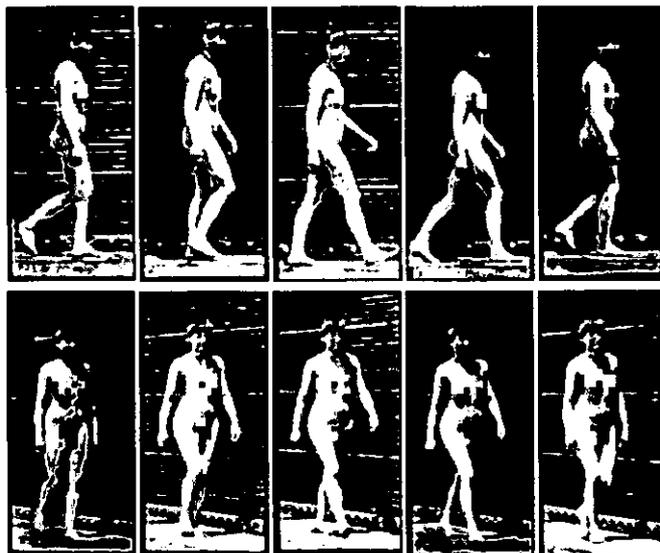


Figure 5: Eadweard Muybridge, *The Human Figure in Motion*, Plate 97: Woman Walking. The first 5 frames show part of a walk cycle from one example view, and the second 5 frames show the same time steps from a different view

simultaneously, one from each battery. The effective 'framerate' of his technique is about two times lower than current video frame rates; a fact which makes tracking a harder problem.. It is to our advantage that he took for each time step three pictures from different viewpoints.

Figure 4.2 and figure 4.2 shows example photo plates. We could initialize the 3D pose by labeling all three views of the first frame and running the minimization procedure over the body dimensions and poses. Figure 4.2 shows one example initialization. Every body segment was visible in at least one of the three camera views, therefore we could track the left and the right side of the person. We applied this technique to a walking woman and a walking man. For the walking woman we had 10 time steps available that contained 60% of a full walk cycle (figure 4.2). For this set of experiments we extended our kinematic model to 19 DOFs. The two hip joints, the two shoulder joints, and the neck joint, were modeled by 3 DOFs. The two knee joints and two elbow joints were modeled just by one rotation axis. Figure 4.2 shows the tracking results with the model overlaid. As you see, we could successfully track the complete sequence. To animate the tracking results we mirrored the left and right side angles to produce the remaining frames of a complete walk cycle. We animated the 3D motion capture data with a stick figure model and a volumetric model (figure 10), and it looks very natural. The video shows some of the tracking and animation sequences from several novel camera views, replicating the walk cycle performed over a century ago on the grounds of University of Pennsylvania.

For the visualization of the walking man sequence, we did not apply the mirroring, because he was carrying a boulder on his shoulder. This made the walk asymmetric. We re-animated the original tracked motion (figure 4.2) capture data for the man, and it also looked very natural.

Given the successful application of our tracking technique to multi-view data, we are planning to record with higher frame-rates our own multi-view video footage. We also plan to record a wider range of gestures.

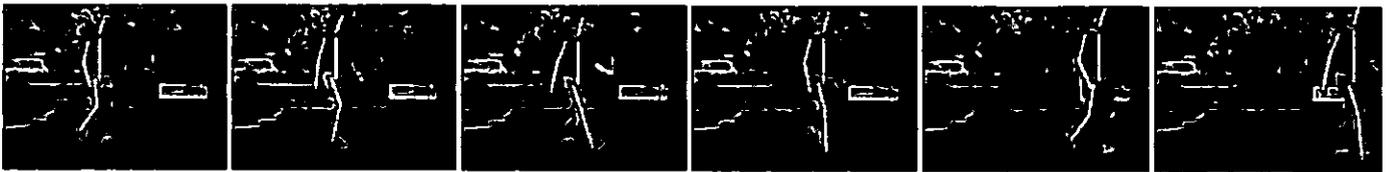


Figure 4: Example configurations of the estimated kinematic structure of a person seen from an oblique view.

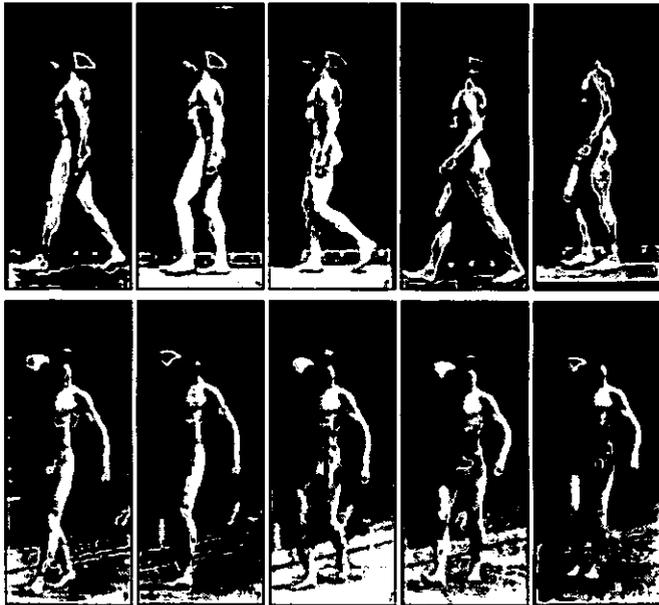


Figure 6: Eadweard Muybridge, *The Human Figure in Motion*, Plate 7: Man walking and carrying 75-LB boulder on shoulder. The first 5 frames show part of a walk cycle from one example view, and the second 5 frames show the same time steps from a different view

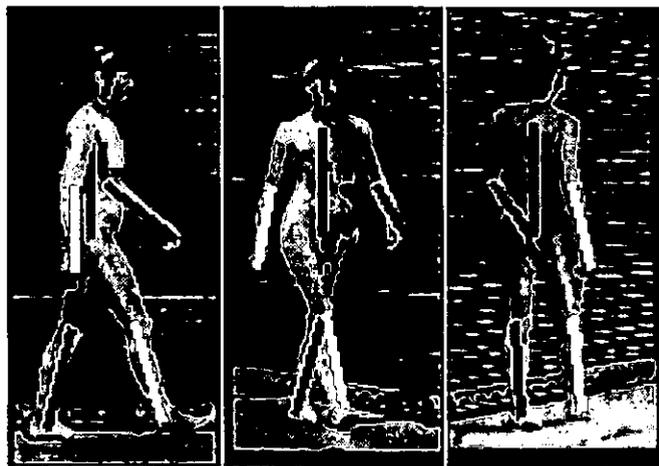


Figure 7: Initialization of Muybridge's Woman Walking: This visualizes the initial angular configuration projected to 3 example views.

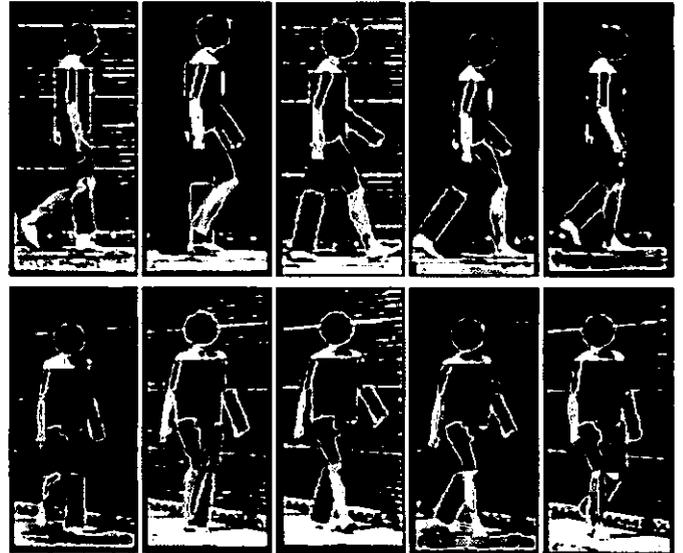


Figure 8: Muybridge's Woman Walking: Motion Capture results. This shows the tracked angular configurations and its volumetric model projected to 2 example views.

5 Conclusion

In this paper, we have developed and demonstrated a new technique for video motion capture. The approach does not require any markers, body suits or any other devices attached to the body of the actor. The actor can move about wearing his or her regular clothes. We demonstrated results on video recordings of people walking both in frontoparallel and oblique views, as well as on the classic Muybridge photographic sequences recorded more than a century ago.

Visually tracking human motion at the level of individual joints is a very challenging problem. Our results are due, in large measure, to the introduction of a novel mathematical technique, the product of exponential maps and twist motions, and its integration into a differential motion estimation scheme. The advantage of this particular formulation is that it results in the equations that need to be solved to update the kinematic chain parameters from frame to frame being linear, and that it is not necessary to solve for any redundant or unnecessary variables.

Future work will concentrate on dealing with very large motions, as may happen, for instance, in videotapes of high speed running. The approach developed in this paper is a differential method, and therefore may be expected to fail when the motion from frame-to-frame is very large. We propose to augment the technique by the use of an initial coarse search stage. Given a close enough starting value, the differential method will converge correctly.

Acknowledgements

We would like to thank Charles Ying for creating the Open-GL animations and video editing, Shankar Sastry, Lara Crawford, Jerry Feldman, John Canny, and Jianbo Shi for fruitful discussions, Chad Carson for helping to write this document, and Interval Research Corp, and the California State MICRO program for supporting this research.

References

- [1] S. Basu, I.A. Essa, and A.P. Pentland. Motion regularization for model-based head tracking. In *International Conference on Pattern Recognition*, 1996.
- [2] J.R. Bergen, P. Anandan, K.J. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *ECCV*, pages 237–252, 1992.
- [3] M.J. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1):75–104, Jan 1996.
- [4] G. Cameron, A. Bustanoby, K. Cope, S. Greenberg, C. Hayes, and O. Ozoux. Panel on motion capture and cg character animation. *SIGGRAPH 97*, pages 442–445, 1997.
- [5] L. Concalves, E.D. Bernardo, E. Ursella, and P. Perona. Monocular tracking of the human arm in 3d. In *Proc. Int. Conf. Computer Vision*, 1995.
- [6] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39, 1977.
- [7] D.M. Gavrilu and L.S. Davis. Towards 3-d model-based tracking and recognition of human movement: a multi-view approach. In *Proc. of the Int. Workshop on Automatic Face- and Gesture-Recognition, Zurich, 1995*, 1995.
- [8] D. Hogg. A program to see a walking person. *Image Vision Computing*, 5(20), 1983.
- [9] A. Jepson and M.J. Black. Mixture models for optical flow computation. In *Proc. IEEE Conf. Computer Vision Pattern Recognition*, pages 760–761, New York, 1993.
- [10] S.X. Ju, M.J. Black, and Y. Yacoob. Cardboard people: A parameterized model of articulated motion. In *2nd Int. Conf. on Automatic Face- and Gesture-Recognition, Killington, Vermont*, pages 38–44, 1996.
- [11] I.A. Kakadiaris and D. Metaxas. Model-based estimation of 3d human motion with occlusion based on active multi-viewpoint selection. In *CVPR*, 1996.
- [12] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *Proc. 7th Int. Joint Conf. on Art. Intell.*, 1981.
- [13] M.P. Murray, A.B. Drought, and R.C. Kory. Walking patterns of normal men. *Journal of Bone and Joint Surgery*, 46-A(2):335–360, March 1964.
- [14] R.M. Murray, Z. Li, and S.S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.



Figure 9: Muybridge's Man Walking: Motion Capture results. This shows the tracked angular configurations and its volumetric model projected to 2 example views.

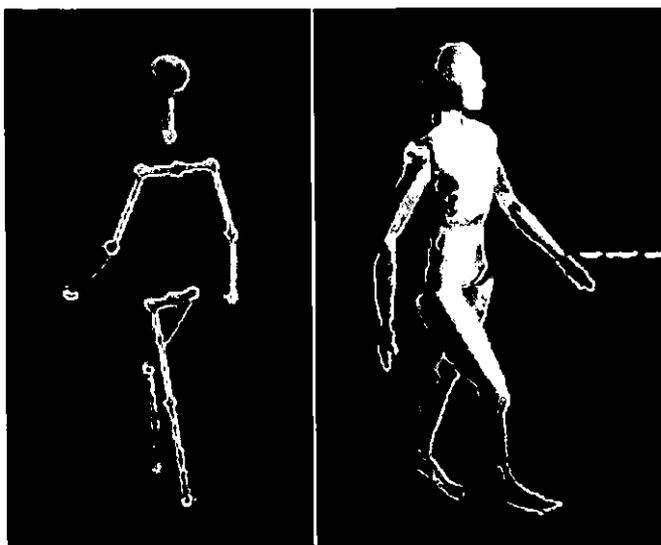


Figure 10: Computer models used for the animation of the Muybridge motion capture. Please check out the video to see the quality of the animation.

- [15] Eadweard Muybridge. *The Human Figure In Motion*. Various Publishers, latest edition by Dover Publications, 1901.
- [16] J. O'Rourke and N. I. Badler. Model-based image analysis of human motion using constraint propagation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2(6):522–536, November 1980.
- [17] J.M. Rehg and T. Kanade. Model-based tracking of self-occluding articulated objects. In *Proc. Int. Conf. Computer Vision*, 1995.
- [18] K. Rohr. Incremental recognition of pedestrians from image sequences. In *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn.*, pages 8–13, New York City, June, 1993.
- [19] J. Shi and C. Tomasi. Good features to track. In *CVPR*, 1994.
- [20] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. In *SPIE Conference on Integration Issues in Large Commercial Media Delivery Systems*, volume 2615, 1995.

IBR Techniques for Non-Rigid Domains

Video-Based Animation of Human Motion

Christoph Bregler

Computer Science Division
University of California, Berkeley

Includes Work done with
Jitendra Malik, Jerome A. Feldman, Charles H. Ying, UC Berkeley
Michele Covell, Malcolm Stanley, Interval.

Problem Domains

- Motion Capture / Animation



Body Suits, Markers



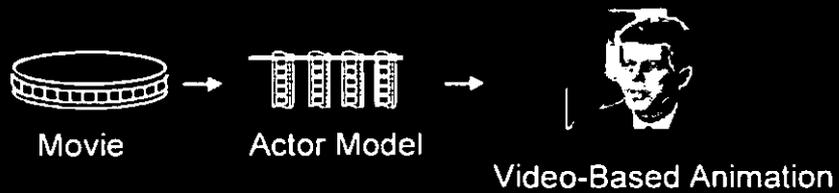
Video Motion Capture

Overview

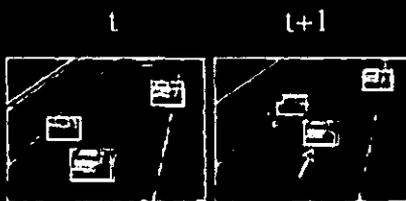
- Survey Technology

Measurement / Learn Models / Recognition

- New Animation Paradigm



Visual Tracking



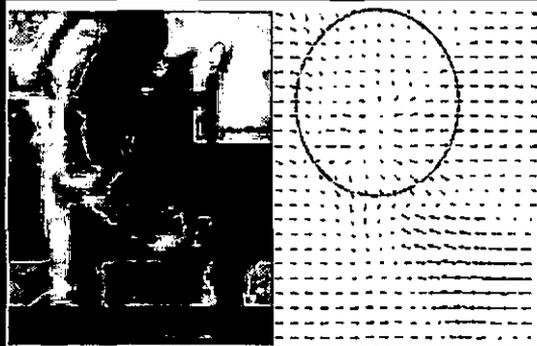
Standard Techniques:

- Template Matching
- Edges / Shape / Color
- Background Subtraction
- Optical Flow

New Challenges:

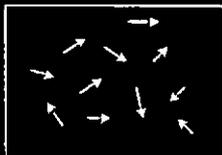
- Complex Variation
- Self Occlusion
- Noise (Folds, Low Contrast)

Optical Flow



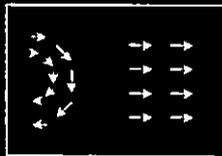
local ambiguities

Motion Constraints



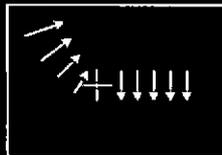
N Pixel

• Optical Flow: $2 \cdot N$ Parameters



L Areas

• Layered Motion: $6 \cdot L$ Parameters

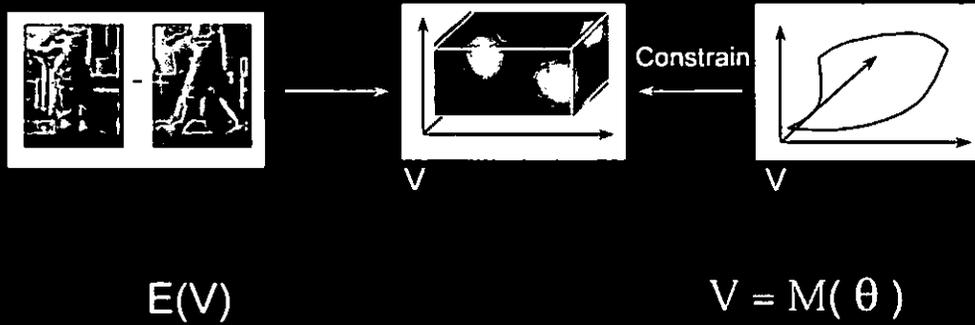


K DOFs

• Articulated Chains: $6 + K$ Parameters

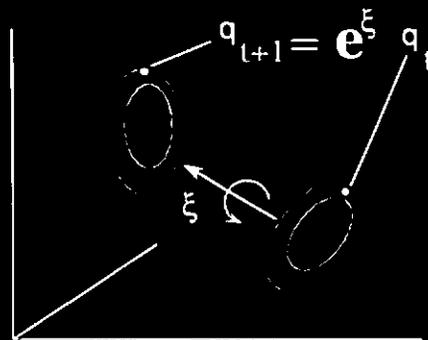
Video Motion Capture

Bregler, Malik

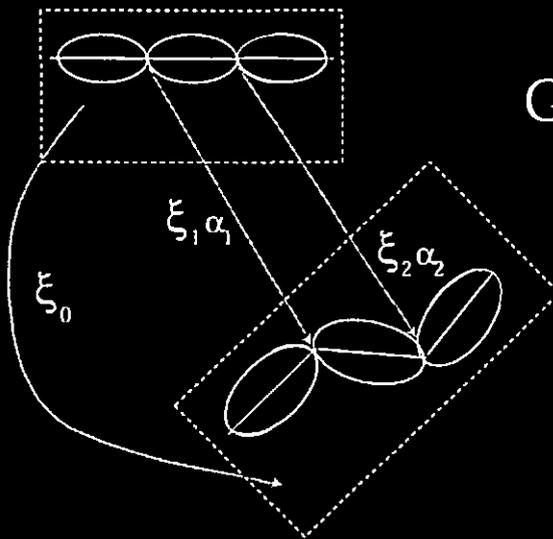


Twist Motion / Exponential Map

Murray, Li, Sastry

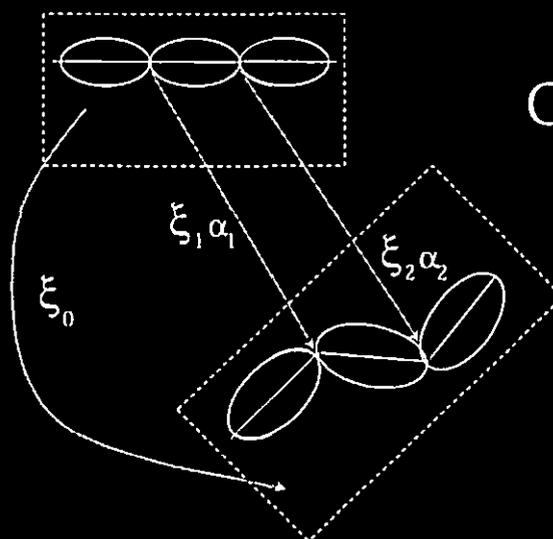


Product of Exponential Map



$$G = e^{\hat{\xi}_0} e^{\hat{\xi}_1 \alpha_1} e^{\hat{\xi}_2 \alpha_2}$$

Product of Exponential Map

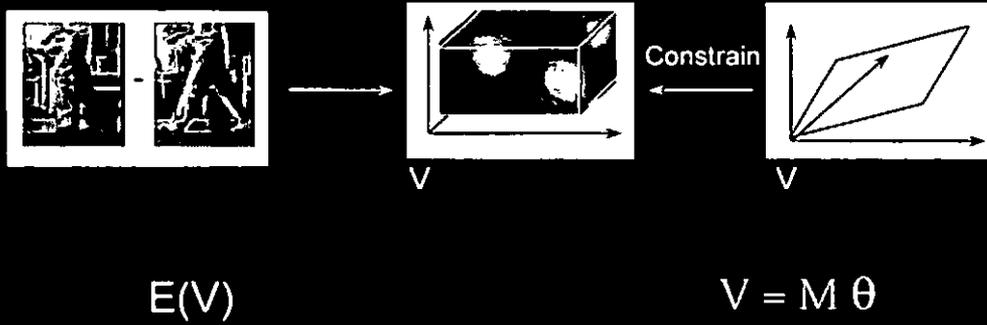


$$G = e^{\hat{\xi}_0} e^{\hat{\xi}_1 \alpha_1} e^{\hat{\xi}_2 \alpha_2}$$

$$\downarrow \frac{\delta}{\delta t}$$

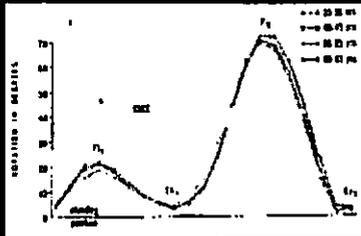
$$v_i = M_i \begin{bmatrix} \dot{\xi} \\ \alpha \end{bmatrix}$$

Video Motion Capture

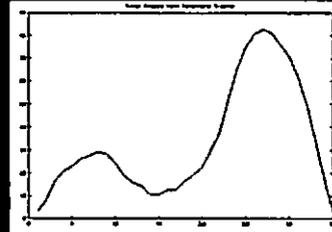


Video

Comparison to Biometric Data

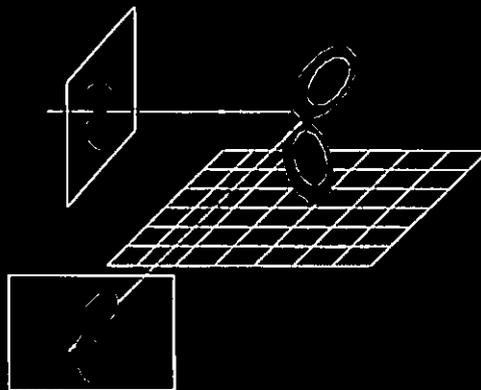


Murray, Drought, Kory, 1964



kinematic tracker

Multiple Views



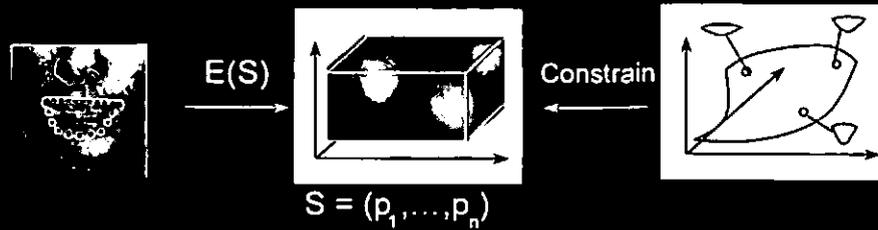
Eadweard Muybridge



Video

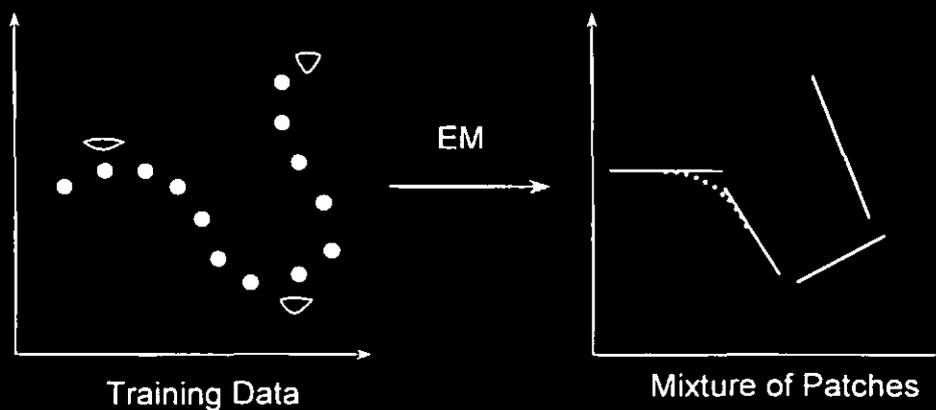
Graphics by Charles Ying

Non-Rigid Measurement

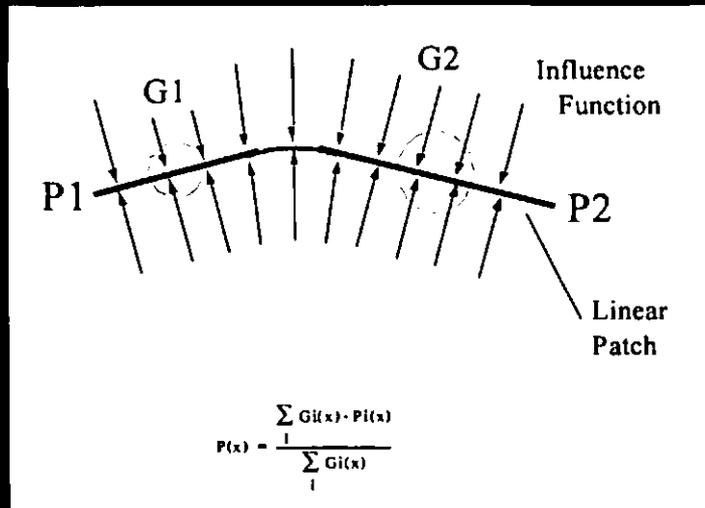


Manifold Learning for Tracking

Chris Bregler, Steve Omohundro



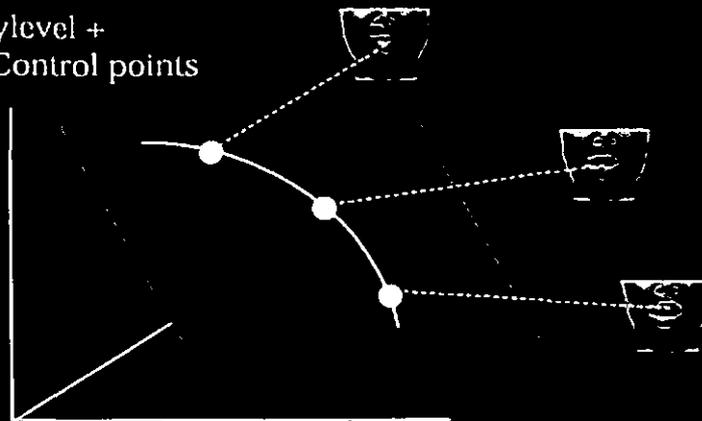
Mixture of Projections



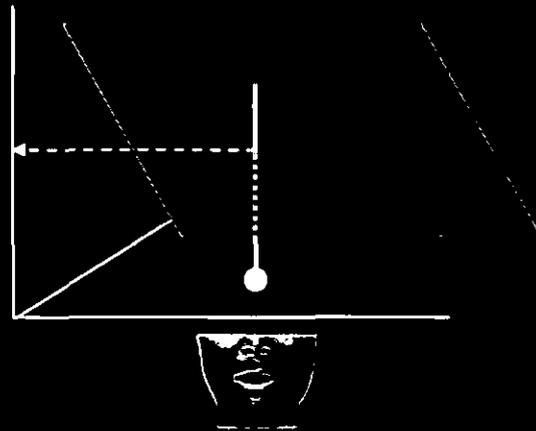
8

Eigenpoints - Training -

Graylevel +
XY Control points

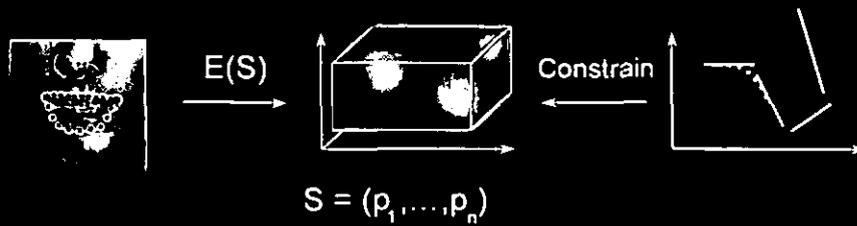


Eigenpoints - Mapping -



Graylevel +
XY Control point
Space

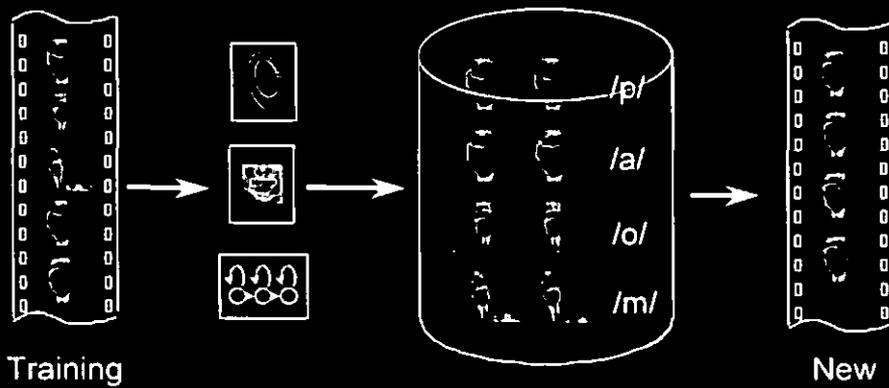
Constrained Tracking



Video

Video Rewrite

C. Bregler, M. Covell, M. Slaney
Interval Research Corp.



Goal: *Photo-realistic Talking Face*

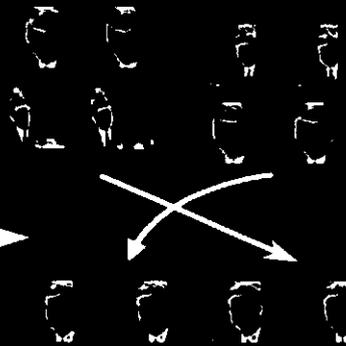
2

Handcoded
3D Model



OR

Video Rewrite



Building Video Model

5

◦ *Phonetic*



◦ *Head Pose*



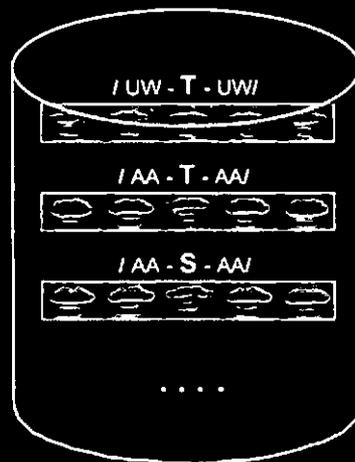
◦ *Mouth Shape*



video

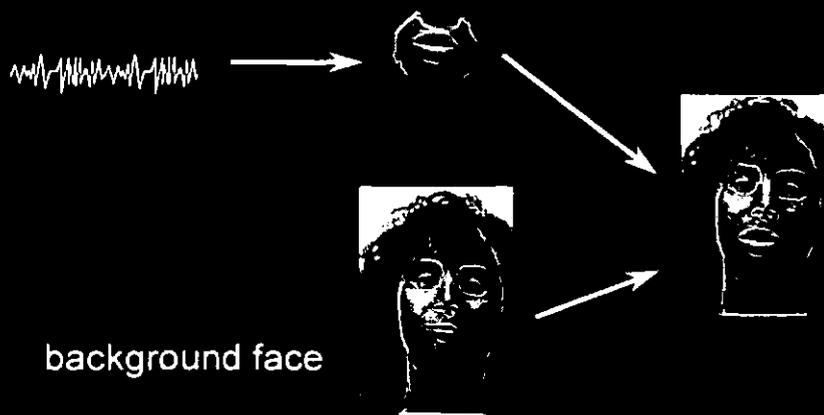
Video Model:

8 min =
1,700 triphones



"Ellen Model"

Synthesis - Overview -

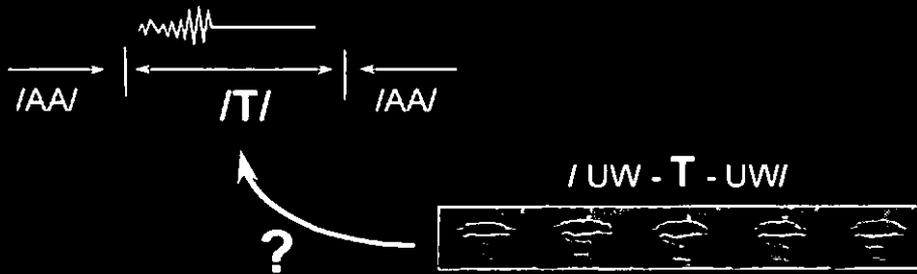


Synthesis:

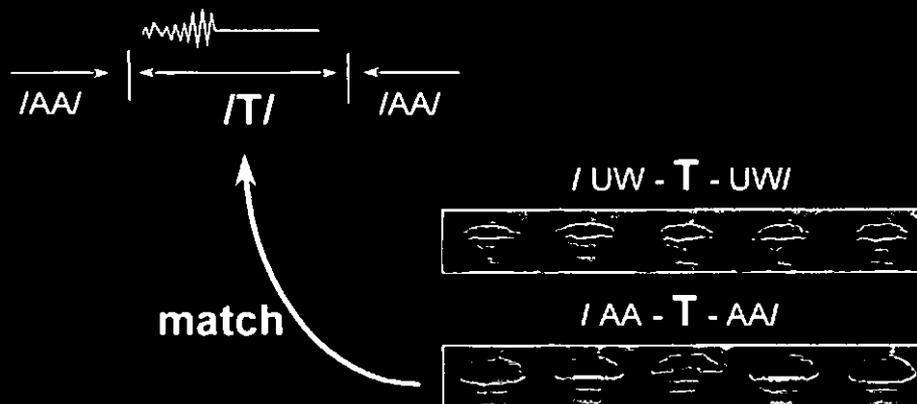
- *Transcribe*
- *Find Lip Clips*
- *Stitch Together*



Matching: Co-Articulation

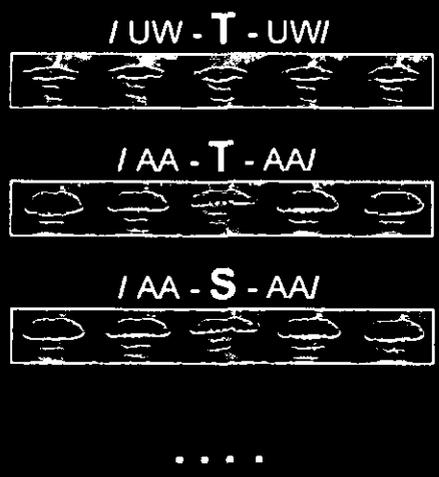


Matching: Co-Articulation

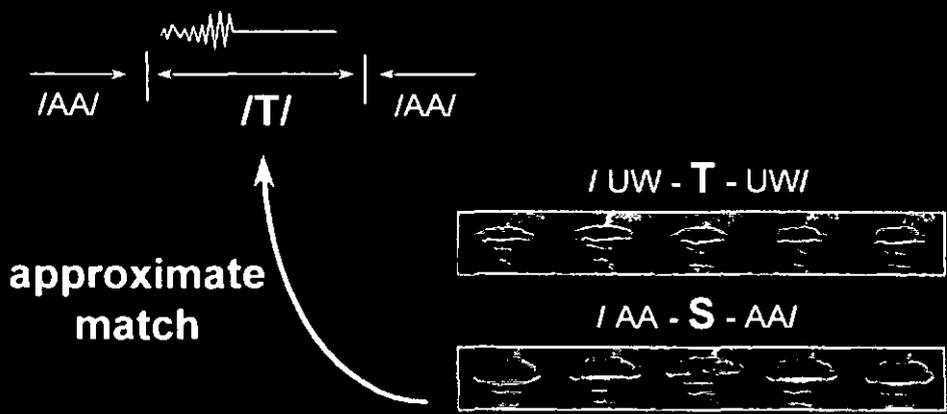


Co-Articulation: *Tri-Phones*

More than
20,000 Tri-Phones
in English



Matching: *Viseme-Distance*

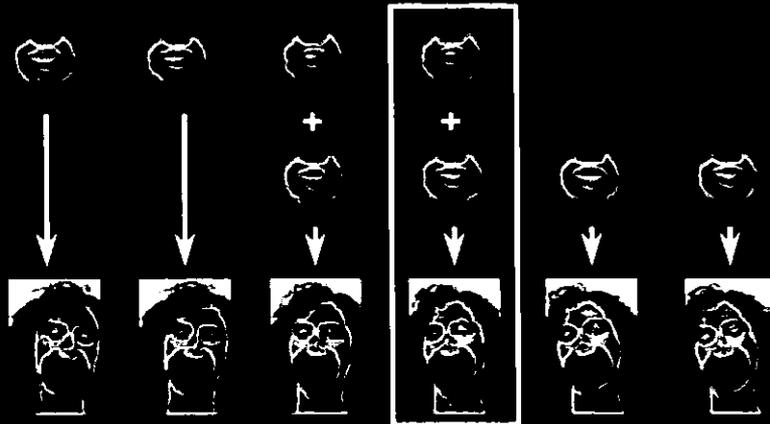


video

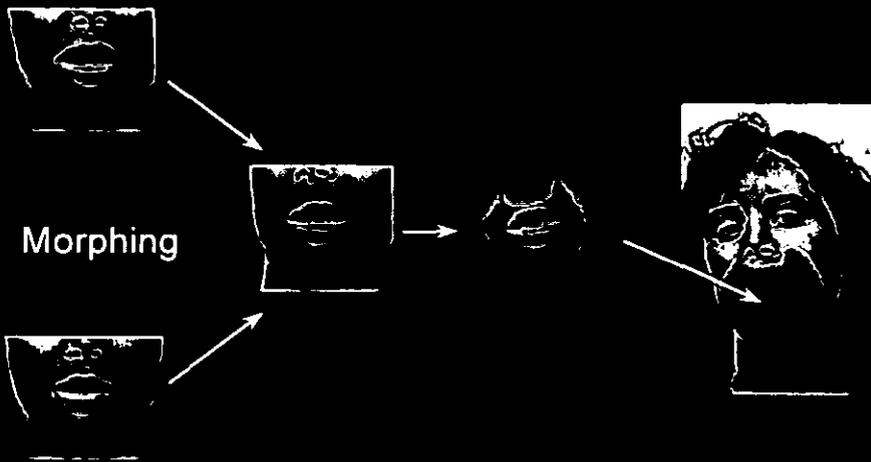
7

Stitching

20



Stitching



Video Rewrite Results



Ellen - Video Model

8 minutes data



JFK - Video Model

2 minutes data

Future: Virtual Actors



Challenges



Key Technologies

Learning



Video-Based
Rendering

Video Rewrite: Driving Visual Speech with Audio

Christoph Bregler, Michele Covell, Malcolm Slaney
Interval Research Corporation

ABSTRACT

Video Rewrite uses existing footage to create automatically new video of a person mouthing words that she did not speak in the original footage. This technique is useful in movie dubbing, for example, where the movie sequence can be modified to sync the actors' lip motions to the new soundtrack.

Video Rewrite automatically labels the phonemes in the training data and in the new audio track. Video Rewrite reorders the mouth images in the training footage to match the phoneme sequence of the new audio track. When particular phonemes are unavailable in the training footage, Video Rewrite selects the closest approximations. The resulting sequence of mouth images is stitched into the background footage. This stitching process automatically corrects for differences in head position and orientation between the mouth images and the background footage.

Video Rewrite uses computer-vision techniques to track points on the speaker's mouth in the training footage, and morphing techniques to combine these mouth gestures into the final video sequence. The new video combines the dynamics of the original actor's articulations with the mannerisms and setting dictated by the background footage. Video Rewrite is the first facial-animation system to automate all the labeling and assembly tasks required to resync existing footage to a new soundtrack.

CR Categories: I.3.3 [Computer Graphics]: Picture/Image Generation—Morphing; I.4.6 [Image Processing]: Segmentation—Feature Detection; I.3.8 [Computer Graphics]: Applications—Facial Synthesis; I.4.10 [Image Processing]: Applications—Feature Transformations.

Additional Keywords: Facial Animation, Lip Sync.

1 WHY AND HOW WE REWRITE VIDEO

We are very sensitive to the synchronization between speech and lip motions. For example, the special effects in *Forest Gump* are compelling because the Kennedy and Nixon footage is lip synched to the movie's new soundtrack. In contrast, close-ups in dubbed movies are often disturbing due to the lack of lip sync. Video Rewrite is a system for automatically synthesizing faces with proper lip sync. It can be used for dubbing movies, teleconferencing, and special effects.

1801 Page Mill Road, Building C, Palo Alto, CA, 94304. E-mail: bregler@cs.berkeley.edu, covell@interval.com, malcolm@interval.com. See the SIGGRAPH Video Proceedings or <http://www.interval.com/papers/1997-012/> for the latest animations.

Permission to make digital/hard copy of all or part of this material for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication and its date appear, and notice is given that copying is by permission of ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

© 1997 ACM.

Video Rewrite automatically pieces together from old footage a new video that shows an actor mouthing a new utterance. The results are similar to labor-intensive special effects in *Forest Gump*. These effects are successful because they start from actual film footage and modify it to match the new speech. Modifying and reassembling such footage in a smart way and synchronizing it to the new sound track leads to final footage of realistic quality. Video Rewrite uses a similar approach but does not require labor-intensive interaction.

Our approach allows Video Rewrite to learn from example footage how a person's face changes during speech. We learn what a person's mouth looks like from a video of that person speaking normally. We capture the dynamics and idiosyncrasies of her articulation by creating a database of video clips. For example, if a woman speaks out of one side of her mouth, this detail is recreated accurately. In contrast, most current facial-animation systems rely on generic head models that do not capture the idiosyncrasies of an individual speaker.

To model a new person, Video Rewrite requires a small number (26 in this work) of hand-labeled images. This is the only human intervention that is required in the whole process. Even this level of human interaction is not a fundamental requirement: We could use face-independent models instead [Kirby90, Covell96].

Video Rewrite shares its philosophy with concatenative speech synthesis [Moulines90]. Instead of modeling the vocal tract, concatenative speech synthesis analyzes a corpus of speech, selects examples of phonemes, and normalizes those examples. Phonemes are the distinct sounds within a language, such as the /IY/ and /P/ in "teapot." Concatenative speech synthesizes new sounds by concatenating the proper sequence of phonemes. After the appropriate warping of pitch and duration, the resulting speech is natural sounding. This approach to synthesis is data driven: The algorithms analyze and resynthesize sounds using little hand-coded knowledge of speech. Yet they are effective at implicitly capturing the nuances of human speech.

Video Rewrite uses a similar approach to create new sequences of visemes. Visemes are the visual counterpart to phonemes. Visemes are visually distinct mouth, teeth, and tongue articulations for a language. For example, the phonemes /B/ and /P/ are visually indistinguishable and are grouped into a single viseme class.

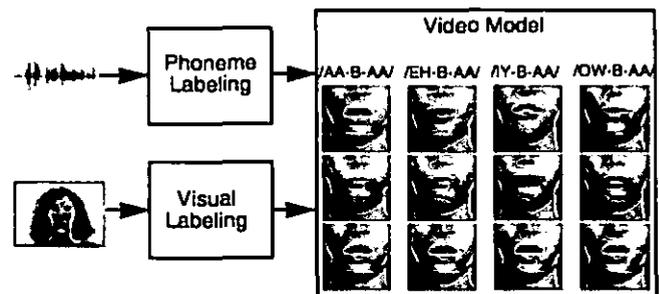


Figure 1: Overview of analysis stage. Video Rewrite uses the audio track to segment the video into triphones. Vision techniques find the orientation of the head, and the shape and position of the mouth and chin in each image. In the synthesis stage, Video Rewrite selects from this video model to synchronize new lip videos to any given audio.

Video Rewrite creates new videos using two steps: analysis of a training database and synthesis of new footage. In the *analysis* stage, Video Rewrite automatically segments into phonemes the audio track of the training database. We use these labels to segment the video track as well. We automatically track facial features in this segmented footage. The phoneme and facial labels together completely describe the visemes in the training database. In the *synthesis* stage, our system uses this video database, along with a new utterance. It automatically retrieves the appropriate viseme sequences, and blends them into a background scene using morphing techniques. The result is a new video with lip and jaw movements that synchronize to the new audio. The steps used in the analysis stage are shown in Figure 1; those of the synthesis stage are shown in Figure 2.

In the remainder of this paper, we first review other approaches to synthesizing talking faces (Section 2). We then describe the analysis and synthesis stages of Video Rewrite. In the analysis stage (Section 3), a collection of video is analyzed and stored in a database that matches sounds to video sequences. In the synthesis stage (Section 4), new speech is labeled, and the appropriate sequences are retrieved from the database. The final sections of this paper describe our results (Section 5), future work (Section 6), and contributions (Section 7).

2 SYNTHETIC VISUAL SPEECH

Facial-animation systems build a model of what a person's speech sounds and looks like. They use this model to generate a new output sequence, which matches the (new) target utterance. On the model-building side (analysis), there are typically three distinguishing choices: how the facial appearance is learned or described, how the facial appearance is controlled or labeled, and how the viseme labels are learned or described. For output-sequence generation (synthesis), the distinguishing choice is how the target utterance is characterized. This section reviews a representative sample of past research in these areas.

2.1 Source of Facial Appearance

Many facial-animation systems use a generic 3D mesh model of a face [Parke72, Lewis91, Guiard-Marigny94], sometimes adding texture mapping to improve realism [Morshima91, Cohen93, Waters95]. Another synthetic source of face data is hand-drawn images [Litwinowicz94]. Other systems use real faces for their source examples, including approaches that use 3D scans [Williams90] and still images [Scott94]. We use video footage to train Video Rewrite's models.

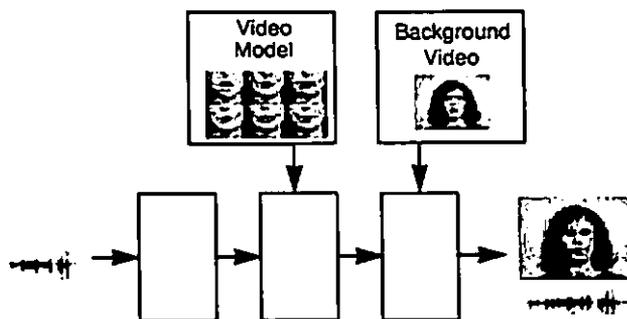


Figure 2: Overview of synthesis stage. Video Rewrite segments new audio and uses it to select triphones from the video model. Based on labels from the analysis stage, the new mouth images are morphed into a new background face.

2.2 Facial Appearance Control

Once a facial model is captured or created, the control parameters that exercise that model must be defined. In systems that rely on a 3D mesh model for appearance, the control parameters are the allowed 3D mesh deformations. Most of the image-based systems label the positions of specific facial locations as their control parameters. Of the systems that use facial-location labels, most rely on manual labeling of each example image [Scott94, Litwinowicz94]. Video Rewrite creates its video model by automatically labeling specific facial locations.

2.3 Viseme Labels

Many facial-animation systems label different visual configurations with an associated *phoneme*. These systems then match these phoneme labels with their corresponding labels in the target utterance. With synthetic images, the phoneme labels are artificial or are learned by analogy [Morshima91]. For natural images, taken from a video of someone speaking, the phonemic labels can be generated manually [Scott94] or automatically. Video Rewrite determines the phoneme labels automatically (Section 3.1).

2.4 Output-Sequence Generation

The goal of facial animation is to generate an image sequence that matches a target utterance. When phoneme labels are used, those for the target utterance can be entered manually [Scott94] or computed automatically [Lewis91, Morshima91]. Another option for phoneme labeling is to create the new utterance with synthetic speech [Parke72, Cohen93, Waters95]. Approaches that do not use phoneme labels include motion capture of facial locations that are artificially highlighted [Williams90, Guiard-Marigny94] and manual control by an animator [Litwinowicz94]. Video Rewrite uses a combination of phoneme labels (from the target utterance) and facial-location labels (from the video-model segments). Video Rewrite derives all these labels automatically.

Video Rewrite is the first facial-animation system to automate all these steps and to generate realistic lip-synched video from natural speech and natural images.

3 ANALYSIS FOR VIDEO MODELING

As shown in Figure 1, the analysis stage creates an annotated database of example video clips, derived from unconstrained footage. We refer to this collection of annotated examples as a video model. This model captures how the subject's mouth and jaw move during speech. These training videos are labeled automatically with the phoneme sequence uttered during the video, and with the locations of fiduciary points that outline the lips, teeth, and jaw.

As we shall describe, the phonemic labels are from a time-aligned transcript of the speech, generated by a hidden Markov model (HMM). Video Rewrite uses the phonemic labels from the HMM to segment the input footage into short video clips, each showing three phonemes or a triphone. These triphone videos, with the fiduciary-point locations and the phoneme labels, are stored in the video model.

In Sections 3.1 and 3.2, we describe the visual and acoustic analyses of the video footage. In Section 4, we explain how to use this model to synthesize new video.

3.1 Annotation Using Image Analysis

Video Rewrite uses any footage of the subject speaking. As her face moves within the frame, we need to know the mouth position and the lip shapes at all times. In the synthesis stage, we use this information to warp overlapping videos such that they have the same lip shapes, and to align the lips with the background face.

Manual labeling of the fiduciary points around the mouth and jaw is error prone and tedious. Instead, we use computer-vision techniques to label the face and to identify the mouth and its shape. A major hurdle to automatic annotation is the low resolution of the images. In a typical scene, the lip region has a width of only 40 pixels. Conventional contour-tracking algorithms [Kass87, Yuille89] work well on high-contrast outer lip boundaries with some user interaction, but fail on inner lip boundaries at this resolution, due to the low signal-to-noise ratios. Grayscale-based algorithms, such as eigenimages [Kirby90, Turk91], work well at low resolutions, but estimate only the location of the lips or jaw, rather than estimating the desired fiduciary points. Eigenpoints [Covell96], and other extensions of eigenimages [Lanitis95], estimate control points reliably and automatically, even in such low-resolution images. As shown in Figure 3, eigenpoints learns how fiduciary points move as a function of the image appearance, and then uses this model to label new footage.

Video Rewrite labels each image in the training video using a total of 54 eigenpoints: 34 on the mouth (20 on the outer boundary, 12 on the inner boundary, 1 at the bottom of the upper teeth, and 1 at the top of the lower teeth) and 20 on the chin and jaw line. There are two separate eigenpoint analyses. The first eigenspace controls the placement of the 34 fiduciary points on the mouth, using 50×40 pixels around the nominal mouth location, a region that covers the mouth completely. The second eigenspace controls the placement of the 20 fiduciary points on the chin and jaw line, using 100×75 pixels around the nominal chin-location, a region that covers the upper neck and the lower part of the face.

We created the two eigenpoint models for locating the fiduciary points from a small number of images. We hand annotated only 26 images (of 14,218 images total; about 0.2%). We extended the hand-annotated dataset by morphing pairs of annotated images to form intermediate images, expanding the original 26 to 351 annotated images without any additional manual work. We then derived eigenpoints models using this extended data set.

We use eigenpoints to find the mouth and jaw and to label their contours. The derived eigenpoint models locate the facial features using six basis vectors for the mouth and six different vectors for the jaw. Eigenpoints then places the fiduciary points around the feature locations: 32 basis vectors place points around the lips and 64 basis vectors place points around the jaw.

Eigenpoints assumes that the features (the mouth or the jaw) are undergoing pure translational motion. It does a comparatively poor job at modeling rotations and scale changes. Yet, Video Rewrite is designed to use unconstrained footage. We expect rotations and scale changes. Subjects may lean toward the camera or turn away from it, tilt their heads to the side, or look up from under their eyelashes.

To allow for a variety of motions, we warp each face image into a standard reference plane, prior to eigenpoints labeling. We

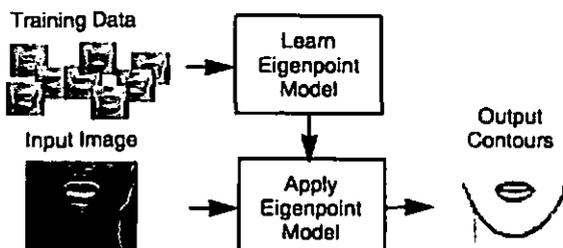


Figure 3: Overview of eigenpoints. A small set of hand-labeled facial images is used to train subspace models. Given a new image, the eigenpoint models tell us the positions of points on the lips and jaw.

find the global transform that minimizes the mean-squared error between a large portion of the face image and a facial template. We currently use an affine transform [Black95]. The mask shown in Figure 4 defines the support of the minimization integral. Once the best global mapping is found, it is inverted and applied to the image, putting that face into the standard coordinate frame. We then perform eigenpoints analysis on this pre-warped image to find the fiduciary points. Finally, we back-project the fiduciary points through the global warp to place them on the original face image.

The labels provided by eigenpoints allow us automatically to (1) build the database of example lip configurations, and (2) track the features in a background scene that we intend to modify. Section 4.2 describes how we match the points we find in step 1 to each other and to the points found in step 2.

3.2 Annotation Using Audio Analysis

All the speech data in Video Rewrite (and their associated video) are segmented into sequences of phonemes. Although single phonemes are a convenient representation for linguistic analysis, they are not appropriate for Video Rewrite. We want to capture the visual dynamics of speech. To do so correctly, we must consider *coarticulation*, which causes the lip shapes for many phonemes to be modified based on the phoneme's context. For example, the /T/ in "beet" looks different from the /T/ in "boot."

Therefore, Video Rewrite segments speech and video into tri-phones: collections of three sequential phonemes. The word "tea-pot" is split into the sequence of triphones /SIL-T-IY/, ¹/T-IY-P/, /IY-P-AA/, /P-AA-T/, and /AA-T-SIL/. When we synthesize a video, we emphasize the middle of each triphone. We cross-fade the overlapping regions of neighboring triphones. We thus ensure that the precise transition points are not critical, and that we can capture effectively many of the dynamics of both forward and backward coarticulation.

Video Rewrite uses HMMs [Rabiner89] to label the training footage with phonemes. We trained the HMMs using the TIMIT speech database [Lameli86], a collection of 4200 utterances with phonemic transcriptions that gives the uttered phonemes and their timing. Each of the 61 phoneme categories in TIMIT is modeled with a separate three-state HMM. The emission probabilities of each state are modeled with mixtures of eight Gaussians with diagonal covariances. For robustness, we split the available data by gender and train two speaker-independent, gender-specific systems, one based on 1300 female utterances, and one based on 2900 male utterances.

We used these gender-specific HMMs to create a fine-grained phonemic transcription of our input footage, using forced Viterbi

1. /SIL/ indicates silence. Two /SIL/ in a row are used at the beginnings and ends of utterances to allow all segments—including the beginning and end—to be treated as triphones.

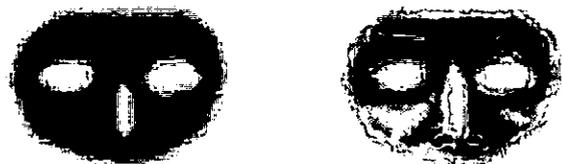


Figure 4: Mask used to estimate the global warp. Each image is warped to account for changes in the head's position, size, and rotation. The transform minimizes the difference between the transformed images and the face template. The mask (left) forces the minimization to consider only the upper face (right).

search [Viterbi67]. Forced Viterbi uses unaligned sentence-level transcriptions and a phoneme-level pronunciation dictionary to create a time-aligned phoneme-level transcript of the speech. From this transcript, Video Rewrite segments the video automatically into triphone videos, labels them, and includes them in the video model.

4 SYNTHESIS USING A VIDEO MODEL

As shown in Figure 2, Video Rewrite synthesizes the final lip-synced video by labeling the new speech track, selecting a sequence of triphone videos that most accurately matches the new speech utterance, and stitching these images into a background video.

The background video sets the scene and provides the desired head position and movement. The background sequence in Video Rewrite includes most of the subject's face as well as the scene behind the subject. The frames of the background video are taken from the source footage in the same order as they were shot. The head tilts and the eyes blink, based on the background frames.

In contrast, the different triphone videos are used in whatever order is needed. They simply show the motions associated with articulation. For all the animations in this paper, the triphone images include the mouth, chin, and part of the cheeks, so that the chin and jaw move and the cheeks dimple appropriately as the mouth articulates. We use illumination-matching techniques [Burt83] to avoid visible seams between the triphone and background images.

The first step in synthesis (Figure 2) is labeling the new soundtrack. We label the new utterance with the same HMM that we used to create the video-model phoneme labels. In Sections 4.1 and 4.2, we describe the remaining steps: selecting triphone videos and stitching them into the background.

4.1 Selection of Triphone Videos

The new speech utterance determines the target sequence of speech sounds, marked with phoneme labels. We would like to find a sequence of triphone videos from our database that matches this new speech utterance. For each triphone in the new utterance, our goal is to find a video example with exactly the transition we need, and with lip shapes that match the lip shapes in neighboring triphone videos. Since this goal often is not reachable, we compromise by choosing a sequence of clips that approximates the desired transitions and shape continuity.

Given a triphone in the new speech utterance, we compute a matching distance to each triphone in the video database. The matching metric has two terms: the *phoneme-context distance*, D_p , and the *distance between lip shapes* in overlapping visual triphones, D_s . The total error is

$$\text{error} = \alpha D_p + (1 - \alpha) D_s,$$

where the weight, α , is a constant that trades off the two factors.

The phoneme-context distance, D_p , is based on categorical distances between phoneme categories and between viseme classes. Since Video Rewrite does not need to create a new soundtrack (it needs only a new video track), we can cluster phonemes into viseme classes, based on their visual appearance.

We use 26 viseme classes. Ten are consonant classes: (1) /CH/, /JH/, /SH/, /ZH/; (2) /K/, /G/, /N/, /L/; (3) /T/, /D/, /S/, /Z/; (4) /P/, /B/, /M/; (5) /F/, /V/; (6) /TH/, /DH/; (7) /W/, /R/; (8) /HH/; (9) /Y/; and (10) /NG/. Fifteen are vowel classes: one each for /EH/, /EY/, /ER/, /UH/, /AA/, /AO/, /AW/, /AY/, /UW/, /OW/, /OY/, /IY/, /IH/, /AE/, /AH/. One class is for silence, /SIL/.

The phoneme-context distance, D_p , is the weighted sum of phoneme distances between the target phonemes and the video-model phonemes within the context of the triphone. If the phonemic categories are the same (for example, /P/ and /P/), then this distance is 0. If they are in different viseme classes (/P/ and /Y/), then the distance is 1. If they are in different phonemic categories but are in the same viseme class (/P/ and /B/), then the distance is a value between 0 and 1. The intraclass distances are derived from published confusion matrices [Owens85].

In D_p , the center phoneme of the triphone has the largest weight, and the weights drop smoothly from there. Although the video model stores only triphone images, we consider the triphone's original context when picking the best-fitting sequence. In current animations, this context covers the triphone itself, plus one phoneme on either side.

The second term, D_s , measures how closely the mouth contours match in overlapping segments of adjacent triphone videos. In synthesizing the mouth shapes for "teapot" we want the contours for the /Y/ and /P/ in the lip sequence used for /T-IY-P/ to match the contours for the /Y/ and /P/ in the sequence used for /Y-P-AA/. We measure this similarity by computing the Euclidean distance, frame by frame, between four-element feature vectors containing the overall lip width, overall lip height, inner lip height, and height of visible teeth.

The lip-shape distance (D_s) between two triphone videos is minimized with the correct time alignment. For example, consider the overlapping contours for the /P/ in /T-IY-P/ and /Y-P-AA/. The /P/ phoneme includes both a silence, when the lips are pressed together, and an audible release, when the lips move rapidly apart. The durations of the initial silence within the /P/ phoneme may be different. The phoneme labels do not provide us with this level of detailed timing. Yet, if the silence durations are different, the lip-shape distance for two otherwise-well-matched videos will be large. This problem is exacerbated by imprecision in the HMM phonemic labels.

We want to find the temporal overlap between neighboring triphones that maximizes the similarity between the two lip shapes. We shift the two triphones relative to each other to find the best temporal offset and duration. We then use this optimal overlap both in computing the lip-shape distance, D_s , and in cross-fading the triphone videos during the stitching step. The optimal overlap is the one that minimizes D_s while still maintaining a minimum-allowed overlap.

Since the fitness measure for each triphone segment depends on that segment's neighbors in both directions, we select the sequence of triphone segments using dynamic programming over the entire utterance. This procedure ensures the selection of the optimal segments.

4.2 Stitching It Together

Video Rewrite produces the final video by stitching together the appropriate entries from the video database. At this point, we have already selected a sequence of triphone videos that most closely matches the target audio. We need to align the overlapping lip images temporally. This internally time-aligned sequence of videos is then time aligned to the new speech utterance. Finally, the resulting sequences of lip images are spatially aligned and are stitched into the background face. We describe each step in turn.

4.2.1 Time Alignment of Triphone Videos

We have a sequence of triphone videos that we must combine to form a new mouth movie. In combining the videos, we want to maintain the dynamics of the phonemes and their transitions. We need to time align the triphone videos carefully before blending

them. If we are not careful in this step, the mouth will appear to flutter open and closed inappropriately.

We align the triphone videos by choosing a portion of the overlapping triphones where the two lips shapes are as similar as possible. We make this choice when we evaluate D_s to choose the sequence of triphone videos (Section 4.1). We use the overlap duration and shift that provide the minimum value of D_s for the given videos.

4.2.2 Time Alignment of the Lips to the Utterance

We now have a self-consistent temporal alignment for the triphone videos. We have the correct articulatory motions, in the correct order to match the target utterance, but these articulations are not yet time aligned with the target utterance.

We align the lip motions with the target utterance by comparing the corresponding phoneme transcripts. The starting time of the center phone in the triphone sequence is aligned with the corresponding label in the target transcript. The triphone videos are then stretched or compressed such that they fit the time needed between the phoneme boundaries in the target utterance.

4.2.3 Combining of the Lips and the Background

The remaining task is to stitch the triphone videos into the background sequence. The correctness of the facial alignment is critical to the success of the recombination. The lips and head are constantly moving in the triphone and background footage. Yet, we need to align them all so that the new mouth is firmly planted on the face. Any error in spatial alignment causes the mouth to jitter relative to the face—an extremely disturbing effect.

We again use the mask from Figure 4 to help us find the optimal global transform to register the faces from the triphone videos with the background face. The combined transforms from the mouth and background images to the template face (Section 3.1) give our starting estimate in this search. Re-estimating the global transform by directly matching the triphone images to the background improves the accuracy of the mapping.

We use a replacement mask to specify which portions of the final video come from the triphone images and which come from the background video. This replacement mask warps to fit the new mouth shape in the triphone image and to fit the jaw shape in the background image. Figure 5 shows an example replacement mask, applied to triphone and background images.

Local deformations are required to stitch the shape of the mouth and jaw line correctly. These two shapes are handled differently. The mouth's shape is completely determined by the triphone images. The only changes made to these mouth shapes are imposed to align the mouths within the overlapping triphone images: The lip shapes are linearly cross-faded between the shapes in the overlapping segments of the triphone videos.



Figure 5: Facial fading mask. This mask determines which portions of the final movie frames come from the background frame, and which come from the triphone database. The mask should be large enough to include the mouth and chin. These images show the replacement mask applied to a triphone image, and its inverse applied to a background image. The mask warps according to the mouth and chin motions.

The jaw's shape, on the other hand, is a combination of the background jaw line and the two triphone jaw lines. Near the ears, we want to preserve the background video's jaw line. At the center of the jaw line (the chin), the shape and position are determined completely by what the mouth is doing. The final image of the jaw must join smoothly together the motion of the chin with the motion near the ears. To do this, we smoothly vary the weighting of the background and triphone shapes as we move along the jawline from the chin towards the ears.

The final stitching process is a three-way tradeoff in shape and texture among the fade-out lip image, the fade-in lip image, and the background image. As we move from phoneme to phoneme, the relative weights of the mouth shapes associated with the overlapping triphone-video images are changed. Within each frame, the relative weighting of the jaw shapes contributed by the background image and of the triphone-video images are varied spatially.

The derived fiduciary positions are used as control points in morphing. All morphs are done with the Beier-Neely algorithm [Beier92]. For each frame of the output image we need to warp four images: the two triphones, the replacement mask, and the background face. The warping is straightforward since we automatically generate high-quality control points using the eigenpoints algorithm.

5 RESULTS

We have applied Video Rewrite to several different training databases. We recorded one video dataset specifically for our evaluations. Section 5.1 describes our methods to collect this data and create lip-sync videos. Section 5.2 evaluates the resulting videos.

We also trained video models using truncated versions of our evaluation database. Finally, we used old footage of John F. Kennedy. We present the results from these experiments in Section 5.3.

5.1 Methods

We recorded about 8 minutes of video, containing 109 sentences, of a subject narrating a fairy tale. During the reading, the subject was asked to directly face the camera for some parts (still-head video) and to move and glance around naturally for others (moving-head video). We use these different segments to study the errors in local deformations separately from the errors in global spatial registration. The subject was also asked to wear a hat during the filming. We use this landmark to provide a quantitative evaluation of our global alignment. The hat is strictly outside all our alignment masks and our eigenpoints models. Thus, having the subject wear the hat does not effect the magnitude or type of errors that we expect to see in the animations—it simply provides us with a reference marker for the position and movement of her head.

To create a video model, we trained the system on all the still-head footage. Video Rewrite constructed and annotated the video model with just under 3500 triphone videos automatically, using HMM labeling of triphones and eigenpoint labeling of facial contours.

Video Rewrite was then given the target sentence, and was asked to construct the corresponding image sequence. To avoid unduly optimistic results, we removed from the database the triphone videos from training sentences similar to the target. A training sentence was considered similar to the target if the two shared a phrase two or more words long. Note that Video Rewrite would not normally pare the database in this manner: Instead, it would take advantage of these coincidences. We remove the similar sentences to avoid biasing our results.

We evaluated our output footage both qualitatively and quantitatively. Our qualitative evaluation was done informally, by a panel

of observers. There are no accepted metrics for evaluating lip-synced footage. Instead, we were forced to rely on the qualitative judgements listed in Section 5.2.

Only the (global) spatial registration is evaluated quantitatively. Since our subject wore a hat that moved rigidly with her upper head, we were able to measure quantitatively our global-registration error on this footage. We did so by first warping the full frame (instead of just the mouth region) of the triphone image into the coordinate frame of the background image. If this global transformation is correct, it should overlay the two images of the hat exactly on top of one another. We measured the error by finding the offset of the correlation peak for the image regions corresponding to the front of the hat. The offset of the peak is the registration error (in pixels).

5.2 Evaluation

Examples of our output footage can be seen at <http://www.interval.com/papers/1997-012/>. The top row of Figure 6 shows example frames, extracted from these videos. This section describes our evaluation criteria and the results.

5.2.1 Lip and Utterance Synchronization

How well are the lip motions synchronized with the audio? We evaluate this measure on the still-head videos. There occasionally are visible timing errors in plosives and stops.

5.2.2 Triphone-Video Synchronization

Do the lips flutter open and closed inappropriately? This artifact usually is due to synchronization error in overlapping triphone videos. We evaluated this measure on the still-head videos. We do not see any artifacts of this type.

5.2.3 Natural Articulation

Assuming that neither of the artifacts from Sections 5.2.1 or 5.2.2 appear, do the lip and teeth articulations look natural? Unnatural-looking articulation can result if the desired sequence of phonemes is not available in the database, and thus another sequence is used in its place. In our experiments, this replacement occurred on 31 percent of the triphone videos. We evaluated this measure on the

still-head videos. We do not see this type of error when we use the full video model. Additional experiments in this area are described in Section 5.3.1.

5.2.4 Fading-Mask Visibility and Extent

Does the fading mask show? Does the animation have believable texture and motion around the lips and chin? Do the dimples move in sync with the mouth? We evaluated this measure on all the output videos. The still-head videos better show errors associated with the extent of the fading mask, whereas the moving-head videos better show errors due to interactions between the fading mask and the global transformation. Without illumination correction, we see artifacts in some of the moving-head videos, when the subject looked down so that the lighting on her face changed significantly. These artifacts disappear with adaptive illumination correction [Burt83].

5.2.5 Background Warping

Do the outer edges of the jaw line and neck, and the upper portions of the cheeks look realistic? Artifacts in these areas are due to incorrect warping of the background image or to a mismatch between the texture and the warped shape of the background image. We evaluated this measure on all the output videos. In some segments, we found minor artifacts near the outer edges of the jaw.

5.2.6 Spatial Registration

Does the mouth seem to float around on the face? Are the teeth rigidly attached to the skull? We evaluated this measure on the moving-head videos. No registration errors are visible.

We evaluated this error quantitatively as well, using the hat-registration metric described in Section 5.1. The mean, median, and maximum errors in the still-head videos were 0.6, 0.5, and 1.2 pixels (standard deviation 0.3); those in the moving-head videos were 1.0, 1.0, and 2.0 pixels (standard deviation 0.4). For comparison, the face covers approximately 85×120 pixels.

5.2.7 Overall Quality

Is the lip-sync believable? We evaluated this measure on all the output videos. We judged the overall quality as excellent.

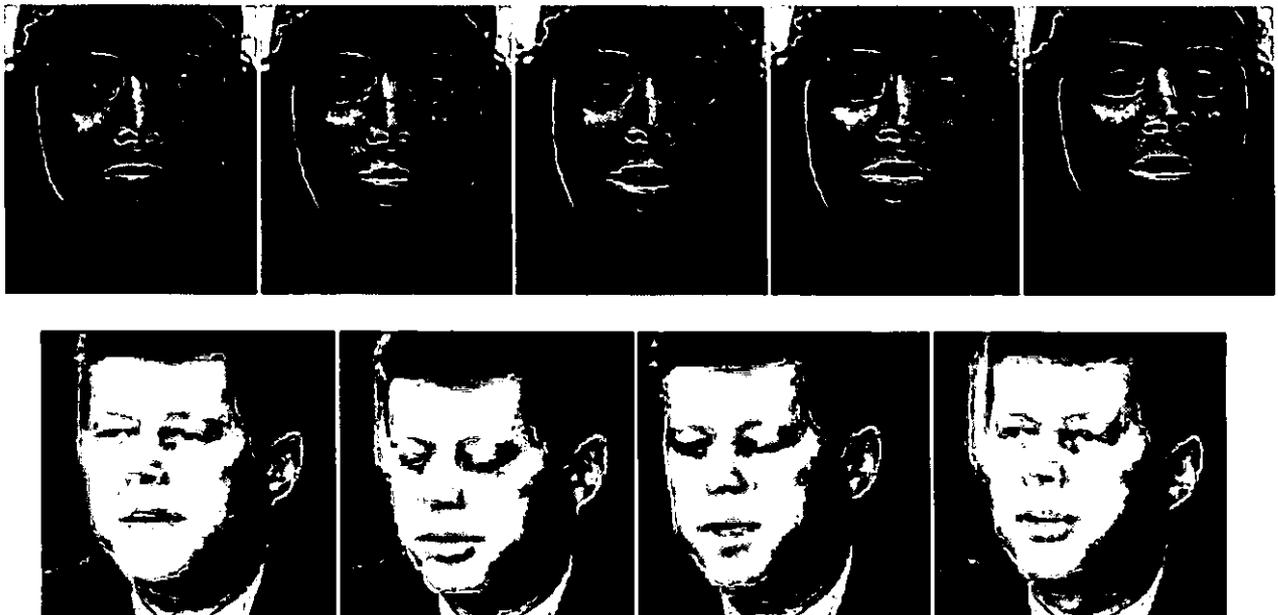


Figure 6: Examples of synthesized output frames. These frames show the quality of our output after triphone segments have been stitched into different background video frames.

5.3 Other Experiments

In this section, we examine our performance using steadily smaller training databases (Section 5.3.1) and using historic footage (Section 5.3.2).

5.3.1 Reduction of Video Model Size

We wanted to see how the quality fell off as the number of data available in the video model were reduced. With the 8 minutes of video, we have examples of approximately 1700 different triphones (of around 19,000 naturally occurring triphones); our animations used triphones other than the target triphones 31 percent of the time. What happens when we have only 1 or 2 minutes of data? We truncated our video database to one-half, one-quarter, and one-eighth of its original size, and then reanimated our target sentences. The percent of mismatched triphones increased by about 15 percent with each halving of the database (that is, 46, 58, and 74 percent of the triphones were replaced in the reduced datasets). The perceptual quality also degraded smoothly as the database size was reduced. The video from the reduced datasets are shown on our web site.

5.3.2 Reanimation of Historic Footage

We also applied Video Rewrite to public-domain footage of John F. Kennedy. For this application, we digitized 2 minutes (1157 triphones) of Kennedy speaking during the Cuban missile crisis. Forty-five seconds of this footage are from a close-up camera, about 30 degrees to Kennedy's left. The remaining images are medium shots from the same side. The size ratio is approximately 5:3 between the close-up and medium shots. During the footage, Kennedy moves his head about 30 degrees vertically, reading his speech from notes on the desk and making eye contact with a center camera (which we do not have).

We used this video model to synthesize new animations of Kennedy saying, for example, "Read my lips" and "I never met Forrest Gump." These animations combine the footage from both camera shots and from all head poses. The resulting videos are shown on our web site. The bottom row of Figure 6 shows example frames, extracted from these videos.

In our preliminary experiments, we were able to find the correct triphone sequences just 6% of the time. The lips are reliably synchronized to the utterance. The fading mask is not visible, nor is the background warping. However, the overall animation quality is not as good as our earlier results. The animations include some lip fluttering, because of the mismatched triphone sequences.

Our quality is limited for two reasons. The available viseme footage is distributed over a wide range of vertical head rotations. If we choose triphones that match the desired pose, then we cannot find good matches for the desired phoneme sequence. If we choose triphones that are well matched to the desired phoneme sequence, then we need to dramatically change the pose of the lip images. A large change in pose is difficult to model with our global (affine) transform. The lip shapes are distorted because we assumed, implicitly in the global transform, that the lips lie on a flat plane. Both the limited-triphone and pose problems can be avoided with additional data.

6 FUTURE WORK

There are many ways in which Video Rewrite could be extended and improved. The phonemic labeling of the triphone and background footage could consider the mouth- and jaw-shape information, as well as acoustic data [Bregler95]. Additional lip-image data and multiple eigenpoints models could be added, allowing larger out-of-plane head rotations. The acoustic data could be used in selecting the triphone videos, because facial expressions affect

voice qualities (you can hear a smile). The synthesis could be made real-time, with low-latency.

In Sections 6.1 through 6.3, we explore extensions that we think are most promising and interesting.

6.1 Alignment Between Lips and Target

We currently use the simplest approach to time aligning the lip sequences with the target utterance: We rely on the phoneme boundaries. This approach provides a rough alignment between the motions in the lip sequence and the sounds in the target utterance. As we mentioned in Section 4.1, however, the phoneme boundaries are both imprecise (the HMM alignment is not perfect) and coarse (significant visual and auditory landmarks occur within single phonemes).

A more accurate way to time align the lip motions with the target utterance uses dynamic time warping of the audio associated with each triphone video to the corresponding segment of the target utterance. This technique would allow us to time align the auditory landmarks from the triphone videos with those of the target utterance, even if the landmarks occur at subphoneme resolution. This time alignment, when applied to the triphone image sequence, would then align the visual landmarks of the lip sequence with the auditory landmarks of the target utterance.

The overlapping triphone videos would provide overlapping and conflicting time warpings. Yet we want to keep fixed the time alignment of the overlapping triphone videos, as dictated by the visual distances (Section 4.1 and 4.2). Research is needed in how best to trade off these potentially conflicting time-alignment maps.

6.2 Animation of Facial Features

Another promising extension is animation of other facial parts, based on simple acoustic features or other criteria. The simplest version of this extension would change the position of the eyebrows with pitch [Ohala94]. A second extension would index the video model by both triphone and expression labels. Using such labels, we would select smiling or frowning lips, as desired. Alternatively, we could impose the desired expression on a neutral mouth shape, for those times when the appropriate combinations of triphones and expression are not available. To do this imposition correctly, we must separate which deformations are associated with articulations, and which are associated with expressions, and how the two interact. This type of factorization must be learned from examples [Tenenbaum97].

6.3 Perception of Lip Shapes

In doing this work, we solved many problems—automatic labeling, matching, and stitching—yet we found many situations where we did not have sufficient knowledge of how people perceive speaking faces. We would like to know more about how important the correct lip shapes and motions are in lip synching. For example, one study [Owens85] describes the confusibility of consonants in vowel-consonant-vowel clusters. The clustering of consonants into viseme class depends on the surrounding vowel context. Clearly, we need more sophisticated distance metrics within and between viseme classes.

7 CONTRIBUTIONS

Video Rewrite is a facial animation system that is driven by audio input. The output sequence is created from real video footage. It combines background video footage, including natural facial movements (such as eye blinks and head motions) with natural footage of mouth and chin motions. Video Rewrite is the first facial-animation system to automate all the audio- and video-labeling tasks required for this type of reanimation.

Video Rewrite can use images from unconstrained footage both to create the video model of the mouth and chin motions and to provide a background sequence for the final output footage. It preserves the individual characteristics of the subject in the original footage, even while the subject appears to mouth a completely new utterance. For example, the temporal dynamics of John F. Kennedy's articulatory motions can be preserved, reorganized, and reimposed on Kennedy's face.

Since Video Rewrite retains most of the background frame, modifying only the mouth area, it is well suited to applications such as movie dubbing. The setting and action are provided by the background video. Video Rewrite maintains an actor's visual mannerisms, using the dynamics of the actor's lips and chin from the video model for articulatory mannerisms, and using the background video for all other mannerisms. It maintains the correct timing, using the action as paced by the background video and speech as paced by the new soundtrack. It undertakes the entire process without manual intervention. The actor convincingly mouths something completely new.

ACKNOWLEDGMENTS

Many colleagues helped us. Ellen Tauber and Marc Davis graciously submitted to our experimental manipulation. Trevor Darrell and Subutai Ahmad contributed many good ideas to the algorithm development. Trevor, Subutai, John Lewis, Bud Lassiter, Gaile Gordon, Kris Rahardja, Michael Bajura, Frank Crow, Bill Verplank, and John Woodfill helped us to evaluate our results and the description. Bud Lassiter and Chris Seguin helped us with the video production. We offer many thanks to all.

REFERENCES

- [Beier92] T. Beier, S. Neely. Feature-based image metamorphosis. *Computer Graphics*, 26(2):35-42, 1992. ISSN 0097-8930.
- [Black95] M.J. Black, Y. Yacoob. Tracking and recognizing rigid and non-rigid facial motions using local parametric models of image motion. *Proc. IEEE Int. Conf. Computer Vision*, Cambridge, MA, pp. 374-381, 1995. ISBN 0-8186-7042-8.
- [Bregler95] C. Bregler, S. Omohundro. Nonlinear manifold learning for visual speech recognition. *Proc. IEEE Int. Conf. Computer Vision*, Cambridge, MA, pp. 494-499, 1995. ISBN 0-8186-7042-8.
- [Burt83] P.J. Burt, E.H. Adelson. A multiresolution spline with application to image mosaics. *ACM Trans. Graphics*, 2(4): 217-236, 1983. ISSN 0730-0301.
- [Cohen93] M.M. Cohen, D.W. Massaro. Modeling coarticulation in synthetic visual speech. In *Models and Techniques in Computer Animation*, ed. N.M. Thalmann, D. Thalmann, pp. 139-156, Tokyo: Springer-Verlag, 1993. ISBN 0-3877-0124-9.
- [Covell96] M. Covell, C. Bregler. Eigenpoints. *Proc. Int. Conf. Image Processing*, Lausanne, Switzerland, Vol. 3, pp. 471-474, 1996. ISBN 0-7803-3258-x.
- [Guiard-Marigny94] T. Guiard-Marigny, A. Adjoudani, C. Benoit. A 3-D model of the lips for visual speech synthesis. *Proc. ESCA/IEEE Workshop on Speech Synthesis*, New Paltz, NY, pp. 49-52, 1994.
- [Kass87] M. Kass, A. Witkin, D. Terzopoulos. Snakes: Active contour models. *Int. J. Computer Vision*, 1(4):321-331, 1987. ISSN 0920-5691.
- [Kirby90] M. Kirby, L. Sirovich. Application of the Karhunen-Loeve procedure for the characterization of human faces. *IEEE PAMI*, 12(1):103-108, Jan. 1990. ISSN 0162-8828.
- [Lamel86] L. F. Lamel, R. H. Kessel, S. Seneff. Speech database development: Design and analysis of the acoustic-phonetic corpus. *Proc. Speech Recognition Workshop (DARPA)*, Report #SAIC-86/1546, pp. 100-109, McLean VA: Science Applications International Corp., 1986.
- [Lanitis95] A. Lanitis, C.J. Taylor, T.F. Cootes. A unified approach for coding and interpreting face images. *Proc. Int. Conf. Computer Vision*, Cambridge, MA, pp. 368-373, 1995. ISBN 0-8186-7042-8.
- [Lewis91] J.Lewis. Automated lip-sync: Background and techniques. *J. Visualization and Computer Animation*, 2(4):118-122, 1991. ISSN 1049-8907.
- [Litwinowicz94] P. Litwinowicz, L. Williams. Animating images with drawings. *SIGGRAPH 94*, Orlando, FL, pp. 409-412, 1994. ISBN 0-89791-667-0.
- [Morishima91] S. Morishima, H. Harashima. A media conversion from speech to facial image for intelligent man-machine interface. *IEEE J Selected Areas Communications*, 9 (4):594-600, 1991. ISSN 0733-8716.
- [Moulines90] E. Moulines, P. Emerard, D. Larreur, J. L. Le Saint, L. Le Faucheur, F. Marty, F. Charpentier, C. Sorin. A real-time French text-to-speech system generating high-quality synthetic speech. *Proc. Int. Conf. Acoustics, Speech, and Signal Processing*, Albuquerque, NM, pp. 309-312, 1990.
- [Ohala94] J.J. Ohala. The frequency code underlies the sound symbolic use of voice pitch. In *Sound Symbolism*, ed. L. Hinton, J. Nichols, J. J. Ohala, pp. 325-347, Cambridge UK: Cambridge Univ. Press, 1994. ISBN 0-5214-5219-8.
- [Owens85] E. Owens, B. Blazek. Visemes observed by hearing-impaired and normal-hearing adult viewers. *J. Speech and Hearing Research*, 28:381-393, 1985. ISSN 0022-4685.
- [Parke72] F. Parke. Computer generated animation of faces. *Proc. ACM National Conf.*, pp. 451-457, 1972.
- [Rabiner89] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Readings in Speech Recognition*, ed. A. Waibel, K. F. Lee, pp. 267-296, San Mateo, CA: Morgan Kaufmann Publishers, 1989. ISBN 1-5586-0124-4.
- [Scott94] K.C. Scott, D.S. Kagels, S.H. Watson, H. Rom, J.R. Wright, M. Lee, K.J. Hussey. Synthesis of speaker facial movement to match selected speech sequences. *Proc. Australian Conf. Speech Science and Technology*, Perth Australia, pp. 620-625, 1994. ISBN 0-8642-2372-2.
- [Tenenbaum97] J. Tenenbaum, W. Freeman. Separable mixture models: Separating style and content. In *Advances in Neural Information Processing 9*, ed. M. Jordan, M. Mozer, T. Patsche, Cambridge, MA: MIT Press, (in press).
- [Turk91] M. Turk, A. Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3(1):71-86, 1991. ISSN 0898-929X
- [Viterbi67] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimal decoding algorithm. *IEEE Trans. Inform. Theory*, IT-13:260-269, 1967. ISSN 0018-9448.
- [Waters95] K. Waters, T. Levergood. DECface: A System for Synthetic Face Applications. *J. Multimedia Tools and Applications*, 1 (4):349-366, 1995. ISSN 1380-7501.
- [Williams90] L. Williams. Performance-Driven Facial Animation. *Computer Graphics (Proceedings of SIGGRAPH 90)*, 24(4):235-242, 1990. ISSN 0097-8930.
- [Yuille89] A.L. Yuille, D.S. Cohen, P.W. Hallinan. Feature extraction from faces using deformable templates. *Proc. IEEE Computer Vision and Pattern Recognition*, San Diego, CA, pp. 104-109, 1989. ISBN 0-8186-1952-x.

Video Motion Capture

Christoph Bregler

Jitendra Malik

Computer Science Division
University of California, Berkeley
Berkeley, CA 94720-1776
bregler@cs.berkeley.edu, malik@cs.berkeley.edu

Abstract

This paper demonstrates a new vision based motion capture technique that is able to recover high degree-of-freedom articulated human body configurations in complex video sequences. It does not require any markers, body suits, or other devices attached to the subject. The only input needed is a video recording of the person whose motion is to be captured. For visual tracking we introduce the use of a novel mathematical technique, the product of exponential maps and twist motions, and its integration into a differential motion estimation. This results in solving simple linear systems, and enables us to recover robustly the kinematic degrees-of-freedom in noise and complex self occluded configurations. We demonstrate this on several image sequences of people doing articulated full body movements, and visualize the results in re-animating an artificial 3D human model. We are also able to recover and re-animate the famous movements of Eadweard Muybridge's motion studies from the last century. To the best of our knowledge, this is the first computer vision based system that is able to process such challenging footage and recover complex motions with such high accuracy.

CR Categories:

Keywords: Computer Vision, Animation, Motion Capture, Visual Tracking, Twist Kinematics, Exponential Maps, Muybridge

1 Introduction

In this paper, we offer a new approach to motion capture based just on ordinary video recording of the actor performing naturally. The approach does not require any markers, body suits or any other devices attached to the body of the actor. The actor can move about wearing his or her regular clothes. This implies that one can use historical footage—motion capture Charlie Chaplin's inimitable walk, for instance. Indeed in this paper we shall go even further back historically and show motion capture results from Muybridge sequences—the first examples of photographically recorded motion [15].

Motion capture occupies an important role in the creation of special effects. Its application to CG character animation has been much more controversial; SIGGRAPH 97 featured a lively panel debate[4] between its proponents and opponents. Our goal in this paper is not to address that debate. Rather we take it as a given that motion capture, like any other technology, can be correctly or incorrectly applied and we are merely extending its possibilities.

Our approach, from a user's point of view, is rather straightforward. The user marks limb segments in an initial frame; if multiple video streams are available from synchronized cameras, then the limb segments are marked in the corresponding initial frames in all of them. The computer program does the rest—tracking the multiple degrees of freedom of the human body configuration from frame to frame.

Attempts to track the human body without special markers go back quite a few years – we review past work in Sec. 2. However in spite of many years of work in computer vision on this problem, it is fair to describe it as not yet solved. There are many reasons why human body tracking is very challenging, compared to tracking other objects such as footballs, robots or cars. These include

1. *High Accuracy Requirements.* Especially in the context of motion capture applications, one desires to record all the degrees of freedom of the configuration of arms, legs, torso, head etc accurately from frame to frame. At playback time, any error will be instantly noticed by a human observer.
2. *Frequent inter-part occlusion* During normal motion, from any camera angle some parts of the body are occluded by other parts of the body
3. *Lack of contrast* Distinguishing the edge of a limb from, say the torso underneath, is made difficult by the fact that typically the texture or color of the shirt is usually the same in both regions.

Our contribution to this problem is the introduction of a novel mathematical technique, the product of exponential maps and twist motions, and its integration into a differential motion estimation scheme. This formalism will be explained fully in Section 3. The advantage of this particular formulation is that it results in the equations that need to be solved to update the kinematic chain parameters from frame to frame being *linear*. Also the only parameters that need to be solved for are the true degrees of freedom and pose parameters—there are no intermediate stages which may be unnecessarily hard. For instance recovering the local affine motion parameters of each and every limb segment separately is harder than the final goal of knowing the configuration of all the joints from frame to frame—the fact that the joints are constrained to move together reduces considerably the number of degrees of freedom. This in turn provides robustness to self-occlusions, loss of contrast, large motions etc.

We applied this technique to several video recordings of walking people and to the famous photo plates of Eadweard Muybridge. We achieved accurate tracking results with high degree-of-freedom full body models and could successfully re-animate the data. The accompanying video shows the tracking results and the naturalness of the animated motion capture data.

Section 2 reviews previous video tracking techniques, section 3 introduces the new motion tracking framework and its mathematical formulation, section 4 details our experiments, and we discuss the results and future directions in section 5.

2 Review

The earliest computer vision attempt to recognize human movements was reported by O'Rourke and Badler [16] working on syn-

thetic images using a 3D structure of rigid segments, joints, and constraints between them.

Marker-free visual tracking on video recordings of human bodies goes back to work by Hogg and by Rohr [8, 18]. Both systems are specialized to one degree-of-freedom walking models. Edge and line features are extracted from images and matched to a cylindrical 3D human body model. Higher degree-of-freedom articulated hand configurations are tracked by Regh and Kanade [17], full body configurations by Gravrila and Davis [7], and arm configurations by Kakadiaris and Metaxas [11] and by Goncalves and Perona [5]. All these approaches are demonstrated in constrained environments with high contrast edge boundaries. In most cases this is achieved by uniform backgrounds, and skintight clothing of uniform color. Also, in order to estimate 3D configurations, a camera calibration is needed. Alternatively, Weng et. al demonstrated how to track full bodies with color features [20], and Ju et. al showed motion based tracking of leg configurations [10]. No 3D kinematic chain models were used in the last two cases.

To the best of our knowledge, there is no system reported so far, which would be able to successfully track accurate high-degree-of-freedom human body configurations in the challenging footage that we will demonstrate here.

3 Articulated Tracking

There exist a wide range of visual tracking techniques in the literature ranging from edge feature based to region based tracking, and brute-force search methods to differential approaches.

Edge feature based tracking techniques usually require clean data with high contrast object boundaries. Unfortunately on human bodies such features are very noisy. Clothes have many folds. Also if the left and right leg have the same color and they overlap, they are separated only by low contrast boundaries.

Region based techniques can track objects with arbitrary texture. Such techniques attempt to match areas between consecutive frames. For example if the area describes a rigid planar object, a 2D affine deformation of this area has to be found. This requires the estimation of 6 free parameters that describe this deformation (x/y translation, x/y scaling, rotation, and shear). Instead of exhaustively searching over these parameters, differential methods link local intensity changes to parameter changes, and allow for Newton-step like optimizations.

In the following we will introduce a new region based differential technique that is tailored to articulated objects modeled by kinematic chains. We will first review a commonly used motion estimation framework [2, 19], and then show how this can be extended for our task, using the twist and product of exponential formulation [14].

3.1 Preliminaries

Assuming that changes in image intensity are only due to translation of local image intensity, a parametric image motion between consecutive time frames t and $t + 1$ can be described by the following equation:

$$I(x + u_x(x, y, \phi), y + u_y(x, y, \phi), t + 1) = I(x, y, t) \quad (1)$$

$I(x, y, t)$ is the image intensity. The motion model $u(x, y, \phi) = [u_x(x, y, \phi), u_y(x, y, \phi)]^T$ describes the pixel displacement dependent on location (x, y) and model parameters ϕ . For example, a 2D affine motion model with parameters $\phi = [a_1, a_2, a_3, a_4, d_x, d_y]^T$ is defined as

$$u(x, y, \phi) = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix} \quad (2)$$

The first-order Taylor series expansion of (1) leads to the commonly used gradient formulation [12]:

$$I_t(x, y) + [I_x(x, y), I_y(x, y)] \cdot u(x, y, \phi) = 0 \quad (3)$$

$I_t(x, y)$ is the temporal image gradient and $[I_x(x, y), I_y(x, y)]$ is the spatial image gradient at location (x, y) . Assuming a motion model of K degrees of freedom (in case of the affine model $K = 6$) and a region of $N > K$ pixels, we can write an over-constrained set of N equations. For the case that the motion model is linear (as in the affine case), we can write the set of equations in matrix form (see [2] for details):

$$\mathbf{H} \cdot \phi + \vec{z} = \vec{0} \quad (4)$$

where $\mathbf{H} \in \mathbb{R}^{N \times K}$, and $\vec{z} \in \mathbb{R}^N$. The least squares solution to (3) is:

$$\phi = -(\mathbf{H}^T \cdot \mathbf{H})^{-1} \cdot \mathbf{H}^T \vec{z} \quad (5)$$

Because (4) is the first-order Taylor series linearization of (1), we linearize around the new solution and iterate. This is done by warping the image $I(t + 1)$ using the motion model parameters ϕ found by (5). Based on the re-warped image we compute the new image gradients (3). Repeating this process is equivalent to a Newton-Raphson style minimization.

A convenient representation of the shape of an image region is a probability mask $w(x, y) \in [0, 1]$. $w(x, y) = 1$ declares that pixel (x, y) is part of the region. Equation (5) can be modified, such that it weights the contribution of pixel location (x, y) according to $w(x, y)$:

$$\phi = -((\mathbf{W} \cdot \mathbf{H})^T \cdot \mathbf{H})^{-1} \cdot (\mathbf{W} \cdot \mathbf{H})^T \vec{z} \quad (6)$$

\mathbf{W} is an $N \times N$ diagonal matrix, with $\mathbf{W}(i, i) = w(x_i, y_i)$. We assume for now that we know the exact shape of the region. For example, if we want to estimate the motion parameters for a human body part, we supply a weight matrix \mathbf{W} that defines the image support map of that specific body part, and run this estimation technique for several iterations. Section 3.4 describes how we can estimate the shape of the support maps as well.

Tracking over multiple frames can be achieved by applying this optimization technique successively over the complete image sequence.

3.2 Twists and the Product of Exponential Formula

In the following we develop a motion model $u(x, y, \phi)$ for a 3D kinematic chain under scaled orthographic projection and show how these domain constraints can be incorporated into one linear system similar to (6). ϕ will represent the 3D pose and angle configuration of such a kinematic chain and can be tracked in the same fashion as already outlined for simpler motion models.

3.2.1 3D pose

The pose of an object relative to the camera frame can be represented as a rigid body transformation in \mathbb{R}^3 using homogeneous coordinates (we will use the notation from [14]):

$$q_c = \mathbf{G} \cdot q_o \quad \text{with} \quad \mathbf{G} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & d_x \\ r_{2,1} & r_{2,2} & r_{2,3} & d_y \\ r_{3,1} & r_{3,2} & r_{3,3} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

$q_o = [x_o, y_o, z_o, 1]^T$ is a point in the object frame and $q_c = [x_c, y_c, z_c, 1]^T$ is the corresponding point in the camera frame. Using scaled orthographic projection with scale s , the point q_c in the camera frame gets projected into the image point $[x_{im}, y_{im}]^T = s \cdot [x_c, y_c]^T$.

The 3D translation $[d_x, d_y, d_z]^T$ can be arbitrary, but the rotation matrix:

$$\mathbf{R} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{bmatrix} \in SO(3) \quad (8)$$

has only 3 degrees of freedom. Therefore the rigid body transformation $\mathbf{G} \in SE(3)$ has a total of 6 degrees of freedom.

Our goal is to find a model of the image motion that is parameterized by 6 degrees of freedom for the 3D rigid motion and the scale factor s for scaled orthographic projection. Euler angles are commonly used to constrain the rotation matrix to $SO(3)$, but they suffer from singularities and don't lead to a simple formulation in the optimization procedure (for example [1] propose a 3D ellipsoidal tracker based on Euler angles). In contrast, the *twist* representation provides a more elegant solution [14] and leads to a very simple linear representation of the motion model. It is based on the observation that every rigid motion can be represented as a rotation around a 3D axis and a translation along this axis. A twist ξ has two representations: (a) a 6D vector, or (b) a 4×4 matrix with the upper 3×3 component as a skew-symmetric matrix:

$$\xi = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \quad \text{or} \quad \hat{\xi} = \begin{bmatrix} 0 & -\omega_x & \omega_y & v_1 \\ \omega_x & 0 & -\omega_z & v_2 \\ -\omega_y & \omega_z & 0 & v_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad (9)$$

ω is a 3D unit vector that points in the direction of the rotation axis. The amount of rotation is specified with a scalar angle θ that is multiplied by the twist: $\xi\theta$. The v component determines the location of the rotation axis and the amount of translation along this axis. See [14] for a detailed geometric interpretation. For simplicity, we drop the constraint that ω is unit, and discard the θ coefficient. Therefore $\xi \in \mathbb{R}^6$.

It can be shown [14] that for any arbitrary $\mathbf{G} \in SE(3)$ there exists a $\xi \in \mathbb{R}^6$ twist representation.

A twist can be converted into the \mathbf{G} representation with following exponential map:

$$\mathbf{G} = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} & d_x \\ r_{2,1} & r_{2,2} & r_{2,3} & d_y \\ r_{3,1} & r_{3,2} & r_{3,3} & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ = e^{\hat{\xi}} = \mathbf{I} + \hat{\xi} + \frac{(\hat{\xi})^2}{2!} + \frac{(\hat{\xi})^3}{3!} + \dots \quad (10)$$

3.2.2 Twist motion model

At this point we would like to track the 3D pose of a rigid object under scaled orthographic projection. We will extend this formulation in the next section to a kinematic chain representation. The pose of an object is defined as $[s, \xi^T]^T = [s, v_1, v_2, v_3, \omega_x, \omega_y, \omega_z]^T$. A point q_o in the object frame is projected to the image location (x_{im}, y_{im}) with:

$$\begin{bmatrix} x_{im} \\ y_{im} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot s \cdot e^{\hat{\xi}} \cdot q_o \quad (11)$$

The image motion of point (x_{im}, y_{im}) from time t to time $t + 1$ is:

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} x_{im}(t+1) - x_{im}(t) \\ y_{im}(t+1) - y_{im}(t) \end{bmatrix} \\ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \left(s(t+1) \cdot e^{\hat{\xi}(t+1)} \cdot q_o - s(t) \cdot e^{\hat{\xi}(t)} \cdot q_o \right) \\ = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \left((1+s') \cdot e^{\hat{\xi}'} - \mathbf{I} \right) \cdot s(t) q_c \quad (12)$$

$$\text{with } \hat{\xi}(t+1) = \hat{\xi}(t) + \hat{\xi}' \\ s(t+1) = s(t) \cdot (1+s')$$

Using the first order Taylor expansion from (10) we can approximate:

$$(1+s') \cdot e^{\hat{\xi}'} \approx (1+s') \cdot \mathbf{I} + (1+s') \cdot \hat{\xi}' \quad (13)$$

and can rewrite (12) as:

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} s' & -\omega'_x & \omega'_y & v'_1 \\ \omega'_x & s' & -\omega'_z & v'_2 \end{bmatrix} \cdot q_c \quad (14)$$

with

$$\omega(t+1) = \omega(t) + \frac{1}{1+s'} \cdot \omega' \\ v(t+1) = v(t) + \frac{1}{1+s'} \cdot v'$$

$\phi = [s', v'_1, v'_2, \omega'_x, \omega'_y, \omega'_z]^T$ codes the relative scale and twist motion from time t to $t + 1$. Note that (14) does not include v'_3 . Translation in the Z direction of the camera frame is not measurable under scaled orthographic projection.

Equation (14) describes the image motion of a point (x_i, y_i) in terms of the motion parameters ϕ and the corresponding 3D point $q_c(i)$ in the camera frame. The 3D point $q_c(i)$ is computed by intersecting the camera ray of the image point (x_i, y_i) with the 3D model. In this paper we assume that the body segments can be approximated by ellipsoidal 3D blobs. Therefore q_c is the solution of a quadratic equation. This computation has to be done only once for each new image. It is outside the Newton-Raphson iterations. It could be replaced by more complex models and rendering algorithms.

Inserting (14) into (3) leads to:

$$I_t + I_x \cdot [s', -\omega'_x, \omega'_y, v'_1] \cdot q_c + I_y \cdot [\omega'_x, s', -\omega'_z, v'_2] \cdot q_c = 0 \\ \Leftrightarrow I_t(i) + H_i \cdot [s, v'_1, v'_2, \omega'_x, \omega'_y, \omega'_z]^T = 0 \quad (15)$$

with $I_t := I_t(x_i, y_i)$, $I_x := I_x(x_i, y_i)$, $I_y := I_y(x_i, y_i)$

For N pixel positions we have N equations of the form (15). This can be written in matrix form:

$$\mathbf{H} \cdot \phi + \bar{z} = 0 \quad (16)$$

with

$$\mathbf{H} = \begin{bmatrix} H_1 \\ H_2 \\ \dots \\ H_N \end{bmatrix} \quad \text{and} \quad \bar{z} = \begin{bmatrix} I_t(x_1, y_1) \\ I_t(x_2, y_2) \\ \dots \\ I_t(x_N, y_N) \end{bmatrix}$$

Finding the least-squares solution (3D twist motion ϕ) for this equation is done using (6).

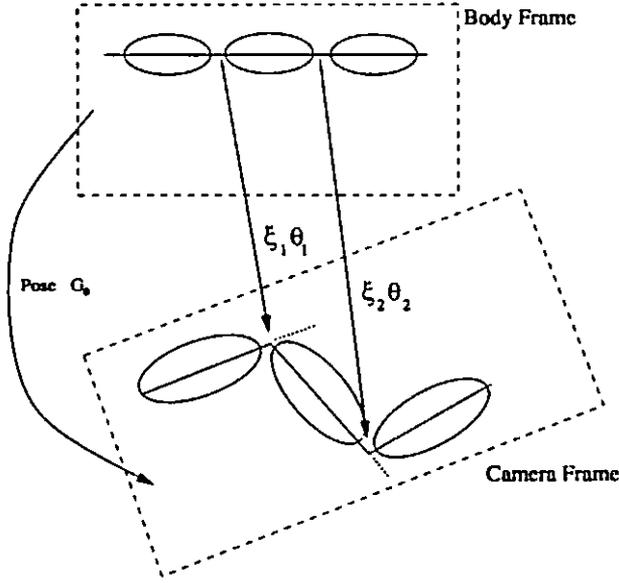


Figure 1: Kinematic chain defined by twists

3.2.3 Kinematic chain as a Product of Exponentials

So far we have parameterized the 3D pose and motion of a body segment by the 6 parameters of a twist ξ . Points on this body segment in a canonical object frame are transformed into a camera frame by the mapping $G_0 = e^{\xi}$. Assume that a second body segment is attached to the first segment with a joint. The joint can be defined by an axis of rotation in the object frame. We define this rotation axis in the object frame by a 3D unit vector ω_1 along the axis, and a point q_1 on the axis (figure 1). This is a so called revolute joint, and can be modeled by a twist ([14]):

$$\xi_1 = \begin{bmatrix} -\omega_1 \times q_1 \\ \omega_1 \end{bmatrix} \quad (17)$$

A rotation of angle θ_1 around this axis can be written as:

$$g_1 = e^{\xi_1 \cdot \theta_1} \quad (18)$$

$$(19)$$

The global mapping from object frame points on the first body segment into the camera frame is described by the following product:

$$\begin{aligned} g(\theta_1) &= G_0 \cdot e^{\xi_1 \cdot \theta_1} \\ q_c &= g(\theta_1) \cdot q_0 \end{aligned} \quad (20)$$

If we have a chain of $K+1$ segments linked with K joints (kinematic chain) and describe each joint by a twist ξ_k , a point on segment k is mapped from the object frame into the camera frame dependent on G_0 and angles $\theta_1, \theta_2, \dots, \theta_k$:

$$g_k(\theta_1, \theta_2, \dots, \theta_k) = G_0 \cdot e^{\xi_1 \cdot \theta_1} \cdot e^{\xi_2 \cdot \theta_2} \cdot \dots \cdot e^{\xi_k \cdot \theta_k} \quad (21)$$

This is called the **product of exponential maps** for kinematic chains.

The velocity of a segment k can be described with a twist V_k that is a linear combination of twists $\xi_1, \xi_2, \dots, \xi_k$ and the angular velocities $\dot{\theta}_1, \dot{\theta}_2, \dots, \dot{\theta}_k$ (see [14] for the derivations):

$$\begin{aligned} V_k &= \xi_1' \cdot \dot{\theta}_1 + \xi_2' \cdot \dot{\theta}_2 + \dots + \xi_k' \cdot \dot{\theta}_k \\ \xi_k' &= \text{Ad}_{e^{\xi_1 \cdot \theta_1} \dots e^{\xi_{k-1} \cdot \theta_{k-1}}} \xi_k \end{aligned} \quad (22)$$

Ad_g is the adjoint transformation associated with g .¹

Given a point q_c on the k 'th segment of a kinematic chain, its motion vector in the image is related to the angular velocities by:

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot [\hat{\xi}_1' \cdot \dot{\theta}_1 + \hat{\xi}_2' \cdot \dot{\theta}_2 + \dots + \hat{\xi}_k' \cdot \dot{\theta}_k] \cdot q_c \quad (23)$$

Recall (15) relates the image motion of a point q_c to changes in pose G_0 . We combine (15) and (23) to relate the image motion to the combined vector of pose change and angular change $\Phi = [s', v_1', v_2', \omega_x', \omega_y', \omega_z', \phi_1, \phi_2, \dots, \phi_K]^T$:

$$I_t + H_i \cdot [s, v_1', v_2', \omega_x', \omega_y', \omega_z']^T + J_i \cdot [\dot{\theta}_1, \dot{\theta}_2, \dots, \dot{\theta}_K]^T = 0 \quad (24)$$

$$[H, J] \cdot \Phi + \bar{z} = 0 \quad (25)$$

with

$$J = \begin{bmatrix} J_1 \\ J_2 \\ \dots \\ J_N \end{bmatrix} \quad \text{and } H, \bar{z} \text{ as before}$$

$$J_i = [J_{i1}, J_{i2}, \dots, J_{iK}]$$

$$J_{ik} = \begin{cases} [I_x, I_y] \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \cdot \hat{\xi}_k \cdot q_c \\ 0 & \text{if pixel } i \text{ is on a segment that} \\ & \text{is not affected by joint } \xi_k \end{cases}$$

The least squares solution to (25) is:

$$\Phi = -([H, J]^T \cdot [H, J])^{-1} \cdot [H, J]^T \cdot \bar{z} \quad (26)$$

Φ is the new estimate of the pose and angular change between two consecutive images. As outlined earlier, this solution is based on the assumption that the local image intensity variations can be approximated by the first-order Taylor expansion (3). We linearize around this new solution and iterate. This is done in warping the image $I(t+1)$ using the solution Φ . Based on the re-warped image we compute the new image gradients. Repeating this process of warping and solving (26) is equivalent to a Newton-Raphson style minimization.

3.3 Multiple Camera Views

In cases where we have access to multiple synchronized cameras, we can couple the different views in one equation system. Let's assume we have C different camera views at the same time. View c corresponds to following equation system (from (25)):

$$[H_c, J_c] \cdot \begin{bmatrix} \Omega_c \\ \phi_1 \\ \phi_2 \\ \dots \\ \phi_K \end{bmatrix} + \bar{z}_c = 0 \quad (27)$$

$\Omega_c = [s'_{c}, v'_{1,c}, v'_{2,c}, \omega'_{x,c}, \omega'_{y,c}, \omega'_{z,c}]^T$ describes the pose seen from view c . All views share the same angular parameters, because

$${}^1\text{Ad}_g = \begin{bmatrix} R & \hat{p} \cdot R \\ 0 & R \end{bmatrix}, \text{ and } g = \begin{bmatrix} R & p \\ 000 & 1 \end{bmatrix}$$

the cameras are triggered at the same time. We can simply combine all C equation systems into one large equation system:

$$\begin{bmatrix} \mathbf{H}_1 & \mathbf{0} & \dots & \mathbf{0} & \mathbf{J}_1 \\ \mathbf{0} & \mathbf{H}_2 & \dots & \mathbf{0} & \mathbf{J}_2 \\ \dots & \dots & \dots & \dots & \dots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{H}_C & \mathbf{J}_C \end{bmatrix} \cdot \begin{bmatrix} \Omega_1 \\ \Omega_2 \\ \dots \\ \Omega_C \\ \phi_1 \\ \phi_2 \\ \dots \\ \phi_K \end{bmatrix} + \begin{bmatrix} \tilde{z}_1 \\ \tilde{z}_2 \\ \dots \\ \tilde{z}_C \end{bmatrix} = \mathbf{0} \quad (28)$$

Operating with multiple views has three main advantages. The estimation of the angular parameters is more robust: (1) the number of measurements and therefore the number of equations increases with the number of views, (2) some angular configurations might be close to a singular pose in one view, whereas they can be estimated in a orthogonal view much better. (3) With more camera views, the chance decreases that one body part is occluded in all views.

3.4 Adaptive Support Maps using EM

As in (3), the update can be constrained to estimate the motion only in a weighted support map \mathbf{W}_k for each segment k using:

$$\Phi = -((\mathbf{W}_k \cdot [\mathbf{H}, \mathbf{J}])^T \cdot [\mathbf{H}, \mathbf{J}])^{-1} \cdot (\mathbf{W}_k \cdot [\mathbf{H}, \mathbf{J}])^T \tilde{\mathbf{z}} \quad (29)$$

We approximate the shape of the body segments as ellipsoids, and can compute the support map as the projection of the ellipsoids into the image. Such a support map usually covers a larger region, including pixels from the environment. That distracts the exact motion measurement. Robust statistics would be one solution to this problem [3]. Another solution is an EM-based layered representation [6, 9]. It is beyond the scope of this paper to describe this method in detail, but we would like to outline the method briefly: We start with an initial guess of the support map (ellipsoidal projection in this case). Given the initial \mathbf{W}_k , we compute the motion estimate Φ (M-step). Given such a Φ we can compute for each pixel location the probability that it complies with the motion model defined by Φ . We do this for each blob and the background (dominant motion) and normalize the sum of all probabilities per pixel location to 1. This results in new \mathbf{W}_k maps that are better "tuned" to the real shape of the body segment. In this paper we repeat the EM iteration only once.

3.5 Tracking Recipe

We summarize the algorithm for tracking the pose and angles of a kinematic chain in an image sequence:

- **Input:** $I(t)$, $I(t+1)$, $\mathbf{G}_0(t)$, $\theta_1(t)$, $\theta_2(t)$, ..., $\theta_K(t)$
(Two images and the pose and angles for the first image).
 - **Output:** $\mathbf{G}_0(t+1)$, $\theta_1(t+1)$, $\theta_2(t+1)$, ..., $\theta_K(t+1)$.
(Pose and angles for second image).
1. Compute for each image location (x_i, y_i) in $I(t)$ the 3D point $q_c(i)$ (using ellipsoids or more complex models and rendering algorithm).
 2. Compute for each body segment the support map \mathbf{W}_k .
 3. Set $\mathbf{G}_0(t+1) := \mathbf{G}_0(t)$, $\forall k: \theta_k(t+1) := \theta_k(t)$.

4. Iterate:

- (a) Compute spatiotemporal image gradients: I_t, I_x, I_y .
- (b) Estimate Φ using (29)
- (c) Update $\mathbf{G}_0(t+1) := \mathbf{G}_0(t+1) \cdot (1+s') \cdot e^{\frac{\mu}{1+s'}}$
- (d) $\forall k$ Update $\theta_k(t+1) := \theta_k(t+1) + \hat{\theta}_k$.
- (e) $\forall k$ Warp the region inside \mathbf{W}_k of $I(t+1)$ by $\mathbf{G}_0(t+1) \cdot g_k(t+1) \cdot (\mathbf{G}^t) \cdot g_k(t)^{-1}$.

3.6 Initialization

The visual tracking is based on an initialized first frame. We have to know the initial pose and the initial angular configuration. If more than one view is available, all views for the first time step have to be known. A user clicks on the 2D joint locations in all views at the first time step. Given that, the 3D pose and the image projection of the matching angular configuration is found in minimizing the sum of squared differences between the projected model joint locations and the user supplied model joint locations. The optimization is done over the poses, angles, and body dimensions. Example body dimensions are "upper-leg-length", "lower-leg-length", or "shoulder-width". The dimensions and angles have to be the same in all views, but the pose can be different. Symmetry constraints, that the left and right body lengths are the same, are enforced as well. Minimizing only over angles, or only over model dimensions results in linear equations similar to what we have shown so far. Unfortunately the global minimization criteria over all parameters is a tri-linear equation system, that cannot be easily solved by simple matrix inversions. There are several possible techniques for minimizing such functions. We achieved good results with a Quasi-Newton method and a mixed quadratic and cubic line search procedure.

4 Results

We applied this technique to video recordings in our lab and to photo-plate sequence of Eadweard Muybridge's motion studies.

4.1 Single camera recordings

Our lab video recordings were done with a single camera. Therefore the 3D pose and some parts of the body can not be estimated completely. Figure 2 shows one example sequences of a person walking in a frontparallel plane. We defined a 6 DOF kinematic structure: One blob for the body trunk, three blobs for the frontal leg and foot, connected with a hip joint, knee joint, and ankle joint, and two blobs for the arm connected with a shoulder and elbow joint. All joints have an axis orientation parallel to the Z-axis in the camera frame. The head blob was connected with one joint to the body trunk. The first image in figure 2 shows the initial blob support maps.

After the hand-initialization we applied the motion tracker to a sequence of 53 image frames. We could successfully track all body parts in this video sequence (see video). The video shows that the appearance of the upper leg changes significantly due to moving folds on the subject's jeans. The lower leg appearance does not change to the same extent. The constraints were able to enforce compatible motion vectors for the upper leg, based on more reliable measurements on the lower leg.

We can compare the estimated angular configurations with motion capture data reported in the literature. Murray, Brought, and

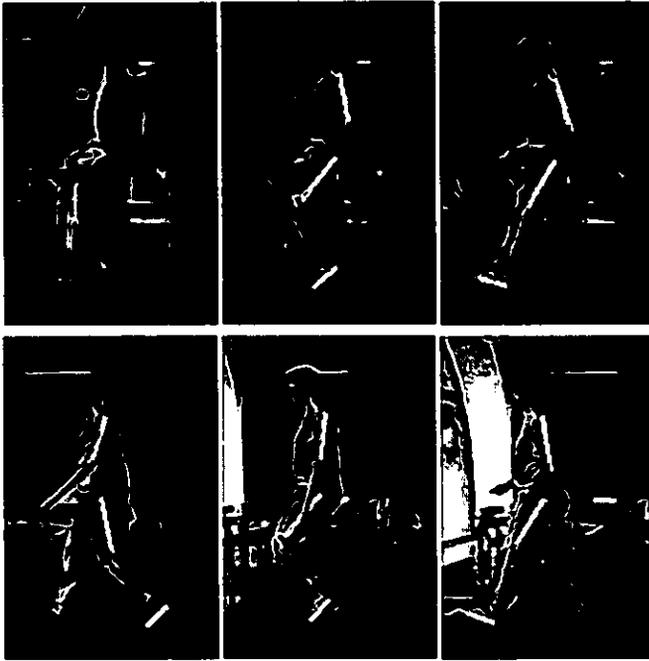


Figure 2: Example configurations of the estimated kinematic structure. First image shows the support maps of the initial configuration. In subsequent images the white lines show blob axes. The joint is the position on the intersection of two axes.

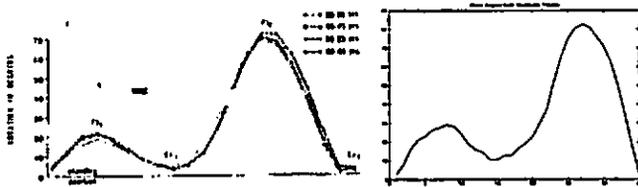


Figure 3: Comparison of a) data from [Murray et al] (left) and b) our motion tracker (right).

Kory published [13] such measurements for the hip, knee, and ankle joints. We compared our motion tracker measurements with the published curves and found good agreement. Figure 4.1a shows the curves for the knee and ankle reported in [13], and figure 4.1b shows our measurements.

We also experimented with a walking sequence of a subject seen from an oblique view with a similar kinematic model. As seen in figure 4, we tracked the angular configurations and the pose successfully over the complete sequence of 45 image frames. Because we use a scaled orthographic projection model, the perspective effects of the person walking closer to the camera had to be compensated by different scales. The tracking algorithm could successfully estimate the scale changes.

4.2 Digital Muybridge

The final set of experiments was done on historic footage recorded by Eadward Muybridge in 1884. His methods are of independent interest, as they predate motion pictures. Muybridge had his models walk in an open shed. Parallel to the shed was a fixed battery of 24 cameras. Two portable batteries of 12 cameras each were positioned at both ends of the shed, either at an angle of 90 deg relative to the shed or an angle of 60 deg. Three photographs were take

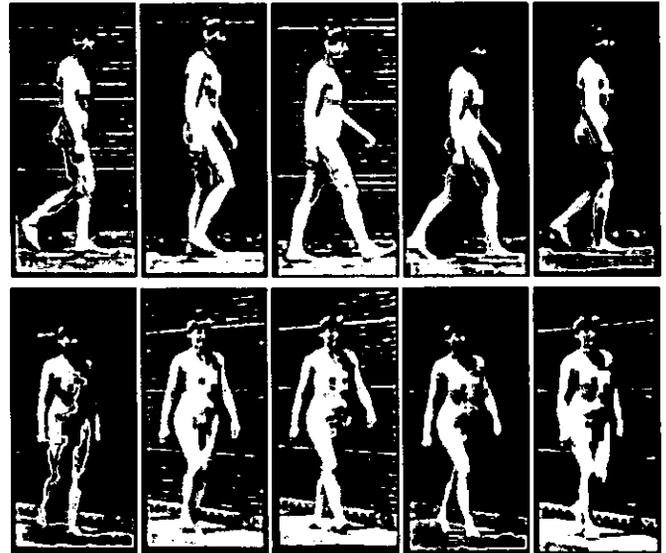


Figure 5: Eadward Muybridge, *The Human Figure in Motion*, Plate 97: Woman Walking. The first 5 frames show part of a walk cycle from one example view, and the second 5 frames show the same time steps from a different view

simultaneously, one from each battery. The effective 'framerate' of his technique is about two times lower than current video frame rates; a fact which makes tracking a harder problem. It is to our advantage that he took for each time step three pictures from different viewpoints.

Figure 4.2 and figure 4.2 shows example photo plates. We could initialize the 3D pose by labeling all three views of the first frame and running the minimization procedure over the body dimensions and poses: Figure 4.2 shows one example initialization. Every body segment was visible in at least one of the three camera views, therefore we could track the left and the right side of the person. We applied this technique to a walking woman and a walking man. For the walking woman we had 10 time steps available that contained 60% of a full walk cycle (figure 4.2). For this set of experiments we extended our kinematic model to 19 DOFs. The two hip joints, the two shoulder joints, and the neck joint, were modeled by 3 DOFs. The two knee joints and two elbow joints were modeled just by one rotation axis. Figure 4.2 shows the tracking results with the model overlaid. As you see, we could successfully track the complete sequence. To animate the tracking results we mirrored the left and right side angles to produce the remaining frames of a complete walk cycle. We animated the 3D motion capture data with a stick figure model and a volumetric model (figure 10), and it looks very natural. The video shows some of the tracking and animation sequences from several novel camera views, replicating the walk cycle performed over a century ago on the grounds of University of Pennsylvania.

For the visualization of the walking man sequence, we did not apply the mirroring, because he was carrying a boulder on his shoulder. This made the walk asymmetric. We re-animated the original tracked motion (figure 4.2) capture data for the man, and it also looked very natural.

Given the successful application of our tracking technique to multi-view data, we are planning to record with higher frame-rates our own multi-view video footage. We also plan to record a wider range of gestures.



Figure 4: Example configurations of the estimated kinematic structure of a person seen from an oblique view.

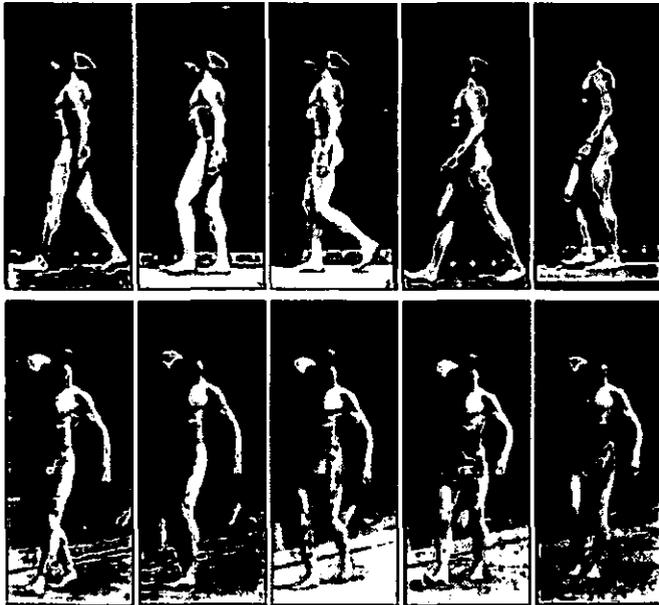


Figure 6: Eadweard Muybridge, *The Human Figure in Motion*, Plate 7: Man walking and carrying 75-LB boulder on shoulder. The first 5 frames show part of a walk cycle from one example view, and the second 5 frames show the same time steps from a different view

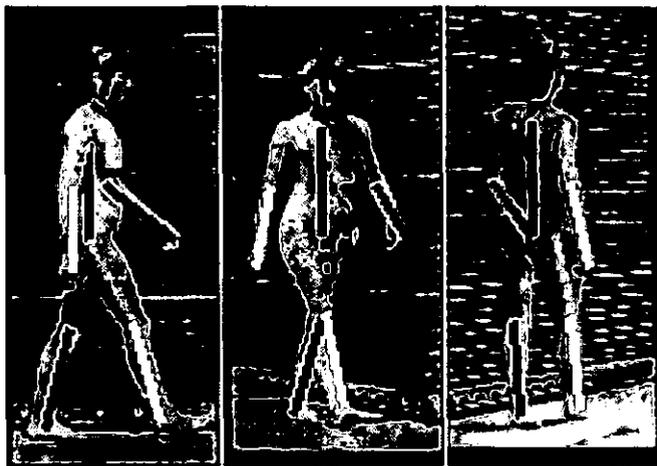


Figure 7: Initialization of Muybridge's Woman Walking: This visualizes the initial angular configuration projected to 3 example views.



Figure 8: Muybridge's Woman Walking: Motion Capture results. This shows the tracked angular configurations and its volumetric model projected to 2 example views.

5 Conclusion

In this paper, we have developed and demonstrated a new technique for video motion capture. The approach does not require any markers, body suits or any other devices attached to the body of the actor. The actor can move about wearing his or her regular clothes. We demonstrated results on video recordings of people walking both in frontoparallel and oblique views, as well as on the classic Muybridge photographic sequences recorded more than a century ago.

Visually tracking human motion at the level of individual joints is a very challenging problem. Our results are due, in large measure, to the introduction of a novel mathematical technique, the product of exponential maps and twist motions, and its integration into a differential motion estimation scheme. The advantage of this particular formulation is that it results in the equations that need to be solved to update the kinematic chain parameters from frame to frame being linear, and that it is not necessary to solve for any redundant or unnecessary variables.

Future work will concentrate on dealing with very large motions, as may happen, for instance, in videotapes of high speed running. The approach developed in this paper is a differential method, and therefore may be expected to fail when the motion from frame-to-frame is very large. We propose to augment the technique by the use of an initial coarse search stage. Given a close enough starting value, the differential method will converge correctly.

Acknowledgements

We would like to thank Charles Ying for creating the Open-GL animations and video editing, Shankar Sastry, Lara Crawford, Jerry Feldman, John Canny, and Jianbo Shi for fruitful discussions, Chad Carson for helping to write this document, and Interval Research Corp, and the California State MICRO program for supporting this research.

References

- [1] S. Basu, I.A. Essa, and A.P. Pentland. Motion regularization for model-based head tracking. In *International Conference on Pattern Recognition*, 1996.
- [2] J.R. Bergen, P. Anandan, K.J. Hanna, and R. Hingorani. Hierarchical model-based motion estimation. In *ECCV*, pages 237–252, 1992.
- [3] M.J. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1):75–104, Jan 1996.
- [4] G. Cameron, A. Bustanoby, K. Cope, S. Greenberg, C. Hayes, and O. Ozoux. Panel on motion capture and cg character animation. *SIGGRAPH 97*, pages 442–445, 1997.
- [5] L. Concalves, E.D. Bernardo, E. Ursella, and P. Perona. Monocular tracking of the human arm in 3d. In *Proc. Int. Conf. Computer Vision*, 1995.
- [6] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39, 1977.
- [7] D.M. Gavrila and L.S. Davis. Towards 3-d model-based tracking and recognition of human movement: a multi-view approach. In *Proc. of the Int. Workshop on Automatic Face- and Gesture-Recognition, Zurich, 1995*, 1995.
- [8] D. Hogg. A program to see a walking person. *Image Vision Computing*, 5(20), 1983.
- [9] A. Jepson and M.J. Black. Mixture models for optical flow computation. In *Proc. IEEE Conf. Computer Vision Pattern Recognition*, pages 760–761, New York, 1993.
- [10] S.X. Ju, M.J. Black, and Y. Yacoob. Cardboard people: A parameterized model of articulated motion. In *2nd Int. Conf. on Automatic Face- and Gesture-Recognition, Killington, Vermont*, pages 38–44, 1996.
- [11] I.A. Kakadiaris and D. Metaxas. Model-based estimation of 3d human motion with occlusion based on active multi-viewpoint selection. In *CVPR*, 1996.
- [12] B.D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *Proc. 7th Int. Joint Conf. on Art. Intell.*, 1981.
- [13] M.P. Murray, A.B. Drought, and R.C. Kory. Walking patterns of normal men. *Journal of Bone and Joint Surgery*, 46-A(2):335–360, March 1964.
- [14] R.M. Murray, Z. Li, and S.S. Sastry. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

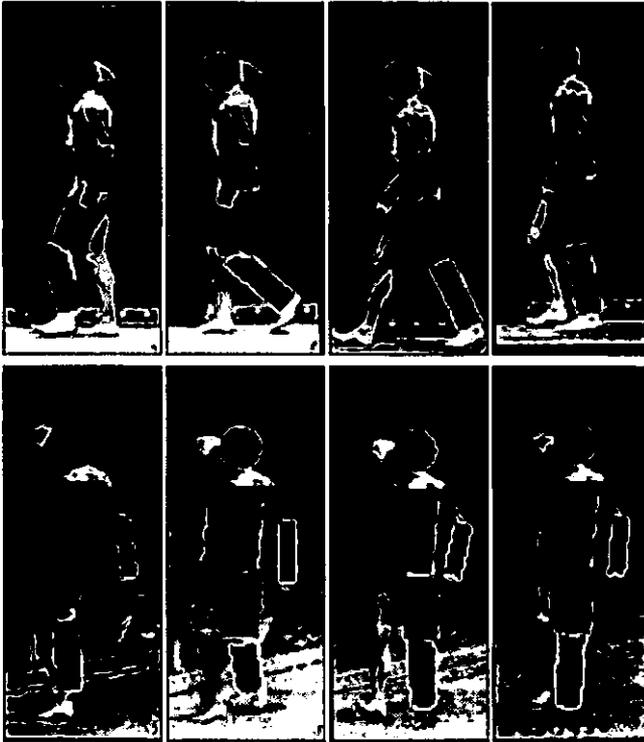


Figure 9: Muybridge's Man Walking: Motion Capture results. This shows the tracked angular configurations and its volumetric model projected to 2 example views.

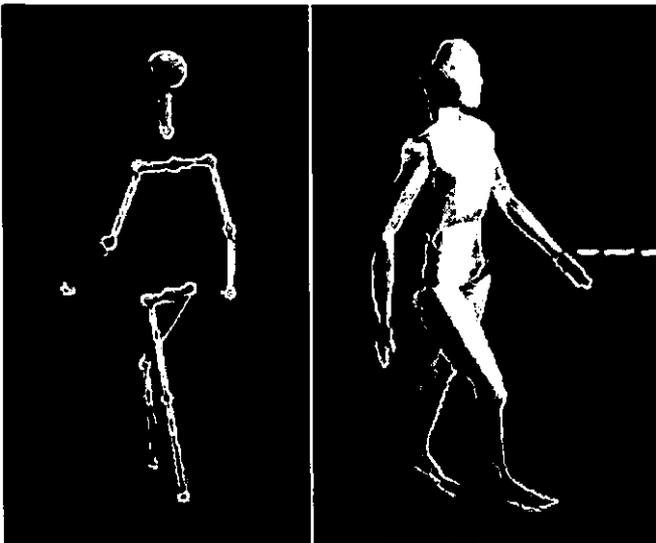


Figure 10: Computer models used for the animation of the Muybridge motion capture. Please check out the video to see the quality of the animation.

- [15] Eadweard Muybridge. *The Human Figure In Motion*. Various Publishers, latest edition by Dover Publications, 1901.
- [16] J. O'Rourke and N. I. Badler. Model-based image analysis of human motion using constraint propagation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2(6):522-536, November 1980.
- [17] J.M. Rehg and T. Kanade. Model-based tracking of self-occluding articulated objects. In *Proc. Int. Conf. Computer Vision*, 1995.
- [18] K. Rohr. Incremental recognition of pedestrians from image sequences. In *Proc. IEEE Comput. Soc. Conf. Comput. Vision and Pattern Recogn.*, pages 8-13, New York City, June, 1993.
- [19] J. Shi and C. Tomasi. Good features to track. In *CVPR*, 1994.
- [20] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfunder: Real-time tracking of the human body. In *SPIE Conference on Integration Issues in Large Commercial Media Delivery Systems*, volume 2615, 1995.