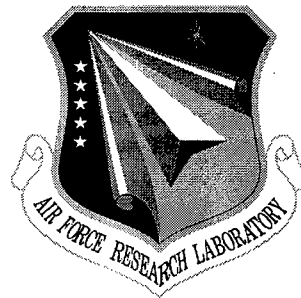AFRL-IF-RS-TR-2000-37
Final Technical Report
April 2000

STEERABLE GAZE CONTROL FOR A VIDEO-BASED VIRTUAL SURVEILLANT

Charles R. Dyer

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK**

20000530 048

DTIC QUALITY INSPECTED 3

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.
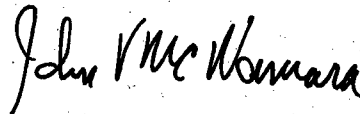
AFRL-IF-RS-TR-2000-37 has been reviewed and is approved for publication.

APPROVED: *Peter J. Costianes*

PETER J. COSTIANES
Project Engineer

FOR THE DIRECTOR: *John V. McNamara*

JOHN V. MCNAMARA, Technical Advisor
Information & Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE APRIL 2000	3. REPORT TYPE AND DATES COVERED Final Jun 97 - Sep 99

4. TITLE AND SUBTITLE
STEERABLE GAZE CONTROL FOR A VIDEO-BASED VIRTUAL SURVEILLANT

6. AUTHOR(S)
Charles R. Dyer

5. FUNDING NUMBERS
C - F30602-97-1-0138
PE - 62301E
PR - E663
TA - 00
WU - 01

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Wisconsin - Madison
Department of Computer Science
1210 W. Dayton St.
Madison WI 53706

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Agency   Air Force Research Laboratory/IFED
3701 N Fairfax Dr   32 Brooks Road
Arlington VA 22203-1714   Rome NY 13441-4514

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2000-37

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Peter J. Costianes/IFED/(315) 330-4030

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
This is the final report summarizing the video surveillance and monitoring (VSAM) research conducted at the University of Wisconsin - Madison during the period 6 June 1997 to 5 September 1999. In this project, we developed new image-based visualization methods for synthesizing new views of a real scene from a virtual camera. New techniques were devised and evaluated, including methods called view morphing, dynamic view morphing, voxel coloring, and a new structure-from-motion technique.

14. SUBJECT TERMS
Video Surveillance, View Interpolation, Image-Based Rendering, Computer Vision

15. NUMBER OF PAGES
64

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

STEERABLE GAZE CONTROL FOR A VIDEO-BASED VIRTUAL SURVEILLANT
Charles R. Dyer

Contractor: University of Wisconsin
Contract Number: F30602-97-1-0138
Effective Date of Contract: 6 June 1997
Contract Expiration Date: 5 September 1999
Short Title of Work: Steerable Gaze Control for a Video-
                           Based Virtual Surveillant
Period of Work Covered: Jun 97 – Sep 99

Principal Investigator: Charles R. Dyer
            Phone: (608) 262-1965
AFRL Project Engineer: Peter J. Costianes
            Phone: (315) 330-4030

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

List of Figures

List of Tables

Abstract

In this project we developed new image-based visualization tools that enable human or automated viewing of a real scene from a virtual camera. The methods enable capabilities for monitoring areas of interest and for assessing objects' dispositions, as best determined by operator viewing preferences and task-specific targets and activities. In addition, the methods can be used for video compression, gap filling in video, and obstruction removal in image data.

Specific methods for image-based view synthesis that were invented include view morphing, dynamic view morphing, voxel coloring, and a new structure-from-motion technique. *View morphing* takes two images from two widely-separated views of a static scene and creates an interpolated sequence of photorealistic, in-between views. *Dynamic view morphing* extends the view morphing approach to dynamic scenes, producing an interpolation of both viewpoint and scene motion. That is, given two input images taken at different times from different viewpoints, a sequence of images is synthesized that smoothly transitions from the first image's viewpoint at time 0 to the second image's viewpoint at time 1. No scene models or knowledge of the real motions of objects is assumed. *Voxel coloring* is a method we developed that uses information from an arbitrary number of views, creating a voxel representation of the scene by using a correlation test to determine if a region of space is opaque or empty. To make the algorithm fast enough for real-time interactive use, we also investigated several extensions of the basic procedure that exploit spatial and temporal coherence. Finally, we defined and studied a novel *structure-from-motion* technique for recovering scene structure and external camera parameters from a set of images. Our approach overcomes some of the limitations of existing methods.

1 Introduction

This final report summarizes activity conducted at the University of Wisconsin–Madison under Agreement No. F30602-97-1-0138 sponsored by the Defense Advanced Research Projects Agency (DARPA) and monitored by the Air Force Materiel Command, Air Force Research Laboratory (AFRL), titled "Steerable Gaze Control for a Video-Based Virtual Surveillant" for the period 6 June 1997 to 5 September 1999. The DARPA Program Manager was George Lukes, and the AFRL Project Engineer was Peter Costianes.

The major goal of this project was to enhance human and automated surveillance capabilities by developing new technologies that enable scene visualization by a virtual camera. In addition, these technologies enable other modeling, rendering, and virtual-modification operations of a real three-dimensional scene, e.g., urban areas and battlefields, by adaptively combining a set of reference images of that scene. The methods developed will enhance capabilities for monitoring areas of interest and for assessing objects' dispositions, as best determined by operator viewing preferences and task-specific targets and activities. Examples of military activities of this type include battlefield and facility visualizations and flybys, mission rehearsal and planning, site analysis, treaty monitoring, and accident analysis. This is important for such customers as intelligence analysts, special forces operators, combat engineers, and command post planners. For each of the above tasks the raw sensor data may not be well-matched to the tasks that use that data. Different tasks require different views of a scene, and so the "optimal" views for a particular task may not have been captured. Also, a sensor may be time-shared for multiple uses in a single mission, e.g., when a single sensor is slewed between multiple targets and areas of interest. For these reasons it is advantageous to synthesize photorealistic, customized images and videos that are tuned to an operator's viewing preferences.

Our approach is image based in that the input is a set of images or video, and no auxiliary data sources such as terrain data or site models are assumed. Instead, images are leveraged to use the rich information they supply about scene structure and, by definition, photorealistic appearance. The challenge is to obtain much of the flexibility of geometry-based rendering in terms of viewing position and orientation, ability to change lighting, ability to virtually modify the scene itself, and so on.

We assume that views are captured by multiple cameras that are widely separated and arbitrarily positioned around the environment. The views from the cameras are partially overlapping so that multiple cameras view most scene points. The 3D scene can be arbitrarily complex. Output is a sequence of images to be viewed by a person or used as input to other image-understanding algorithms. For both visualization and further processing we focus on producing photorealistic images of novel views and smooth sequences of views. Thus the main emphasis is on image appearance, not surface reconstruction or model building (though this may be a by-product).

Many issues related to image-based view synthesis were investigated under this grant. New methods were developed called view morphing, dynamic view morphing, voxel coloring, real-time voxel coloring, and Euclidean scene reconstruction by projected error refinement. Work done in each of these areas is described in the following section. Summary papers are given in [Dye97, Dye98]. A list of publications associated with the grant is given in Section 3.

2   Technical Accomplishments

2.1   View Morphing

We developed an approach, called view morphing, that produces photorealistic new views given just two reference views, needs only sparse correspondence information, uncalibrated cameras, and widely-separated reference views. These assumptions mean that the method can be used in a wide variety of applications and physical settings.

The problem of synthesizing new views of a real scene by warping a pair of reference views is represented schematically in Figure 1. This problem is interesting because (1) it has applications of practical importance, such as stereo viewing [MB95a], teleconferencing [BP96], latency compensation in VR, video compression, and gap filling between two images; (2) it is amenable to a thorough bottom-up analysis; and (3) it provides a base case for the more general problem of view synthesis from arbitrary sets of views.

Towards this end, the first objective was to demonstrate that this view synthesis problem is indeed solvable, i.e., given two perspective views of a static scene, under what conditions may new views be unambiguously predicted? We point out that this question is nontrivial, given that basic quantities like optical flow and shape are *not* uniquely computable due to inherent ambiguities (e.g., the aperture problem [Mar82]).

The second goal was to develop an algorithm that produces correct, high-quality, synthetic views of a scene from two reference images. The algorithm produces correct views when the underlying assumptions are satisfied, and is also sufficiently robust to cope with large deviations, e.g., non-static scenes or varying illumination.

In the remainder of this section we describe our results in these two areas. Our publications related to this work are given in [SD96c, SD95, SD97c, SD96a, Sei97a, Sei97b].

First, we show that a specific range of perspective views is theoretically determined from two or more reference views, under a generic visibility assumption called *monotonicity*. This result applies when either the relative camera configurations are known or when only the fundamental matrix is available. In addition, we present a simple technique for generating this particular range of views using image interpolation. Importantly, the method relies only on *measurable* image information,
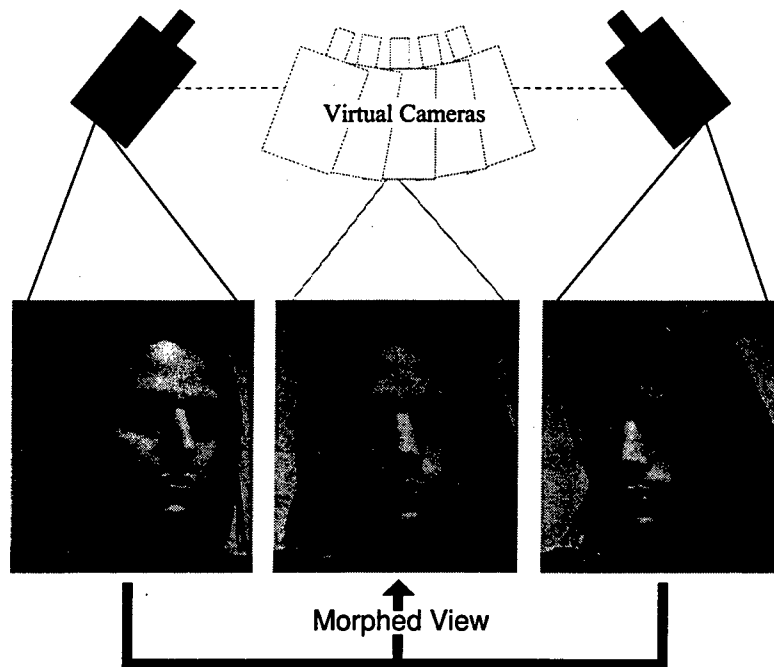
Figure 1: View morphing between two images of an object taken from two different viewpoints produces the illusion of physically moving a virtual camera.

avoiding ill-posed correspondence problems entirely. Furthermore, all processing occurs at the scanline level, effectively reducing the original 3D synthesis problem to a set of simple 1D image transformations that can be implemented efficiently on existing graphics workstations. The work presented here extends to perspective projection previous results on the orthographic case [SD95].

We begin by introducing the *monotonicity* constraint and describing its implications for view synthesis in Section 2.1.1. Section 2.1.2 considers *how* views can be synthesized, and describes a simple and efficient algorithm called *view morphing* for synthesizing new views by interpolating images, under the assumption that the relative geometry of the two cameras is known. Section 2.1.3 investigates the case where the images are *uncalibrated*, i.e., the camera geometry is unknown. Section 2.1.4 presents extensions when three or more basis views are available. Section 2.1.5 presents some results on real images.

2.1.1 View Synthesis and Monotonicity

Can the appearance from new viewpoints of a static three-dimensional scene be predicted from a set of basis views of the same scene? One way of addressing this question is to consider view synthesis as a two-step process—reconstruct the scene from the basis views using stereo or structure-from-motion methods and then reproject to form the new view. The problem with this paradigm is

3
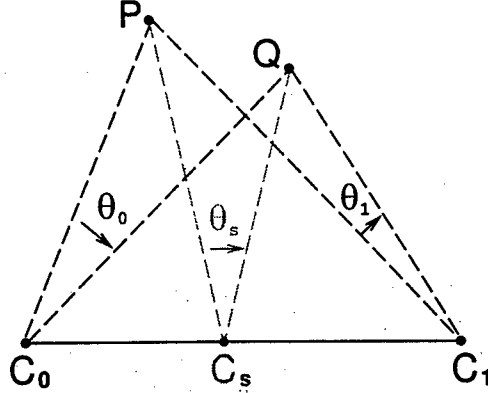
Figure 2: The monotonicity constraint holds when $\theta_0 \theta_1 > 0$ for all pairs of scene points P and Q in the same epipolar plane.

that view synthesis becomes at least as difficult as 3D scene reconstruction. This conclusion is especially unfortunate in light of the fact that 3D reconstruction from sparse images is generally ambiguous—a number of different scenes may be consistent with a given set of images; it is an ill-posed problem. This suggests that view synthesis is also ill-posed.

In this section we present an alternate paradigm for view synthesis that avoids 3D reconstruction and dense correspondence as intermediate steps, instead relying only on *measurable* quantities, computable from a set of basis images. We first consider the conditions under which reconstruction is ill-posed and then describe why these conditions do not impede view synthesis. Ambiguity arises within regions of uniform intensity in the images. Uniform image regions provide shape and correspondence information only at boundaries. Consequently, 3D reconstruction of these regions is not possible without additional assumptions. Note however that boundary information is sufficient to predict the appearance of these regions in new views, since the region's interior is assumed to be uniform. This argument hinges on the notion that uniform regions are "preserved" in different views, a constraint formalized by the condition of *monotonicity* which we introduce next.

Consider two views, V_0 and V_1, with respective optical centers C_0 and C_1, and images I_0 and I_1. Denote $\overline{C_0 C_1}$ as the line segment connecting the two optical centers. Any point P in the scene determines an epipolar plane containing P, C_0, and C_1 that intersects the two images in conjugate epipolar lines. The monotonicity constraint dictates that all visible scene points appear in the same order along conjugate epipolar lines of I_0 and I_1. This constraint is used commonly in stereo matching because the fixed relative ordering of points along epipolar lines simplifies the correspondence problem. Despite its usual definition with respect to epipolar lines and images, monotonicity constrains only the location of the optical centers with respect to points in the scene—the image planes may be chosen arbitrarily. An alternate definition that isolates this dependence more clearly is shown in Figure 2. Any two scene points P and Q in the same
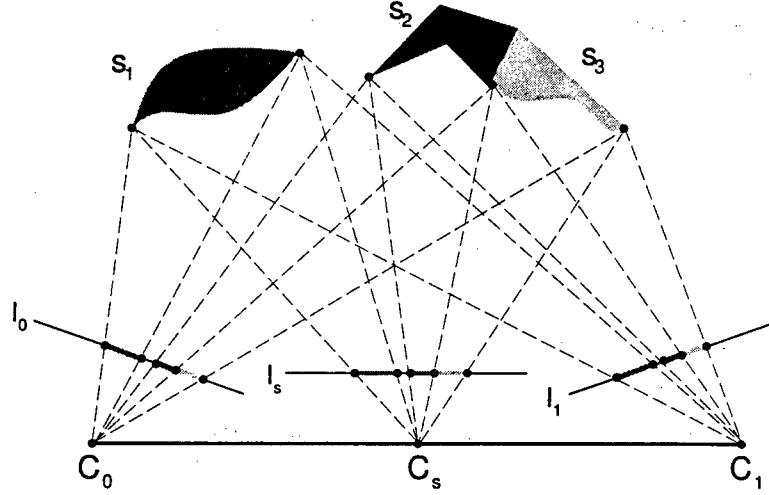
Figure 3: Although the projected intervals in l_0 and l_1 do not provide enough information to reconstruct S_1, S_2 and S_3, they are sufficient to predict the appearance of l_s.

epipolar plane determine angles θ_0 and θ_1 with the optical centers C_0 and C_1. The monotonicity constraint dictates that for all such points θ_0 and θ_1 must be nonzero and of equal sign. The fact that no constraint is made on the image planes is of primary importance for view synthesis because it means that *monotonicity is preserved under homographies*, i.e., under image reprojection. This fact will be essential in the next section for developing an algorithm for view synthesis.

A useful consequence of monotonicity is that it extends to cover a continuous range of views in-between V_0 and V_1. We say that a third view V_s is *in-between* V_0 and V_1 if its optical center C_s is on $\overline{C_0 C_1}$. Observe that monotonicity is violated only when there exist two scene points, P and Q, in the same epipolar plane such that the infinite line PQ through P and Q intersects $\overline{C_0 C_1}$. But PQ intersects $\overline{C_0 C_1}$ if and only if it intersects either $\overline{C_0 C_s}$ or $\overline{C_s C_1}$. Therefore monotonicity applies to in-between views as well, i.e., signs of angles are preserved and visible scene points appear in the same order along conjugate epipolar lines of all views along $\overline{C_0 C_1}$. We therefore refer to the range of views with centers on $\overline{C_0 C_1}$ as a *monotonic range* of viewspace. Notice that this range gives a lower bound on the range of views for which monotonicity is satisfied in the sense that the latter set contains the former. For instance, in Figure 2 monotonicity is satisfied for all views on the open ray from the point $C_0 C_1 \cap PQ$ through both camera centers. However, without *a priori* knowledge of the geometry of the scene, we can infer only that monotonicity is satisfied for the range $\overline{C_0 C_1}$.

The property that monotonicity applies to in-between views is quite powerful and is sufficient to completely predict the appearance of the visible scene from all viewpoints along $\overline{C_0 C_1}$. Consider the projections of a set of uniform Lambertian surfaces (each surface has uniform radiance, but any

5

two surfaces can have different radiances) into views V_0 and V_1. Figure 3 shows cross sections S_1, S_2, and S_3 of three such surfaces projecting into conjugate epipolar lines l_0 and l_1. Each connected cross section projects to a uniform interval (i.e., an interval of uniform intensity) of l_0 and l_1. The monotonicity constraint induces a correspondence between the endpoints of the intervals in l_0 and l_1, determined by their relative ordering. The points on S_1, S_2, and S_3 projecting to the interval endpoints are determined from this correspondence by triangulation. We will refer to these scene points as *visible endpoints* of S_1, S_2, and S_3.

Now consider an in-between view, V_s, with image I_s and corresponding epipolar line l_s. As a consequence of monotonicity, S_1, S_2, and S_3 project to three uniform intervals along l_s, delimited by the projections of their visible endpoints. Notice that the intermediate image does not depend on the specific shapes of surfaces in the scene, only on the positions of their visible endpoints. **Any number of distinct scenes could have produced I_0 and I_1, but each one would also produce the same set of intermediate images.** Hence, all views along $\overline{C_0C_1}$ are determined from I_0 and I_1. This result demonstrates that view synthesis under monotonicity is an inherently well-posed problem—and is therefore much easier than 3D reconstruction and related motion analysis tasks requiring smoothness conditions and regularization techniques.

A final question concerns the *measurability* of monotonicity. That is, can we determine if two images satisfy monotonicity by inspecting the images themselves or must we know the answer *a priori*? Strictly speaking, monotonicity is not measurable in the sense that two images may be consistent with multiple scenes, some of which satisfy monotonicity and others that do not. However, we can determine whether or not two images are *consistent* with a scene for which monotonicity applies, by checking that each epipolar line in the first image is a monotonic warp of its conjugate in the second image.

2.1.2 View Morphing Algorithm

The previous section established that certain views are determined from two basis views under an assumption of monotonicity. In this section we present a simple approach for synthesizing these views based on image interpolation. The procedure takes as input two images, I_0 and I_1, their respective projection matrices, Π_0 and Π_1, and a third projection matrix Π_s representing the configuration of a third view along $\overline{C_0C_1}$. The result is a new image I_s representing how the visible scene appears from the third viewpoint.

We begin with a special case where the image planes are parallel and aligned with $\overline{C_0C_1}$. This configuration is often used in stereo applications and will be referred to as the *parallel configuration*. The situation is expressed algebraically using the projection equations as follows. A camera is represented by a 3×4 homogeneous matrix $\Pi = [H \mid -HC]$. The optical center is given by C and the image plane normal is the last row of H. A scene point (X, Y, Z) is expressed in
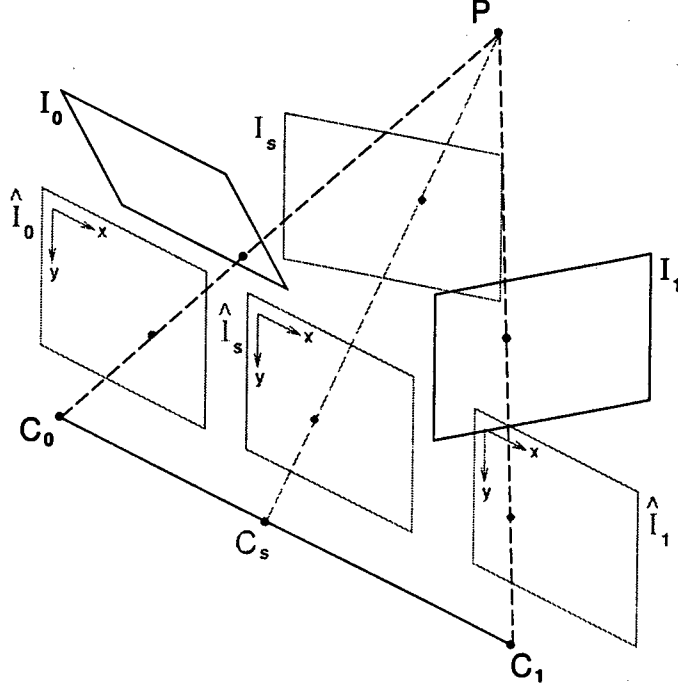
Figure 4: The three steps in view morphing: (1) Original images I_0 and I_1 are prewarped (rectified) to be parallel, (2) $\hat{I}_s$ is produced by interpolation, and (3) $\hat{I}_s$ is postwarped to form I_s.

homogeneous coordinates as $P = [X \ Y \ Z \ 1]^T$ and an image point (x, y) by $p = [x \ y \ 1]^T$. Because homogeneous structures are invariant under scalar multiplication, sP and P represent the same point, and similarly for sp and p. We therefore reserve the notation P and p for points whose last coordinate is 1. All other multiples of these points will be denoted as $\tilde{P}$ and $\tilde{p}$. The perspective projection equation is:

$$\tilde{p} = \Pi P$$

In the parallel configuration, the projection matrices may be chosen so that $\Pi_0 = [I \ | \ -C_0]$ and $\Pi_1 = [I \ | \ -C_1]$, where I is the 3×3 identity matrix. Without loss of generality, we assume that C_0 is at the world origin and $\overline{C_0 C_1}$ is parallel to the world X-axis so that $C_1 = [C_X \ 0 \ 0]^T$. Let p_0 and p_1 be projections of a scene point $P = [X \ Y \ Z \ 1]^T$ in the two views, respectively. Linear interpolation of p_0 and p_1 yields

$$
\begin{aligned}
(1 - s)p_0 + sp_1 &= (1 - s)\frac{1}{Z}\Pi_0 P + s\frac{1}{Z}\Pi_1 P \\
&= \frac{1}{Z}\Pi_s P
\end{aligned}
$$

where

$$\Pi_s = (1 - s)\Pi_0 + s\Pi_1 \tag{1}$$

7

Image interpolation, or *morphing* [BN92], therefore produces a new view whose projection matrix, Π_s, is a linear interpolation of Π_0 and Π_1 and whose optical center is $C_s = [sC_X \ 0 \ 0]^T$. Eq. (1) indicates that in the parallel configuration, any parallel view along $\overline{C_1C_2}$ may be synthesized simply by interpolating corresponding points in the two basis views. In other words, image interpolation induces an interpolation of viewpoint for this special camera geometry.

To interpolate general views with projection matrices $\Pi_0 = [H_0 \ | \ -H_0C_0]$ and $\Pi_1 = [H_1 \ | \ -H_1C_1]$, we first apply homographies H_0^{-1} and H_1^{-1} to convert I_0 and I_1 to a parallel configuration. This procedure is identical to rectification techniques used in stereo vision [RZFH95]. This suggests a three-step procedure for view synthesis:

1. Prewarp: $\hat{I}_0 = H_0^{-1}I_0$, $\hat{I}_1 = H_1^{-1}I_1$

2. Morph: linearly interpolate positions and intensities of corresponding pixels in $\hat{I}_0$ and $\hat{I}_1$ to form $\hat{I}_s$

3. Postwarp: $I_s = H_s\hat{I}_s$

Rectification is possible providing that the epipoles are outside of the respective image borders. If this condition is not satisfied, it is still possible to apply the procedure if the prewarped images are never explicitly constructed, i.e., if the prewarp, morph, and postwarp transforms are concatenated into a pair of aggregate warps [SD96c]. The prewarp step implicitly requires selection of a particular epipolar plane on which to reproject the basis images. Although the particular plane can be chosen arbitrarily, certain planes may be more suitable due to image sampling considerations.

2.1.3 Uncalibrated View Morphing

In order to use the view morphing algorithm presented in Section 2.1.2, we must find a way to rectify the images without knowing the projection matrices. Towards this end, it can be shown [SD96b] that two images are in the parallel configuration when their fundamental matrix is given, up to scalar multiplication, by

$$\hat{F} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix}$$

We seek a pair of homographies H_0 and H_1 such that the prewarped images $\hat{I}_0 = H_0^{-1}I_0$ and $\hat{I}_1 = H_1^{-1}I_1$ have the fundamental matrix given by Eq. (2.1.3). In terms of F the condition on H_0 and H_1 is

$$H_1^T F H_0 = \hat{F} \tag{2}$$

Solutions to Eq. (2) are discussed in [SD96b, RZFH95].

We have established that two images can be rectified, and therefore interpolated, without knowing their projection matrices. As in Section 2.1.2, interpolation of the prewarped images results in new views along $\overline{C_0 C_1}$. In contrast to the calibrated case however, the postwarp step is under-specified; there is no obvious choice for the homography that transforms $\hat{I}_s$ to I_s. One solution is to have the user provide the homography directly or indirectly by specification of a small number of image points [LF94, SD96c]. Another method is to simply interpolate the components of H_0^{-1} and H_1^{-1}, resulting in a continuous transition from I_0 to I_1 [SD96b]. Both methods for choosing the postwarp transforms generally result in the synthesis of *projective* views. A projective view is a perspective view warped by a 2D affine transformation.

2.1.4 Three Views and Beyond

Up to this point we have focused on image synthesis from exactly two basis views. The extension to more views is straightforward. Suppose for instance that we have three basis views that satisfy monotonicity pairwise ((I_0, I_1), (I_0, I_2), and (I_1, I_2) each satisfy monotonicity). Three basis views permit synthesis of a triangular region of viewspace, delimited by the three optical centers. Each pair of basis images determines the views along one side of the triangle, spanned by $\overline{C_0 C_1}$, $\overline{C_1 C_2}$, and $\overline{C_2 C_0}$.

What about interior views, i.e., views with optical centers in the interior of the triangle? Indeed, any interior view can be synthesized by a second interpolation, between a corner and a side view of the triangle. However, the assumption that monotonicity applies pairwise between corner views is not sufficient to infer monotonicity between interior views in the closed triangle $\triangle C_0 C_1 C_2$; monotonicity is not transitive. In order to predict interior views, a slightly stronger constraint is needed. *Strong monotonicity* dictates that for every pair of scene points P and Q, the line PQ does not intersect $\triangle C_0 C_1 C_2$. Strong monotonicity is a direct generalization of monotonicity; in particular, strong monotonicity of $\triangle C_0 C_1 C_2$ implies that monotonicity is satisfied between every pair of views centered in this triangle, and vice-versa. Consequently, strong monotonicity permits synthesis of any view in $\triangle C_0 C_1 C_2$.

Now suppose we have n basis views with optical centers $C_0, ..., C_{n-1}$ and that strong monotonicity applies between each triplet of basis views[1]. By the preceding argument, any triplet of basis views determines the triangle of views between them. In particular, any view on the convex hull $\mathcal{H}$ of $C_0, ..., C_{n-1}$ is determined, as $\mathcal{H}$ is comprised of a subset of these triangles. Furthermore, the interior views are also determined: let C be a point in the interior of $\mathcal{H}$ and choose a corner C_i on $\mathcal{H}$. The line through C and C_i intersects $\mathcal{H}$ in a point K. Since K lies on the convex hull, it represents the optical center of a set of views produced by two or fewer interpolations. Because C lies on $\overline{C_i K}$, all views centered at C are determined as well by one additional interpolation,

---

[1]In fact, strong monotonicity for each triangle on the convex hull of $C_0, ..., C_{n-1}$ is sufficient.

9

providing monotonicity is satisfied between C_i and K. To establish this last condition, observe that for monotonicity to be violated there must exist two scene points P and Q such that PQ intersects $\overline{C_i K}$, implying that PQ also intersects $\mathcal{H}$. Thus, PQ intersects at least one triangle $\triangle C_i C_j C_k$ on $\mathcal{H}$, violating the assumption of strong monotonicity. In conclusion, n basis views determine the 3D range of viewspace contained in the convex hull of their optical centers.

This constructive argument suggests that arbitrarily large regions of viewspace may be constructed by adding more basis views. However, the prediction of any range of view-space depends on the assumption that *all* possible pairs of views within that space satisfy monotonicity. In particular, a monotonic range may span no more than a single aspect of an aspect graph [SD96b], thus limiting the range of views that may be predicted. Nevertheless, it is clear that a discrete set of views implicitly describes scene appearance from a continuous range of viewpoints.

2.1.5 Experimental Results

We have applied the view morphing algorithm to many pairs of reference images, three of which are shown in Figure 5. Each pair of images was uncalibrated and the fundamental matrix was computed from several manually-specified point correspondences.

The first pair of images shows two views of a face. A sparse set of user-specified feature correspondences was used to determine the correspondence map [SD96c]. The synthesized image represents a view halfway between the two basis views. Some artifacts occur in regions where monotonicity is violated, e.g., near the right ear.

The second pair of images shows a wooden mannequin. This is an object that would be difficult to reconstruct due to lack of texture, but is relatively easy to synthesize views. In this example, image correspondences were automatically determined. Some local artifacts are visible where monotonicity is violated (e.g., left foot). Blurring is caused by image resampling, which is done three times in the current implementation. The problem may be ameliorated by super-sampling the intermediate images or by concatenating the multiple image transforms into two aggregate warps and resampling only once [SD96c].

2.1.6 Discussion

We have studied the question of which views of a static scene can be predicted from a set of two or more basis views, under perspective projection. The following results were shown: under monotonicity, two perspective views determine scene appearance from the set of all viewpoints on the line between their optical centers. Second, under strong monotonicity, a volume of viewspace is determined, corresponding to the convex hull of the optical centers of the basis views. Third, new perspective views may be synthesized by rectifying a pair of images and then interpolating

10
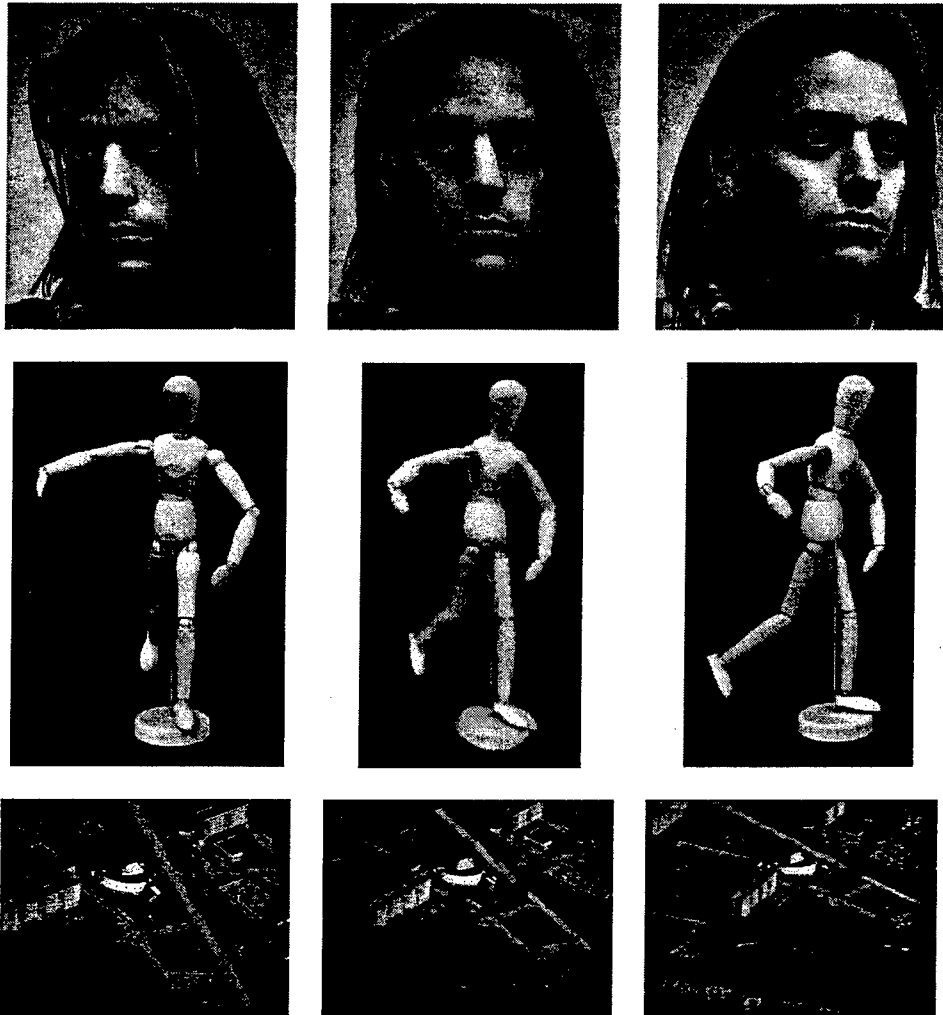
Figure 5: Reference views (left and right) of a face (top), mannequin (middle) and outdoor scene from Predator (bottom), with a synthesized view (center) halfway in-between each pair.

corresponding pixels, one scanline at a time, using a procedure called *view morphing*. Fourth, view synthesis is possible even when the views are uncalibrated, provided the *fundamental matrix* is known. In the uncalibrated case, the synthesized images represent *projective* views of the scene.

2.2 Dynamic View Morphing

View interpolation [CW93] involves creating a series of virtual views of a scene that, taken together, represent a continuous and physically-correct transition between two reference views of the scene. Previous work on view interpolation has been restricted to static scenes. Dynamic scenes change over time and, consequently, these changes will be evident in two reference views that are captured at different times. Therefore, view interpolation for dynamic scenes must portray a continuous
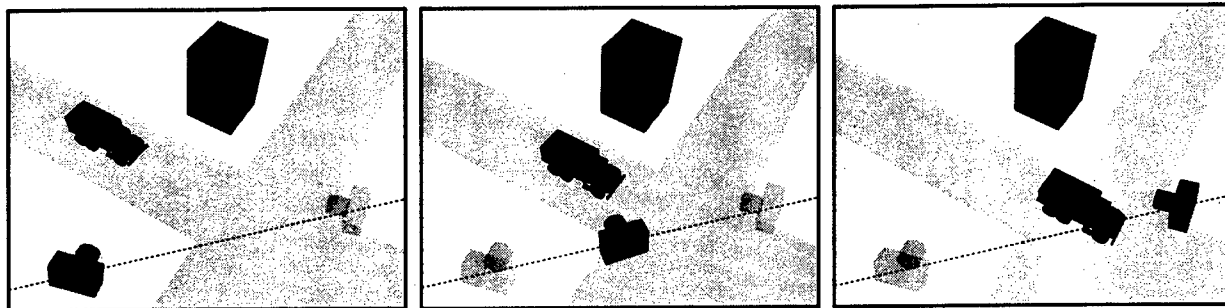
Figure 6: A dynamic scene at three different times. The goal of view interpolation for dynamic scenes is to synthesize the view from the camera in the middle frame starting with only the two reference views from the cameras in the left and right frames.

change in viewpoint *and* a continuous change in the scene itself in order to transition smoothly between the reference views (Figure 6).

Our approach to this problem is based on our earlier work on *view morphing* [SD96c], which provides a method for interpolating between two widely-spaced views of a static scene. The technique has several strengths that make it suitable for practical applications. First, only two reference views are assumed. Second, it does not require that camera calibration be provided nor does it need to calculate the camera parameters. Third, the method works even when only a sparse set of correspondences between the reference views is known. If more information about the reference views is available, this information can be used for added control over the output and for increased realism.

In addition to view morphing, numerous existing methods could be used to create view interpolations for static scenes [Fau92, MB95b, AS97, SD97a, TK92]. However, none of these methods is directly applicable to dynamic scenes. Avidan and Sashua [AS98] provide a method for recovering the geometry of dynamic scenes in which the objects move along straight-line trajectories. Once the geometry is recovered, dynamic view interpolations could be created using the standard graphics pipeline. However, their algorithm does not apply to the problem discussed in this paper because it assumes that five or more views are available and that the camera matrix for each view is known or can be recovered. There are several mosaicing techniques for dynamic scenes [IAH95, Dav98], but mosaicing involves piecing together several small-field views to create a single large-field view, whereas view interpolation involves synthesizing new views from vantage points not in the reference set.

Because the original view morphing algorithm assumes a static scene, we refer to it as *static view morphing* to distinguish it from the *dynamic view morphing* technique described here. Our publications on dynamic view morphing are [MD98a, MD98b, MD99, MD00].

We seek to perform view interpolation directly from the reference views, without additional

12

information about the scene. Consequently, there will be a missing interval of time between when the reference views were captured, and it will be impossible to know for certain what occurred during the missing interval. It is not our goal in this work to try and deduce the most likely manner in which the scene changed. Instead, we are interested in portraying *some* possible way in which the scene could have changed, and we want the portrayal to be physically correct and continuous.

Our method is for dynamic scenes that satisfy the following assumption: For each object in the scene, all of the changes that the object undergoes during the missing time interval, when taken together, are equivalent to a single, rigid translation.

The term *object* has a specific meaning in this paper, defined by the condition given above: An object is a group of particles in a scene for which there exists a fixed vector $\mathbf{u} \in \Re^3$ such that each particle's total motion during the missing time interval is equal to $\mathbf{u}$.

A method for dynamic view interpolation, even if it is physically accurate, may be unsatisfactory if it portrays objects moving along unreasonable trajectories. For instance, when portraying a car driving across a bridge, it is essential that the car stay on the bridge during the entire sequence. To address this problem, we have developed techniques for portraying both straight-line motion (in a camera-based coordinate frame) and straight-line, constant-velocity motion (in camera and world coordinate frames). For brevity, we refer to the latter style of portrayal as *linear motion*. Figure 6 depicts a linear motion view interpolation.

If the reference cameras share the same position in world coordinates, then the virtual camera shares that position as well and straight-line motion relative to the virtual camera also implies straight-line motion in world coordinates. However, this may not be the case if the virtual camera moves during the view interpolation, as Figure 7 demonstrates. It is easy to show that if all objects can be portrayed undergoing linear motion in camera coordinates, then the virtual camera can be considered undergoing linear motion in world coordinates, in which case all the objects will undergo linear motion in world coordinates as well.

2.2.1 Preliminary Concepts

We assume the two reference views are captured at time $t = 0$ and time $t = 1$ through pinhole cameras, which are denoted *camera A* and *camera B*, respectively.

We always use a *fixed-camera formulation*, meaning we assume that the two reference cameras are at the same location and that the world moves around them; this is accomplished by subtracting the actual displacement between the two cameras from the motion vectors of all objects in the scene. Under this assumption, the camera matrices are just 3×3 and each camera is equivalent to a basis for $\Re^3$. Note that no assumption is made about the cameras other than that they share the same
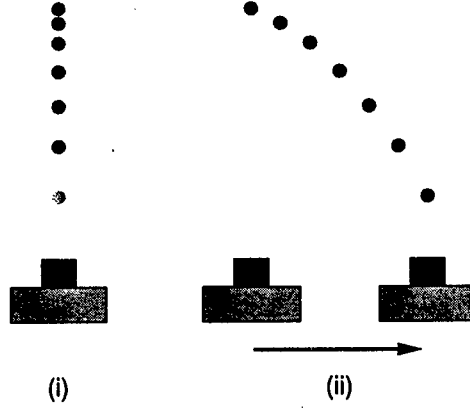
Figure 7: (i) A round object is filmed moving along a trajectory that is a straight line in the camera's frame of reference. The object is shown at equal time intervals and does not move at constant velocity. (ii) If the camera was in motion during the filming, then the object did not follow a straight-line trajectory in world coordinates.

optical center; the camera matrices can be completely different.

We let U denote the "universal" or "world" coordinate frame, and use the notation $\mathcal{T}_{UA}$ to mean the transformation between basis U and basis A. Hence $\mathcal{T}_{UA}$ is the camera matrix for A. Of particular interest to our work is the matrix $\mathcal{T}_{AB}$. Note that capital script letters will always represent 3×3 matrices; in particular, $\mathcal{I}$ is the identity matrix.

A position or a direction in space exists independently of what basis is used to measure it; we will use a subscript letter when needed to denote a particular basis. For instance, if e is the direction between two cameras (that are not at the same location), then e_A is e measured in basis A. The quantity e is called the *epipole*. The fundamental matrix $\mathcal{F}$ for two cameras A and B that are at different locations has the following representation [Har94]:

$$\mathcal{F} = [\mathbf{e}_B]_\times \mathcal{T}_{AB} \tag{3}$$

Here $[\cdot]_\times$ denotes the cross product matrix. When the two cameras share the same optical center, the fundamental matrix is 0 and has no meaning. However, for each moving object Ω in the scene, we can define a new kind of fundamental matrix. If, after making the fixed-camera assumption, Ω is moving in direction $\mathbf{u}$, then the fundamental matrix *for the object* is:

$$\mathcal{F}_\Omega = [\mathbf{u}_B]_\times \mathcal{T}_{AB} \tag{4}$$

The epipoles of $\mathcal{F}_\Omega$ are the vanishing points of Ω as viewed from the two reference cameras, and the epipolar lines trace out trajectories for points on Ω.
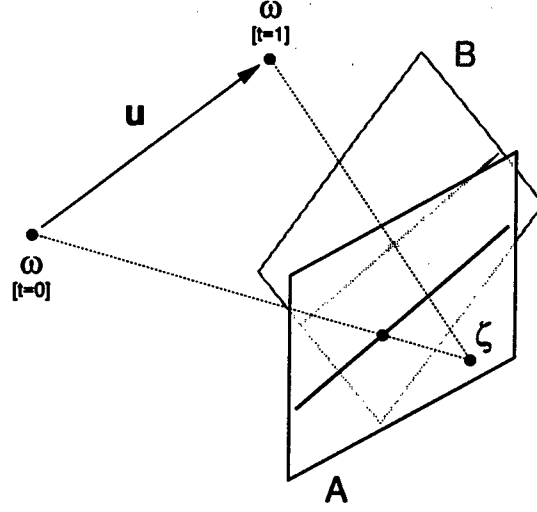
Figure 8: Cameras A and B share the same optical center ζ and are viewing a point on an object that translates by $\mathbf{u}$. The image planes of the cameras are parallel to each other and to $\mathbf{u}$, and hence interpolation will produce a physically-correct view of the object. On each image plane a line parallel to $\mathbf{u}$ is shown.

2.2.2 View Interpolation for a Single Moving Object

Assume the two reference·cameras share the same optical center and are viewing a point ω that is part of an object Ω whose translation vector is $\mathbf{u}$. Let $\mathbf{q}$ and $\mathbf{q}+\mathbf{u}$ denote the position of ω at time $t = 0$ and $t = 1$, respectively (Figure 8).

Assume for this subsection that the image planes of the cameras are parallel to each other and to $\mathbf{u}$. The first half of this condition means that the third row of $\mathcal{T}_{UA}$ equals the third row of $\mathcal{T}_{UB}$ scaled by some constant λ. The second half means that $(\mathcal{T}_{UA}\mathbf{u}_U)_z = (\mathcal{T}_{UB}\mathbf{u}_U)_z = 0$, where $(\cdot)_z$ denotes the z-coordinate of a vector. Note that the condition can be met retroactively by using standard rectification methods [SD96c, MD98a]; this is part of "prewarping" the reference views.

Setting $\xi = (\mathcal{T}_{UA}\mathbf{q}_U)_z = \lambda(\mathcal{T}_{UB}(\mathbf{q} + \mathbf{u})_U)_z$, the linear interpolation of the projection of ω into both cameras is

$$(1 - s)\frac{1}{\xi}\mathcal{T}_{UA}\mathbf{q}_U + s\frac{\lambda}{\xi}\mathcal{T}_{UB}(\mathbf{q} + \mathbf{u})_U \tag{5}$$

Now define a virtual camera V by the matrix

$$\mathcal{T}_{UV} = (1 - s)\mathcal{T}_{UA} + s\lambda\mathcal{T}_{UB} \tag{6}$$

Then the linear interpolation (5) is equal to the projection of scene point $\mathbf{q}(s)$ onto the image plane

15

if prewarps make...	then interpolation is...
image planes parallel	physically correct
...*and* conjugate directions equal up to a scalar	...*and* depicts straight–line motion
...*and* the scalar is λ	...*and* the motion is constant–velocity

Figure 9: How the interpolation sequence is related to different preconditions on the reference views. Stricter preconditions lead to increased control over the output.

of camera V, where

$$q(s) = q + u(s) \tag{7}$$

$$u(s)_V = s\lambda u_B \tag{8}$$

Notice that $u(s)$ depends only on u and the camera matrices and not on the starting location q. Thus linear interpolation of conjugate object points by a factor s creates a physically-valid view of the object. The object is seen as it would appear through camera V if it had been translated by $u(s)$ from its starting position.

Note that in Eq. 8, $u(s)$ is represented in basis V. Since V changes with s it is difficult in general to characterize the trajectory in world coordinates. To have greater control over the interpolation process, we now prove that straight-line motion is achieved when $u_A = u_B$ up to an arbitrary scalar, and constant-velocity straight-line motion (i.e., linear motion) is achieved when $u_A = \lambda u_B$ (Figure 9):

Assume $u_A = ku_B$ for some scalar k. Multiplying both sides of Eq. 6 on the right by $\mathcal{T}_{BU}$ yields

$$\mathcal{T}_{BV} = (1 - s)\mathcal{T}_{BA} + s\lambda\mathcal{I} \tag{9}$$

By multiplying both sides of Eq. 9 on the right by u_B and on the left by $\mathcal{T}_{VB}$ the following can be derived:

$$\mathcal{T}_{VB}u_B = \frac{1}{(1 - s)k + s\lambda}u_B \tag{10}$$

Multiplying both sides of Eq. 8 by $\mathcal{T}_{VB}$ now yields:

$$u(s)_B = \frac{s\lambda}{(1 - s)k + s\lambda}u_B \tag{11}$$

16

The basis V no longer plays a role and the virtual trajectory, given by $\mathbf{u}(s)_B$, is a straight-line in basis B. If $k = \lambda$ then $\mathbf{u}(s)_B = s\mathbf{u}_B$ and the virtual object moves at constant velocity. The results are in basis B, but multiplying by $\mathcal{T}_{BU}$ or $\mathcal{T}_{BA}$ indicates that the results also hold in world coordinates and camera A's coordinates, thus completing the proof. Keep in mind that the world coordinate system used in this context has its origin at the shared optical center of the reference cameras.

If $\mathcal{T}_{BA}$ is known then the camera matrix for B can be transformed into the camera matrix for A. This allows the view from camera B at time $t = 1$ to be transformed into the view from camera A at time $t = 1$, thus producing two views of the scene from camera A at different times. For this reason, we call $\mathcal{T}_{BA}$ the *camera-to-camera transformation*. By applying the earlier results to this special case, we derive the following corollary which forms the basis of the algorithm in Section 2.2.3:

If both camera matrices are equal and if $(\mathcal{T}_{UA}\mathbf{u})_z = 0$, then the camera matrix for the virtual camera V is just $\mathcal{T}_{UA}$ and, because $\lambda = 1$ and $\mathbf{u}_A = \mathbf{u}_B$, the virtual object moves at constant velocity along a straight-line path.

2.2.3 Linear Motion Dynamic View Morphing Algorithm

We now present a dynamic view interpolation algorithm that will portray linear motion. The algorithm requires knowledge of $\mathcal{T}_{AB}$.

(Step 1) Segment both views into layers, with each layer representing a different moving object. Order the layers from nearest object to farthest object (Figure 10).

(Step 2) Transform each layer of view B by $\mathcal{T}_{BA}$, thus creating a view from camera A.

(Step 3) Apply static view morphing to each layer separately.

(Step 4) Recombine the new, virtual layers in the correct depth order.

(Step 5) (Optional) Postwarp the new view.

In step (3), the virtual camera will be the same for each layer by the corollary of the previous section. Furthermore, each layer will portray its corresponding object undergoing linear motion. Consequently, step (4) produces the desired linear portrayal of the entire scene.

2.2.4 Special Case: Parallel Motion

In this and the following section we examine some special-case scenarios for which dynamic view interpolations can be produced without knowledge of $\mathcal{T}_{AB}$.

Assume a fixed-camera formulation and let $\mathbf{u}_i$ denote the displacement between the position of

17

original view

layer 0 (background)   layer 1   layer 2
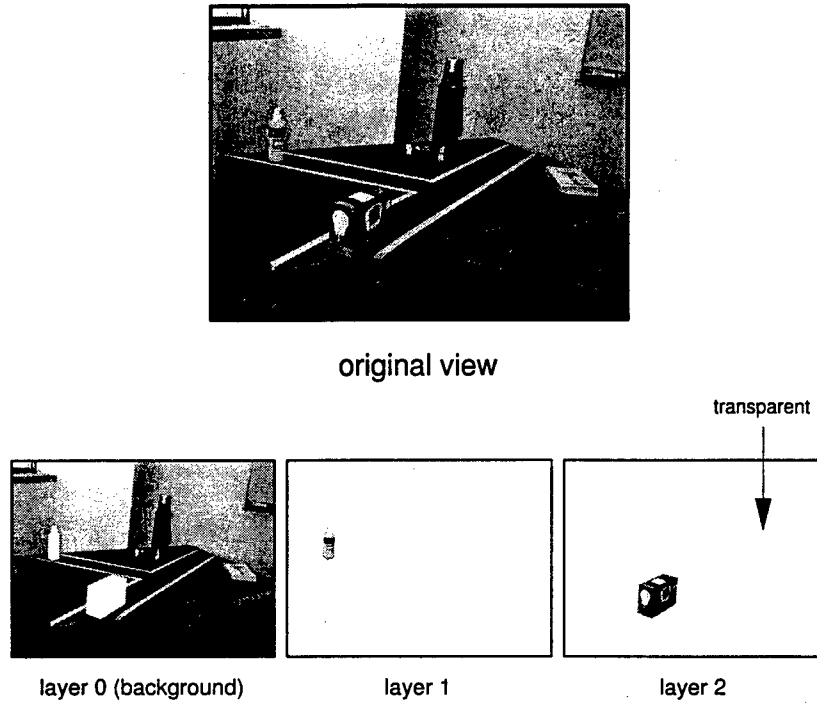
Figure 10: A view divided into layers. Each layer corresponds to a moving object. The single "background" object contains many different objects that all translate by the same amount.

object i at time $t = 0$ and its position at time $t = 1$. We will say the scene consists of *parallel motion* if all the $\mathbf{u}_i$ are parallel in space.

Dynamic view morphing algorithm for parallel motion case: *Segment each view into layers corresponding to objects. Apply static view morphing to each layer and recomposite the results.*

The algorithm works because the fundamental matrix with respect to each object is the same, so the same prewarp works for each layer. The prewarp will make the direction of motion for each object be parallel to the x-axis in both views; consequently, the virtual objects will follow straight-line trajectories as measured in the camera frame. If we assume that the background object has no motion in world coordinates, then the virtual camera moves parallel to the motion of all the objects and hence the virtual object motion is straight-line in world coordinates.

2.2.5 Special Case: Planar Parallel Motion

We now consider the case in which all the $\mathbf{u}_i$ are parallel to some fixed plane in space. Note that this does not mean all the objects are translating in the *same* plane. Also note that this case applies whenever there are two moving objects.

Recall that in Section 2.2.2 the only requirement for the virtual view to be a physically-accurate

portrayal of an object that translates by **u** is that the image planes of both reference views be parallel to **u** and to each other. In the planar parallel motion case, it is possible to prewarp the reference views so that their image planes are parallel to each other and to the displacements of all the objects simultaneously.

Dynamic view morphing algorithm for planar parallel motion case: *Segment each view into layers corresponding to objects. For each reference view, find a single prewarp that sends the z coordinate of the vanishing point of each object to 0. Using this prewarp, apply static view morphing to each layer and recomposite the results.*

The algorithm given above only guarantees physical correctness, not straight-line or linear motion. The *appearance* of straight-line motion can be created by first making the conjugate motion vectors parallel during the prewarp step [MD98a].

2.2.6 Dynamic Scene Hierarchy

This section interrelates the algorithms of the previous three sections. As always, we assume a fixed-camera formulation, meaning we choose to interpret the two reference views as having been captured by cameras that shared the same optical center.

Consider classifying each object in the scene based on the direction of its translation vector, with two objects being placed in the same class if their translation vectors are parallel. A natural hierarchy emerges based on the number of distinct parallel motion classes the scene contains.

First consider scenes that have only one motion class. If the class corresponds to the null direction vector, then the scene is static and view interpolation reduces to mosaicing. If the direction vector is non-null, view interpolations can be produced via the parallel motion algorithm (Section 2.2.4).

When the scene has two motion classes, the planar-parallel motion algorithm applies (Section 2.2.5). With four or more motion classes, $\mathcal{T}_{AB}$ can be determined as described in Section 2.2.8 from the four directions associated with the classes, and the linear motion algorithm applies (Section 2.2.3). For scenes with exactly three motion classes, either the planar-parallel algorithm applies or else $\mathcal{T}_{AB}$ can be approximated after making reasonable assumptions about the reference cameras [MD98a].

2.2.7 Affine Cameras

The mathematical development for affine cameras, which includes orthographic cameras, is similar to that for pinhole cameras. However, except in special cases, no camera-to-camera transformation exists between the reference cameras. Hence it is typically impossible to guarantee linear motion

for the virtual objects. On the other hand, interpolation of conjugate points always produces a physically-valid virtual view, without needing to make the image planes parallel. Prewarps can be applied to align conjugate directions and thus achieve straight-line motion. However, in general it is only possible to align at most three conjugate directions. For a complete discussion, see [MD98a].

2.2.8 Finding $\mathcal{T}_{AB}$

The problem of determining $\mathcal{T}_{AB}$ is central to the linear motion algorithm of Section 2.2.3. $\mathcal{T}_{AB}$ can be determined from four conjugate directions by a well-known result used in mosaicing [Sze96] (because conjugate directions become conjugate points if we treat the reference cameras as being co-centered).

If the fundamental matrix can be determined for two objects in the scene and if the objects are not moving parallel to each other, then $\mathcal{T}_{AB}$ can be determined directly from these two fundamental matrices. The previous fact is proven in [MD98a], which also gives a method for approximating $\mathcal{T}_{AB}$ from two conjugate directions by making a reasonable assumption about the internal parameters of typical cameras.

2.2.9 Applications

Dynamic view morphing has many potential applications; we list a few here: filling a missing gap in a movie, creating a "hand-off" sequence to switch from one camera view to another, creating virtual views of a scene, removing obstructions or moving objects from a sequence, adding synthetic moving objects to real scenes, projecting motion into the future or past, stabilizing and compressing movie sequences, and creating movies from still images.

2.2.10 Experimental Results

We tested our method on a variety of scenarios. Figure 11 shows the results of three tests, each as a series of still frames from a view interpolation sequence. The left-most and right-most frames of each strip are the original reference views, while the center two frames are virtual views created by the algorithm.

To create each sequence, two preprocessing steps were performed manually. First, the two reference views were divided into layers corresponding to the moving objects. Second, for each corresponding layer a set of conjugate points between the two views was determined. Since our implementation uses the Beier-Neely algorithm [BN92] for the morphing step we actually determined a series of line-segment correspondences instead of point correspondences. For each sequence, between 30 and 50 line-segment correspondences were used (counting every layer).
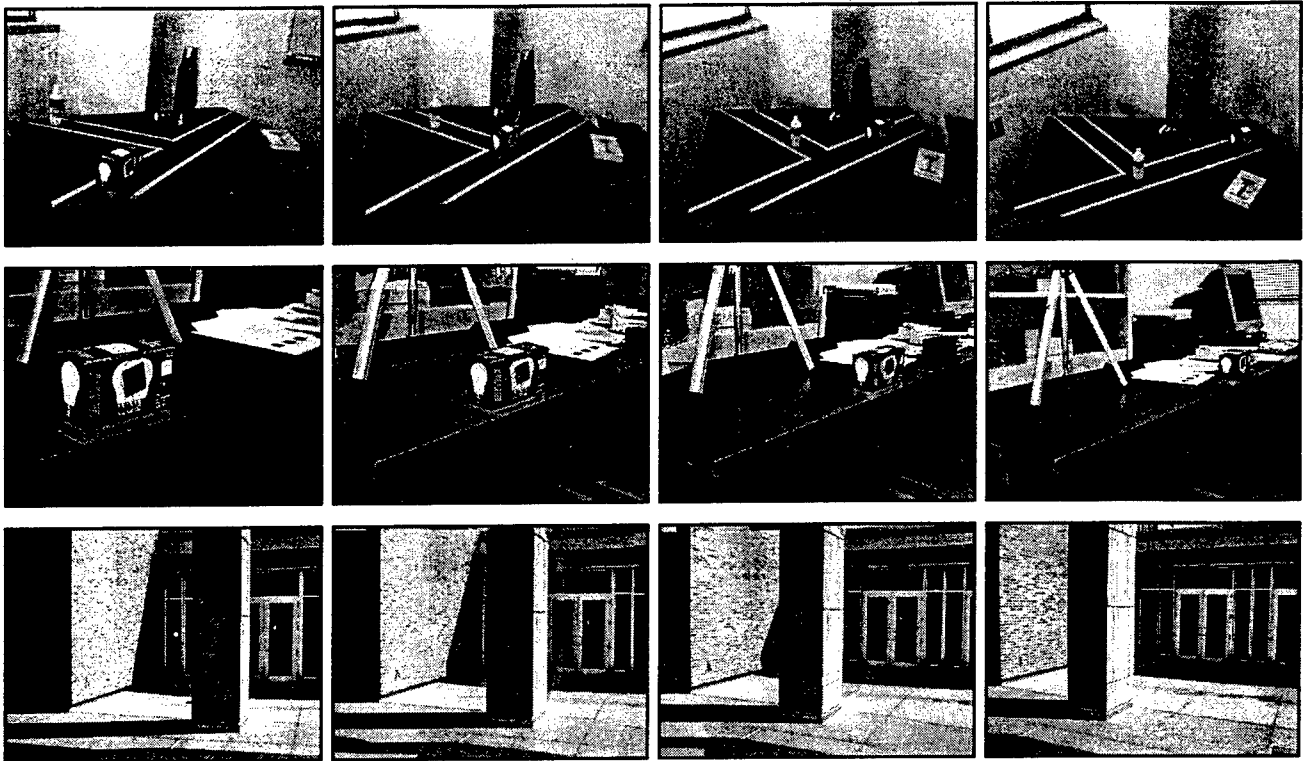
Figure 11: Experimental results.

For all the sequences, the camera calibration was completely unknown, the focal lengths were different, and the cameras were at different locations.

The first sequence is from a test involving three moving objects (counting the background object). Since $\mathcal{T}_{AB}$ could only be approximated, the appearance of straight-line motion was achieved by aligning the conjugate directions of motion for each object during the prewarp step [MD98a]. An object's direction of motion is given by the epipoles of the object's fundamental matrix. Instead of calculating the objects' fundamental matrices, we determined the epipoles directly from the vanishing points of the tape "roads."

The second sequence involves two moving objects (counting the background object) and a dramatic change in focal length. The third sequence demonstrates the parallel motion algorithm (Section 2.2.4). The scene is actually static, but the pillar in the foreground and the remaining background elements are treated as two separate objects that are moving parallel to each other.

2.2.11   Discussion

We have developed a method for interpolating between two views of a dynamic scene. The method requires that, for each object in the scene, the movement that occurs between the first and second

21

views must be equivalent to a rigid translation. The algorithm produces virtual views that portray one version of what might have occurred in the scene. It is only necessary that the image planes of the reference cameras be parallel to each other and to the motion of an object for the interpolated view of the object to be physically correct. With more conditions on the reference cameras, the object can be portrayed moving along a straight-line path and even moving at constant velocity along a straight-line path. Interpolated views of a complete dynamic scene can be created by separately creating interpolated views of the scene's component objects and then combining the results.

By choosing to interpret the views as coming from the same position in space, a single theory has been created which applies to many different possible situations. In particular, the same theory applies whether or not the original reference cameras were actually co-centered. Since it is impossible to know from the reference views themselves how the original reference cameras were positioned relative to each other, the fixed-camera formulation is a natural default assumption. The virtual camera can be chosen to move along *any* trajectory; the choice simply alters the interpretation of the virtual views. The fixed-camera formulation also allows for a simple and intuitive development of the underlying mathematics of the theory.

Finally, it has been shown that each object in a dynamic scene has a corresponding fundamental matrix as long as the assumption of translational motion holds. From two such (distinct) fundamental matrices, the camera-to-camera transformation can be determined.

2.3   Voxel Coloring

View morphing demonstrated the feasibility of view synthesis and provided a robust algorithm for interpolating a pair of images. However, scene visibility is necessarily limited to what appears in only two reference views. Consequently, we devised an algorithm capable of synthesizing *arbitrary* new views of a static scene from a set of reference views that are widely distributed around the environment. Specifically, our objectives were:

- **Photo-integrity**: The synthesized views should accurately reproduce the input images, preserving color, texture and pixel resolution

- **Broad Viewpoint Coverage**: New views should be accurate over a large range of target viewpoints. Therefore, the *reference views* should be widely distributed around the environment

In principle, adding more reference views should improve the fidelity of synthesized views. However, the additional images introduce a whole range of new problems, like occlusion, calibration, correspondence, and representation issues. Whereas the two-image problem has been thoroughly

studied in computer vision, theories of multi-image projective geometry, calibration, and correspondence have only recently begun to emerge [Sha94, Har94, LV94, Tri95, FM95, HA95]. Furthermore, the view synthesis problem as presently formulated raises a number of unique challenges that push the limits of existing multi-image techniques.

In this section, we describe an approach for view synthesis from multiple basis views that seeks to bypass the limitations of the two view approach, e.g., limited scene visibility, while retaining many of the theoretical and practical advantages of the view morphing algorithm, e.g., uniqueness properties and performance. Instead of approaching this problem as one of shape reconstruction, we formulate a *color reconstruction* problem, in which the goal is an assignment of colors (radiances) to points in an (unknown) approximately Lambertian scene. As a solution, we present a *voxel coloring* technique that traverses a discretized 3D space in "depth-order" to identify voxels that have a unique coloring, constant across all possible interpretations of the scene. This approach has several advantages over existing stereo and structure-from-motion approaches to pixel correspondence and scene reconstruction. First, occlusions are explicitly modeled and accounted for. Second, the cameras can be positioned far apart without degrading accuracy or run-time. Third, the technique integrates numerous images to yield dense reconstructions that are accurate over a wide range of target viewpoints.

Our publications on voxel coloring give more complete details of our work [SD97a, SD97b, SK98, Sei97b, SD00].

The remainder of this section describes the voxel coloring problem in detail. The main results require a visibility property that constrains the camera placement relative to the scene, but still permits the input cameras to be spread widely throughout the scene. The visibility property defines a fixed occlusion ordering, enabling scene reconstruction with a single pass through the voxels in the scene.

We assume that the scene is entirely composed of rigid Lambertian surfaces under fixed illumination. Under these conditions, the radiance at each point is isotropic and can therefore be described by a scalar value which we call *color*. We also use the term color to refer to the irradiance of an image pixel. The term's meaning should be clear by context.

2.3.1 Notation

A 3D scene S is represented as a finite[2] set of opaque voxels (volume elements), each of which occupies a finite and homogeneous scene volume and has a fixed color. We denote the set of all voxels with the symbol V. An image is specified by the set I of all its pixels. For now, assume that pixels are infinitesimally small.

---

[2]It is assumed that the visible scene is spatially bounded.
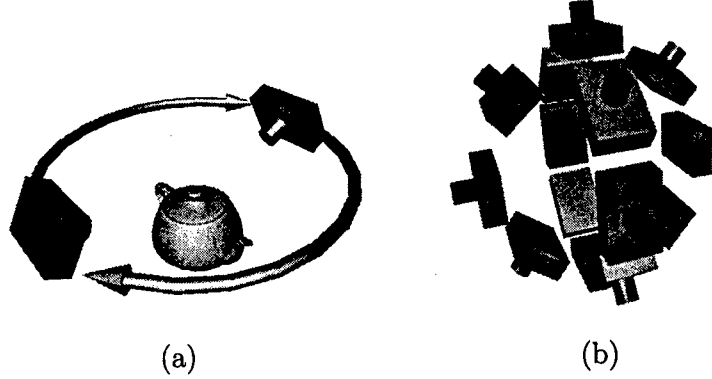
Figure 12: Two camera geometries that satisfy the ordinal visibility constraint.
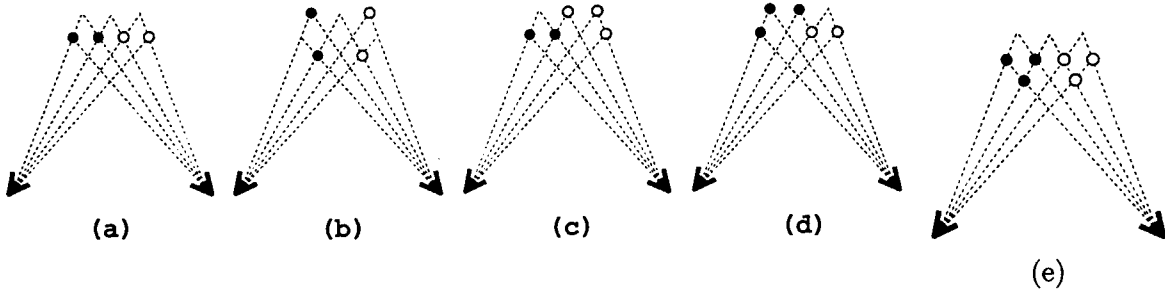
Figure 13: (a-d) Four scenes that are indistinguishable from these two viewpoints. Shape ambiguity: scenes (a) and (b) have no points in common—no hard points exist. Color ambiguity: (c) and (d) share a point that has a different color assignment in the two scenes. (e) The voxel coloring produced from the two images in (a-d). These six points have the same color in every consistent scene that contains them.

Given an image pixel p and scene $\mathcal{S}$, we refer to the voxel $V \in \mathcal{S}$ that is visible and projects to p by $V = \mathcal{S}(p)$. The color of an image pixel $p \in \mathcal{I}$ is given by $color(p, \mathcal{I})$ and of a voxel V by $color(V, \mathcal{S})$. A scene $\mathcal{S}$ is said to be *complete* with respect to a set of images if, for every image $\mathcal{I}$ and every pixel $p \in \mathcal{I}$, there exists a voxel $V \in \mathcal{S}$ such that $V = \mathcal{S}(p)$. A complete scene is said to be *consistent* with a set of images if, for every image $\mathcal{I}$ and every pixel $p \in \mathcal{I}$,

$$color(p, \mathcal{I}) = color(\mathcal{S}(p), \mathcal{S}) \tag{12}$$

2.3.2 Camera Geometry

A pinhole perspective projection model is assumed, although the main results use a visibility assumption that applies equally to other camera models such as orthographic and aperture-based models. We require that the viewpoints (camera positions) are distributed so that ordinal visibility relations between scene points are preserved. That is, if scene point P occludes Q in one image,

24

Q cannot occlude P in any other image. This is accomplished by ensuring that all viewpoints are "on the same side" of the object. For instance, suppose the viewpoints are distributed on a single plane, as shown in Figure 12(a). For every such viewpoint, the relative visibility of any two points depends entirely on which point is closer to the plane. Because the visibility order is fixed for every viewpoint, we say that this range of viewpoints preserves ordinal visibility.

Planarity, however, is not required; the ordinal visibility constraint is satisfied for a relatively wide range of viewpoints, allowing significant flexibility in the image acquisition process. Observe that the constraint is violated only when there exist two scene points P and Q such that P occludes Q in one view while Q occludes P in another. This condition implies that P and Q lie on the line segment between the two camera centers. Therefore, a sufficient condition for the ordinal visibility constraint to be satisfied is that **no scene point be contained within the convex hull C of the camera centers**. For convenience, C will be referred to as the *camera volume*. We use the notation $dist(V, C)$ to denote the distance of a voxel V to the camera volume. Figure 12 shows two useful camera geometries that satisfy this constraint, one a downward facing camera moved 360 degrees around an object, and the other outward facing cameras on a sphere.

2.3.3 Color Invariance

It is well known that a set of images can be consistent with more than one rigid scene. Determining a scene's spatial occupancy is therefore an ill-posed task because a voxel contained in one consistent scene may not be contained in another (Figure 13(a,b)). Alternatively, a voxel may be part of two consistent scenes, but have different colors in each (Figure 13(c,d)).

Given a multiplicity of solutions to the reconstruction problem, the only way to recover intrinsic scene information is through *invariants*— properties that are satisfied by every consistent scene. For instance, consider the set of voxels that are present in every consistent scene. Laurentini [Lau95] described how these invariants, called *hard points*, could be recovered by volume intersection from binary images. Hard points are useful in that they provide absolute information about the true scene. However, such points can be difficult to come by; some images may yield none (e.g., Figure 13). In this section we describe a more frequently occurring type of invariant relating to color rather than shape.

> A voxel V is a **color invariant** with respect to a set of images if, for every pair of scenes S and S' consistent with the images, $V \in S, S'$ implies $color(V, S) = color(V, S')$

Unlike shape invariance, color invariance does not require that a point be present in every consistent scene. As a result, color invariants tend to be more common than hard points. In particular, any set of images satisfying the ordinal visibility constraint yields enough color invariants to form a complete scene reconstruction, as will be shown.

Let $\mathcal{I}_1, \ldots, \mathcal{I}_m$ be a set of images. For a given image point $p \in \mathcal{I}_j$ define V_p to be the voxel in $\{\mathcal{S}(p) \mid \mathcal{S}\ consistent\}$ that is closest to the camera volume. We claim that V_p is a color invariant. To establish this, observe that $V_p \in \mathcal{S}$ implies $V_p = \mathcal{S}(p)$, for if $V_p \neq \mathcal{S}(p)$, $\mathcal{S}(p)$ must be closer to the camera volume, which is impossible by the construction of V_p. It then follows from Eq. (12) that V_p has the same color in every consistent scene; V_p is a color invariant.

The **voxel coloring** of an image set $\mathcal{I}_1, \ldots, \mathcal{I}_m$ is defined to be:
$$\overline{\mathcal{S}} \; = \; \{V_p \mid p \in \mathcal{I}_i, \; 1 \leq i \leq m\}$$

Figure 13(e) shows the voxel coloring resulting from a pair of views. These six points have a unique color interpretation, constant in every consistent scene. They also comprise the closest consistent scene to the cameras in the following sense—every point in each consistent scene is either included in the voxel coloring or is fully occluded by points in the voxel coloring. An interesting consequence of this closeness bias is that neighboring image pixels of the same color produce cusps in the voxel coloring, i.e., protrusions toward the camera volume. This phenomenon is clearly shown in Figure 13(e) where the white and black points form two separate cusps. Also, observe that the voxel coloring is not a minimal reconstruction; removing the two closest points in Figure 13(e) still leaves a consistent scene.

2.3.4 Computing the Voxel Coloring

In this section we describe how to compute the voxel coloring from a set of images. In addition it will be shown that the set of voxels contained in a voxel coloring form a scene reconstruction that is consistent with the input images.

The voxel coloring is computed one voxel at a time in an order that ensures agreement with the images at each step, guaranteeing that all reconstructed voxels satisfy Eq. (12). To demonstrate that voxel colorings form consistent scenes, we also have to show that they are complete, i.e., they account for every image pixel as defined in Section 2.3.1.

In order to make sure that the construction is incrementally consistent, i.e., agrees with the images at each step, we need to introduce a weaker form of consistency that applies to incomplete voxel sets. Accordingly, we say that a set of points with color assignments is *voxel-consistent* if its projection agrees fully with the subset of every input image that it overlaps. More formally, a set $\mathcal{S}$ is said to be voxel-consistent with images $\mathcal{I}_1, \ldots, \mathcal{I}_m$ if for every voxel $V \in \mathcal{S}$ and image pixels $p \in \mathcal{I}_i$ and $q \in \mathcal{I}_j$, $V = \mathcal{S}(p) = \mathcal{S}(q)$ implies $color(p, \mathcal{I}_i) = color(q, \mathcal{I}_j)$. For notational convenience, define $\mathcal{S}_V$ to be the set of all voxels in $\mathcal{S}$ that are closer than V to the camera volume. Scene consistency and voxel consistency are related by the following properties:

1. If $\mathcal{S}$ is a consistent scene then $\{V\} \cup \mathcal{S}_V$ is a voxel-consistent set for every $V \in \mathcal{S}$.

26

2. Suppose $\mathcal{S}$ is complete and, for each point $V \in \mathcal{S}$, $V \cup \mathcal{S}_V$ is voxel-consistent. Then $\mathcal{S}$ is a consistent scene.

A consistent scene may be created using the second property by incrementally moving further from the camera volume and adding voxels to the current set that maintain voxel-consistency. To formalize this idea, we define the following partition of 3D space into voxel layers of uniform distance from the camera volume:

$$\mathcal{V}_{\mathcal{C}}^d = \{V \mid dist(V, \mathcal{C}) = d\} \tag{13}$$

$$\mathcal{V} = \bigcup_{i=1}^{r} \mathcal{V}_{\mathcal{C}}^{d_i} \tag{14}$$

where $d_1, \ldots, d_r$ is an increasing sequence of numbers.

The voxel coloring is computed inductively as follows:

$$SP_1 = \{V \mid V \in \mathcal{V}_{d_1}, \{V\} \text{ voxel-consistent}\}$$
$$SP_k = \{V \mid V \in \mathcal{V}_{d_k},$$
$$\{V\} \cup SP_{k-1} \text{ voxel-consistent}\}$$
$$SP = \{V \mid V = SP_r(p) \text{ for some pixel } p\}$$

We claim $SP = \overline{\mathcal{S}}$. To prove this, first define $\overline{\mathcal{S}}_i = \{V \mid V \in \overline{\mathcal{S}}, dist(V, \mathcal{C}) \le d_i\}$. $\overline{\mathcal{S}}_1 \subseteq SP_1$ by the first consistency property. Inductively, assume that $\overline{\mathcal{S}}_{k-1} \subseteq SP_{k-1}$ and let $V \in \overline{\mathcal{S}}_k$. By the first consistency property, $\{V\} \cup \overline{\mathcal{S}}_{k-1}$ is voxel-consistent, implying that $\{V\} \cup SP_{k-1}$ is also voxel-consistent, because the second set includes the first and SP_{k-1} is itself voxel-consistent. It follows that $\overline{\mathcal{S}} \subseteq SP_r$. Note also that SP_r is complete, since one of its subsets is complete, and hence consistent by the second consistency property. SP contains all the voxels in SP_r that are visible in any image, and is therefore consistent as well. Therefore SP is a consistent scene such that for each pixel p, $SP(p)$ is at least as close to $\mathcal{C}$ as $\overline{\mathcal{S}}(p)$. Hence $SP = \overline{\mathcal{S}}$. Q.E.D.

In summary, the following properties of voxel colorings have been shown:

- $\overline{\mathcal{S}}$ is a consistent scene

- Every voxel in $\overline{\mathcal{S}}$ is a color invariant

- $\overline{\mathcal{S}}$ is directly computable from any set of images satisfying the ordinal visibility constraint

2.3.5 Reconstruction by Voxel Coloring

In this section we present a voxel coloring algorithm for reconstructing a scene from a set of calibrated images. The algorithm closely follows the voxel coloring construction outlined earlier,

27

adapted to account for image quantization and noise. As before, it is assumed that 3D space has been partitioned into a series of voxel layers $\mathcal{V}_C^{d_1}, \ldots, \mathcal{V}_C^{d_r}$ increasing in distance from the camera volume. The images $\mathcal{I}_1, \ldots, \mathcal{I}_m$ are assumed to be quantized into finite non-overlapping pixels. The cameras are assumed to satisfy the ordinal visibility constraint, i.e., no scene point lies within the camera volume.

If a voxel V is not fully occluded in image $\mathcal{I}_j$, its projection will overlap a nonempty set of image pixels, π_j. Without noise or quantization effects, a consistent voxel should project to a set of pixels with equal color values. In the presence of these effects, we evaluate the correlation of the pixel colors to measure the likelihood of voxel consistency. Let s be the standard deviation and n the cardinality of $\bigcup_{j=1}^{m} \pi_j$. Suppose the sensor error (accuracy of irradiance measurement) is approximately normally distributed with standard deviation σ_0. If σ_0 is unknown, it can be estimated by imaging a homogeneous surface and computing the standard deviation of image pixels. The consistency of a voxel can be estimated using the following likelihood ratio test, distributed as χ^2:

$$\lambda_V = \frac{(n-1)s}{\sigma_0}$$

2.3.6 Voxel Coloring Algorithm

The algorithm is as follows:

```
S = ∅

for i = 1,...,r do

    for every V ∈ V_C^{d_i} do

        project to I_1,...,I_m, compute λ_V

        if λ_V < thresh then S = S ∪ {V}
```

The threshold, *thresh*, corresponds to the maximum allowable correlation error. An overly conservative (small) value of *thresh* results in an accurate but incomplete reconstruction. On the other hand, a large threshold yields a more complete reconstruction, but one that includes some erroneous voxels. In practice, *thresh* should be chosen according to the desired characteristics of the reconstructed model, in terms of accuracy vs. completeness.

The problem of detecting occlusions is greatly simplified by the scene traversal ordering used in the algorithm; the order is such that if V occludes V' then V is visited before V'. Therefore, occlusions can be detected by using a one-bit Z-buffer for each image. The Z-buffer is initialized to 0. When a voxel V is processed, π_i is the set of pixels that overlap V's projection in $\mathcal{I}_i$ and have

(a)·   (b)   (c)

Figure 14: Reconstruction of a dinosaur toy. (a) One of 21 input images taken from slightly above the toy while it was rotated 360°. (b-c) Two views rendered from the reconstruction.

Z-buffer values of 0. Once λ_V is calculated, these pixels are then marked with Z-buffer values of 1.

The algorithm visits each voxel exactly once and projects it into every image. Therefore, the time complexity of voxel coloring is: $O(voxels * images)$. To determine the space complexity, observe that evaluating one voxel does not require access to or comparison with other voxels. Consequently, voxels need not be stored during the algorithm; the voxels making up the voxel coloring will simply be output one at a time. Only the images and one-bit Z-buffers need to be stored. The fact that the complexity of voxel coloring is linear in the number of images is essential in that it enables large sets of images to be processed at once.

The algorithm is unusual in that it does not perform any window-based image matching in the reconstruction process. Correspondences are found implicitly during the course of scene traversal. A disadvantage of this searchless strategy is that it requires very precise camera calibration to achieve the triangulation accuracy of existing stereo methods. Accuracy also depends on the voxel resolution.

Importantly, the approach reconstructs only one of the potentially numerous scenes consistent with the input images. Consequently, it is susceptible to aperture problems caused by image regions of near-uniform color. These regions will produce cusps in the reconstruction (see Figure 13(e)) since voxel coloring seeks the reconstruction closest to the camera volume. This is a bias, just like smoothness is a bias in stereo methods, but one that guarantees a consistent reconstruction even with severe occlusions.
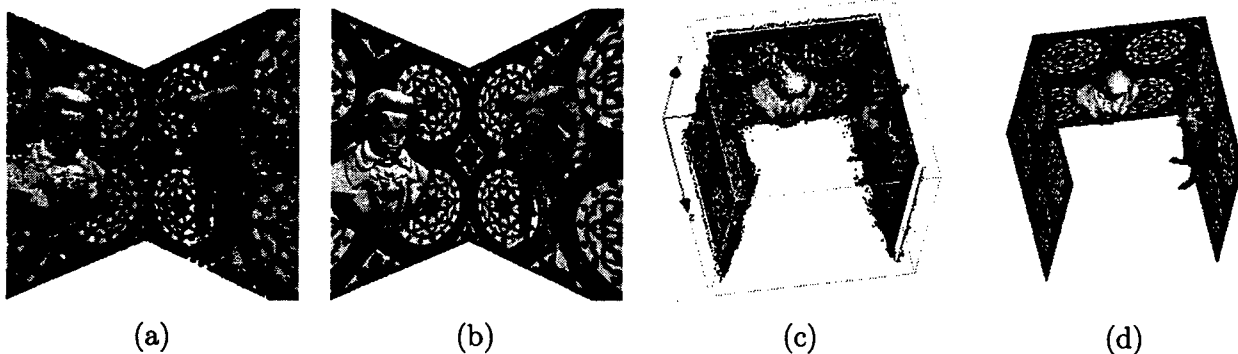
29

(a)   (b)   (c)   (d)

Figure 15: Reconstruction of a synthetic room scene. (a) The voxel coloring. (b) The original model from a new viewpoint. (c) and (d) show the reconstruction and original model, respectively, from a new viewpoint outside the room.

2.3.7   Experimental Results

The first experiment involved reconstructing a dinosaur toy from 21 views spanning a 360 degree rotation of the toy. Figure 14 shows the voxel coloring computed. To facilitate reconstruction, we used a black background and eliminated most of the background points by thresholding the images. While background subtraction is not strictly necessary, leaving this step out results in background-colored voxels scattered around the edges of the scene volume. The threshold may be chosen conservatively since removing most of the background pixels is sufficient to eliminate this background scattering effect. Figure 14(b) shows the reconstruction from approximately the same viewpoint as (a) to demonstrate the photo integrity of the reconstruction. Figure 14(c) shows another view of the reconstructed model. Note that fine details such as the wind-up rod and hand shape were accurately reconstructed. The reconstruction contained 32,244 voxels and took 45 seconds to compute.

A second experiment involved reconstructing a synthetic room from views *inside* the room. The room interior was highly concave, making accurate reconstruction by volume intersection or other contour-based methods impractical. Figure 15 compares the original and reconstructed models from new viewpoints. New views were generated from the room interior quite accurately, as shown in (a), although some details were lost. For instance, the reconstructed walls were not perfectly planar. This point drift effect is most noticeable in regions where the texture is locally homogeneous, indicating that texture information is important for accurate reconstruction. The reconstruction contained 52,670 voxels and took 95 seconds to compute.

Another set of experiments was conducted to evaluate the sensitivity of the approach to factors of texture density, image noise, and voxel resolution. To simplify the analysis of these effects, the experiments were performed using a 2D implementation of the voxel coloring method for which the

30

scene and cameras lie in a common plane. Figure 16(a) shows the synthetic scene (an arc) and the positions of the basis views used in these experiments.

Texture is an important visual cue, and one that is exploited by voxel coloring. To model the influence of texture on reconstruction accuracy, a series of reconstructions were generated in which the texture was systematically varied. The spatial structure of the scene was held fixed. The texture pattern was a cyclic linear gradient, specified as a function of frequency θ and position $t \in [0, 1]$:

$$intensity(t) = 1 - |1 - 2 * frac(\theta * t)|$$

$frac(x)$ returns the fractional portion of x. Increasing the frequency parameter θ causes the density of the texture to increase accordingly. Figure 16(b-j) show the reconstructions obtained by applying voxel coloring for increasing values of θ. For comparison, the corresponding texture patterns and the original arc shapes are also shown. In (b), the frequency is so low that the quantized texture pattern is uniform. Consequently, the problem reduces to reconstruction from silhouettes and the result is similar to what would be obtained by volume intersection [MA91, Sze93, Lau95]. Specifically, volume intersection would yield a closed diamond-shaped region; the reconstructed V-shaped cusp surface in (b) corresponds to the set of surfaces of this diamond that are visible from the basis views.

Doubling θ results in a slightly better reconstruction consisting of two cusps, as shown in (c). Observe that the reconstruction is accurate at the midpoint of the arc, where a texture discontinuity occurs. Progressively doubling θ produces a series of more accurate reconstructions (d-h) with smaller and smaller cusps that approach the true shape. When θ exceeds a certain point, however, the reconstruction degrades. This phenomenon, visible in (i) and (j), results when the projected texture pattern exceeds the resolution of the basis images, i.e., when the Nyquist rate is exceeded. After this point, accuracy degrades and the reconstruction ultimately breaks up.

Figure 16 illustrates the following two points: (1) reconstruction accuracy is strongly dependent upon surface texture, and (2) the errors are highly *structured*. To elaborate on the second point, reconstructed voxels drift from the true surface in a predictable manner as a function of local texture density. When the texture is locally homogeneous, voxels drift *toward* the camera volume. As texture density increases, voxels move monotonically away from the camera volume, toward the true surface. As texture density increases even further, beyond the limits of image resolution, voxels continue to move away from the cameras, and away from the true surface as well, until they ultimately disappear.

We next tested the performance of the algorithm with respect to additive image noise. To simulate noise in the images, we perturbed the intensity of each image pixel independently by adding a random value in the range of $[-\sigma, \sigma]$. To compensate, the error threshold was set to σ. Figure 17 shows the resulting reconstructions. The primary effect of the error and corresponding
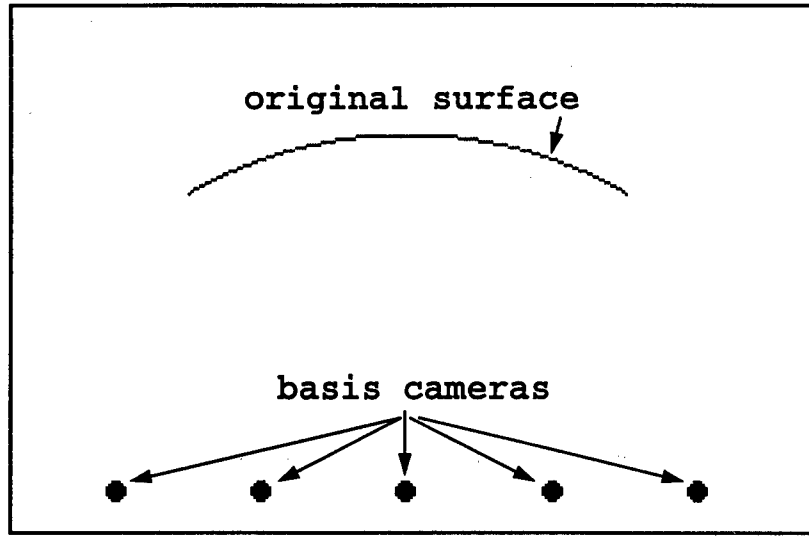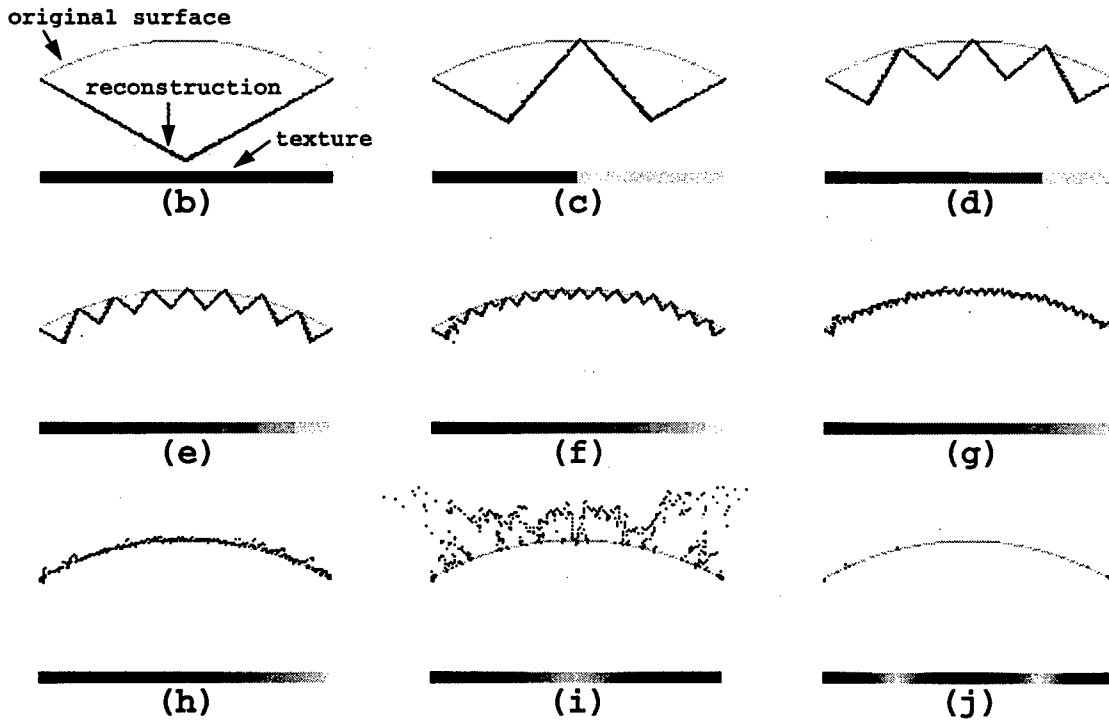
Figure 16: Effects of Texture Density on Voxel Reconstruction. (a): A synthetic arc is reconstructed from five basis views. The arc is textured with a cyclic gradient pattern with a given frequency. Increasing the frequency makes the texture denser and causes the accuracy of the reconstruction to improve, up to a limit. In the case of (b), the texture is uniform so the problem reduces to reconstruction from silhouettes. As the frequency progressively doubles (c-j), the reconstruction converges to the true shape, until a certain point beyond which it exceeds the image resolution (i-j).

32

increase in the threshold was a gradual drift of voxels away from the true surface and toward the cameras. When the error became exceedingly large, the reconstruction ultimately degenerated to the "no texture" solution shown in Figure 16(b). This experiment indicates that image noise, when compensated for by increasing the error threshold, also leads to structured reconstruction errors; higher levels of noise cause voxels to drift progressively closer to the cameras.

The final experiment evaluated the effects of increasing the voxel size on reconstruction accuracy. In principle, the voxel coloring algorithm is only correct in the limit, as voxels become infinitesimally small. In particular, the layering strategy is based on the assumption that points within a layer do not occlude each other. For very small voxels this no-occlusion model is accurate, up to a reasonable approximation. However, as voxels increase in size, the model becomes progressively less accurate.

To more carefully observe the effects of voxel size, we ran the voxel coloring algorithm on the scene in Figure 16(a) for a sequence of increasing voxel sizes. Figure 17 shows the results—the reconstructions are close to optimal, up to the limits of voxel resolution, independent of voxel size. Again, this empirical result is surprising, given the obvious violation of the layering property which is the basis of the algorithm. Some effects of this violation are apparent; some voxels are included in the reconstruction that are clearly invisible, i.e., totally occluded by other voxels from the basis views. For instance, observe that in the reconstruction for voxel size = 10, the top-left and top-right voxels could be deleted without affecting scene appearance from the basis views. These extra voxels are artifacts of the large voxel size and the violation of the layering property. However, these effects are minor and do not adversely affect view synthesis in that adding these voxels does not change scene appearance for viewpoints close to the input images.

2.3.8 Discussion

We have developed a new scene reconstruction technique that incorporates intrinsic color and texture information for the acquisition of photorealistic scene models. Unlike existing stereo and structure-from-motion techniques, the method *guarantees* that a consistent reconstruction is found, even under severe visibility changes, subject to a weak constraint on the camera geometry. A second contribution was the constructive proof of the existence of a set of color invariants. These points are useful in two ways: first, they provide information that is intrinsic, i.e., constant across all possible consistent scenes. Second, together they constitute a volumetric spatial reconstruction of the scene whose projections exactly match the input images.

Our seminal work in this area has lead to a number of other researchers working on this approach. Recent papers include [FK98a, FK98b, KS99, CM99, SK99, CZ00].

noise: $\sigma = 0$ voxel size = 1

noise: $\sigma = 1$ voxel size = 2

noise: $\sigma = 2$ voxel size = 3

noise: $\sigma = 3$ voxel size = 4

noise: $\sigma = 5$ voxel size = 5

noise: $\sigma = 10$ voxel size = 10

noise: $\sigma = 15$ voxel size = 20
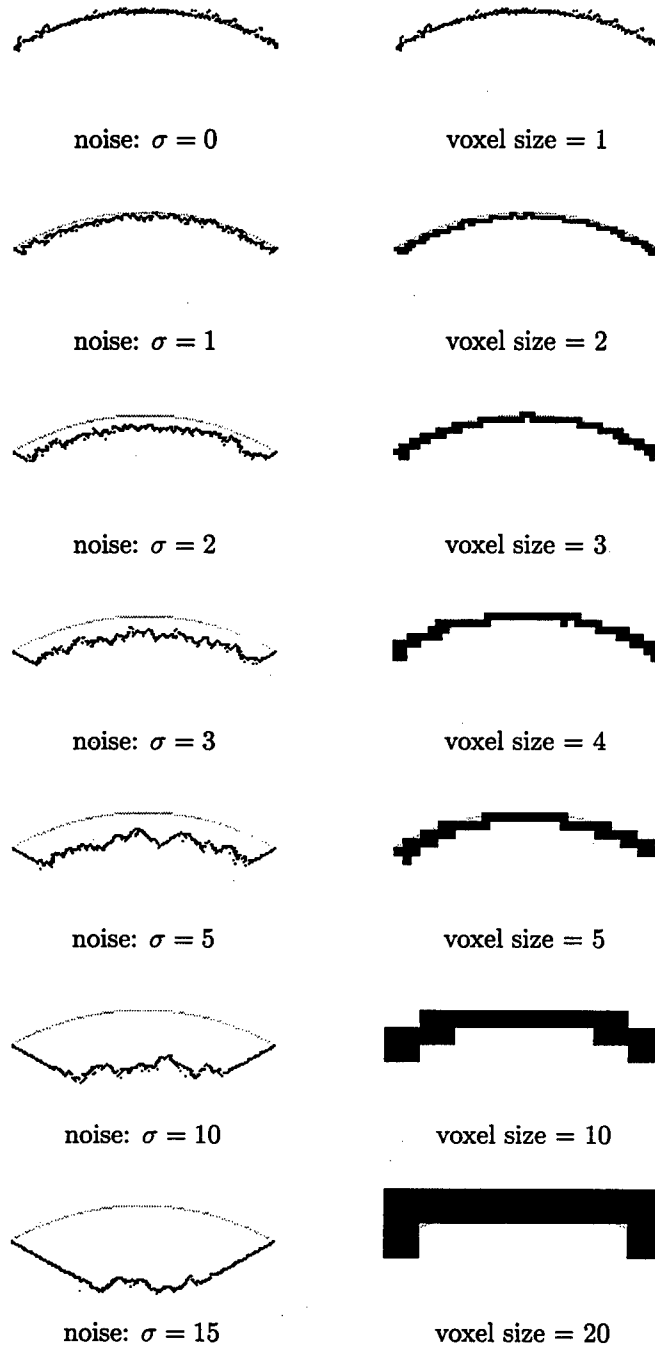
Figure 17: Effects of Image Noise and Voxel Size on Reconstruction. Image noise was simulated by perturbing each pixel by a random value in the range $[-\sigma, \sigma]$. Reconstructions for increasing values of σ are shown at left. To ensure a full reconstruction, the error threshold was also set to σ. Increasing noise caused the voxels to drift from the true surface (shown as light gray). The effects of changing voxel size are shown at right. Notice that the arc shape is reasonably well approximated even for very large voxels.

2.4 Real-Time Voxel Coloring

The straightforward implementation of voxel coloring takes tens of seconds to tens of minutes to reconstruct a scene depending on the spatial resolution being modeled. In this section we describe three methods for speeding up the voxel coloring process. First, texture mapping is used to project the input images onto the voxel layers so as to use hardware acceleration. Second, a coarse-to-fine approach is used. Since voxel coloring produces a 2D surface approximation with voxels, nearly all the space, in most scenes, is empty in the sense that voxel coloring will not color it. The coarse-to-fine approach allows computation time to be focused on the regions in the scene that represent surfaces, thus reducing the overall computation time dramatically. Third, assuming temporal coherence – that is, the scene at successive points in time is similar – we can use the previous time's coloring as input for the next time's coloring. The remainder of the section describes these three approaches in more detail. For further information, see [PD98].

2.4.1 Texture Mapping

The projection of millions of voxels into images is computationally expensive. In particular, for a given layer every voxel must be projected into each image. If we approximate the layer with a plane, this process corresponds to mapping the plane of voxels onto the images, or, inversely, projecting the images onto the plane of voxels. This projection can be implemented on conventional graphics workstations using hardware texture mapping.

For texture mapping to work correctly, the input images must be prewarped so that the transformation from world coordinates to image coordinates can be modeled by a pinhole camera. Also, the overall structure of the algorithm must be modified to change the focus from a per-voxel computation to a per-image computation. Because the images are projected onto the voxel plane, the occlusion information must be stored in the images. This requires updating image pixels on a per-layer basis. The pseudo-code below presents the main loop of the algorithm. Experimental results are summarized in Section 4.

```
prewarp all images
foreach voxel layer i do
    foreach image j do
        texture map layer i with image j
        foreach voxel v in layer i do
            record color value of v
        od
    od
```

35

```
foreach voxel v in layer i do
     if v's colors are correlated
          color voxel v
          update image pixels to
               reflect occlusions
  od
od
```

2.4.2 Coarse-to-Fine Coloring

Coarse-to-fine methods allow processing to be focused on important regions by using relatively low resolution information as input when creating higher resolution solutions. The application of coarse-to-fine methods to voxel coloring allows most of the computation to concentrate on locations in the scene that contain colored voxels. A similar octree strategy was applied in [Sze93], however that method is not suitable here.

2.4.2.1 Naive Approach

2.4.2.1 Naive Approach The main work in voxel coloring is determining which voxels should be colored. The set of colored voxels represent an approximation of the surfaces in the scene. A voxel that contains a small patch of a surface projects to a superset of the pixels which correspond to the actual patch. When the surface only intersects a small fraction of the voxel that is being colored, most pixels that the voxel projects to will not represent the surface. At coarse resolutions this can cause voxels that contain surface patches to go undetected. Voxels that remain uncolored at lower resolutions, but actually contain small opaque sub-regions, should not be eliminated from consideration at higher resolutions. If these regions are lost at a low resolution pass, the resulting coloring will contain noticeable gaps.

Figure 18 shows an example of this problem. The left coloring, (a), was generated by coloring a scene at low resolution. Then, only the colored voxels were then subdivided and subsequently recolored. This process was repeated until the final resolution was reached. Over 13% of the voxels are missing when compared to the correct coloring in (b). This lower density of voxels shows up as gaps in the voxel surface. The gaps and missing voxels are illustrated in (c).

2.4.2.2 Missing Voxels

2.4.2.2 Missing Voxels As resolution is reduced it becomes more and more likely that voxels with smaller occupied sub-volumes will be missed as false negatives. To understand why, consider a simplified version of voxel coloring.

Here voxels are assumed to be spheres, and the sub-volumes that represent occupied space are also spheres wholly contained in the voxel. Let r be the radius of a particular voxel and let r' be the radius of the sub-volume which represents filled space. Let I represent the set of input images.
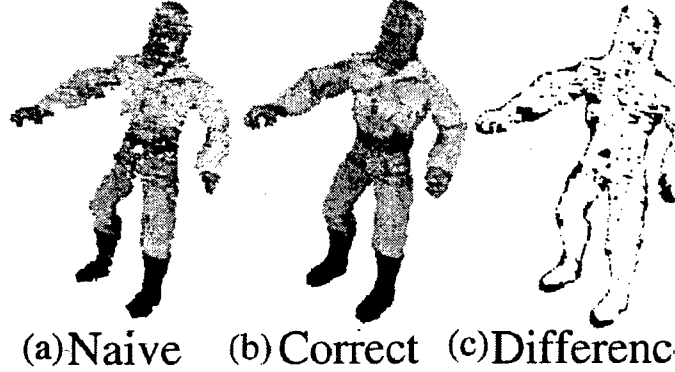
(a)Naive   (b)Correct   (c)Differenc

Figure 18: (a) Naive subdividing of colored voxels. (b) Correct coloring. (c) Projected difference of the two voxel colorings.

To determine whether a voxel should be colored, the pixels to which the voxel projects are analyzed. Let P be the set of pixels in I to which a voxel projects, $c(p)$ be the color of pixel $p \in P$, and $\hat{c}$ be the mean color of the voxel over all pixels in P. We can express the occupation likelihood test, λ, as the average 1-norm from the mean pixel color over P as

$$\lambda = \frac{1}{|P|} \sum_{p \in P} |c(p) - \hat{c}|$$

Now if we denote the set of pixels that correspond to solid space as P', and the set of pixels that correspond to empty space as $\bar{P}$, we can write the summation above as

$$\sum_{p \in P} |c(p) - \hat{c}| = \sum_{p' \in P'} |c(p') - \hat{c}| + \sum_{\bar{p} \in \bar{P}} |c(\bar{p}) - \hat{c}|$$

Thus we can write λ as

$$\lambda = \frac{|P'|}{|P|} \left(\frac{1}{|P'|} \sum_{p' \in P'} |c(p') - \hat{c}| \right) +$$

$$\frac{|\bar{P}|}{|P|} \left(\frac{1}{|\bar{P}|} \sum_{\bar{p} \in \bar{P}} |c(\bar{p}) - \hat{c}| \right)$$

$$= \frac{|P'|}{|P|} \lambda_{v'} + \frac{|\bar{P}|}{|P|} \lambda_{\bar{v}}$$

37

where $\lambda_{v'}$ is the quantity in the first set of parentheses above and $\lambda_{\bar{v}}$ is the second quantity. These can be thought of as the occupation likelihoods for the opaque and non-opaque sub-regions, respectively. For the continuous case the ratio $\frac{|P'|}{|P|}$ is equal to the ratio of the projected area of the solid sub-volumes with respect to the volume. This quantity is simply $\delta^2 = \frac{\pi r'}{\pi r}$. Thus we can approximate the occupation likelihood for the discrete case as

$$\lambda = \delta^2 \lambda_{v'} + (1 - \delta^2)\lambda_{\bar{v}}$$

The quantities $\lambda, \lambda_{v'}$, and $\lambda_{\bar{v}}$ express the color stability of a set of voxels. The occupation likelihood is simply a convex combination of the stability of the two sub-volumes that correspond to solid and empty space.

Consider a solid volume and a background for which both $\lambda_{v'}$ and $\lambda_{\bar{v}}$ are fixed. Then λ is simply a function of δ^2. If $\delta = 1$, the entire voxel is solid, and we have $\lambda = \lambda_{v'}$, as expected. But if $\delta = \frac{1}{2}$, then $\lambda = \frac{1}{4}\lambda_{v'} + \frac{3}{4}\lambda_{\bar{v}}$. By halving the resolution the occupation likelihood becomes dominated by the background. A voxel that would be detected at a given resolution would most likely go undetected if it was the only solid octant of a super voxel.

As a result of the above argument, the naive multi-resolution approach misses a significant number of voxels. To compensate for these missing voxels, some kind of search strategy must be implemented that finds the missing voxels.

2.4.2.3 Searching for False Negatives

Without knowing which of the voxels have been mistakenly left uncolored, the best we can do is to use some kind of heuristic to locate those voxels. We can take advantage of the spatial coherence of the surfaces we are trying to extract by considering only the neighborhood around previously colored voxels.

The search strategy we chose was a nearest neighbor search. All voxels within some 1-norm neighborhood of the original low-resolution set were added to the set. These voxels were then subdivided into octants. This new set of voxels was then traversed in the standard layered order and colored according to the original algorithm. More specifically, with a neighborhood size of one in a two-dimensional scene, all four nearest neighbors are added at the current resolution. Then these voxels are subdivided to create the next higher resolution set of voxels that are candidates for coloring. In the three-dimensional case the neighborhood size used was two.

2.4.3 Static Scene Experiments

This section describes experiments evaluating the performance of the two methods described in Sections 2 and 3. The dataset consisted of eight views of a human figure evenly spaced above the scene (see Figure 19). All of the experiments were run on a 200 MHz R5000 SGI O2. Throughout
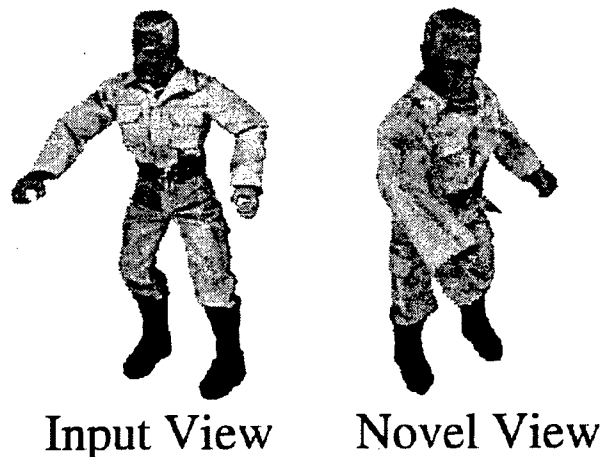
38

Input View   Novel View

Figure 19: Sample input image, and novel reconstructed view.

this section the term *scene resolution* refers to the largest of the three dimensions of the voxel scene space being colored.

2.4.3.1 Input Data Voxel coloring requires widely distributed views of a scene, and corresponding camera calibration information as input. For our experiments, Tsai's method was used to obtain calibration information [Tsa87].

Preprocessing of the input can result in dramatic performance gains. Preprocessing is independent of, and can occur separately from, voxel coloring. Thus it can be implemented in hardware or pipelined with voxel coloring.

Prewarping the input images greatly enhances the performance of the algorithm. Most cameras introduce some amount of radial distortion in images. Tsai's camera calibration method models this radial distortion [Tsa87]. However this method is slow; by prewarping the image we can use a pinhole camera model instead of Tsai's camera model.

Performance of voxel coloring is also enhanced by segmentation of the foreground from the background. This can be done automatically with a staging area using chroma key techniques, or by more elaborate techniques such as motion tracking and snakes. For now we make the assumption that automatic segmentation is available and robust enough for our purposes. For the data presented here the images were segmented manually.

2.4.3.2 Texture Mapping Results In order to perform texture mapping in hardware, the input images were scaled from 640x480 down to 128x128. Because of the reduced resolution,
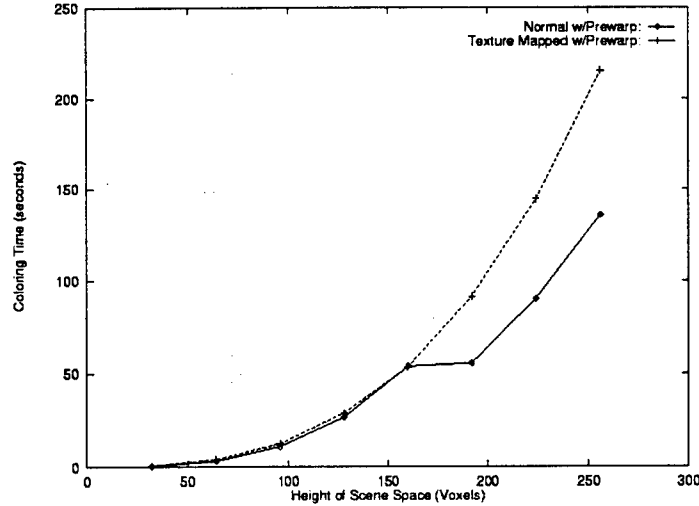
Figure 20: Texture mapping compared to original algorithm with prewarped input.

colorings were only performed at scene resolutions up to 256 voxels.

Texture mapping gives modest performance gains for scene resolutions up to 160 voxels. Figure 20 shows a comparison of texture mapping and the original algorithm with prewarping. For scene resolutions of at least 192 voxels, the texture resolution becomes smaller than the resolution of the voxel layer onto which it is projected. Expansion of textures may be performed more efficiently at this point on the SGI O2 architecture and thus there is a significant speed up.

The colors of voxels produced by texture mapping tend to be mixed with the background color because when the texture is projected the colors are interpolated, causing background pixels to mix with the foreground pixels. This degradation is most noticeable at voxels which correspond to the occluding contours of objects in the input images.

2.4.3.3 Coarse-to-Fine Results

Using this strategy the number of voxels traversed is reduced considerably. The total number of voxels traversed was reduced by 80% to 99% depending on the scene resolution. All of the colorings produced by the coarse-to-fine strategy were identical to the colorings produced by the original algorithm.

Table 1 summarizes the execution times of voxel coloring as the scene resolution (in voxels) increases. The original algorithm is compared to the multi-resolution method applied to prewarped input images. For high scene resolutions (512 voxels), the speedup was over forty times. For lower scene resolutions (128 voxels), the speedup was more modest.

40

Resolution	Orig. (sec)	M/P (sec)	Speedup
32	0.9291	0.7509	1.23
64	5.777	1.512	3.82
128	48.33	4.093	11.8
256	335.8	15.70	21.4
512	2671	64.98	41.1

Table 1: Comparison of original voxel coloring (Orig.) versus multi-resolution coloring with pre-warped input images (M/P).

Figure 21 compares the running time of the original algorithm with the multi-resolution variant as well the effect of prewarping the input images.

2.4.4 Dynamic Voxel Coloring

If we have video of a dynamic scene, we can take advantage of temporal coherence to avoid analyzing regions of scene-space that were determined to contain empty space at the previous time. This will work as long as the scene does not change too quickly and no new objects suddenly appear.

2.4.4.1 Using Temporal Coherence To take advantage of the fact that the scene will be similar between two successive points in time, the lowest resolution coloring from time t_k can be used as the starting point for the next time t_{k+1}, thus eliminating the need to visit every voxel at the lowest resolution. However, rapid motion in the scene may cause this assumption to be locally violated. Again, some sort of search strategy must be used to locate regions of colored space that lie outside the seed coloring.

Search strategies for dynamic scenes can be more complex than those for static scenes. Besides problems of false negatives that arise at low resolutions, the search strategy must correct any false negatives due to object motion. Tracking methods could be used for this purpose. Also, the size of the search window could be varied as a function of the estimated velocity of surfaces.

If we employ the same search strategy as the coarse-to-fine coloring algorithm, rapid movement will cause the surface to be missed. However, as long as the surface does not move completely out of the search window, the algorithm will reconstruct the missing voxels at the next time step.

Assume the voxels used in the initial pass are roughly six inch cubes, and the input video rate is 30 Hz. If the nearest-neighbor search strategy is used, a single voxel would have to move roughly two voxels between frames to escape the search neighborhood in the next time step. This corresponds to a velocity of about 10 meters/sec. If the motion in the scene is less than this threshold, the
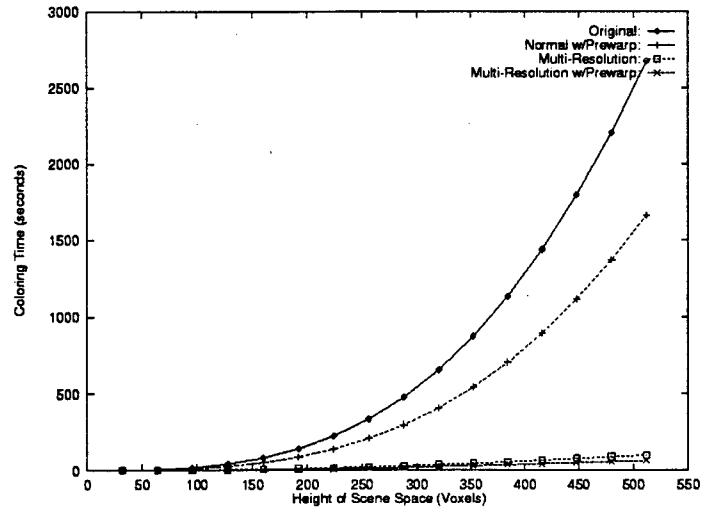
Figure 21: Running time vs. scene complexity for original and multi-resolution variant. Times are with and without prewarped input.

scene will be reconstructed correctly. For the initial implementation of dynamic voxel coloring we made the assumption that all moving objects can be adequately tracked from frame to frame by simply using nearest-neighbor searches.

As each frame is colored, the seed coloring to be used for the next time needs to be updated to reflect any changes due to scene motion. After the seed coloring is augmented, the set of voxels is subdivided. While the subdivided voxels are being colored, the seed coloring for the next time instant is generated. If any voxel in the increased resolution space is colored, then the corresponding super-voxel of the new seed coloring is also colored.

The algorithm for dynamic scene coloring can be summarized as follows:

```
grab all images at time t = t_0
seedcoloring = low res coloring of scene

loop grab all images at current time t do
    augment = seedcoloring plus neighbors
    seedcoloring = emptyset
    for each voxel in augment do
        if voxel should be colored do
            mark supervoxel in
            seedcoloring as
```

42

DISTRIBUTION LIST

| ADDRESSES	QUANTITY
AFRL/IFED (PETER J. COSTIANES) 32 BROOKS ROAD ROME NY 13441-4114	10
AFRL/IFOIL 26 ELECTRONIC PARKWAY ROME NY 13441-4514	1
AFRL/IFOI 26 ELECTRONIC PARKWAY ROME NY 13441-4514	1
ATTN DTIC-OCC DEFENSE TECHNICAL INFO CENTER 8725 JOHN J. KINGMAN RD, STE 0944 FT BELVOIR VA 22060-6218	2
ATTN DR CHARLES R. DYER UNIVERSITY OF WISCONSIN - MADISON DEPARTMENT OF COMPUTER SCIENCE 1210 W. DAYTON ST MADISON WI 53706	5
ATTN GEORGE LUKES DEFENSE ADVANCED RESEARCH PROJECTS AGENCY DARPA/ISO 3701 N FAIRFAX DRIVE ARLINGTON VA 22203-1714	5
TOTAL	24
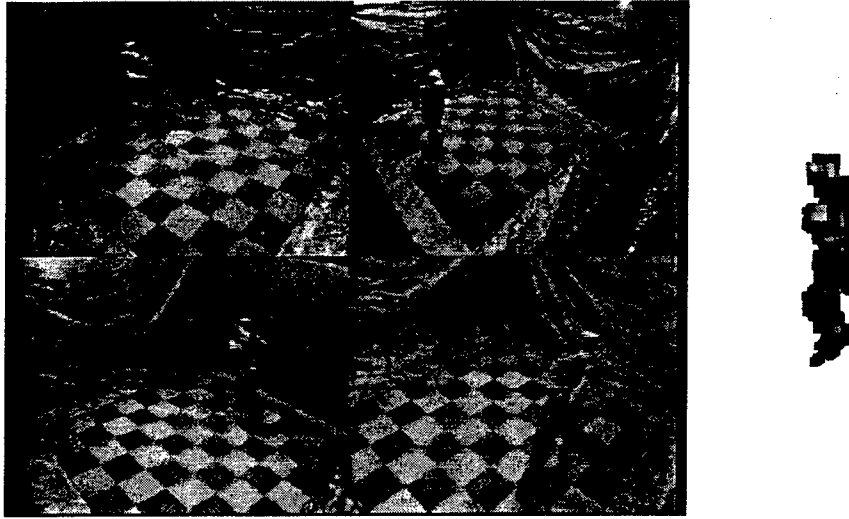
Figure 22: Four sample frames from the input video and corresponding output.

Time Step	Dynamic (sec)	Static (sec)
Initial	0.945	n/a
Time 0	0.540	1.28
Time 1	0.644	1.24
Time 2	0.611	1.19

Table 2: Execution time comparison of dynamic voxel coloring with the standard voxel coloring algorithm.

```
        colored
    od
od
```

2.4.4.2   Results   The dynamic coloring algorithm was applied to a sequence of three time steps using images from four cameras (see Figure 22). The total time to color the sequence was 2.74 seconds. This compares to 3.70 seconds for coloring the three scenes separately. The performance gains are more dramatic if considered on a frame by frame basis. Table 2 summarizes the results and shows that the per frame speedup is more than two times.
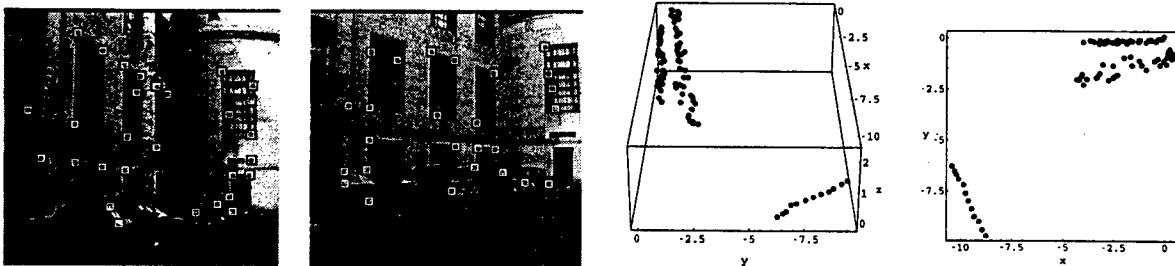
Figure 23: Structure from motion using projected error refinement. The left two images show two of the input views and detected feature points. The right two images show the result of the projected error refinement algorithm. Scene feature points are at the upper-left of the third figure and the upper-right of the right figure. The other points show the recovered camera positions.

2.5 Structure from Motion

We developed a novel structure-from-motion (SFM) method for recovering (static) 3D scene structure and camera positions from a set of images. Our approach overcomes some of the limitations of existing SFM methods by modeling perspective projection, allowing arbitrary camera positions, dealing with feature point outliers (i.e., errors in feature point correspondences and in feature point locations) and occlusion, and being computationally very efficient. The method is a type of bundle adjustment technique we have developed called *Projected Error Refinement* because it formulates the problem as determining the positions of the cameras and feature points so that the projectors (i.e., rays) of corresponding feature points come as close to intersecting as possible. An efficient iterative refinement algorithm takes an initial estimate of the structure and motion parameters and alternately refines the cameras' poses and the positions of the feature points. The solution can be refined to an arbitrary precision, and the algorithm converges rapidly even when the initial estimate is poor. See [Bes98] for complete details of this method.

Figure 23 shows two images of 12 taken of an outdoor scene containing significant perspective effects. 91 feature points were automatically extracted and tracked over the sequence of images, though most features were present in only a few frames. The results of the algorithm are shown in the two views in the right part of Figure 23.

3 Publications under the Grant

The following publications describe more results achieved under this grant. For more information and examples of results, see the web site at `http://www.cs.wisc.edu/vsam`

1. G. S. Bestor, Recovering Feature and Observer Position by Projected Error Refinement, Ph.D. Dissertation, Computer Science Department, University of Wisconsin-Madison, 1998 (available as Computer Science Department Technical Report 1381).

2. C. R. Dyer, Image-based scene rendering and manipulation research at the University of Wisconsin, *Proc. Image Understanding Workshop*, 1997, 63-67.

3. C. R. Dyer, Image-based visualization from widely-separated views, *Proc. Image Understanding Workshop*, 1998, 101-105.

4. R. A. Manning and C. R. Dyer, Dynamic view morphing, Computer Science Department Technical Report 1387, University of Wisconsin-Madison, September 1998.

5. R. A. Manning and C. R. Dyer, Interpolating view and scene motion by dynamic view morphing, *Proc. Image Understanding Workshop*, 1998, 323-330.

6. R. A. Manning and C. R. Dyer, Interpolating view and scene motion by dynamic view morphing, *Proc. Computer Vision and Pattern Recognition Conf.*, 1999, I-388 - I-394.

7. R. A. Manning and C. R. Dyer, Dynamic view interpolation without affine reconstruction, *Proc. NATO Advanced Research Workshop on the Confluence of Computer Vision and Computer Graphics*, Kluwer, Dordrecht, Netherlands, 2000, to appear.

8. A. C. Prock and C. R. Dyer, Towards real-time voxel coloring, *Proc. Image Understanding Workshop*, 1998, 315-321.

9. S. M. Seitz, Image-Based Transformation of Viewpoint and Scene Appearance, Ph.D. Dissertation, Computer Science Department, University of Wisconsin-Madison, 1997 (available as Computer Science Department Technical Report 1354).

10. S. M. Seitz and C. R. Dyer, View morphing: Uniquely predicting scene appearance from basis images, *Proc. Image Understanding Workshop*, 1997, 881-887.

11. S. M. Seitz and C. R. Dyer, Photorealistic scene reconstruction by voxel coloring, *Proc. Image Understanding Workshop*, 1997, 935-942.

12. S. M. Seitz and C. R. Dyer, Photorealistic scene reconstruction by voxel coloring, *Proc. Computer Vision and Pattern Recognition Conf.*, 1997, 1067-1073.

13. S. M. Seitz and K. N. Kutulakos, Plenoptic image editing, *Proc. 6th Int. Conf. Computer Vision*, 1998, 17-24.

14. S. M. Seitz and C. R. Dyer, Photorealistic scene reconstruction by voxel coloring, *Int. Journal of Computer Vision*, to appear.

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of information systems science and

technology for aerospace command and control and its transition to air,

space, and ground systems to meet customer needs in the areas of Global

Awareness, Dynamic Planning and Execution, and Global Information

Exchange is the focus of this AFRL organization. The directorate's areas

of investigation include a broad spectrum of information and fusion,

communication, collaborative environment and modeling and simulation,

defensive information warfare, and intelligent information systems

technologies.