Dynamic View Morphing

Russell A. Manning Charles R. Dyer

Department of Computer Sciences
University of Wisconsin
Madison, Wisconsin 53706

Technical Report #1387
September 1998

Abstract

We present a novel technique for interpolating between two views of a dynamic scene. Our approach extends the concept of view morphing introduced in [SD96] and retains the relative advantages of that method. The interpolation will portray *one possible* physically-valid version of what transpired in the scene during the intervening time between views. The scene is assumed to consist of a small number of *objects*. Each object can undergo any motion during the time between views as long as its total movement is equivalent to a single, rigid translation. The dynamic view morphing technique can work with widely-spaced reference views, sparse point correspondences, and uncalibrated cameras. When the *camera-to-camera transformation* can be determined, the virtual objects can be portrayed moving along straight-line, constant-velocity trajectories. Methods are developed for determining the camera-to-camera transformation from information available in the reference views. It is shown that each moving object in a scene has a corresponding fundamental matrix and that the camera-to-camera transformation can be determined from two distinct fundamental matrices. Dynamic view morphing is developed for both pinhole and orthographic cameras, and the use of three or more reference views is discussed. Static view morphing is made more versatile with respect to occlusion, and mosaicing is combined with dynamic view morphing for the case when both reference views share the same optical center. The resulting combination of techniques can be used to fill-in missing gaps in movies, perform "view hand-offs" between cameras at different locations, create movies from still images, perform movie stabilization and compression, track objects during periods of obstruction, and related tasks.

This work is sponsored by the National Science Foundation under Grant No. IIS-9530985 and by the Defense Advanced Research Projects Agency (DARPA) and Rome Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-1-0138.

1 Introduction

View interpolation [CW93] is the process of using two reference views of a scene to generate a series of virtual views which, taken together, represent a smooth, continuous, and physically accurate transition between the originals. When the original reference views are captured from different locations, a view interpolation will show a transition in viewpoint between the two locations. If the original cameras differ in their internal parameters or their orientation in space, a view interpolation will show a smooth transition in these characteristics as well.

If the two reference views are captured at different times and if the scene has changed during the interim, then a view interpolation must portray in a continuous manner the changes that occurred in the scene so as to provide a smooth transition from one view to the other. Most work relating to view interpolation assumes the scene is static and unchanging; in this paper we introduce a method for interpolating between views of a dynamic scene.

Our work is based upon an earlier technique called *view morphing* [SD96] which provides a method for interpolating between two widely-spaced views of a static scene (Fig. 2). 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 the 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. The practical nature of these advantages is discussed in more detail in Section 1.1, which surveys and compares existing techniques and their possible application to dynamic view interpolation.

The strengths just listed represent minimum requirements. If camera calibration is known or can be estimated, this information can be used to provide added control over the interpolation process. If the sparse point correspondence can be made more dense, then the view morphing algorithm will produce virtual views that are more physically correct and have greater realism. If more than two reference views are available, a wider range of view interpolations is possible.

Algorithms for view interpolation have many practical applications, including filling-in missing or unusable sections of movies, creating hand-off transitions between different cameras, creating movies out of still images, creating new views of a scene, compressing and stabilizing movies, and portraying or tracking objects that would otherwise be obstructed (see Section 6).

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 presented in this paper.

During the interval of time between when the two reference views are captured, a dynamic scene changes in some way. This is a broad statement because there are so many ways a scene could change. The dynamic view morphing algorithm we present applies only to dynamic scenes that change in the following manner: First, we think of dynamic scenes as being composed of a small number of *objects*.

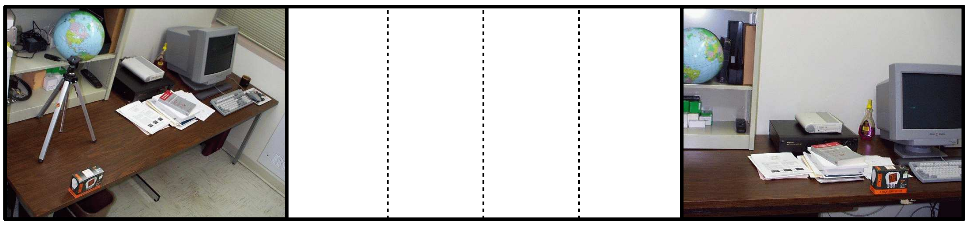

Figure 1: How do these two views relate to each other? How should the missing frames be filled? Notice that one element of the scene (i.e., the box in the foreground) has changed position between views.

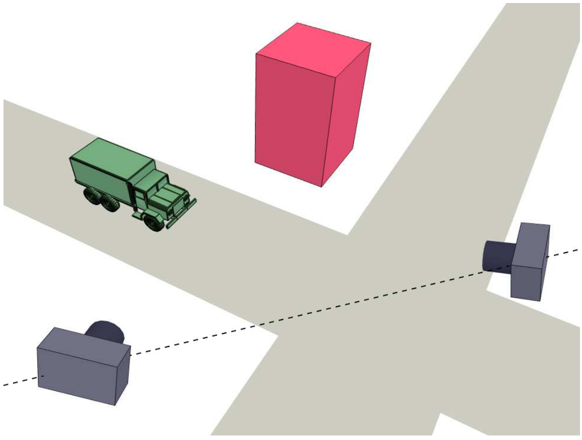

Figure 2: A static view morphing scenario. A building stands at the intersection of two roads, and a truck is parked near the building. The scene is viewed through two cameras. The view morphing algorithm can produce a series of new virtual views of the scene showing a gradual change in viewpoint between the two original camera views. If desired, the new viewing positions can be restricted to the line connecting the original cameras in space (the dotted line in the figure).

Each object can undergo *any* changes during the missing time interval as long as its total change from the start of the interval to the end is equivalent to a single, rigid, translational motion.

It is not our goal in this work to try and deduce the original trajectories for the various objects in a scene, especially from the limited amount of information we have assumed is available. Rather, our goal is to portray *some* possible trajectory for each object. This portrayal must be physically accurate, and it must depict a smooth, continuous transition in the object's location from the beginning of the interval to the end (Fig. 3).

Of course, even if the objects are portrayed in a physically accurate manner, the resulting interpolation may be unsatisfactory if the objects appear to move along unreasonable trajectories. To counter this problem, we have developed techniques for producing virtual trajectories that are straight lines in terms of camera-based coordinates.

A straight-line trajectory typically represents a natural, "best guess" path of motion. Furthermore, in many practical situations objects actually do follow straight-line trajectories. For instance, imagine a view interpolation of a scene in which a truck drives across a straight bridge. The trajectory for the truck must be a straight-line path or else during the interpolation the truck will appear to drive off the bridge and then back on.

Unfortunately, an object that moves along a straight-line trajectory in camera-based coordinates may not follow a straight-line trajectory in world coordinates, as Fig. 4 demonstrates. The solution is to develop a technique whereby the virtual objects and the virtual camera all move along straight-line paths *at constant velocity*. We will use the term *linear motion* to refer to such trajectories. Because of the desirability of portraying linear motion during a view interpolation, we have devoted considerable research effort to this subtopic.

1.1 Related Work

Since we are concerned with using existing reference views to create new views of a scene, we look to the group of computer graphics and machine vision techniques known collectively as *image-based rendering*. This group includes any computer graphics process that incorporates real photographic

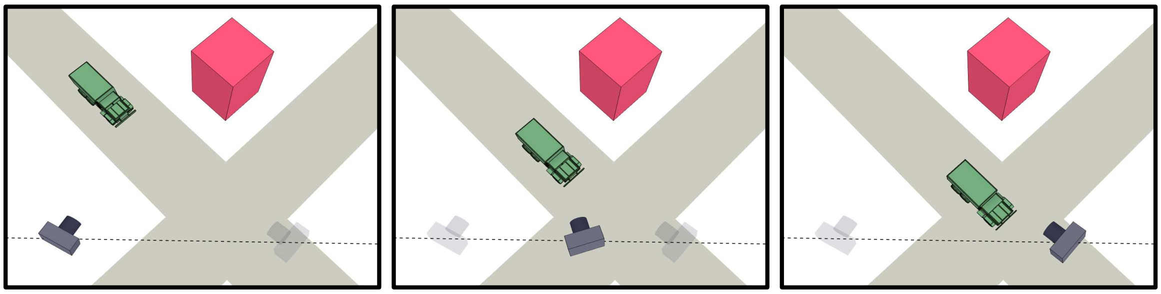

Figure 3: The scene in Fig. 2 becomes dynamic if the truck is driving down the road. The left and right frames above show the scene configuration when the reference views are taken at time $t = 0$ and $t = 1$, respectively. In both frames, the location of the corresponding reference camera is highlighted. The middle frame, which highlights the virtual camera, shows *one possible* recreation of the original scene at time $t = 0.6$. To recreate the scene, linear motion has been assumed (i.e., it has been assumed that the truck and the virtual camera both moved at constant velocity along straight-line paths). Dynamic view morphing produces a view of the recreated scene as captured by the virtual camera. The scene above has *two* moving objects: one is the truck and the other, called the "background object," consists of all stationary elements of the scene (such as the building and the road). The background object appears to move because the reference cameras are at different locations.

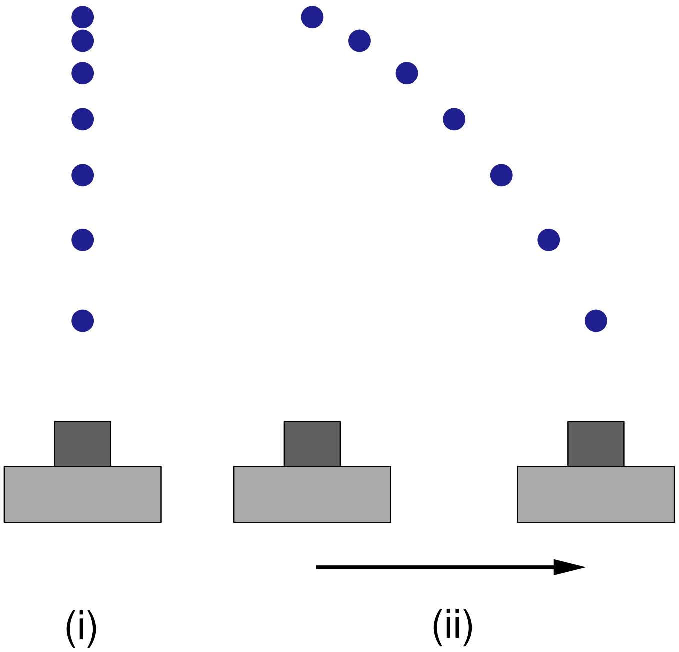

Figure 4: *(i)* A round object is filmed moving along a trajectory that is a straight line in the camera's frame of reference. The object is shown at equal time intervals and does not move at constant velocity. *(ii)* If the camera was in motion during the filming, then the object did not follow a straight-line trajectory in world coordinates.

images (e.g., texture mapping and morphing). In this section, we consider how various existing image-based rendering techniques might be used to create view interpolations for dynamic scenes. As it turns out, some of the techniques are simply not applicable to the problem, while others require more information than we are assuming is available.

One major category of image-based rendering techniques is called *mosaicing* and includes such work as [Sze94, Che95, Sze96, Int97, RPFRA98, SS97, Dav98]. As the name implies, these techniques seek to create large views of a scene by combining many small extant views. Most work on mosaicing is limited to static scenes, making the results not directly applicable to dynamic view interpolation. Davis [Dav98] worked with dynamic scenes, but his goal was to *remove* the motion from the dynamic scene in order to create one single, large, frozen view. Irani, et. al., [IAH95] described mosaicing of video sequences of dynamic scenes. Their technique assumes many reference views taken from a single vantage point; it does not involve the creation of views from novel positions in space nor the creation of virtual motion for objects in the scene. In general, mosaicing techniques will not directly apply to view interpolation problems because mosaicing involves piecing together existing views to create larger views, while view interpolation requires the creation of new views from novel vantage points. However, mosaicing techniques can be crucial in creating dynamic view morphs from co-centered reference cameras (see Section 5.1).

Another major group of image-based rendering techniques can be termed *image-based modeling* [MB95, AS97, SD97, TK92]. These techniques attempt to reconstruct three-dimensional scene geometry from reference views. Once the geometry is recovered, new views of the scene can be generated from arbitrary positions in space. All of these techniques make use of camera calibration and dense point correspondences in some way. The techniques of [MB95] and [AS97] automatically recover camera calibration and dense point correspondences and work from a small number of input views (e.g., two or three). However, they will fail if the original views are not closely spaced. Plenoptic modeling [MB95] requires panoramic reference views and also assumes a sparse set of point correspondences is initially provided. Voxel coloring [SD97] requires accurate camera calibration and a large number of input views, but will recover dense point correspondences automatically. The Factorization Method [TK92] by itself is just a method for recovering scene geometry, but it can be the basis for image-based modeling techniques such as one being developed at K^2T, Inc. The method assumes a very large number of closely spaced views (e.g., as in a movie), from which feature points can be tracked automatically and whose inherent redundancy provides numerical stability. The original technique assumed orthography, but has since been extended to other camera models.

An advantage of image-based modeling techniques is that, once the scene geometry has been recovered, new views can be generated from arbitrary viewpoints. Information about scene measurements also becomes available. Once the various scene objects have been modeled, they can be moved around the scene arbitrarily. Existing object models can be inserted into the scene, and scene elements can be removed or altered. The standard graphics pipeline can then be used to create view interpolations.

All of the methods cited for the previous category assume that the reference views are of static scenes. If the reference views contain object motion, then these techniques cannot be directly applied.

A third general category consists of image-based rendering techniques specifically designed for *view interpolation* [CW93, SD96]. Our work belongs to this group. The methods of this category are all pure image-based rendering techniques, in that three-dimensional scene geometry is not recovered. Instead, new views are created by moving pixels around or otherwise stretching and distorting the original views. As with the image-based modeling techniques but unlike mosaicing, these methods can produce new views of a scene from novel positions in space. View morphing [SD96] can work with unknown camera calibration and very sparse point correspondences. Both [CW93] and [SD96] are for static scenes and cannot be directly applied to dynamic view interpolation except in very limited circumstances (see Section 2.2).

A few other prominent image-based rendering techniques deserve mention with respect to the dynamic view interpolation problem. Talisman [TK96] is an image-based technique for speeding-up the traditional graphics pipeline. As in traditional three-dimensional graphics systems, Talisman starts with a complete model of the scene it seeks to portray. The key concept of Talisman is to avoid expensive rerendering of scene objects for each new view when a simple, fast affine transformation of a previously rendered object will serve as a suitable approximation. Talisman clearly solves a different problem than dynamic view interpolation; for instance, complete scene geometry is assumed and the output is meant to be approximate rather than physically correct.

Bregler [BCS97] presents a method for creating views of a novel dynamic scene from existing video footage of a closely related scene. Specifically, the technique starts with footage of a person talking and creates a new sequence showing the person saying new words with corresponding mouth and jaw movements. Bregler's system is an image-based technique for dynamic scenes, but it does not apply to general dynamic view interpolation problems nor does it use the kind of limited input assumed by our dynamic view interpolation technique.

2 Static View Morphing

2.1 Review of Static View Morphing

In this section, we review the static view morphing algorithm and related terminology. The algorithm can produce a series of new virtual views of a static scene showing a gradual change in viewpoint between two original camera views. If desired, the new viewing positions can be restricted to the line connecting the original cameras in space (Fig. 2).

The static view morphing algorithm takes as input two reference views of a static scene and a set of point correspondences between the views. A point correspondence is the projection of a particular scene point into both views; the pair of projections is the correspondence. We will use the term *conjugate points* to refer to such point correspondences. Each pair of conjugate points represents a particular scene point; if the calibration and position of both cameras are known, we can intersect the two rays generated by the conjugate points and locate the associated scene point. We will refer to these locations as *marked scene points*.

When the two reference views are sufficiently close, some existing stereo algorithms can produce a reliable set of point correspondences (for instance [ZDFL95]). However, when the views are closely spaced it is also possible to use techniques other than view morphing to produce new virtual views [MB95, AS97]. These techniques offer more flexibility in positioning the virtual camera. They also generate *dense correspondences*, leading to improved image quality. View morphing is most applicable when the views are widely spaced, in which case other techniques cannot be used. For widely-spaced views we know of no automatic technique for determining point correspondences reliably, and consequently in our work we determine point correspondences by hand.

From the point correspondences, the fundamental matrix is determined. At this point in the algorithm it is important to identify which camera model is being used: pinhole or orthographic. Depending on the model, the fundamental matrix is calculated differently and has a somewhat different interpretation. To simplify the presentation, we adopt the pinhole camera model everywhere except in Section 3.7. The orthographic model is nonetheless extremely important in practice, and all of Section 3.7 is devoted to it. In particular, the orthographic model will apply when the original views are captured through long distance aerial photography.

Finding the fundamental matrix requires at least eight conjugate points when the pinhole camera model is used; typically many more correspondences are determined so that we are rarely concerned with such minimums.

By using information contained in the fundamental matrix, the two reference views are transformed into *rectified* views (see Appendix in [SD96]). Two cameras and their corresponding views are rectified if both of the following hold simultaneously:

- the image planes of both cameras are *coplanar*
- the line through the optical centers of the two cameras is *colinear* with the x-axis of each camera

The meaning of rectification is somewhat different for orthographic views. For a description of what it means to transform a view, see section (3.1). The rectifying transformations are termed *prewarps* and this step is referred to as *prewarping* the views. Later in this paper, the term "prewarp" will be used more generally, referring to any transformation used to prepare the reference views for interpolation. This can be as simple as making both image planes parallel to each other (see Section 3.4) or even just rotating one of the images (see Section 3.7).

Rectified views have a special property that makes view morphing possible: Any linear interpolation of conjugate points corresponds to a view of the marked scene points from some position in space through some camera V. The new camera V is the virtual camera. If the two views meet both of the rectification conditions listed above, then V lies somewhere on the line connecting the original cameras, and V is rectified with both cameras. As is shown in Section 3.4, if only the first condition is met then the interpolation is still physically correct but the position of V has no simple characterization (see Section 3.3, Eqs. 7–9).

Using these observations, once the two views have been prewarped each pair of conjugate points is linearly interpolated by some fixed factor s. The result is a new view of the marked scene points. Each choice of s yields a new virtual view of the scene: choosing $s = 0$ or $s = 1$ yields the original views, while varying s smoothly between 0 and 1 yields a smooth transition in viewpoint between the original views and choosing $s < 0$ or $s > 1$ yields views from beyond the original reference positions.

In typical applications only a small, sparse set of conjugate points is given as input. The linear interpolation of this small set of conjugate points produces only a small section of the desired virtual view; the remainder of the virtual view is created through standard image morphing techniques (such as [BN92]) using the interpolated points to guide the morphing process. The result is a compelling, if approximate, new view of the original scene. Since the linear interpolation of conjugate points is physically valid, increasing the density of the original correspondence leads to increased realism.

We describe the new view as *physically correct* or *physically valid* even though only the marked scene points are guaranteed to be correctly projected into the virtual camera. In using a morphing algorithm to fill in the remainder of the virtual view, we make an underlying (Copernican) assumption that points that appear near each other in a given view are related in some way and thus will behave in a similar manner when viewed from a nearby position. When a sparse set of conjugate points is used, the realism of the virtual view will rely heavily on the accuracy of this assumption.

Although the marked scene points will be correctly projected into the virtual view, it is possible that a marked point should not be visible from the virtual camera position due to occlusion. Occlusion issues will be largely ignored in this paper. A technique for handling some types of occlusion, both in static and dynamic scenes, is provided through layering as discussed in Section 2.3 and Section 3.4 and demonstrated in sequence (iv) of Fig. 14. Almost all real scenes will have at least some limited occlusion, which will in turn cause some limited disruption in the virtual views due to the morphing algorithm. Technically, it is possible to depth order the marked scene points once the views are rectified: in a static scene, points with the smallest displacement are the farthest away. Such a depth ordering might be used to handle occlusion better than the straight-forward morphing algorithm can, and could be a topic of future research.

If desired, the virtual view can be transformed to make it the same size and shape as the original views or to give it some other desired property. This optional last step is termed *postwarping*.

2.2 Trivial Dynamic Scenes

The underlying theory of view morphing assumes static scenes. In practice, the algorithm handles certain types of small movement reasonably well. For instance, [SD96] gives an example depicting two reference views of a face in which the eyes change gaze direction. A smooth series of new views is generated showing a gradual change in the gaze direction of the eyes. In general, small changes in flat surface features are reasonably accommodated due to the underlying morphing process.

While view morphing can handle some limited dynamic scenes, it will produce unsatisfactory results on scenes containing large amounts of movement, especially when camera movement is not parallel to object movement.

2.3 Nontrivial Dynamic Scenes

Consider what happens when the static view morphing algorithm is applied directly to the dynamic scene in Fig. 3:

(Step 1) First, a set of conjugate points is determined for the two views.

(Step 2) Next, in order to perform rectification a fundamental matrix is found.

Already a problem arises: The fundamental matrix concept only applies to views of a static scene — what about the moving truck?

It is shown later that a separate fundamental matrix exists for each moving object in a dynamic scene. For our purposes here, assume that the direction of the truck's displacement is parallel (in space) to the direction of the camera's displacement. In this case, the same fundamental matrix applies to both the stationary background and the moving truck (see Eq. 2). All the conjugate points, regardless of whether they belong to the truck or the background, can be incorporated into finding this one fundamental matrix.

(Step 3) Rectify the images. Since the prewarps are determined from the fundamental matrix and the same fundamental matrix applies to both the background and the truck, the same rectification process is appropriate for both scene elements.

(Step 4) Interpolate conjugate points and morph the rectified views.

There is a serious problem with this last step, even for the special case under consideration. During the image morphing step, the original views will be literally ripped apart due to the motion of the truck. Control points on the truck will pull nearby background elements along with them as the truck moves. Similarly, control points for background elements near the truck will hold onto the truck, also causing a tearing effect.

The culprit is that, in a typical image morphing algorithm, control points that are near a pixel will have the most influence over that pixel (the Copernican assumption mentioned earlier). This is a good assumption for two points on the same side of the truck, but is a bad assumption for, say, a point at the bottom of the truck and a nearby point on the road.

The solution is to separate unrelated scene points. We accomplish this by segmenting the scene into different layers, with each layer representing a group of related scene points. A similar technique appears in [WA94]. Typically, we will have a separate layer for each moving object in the scene and one for the stationary "background" object. For instance, in the example just discussed there would be two layers: one for the truck and the other for the remaining background elements. Fig. 8 demonstrates layering.

A similar "tearing" effect can occur in a completely static scene due to objects being located at different depths. A *monotonicty assumption* was made in [SD96] to preclude this problem; however, the concept of layering can also be applied to static view morphing to deal with tearing effects (see Section 7, sequence (iv)).

While layering fixes some problems, it is only a first step towards a view morphing algorithm for dynamic scenes.

3 Dynamic View Morphing

3.1 Preliminary Concepts

We now introduce some notation and concepts that are used in discussing the dynamic view morphing algorithm.

First we assume the two reference views are captured through pinhole cameras, which are denoted *camera* A and *camera* B. Camera A will always be associated with time $t = 0$ and camera B with time $t = 1$. The pinhole camera model is equivalent to a frame of reference for measuring point locations in the scene: Associated with each camera is an origin, which is the camera's optical center, and a basis, which is a system of three linearly independent basis vectors used to take measurements in the scene.

In addition to the camera reference frames, we will use a *world* or *universal* frame of reference denoted U and consisting of an orthonormal system of basis vectors. The orthonormal basis ensures that measurements in U correspond to our intuitive sense of distance and angle.

If frames A, B, and U all share the same origin, then we can drop any reference to the location of the origins and instead think of A, B, and U as being simply bases for $\Re^3$. In this case, there exists a 3×3 matrix $\mathcal{T}_{UA}$ representing the basis transformation from U to A, and $\mathcal{T}_{UA}$ will correspond to the standard notion of a camera matrix. Note that throughout this paper we will use script capital letters to denote 3×3 matrices.

In general we will use the notation $\mathcal{T}_{GH}$ to represent the transformation between basis G and basis H, so in particular, $\mathcal{T}_{AU} = \mathcal{T}_{UA}^{-1}$. A major focus of this paper is the matrix $\mathcal{T}_{AB}$, termed the *camera-to-camera transformation.*

If the camera's optical center needs to be represented, the following 3×4 camera matrix can be used (in conjunction with homogeneous coordinates):

$$\left[\ \mathcal{T}_{UA}\ \middle|\ -\mathcal{T}_{UA}(A_{center})_U\ \right] \tag{1}$$

Here A_{center} refers to the location of camera A's optical center as a position in space (independent of frame of reference). The subscript U indicates this position as measured in frame of reference U. We will often use subscripts in this manner to indicate in which frame or basis a measurement is taken.

Assume for the moment that A and B have different optical centers, and define $\mathbf{e} = B_{center} - A_{center}$. The letter "e" here stands for "epipole" because when $\mathbf{e}$ is measured in either frame A or frame B, it corresponds to the standard notion of an epipole.

The fundamental matrix $\mathcal{F}$ for the two cameras A and B has the following representation:

$$\mathcal{F} = [\mathbf{e}_B]_\times^+ \mathcal{T}_{AB} \tag{2}$$

Here we use the standard notation of $[v]_\times$ for the cross product matrix, so that $v \otimes u = [v]_\times u$. Since the cross product matrix is antisymmetric and the fundamental matrix is only defined up to a scalar, we can drop the transpose and write the above as

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

The fundamental matrix can be used to test for a certain coplanarity condition: If two points $p, q \in \Re^3$ are such that $q = p + e$, then:

$$\mathbf{q}_B^+ \mathcal{F} \mathbf{p}_A = 0 \tag{4}$$

This follows because $\mathcal{T}_{AB}\ \mathbf{p}_A = \mathbf{p}_B$. Consequently, from Eq. 3 the left-hand side of (4) becomes $\mathbf{q}_B \cdot (\mathbf{e}_B \otimes \mathbf{p}_B)$. This triple product is 0 exactly when $\mathbf{q}$,$\mathbf{e}$,and $\mathbf{p}$ are coplanar, which is certainly true when $\mathbf{q} = \mathbf{p} + \mathbf{e}$. It is also true under many other conditions, a fact we will exploit later.

Alternatively, $\mathcal{F}$ can be written as:

$$\mathcal{F} = \mathcal{T}_{BA}^+ [\mathbf{e}_A]_\times$$

and still be used in the same coplanarity test.

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 Ω is moving in direction $\mathbf{u}$, the fundamental matrix *for the object* is:

$$\mathcal{F}_\Omega = [\mathbf{u}_B]_\times^+ \mathcal{T}_{AB} \tag{5}$$

The same fundamental matrix would arise if the object was stationary and the camera was moving in the direction $-\mathbf{u}$. In fact, the same fundamental matrix (up to a scalar multiple) would arise for any motion parallel to $\mathbf{u}$ (this is used in Section 3.5). Notice how the role of the epipole in Eq. 3 has been replaced by the object's motion vector $\mathbf{u}$ in Eq. 5. The projection of $\mathbf{u}$ onto a camera's image plane coincides with the *vanishing point* of Ω as viewed through the camera.

In this paper, the term "object" has a specific meaning that must be met for the underlying theory to apply. An *object* is any group of particles in the scene for which there exists a fixed vector $\mathbf{u}$ such that each particle at time $t = 1$ has been displaced by $\mathbf{u}$ from its position at time $t = 0$. In other words, an object is a group of particles that *may have* undergone the same fixed translational motion in the interim time between $t = 0$ and $t = 1$, at least as far as can be deduced from the two reference views. For example, if the truck in Fig. 2 is driving down the road without turning then it is an object. The building and the road are also objects since they undergo a translation of $\mathbf{u} = 0$. In fact, the building, the road, and all the other stationary elements can be thought of as one single, large object. We usually refer to this one large stationary object as the *background object* because it typically fills in the background of the reference views. If the two original cameras are at different locations then, under the fixed camera formulation introduced in the next section, the background object will be just another moving object in the scene.

A point in a view is, strictly speaking, a vector in $\Re^2$. However, we will often treat such points as being in $\Re^3$. We do this by taking the point's z-coordinate to be 1. The reasoning is that a camera's image plane is the plane $z = 1$ in terms of the camera's basis. The resulting three-dimensional vector is really a ray directed from the camera's optical center towards the original scene point being viewed. The ray has a color corresponding to the scene point's color. Transforming a view by a 3×3 matrix involves using the matrix to transform all of these colored rays, then reintersecting the rays with the plane $z = 1$ and coloring the plane appropriately.

3.2 Fixed Camera Formulation

Consider the scenario depicted in Fig. 9 in which a car drives down a straight road and is viewed by a camera flying overhead. The camera's translation $\mathbf{u}$ is indicated by a vector in the figure.

An attempt to directly model this scenario might include the camera's translation. Instead of taking this approach, we reformulate the scenario so that the optical centers of both cameras are at the same fixed location. The cameras may still have different internal parameters and may be oriented in different directions. We convert the original scenario to this new *fixed camera formulation* by simply adding $-\mathbf{u}$ to the translation of each element in the scene, including the stationary background elements (such as the road and the building). The reformulated scenario is also shown in Fig. 9.

In general, we can always convert any view morphing scenario into a fixed camera scenario simply by subtracting out the camera displacement; the reference views will be the same in either case.

Using the fixed camera formulation, the camera locations become unimportant and the camera matrices become simple 3×3 invertible matrices (e.g., $\mathcal{T}_{UA}$ of the previous section). Besides being simpler than 3×4 matrices, these matrices do not require the use of homogeneous coordinates and thus lead to more intuitive results.

This reformulation also allows us to apply view morphing techniques to scenarios in which the cameras really do share a fixed optical center (for instance, when a single camera mounted on a tripod is used for both views). Thus by fixing the camera location we unify a range of potential scenarios under one theory.

A complication arises in using the fixed-camera formulation. If an image interpolation is performed under this formulation, then how does the output relate to the original scene in world coordinates? If the reference cameras were actually co-centered, then the relationship is clear. If the reference cameras were not co-centered but the virtual trajectory of each object is linear in camera coordinates, then it is always possible to assume that the virtual camera follows a linear trajectory while maintaining the linear virtual motion of the objects in world coordinates.

In general, we can choose *any* trajectory for the virtual camera to follow as long as it is consistent with rigid, translational motion; we just have to interpret the results appropriately. Typically, the viewer will interpret the background object as being stationary. This means that the virtual camera in world coordinates moves equal and opposite to the background object's virtual motion. This can result in unintuitive virtual trajectories for the remaining objects in world coordinates. Consequently, it is desirable to produce linear virtual motion for all virtual objects.

3.3 View Interpolation for a Single Moving Object

The following discussion assumes pinhole cameras; see Section 3.7 for an analogous discussion involving orthographic cameras.

Assume the two reference cameras share the same optical center and are viewing a point ω that is part of an object Ω in motion. By definition, each object is associated with a translational motion $\mathbf{u}$. Let $\mathbf{q}$ and $\mathbf{q}+\mathbf{u}$ denote the position of ω at time $t=0$ and $t=1$, respectively (Fig. 5).

We initially require that the image planes of the two cameras be parallel to each other and to the vector $\mathbf{u}$; this will be enough to ensure that the virtual view is physically valid. Later we discuss how adding to this requirement can lead to greater control over the virtual motion of the object.

The image planes of the cameras are parallel to each other only if the third row of $\mathcal{T}_{UA}$ equals the third row of $\mathcal{T}_{UB}$ scaled by some constant λ. The image planes are parallel to $\mathbf{u}$ only if $(\mathcal{T}_{UA}\mathbf{u}_U)_z = (\mathcal{T}_{UB}\mathbf{u}_U)_z = 0$, where $(\cdot)_z$ denotes the z-coordinate of a vector. Setting $Z = (\mathcal{T}_{UA}\mathbf{q}_U)_z = \lambda(\mathcal{T}_{UB}(\mathbf{q}+\mathbf{u})_U)_z$, the linear interpolation of the projection of ω into both cameras is

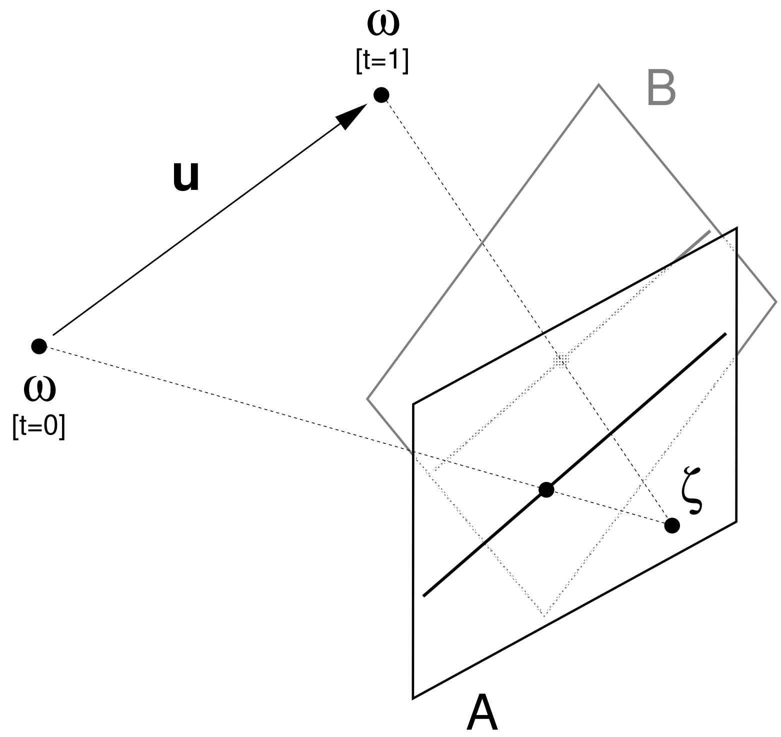

Figure 5: Cameras A and B share the same optical center ζ and are viewing a point on an object that translates by $\mathbf{u}$. The image planes of the cameras are parallel to each other and to $\mathbf{u}$, and hence interpolation will produce a physically correct view of the object. On each image plane a line parallel to $\mathbf{u}$ is shown.

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

Figure 6: How the virtual view of an object is related to different preconditions on the reference views. Stricter preconditions lead to increased control over the output.

$$(1-s)\frac{1}{Z}\mathcal{T}_{UA}\mathbf{q}_U + s\frac{\lambda}{Z}\mathcal{T}_{UB}(\mathbf{q}+\mathbf{u})_U \tag{6}$$

Now define a virtual camera V by the camera matrix

$$\mathcal{T}_{UV} = (1-s)\mathcal{T}_{UA} + s\lambda\mathcal{T}_{UB} \tag{7}$$

It is implicit here that V shares the same optical center as A and B. With this definition of V, the linear interpolation (6) is equal to the projection of scene point $\mathbf{q}(s)$ onto the image plane of camera V, where

$$\begin{aligned} \mathbf{q}(s) &= \mathbf{q} + \mathbf{u}(s) & (8) \\ \mathbf{u}(s)_V &= s\lambda\mathbf{u}_B & (9) \end{aligned}$$

Notice that $\mathbf{u}(s)$ depends only on $\mathbf{u}$ and the camera matrices and not on the starting location $\mathbf{q}$. 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 $\mathbf{u}(s)$ from its starting position. In Eq. 9, $\mathbf{u}(s)$ is represented in basis V. This representation must be transformed by $\mathcal{T}_{VU}$ to arrive at a world-coordinate representation. Furthermore, if we choose to think of the virtual camera as moving in world coordinates, then $\mathbf{u}(s)$ must be translated by the opposite of the chosen virtual camera displacement at time s to create the final world-coordinate representation. All of this is complicated by the fact that basis V changes with s.

As s varies, the change in $\mathbf{u}(s)$ will define the virtual object's motion through space. When $s = 0$ and $s = 1$ the virtual object will be positioned exactly as viewed in cameras A and B respectively.

The motion path given above is unpredictable and unintuitive. We would like the virtual object to travel along a straight-line path and, if possible, at a constant velocity. Straight-line motion is achieved when $\mathbf{u}_A = \mathbf{u}_B$ up to an arbitrary scalar; constant-velocity straight-line motion (i.e., linear motion) is achieved when $\mathbf{u}_A = \lambda\mathbf{u}_B$. Fig. 6 concisely summarizes the observations made so far. Proof of these claims follows:

Assume $\mathbf{u}_A = k\mathbf{u}_B$ for some scalar k. Multiplying both sides of Eq. 7 on the right by $\mathcal{T}_{BU}$ yields

$$\mathcal{T}_{BV} = (1-s)\mathcal{T}_{BA} + s\lambda\mathcal{I} \tag{10}$$

By multiplying both sides of Eq. 10 on the right by $\mathbf{u}_B$ and on the left by $\mathcal{T}_{VB}$ the following can be derived:

$$\mathcal{T}_{VB}\mathbf{u}_B = \frac{1}{(1-s)k + s\lambda}\mathbf{u}_B \tag{11}$$

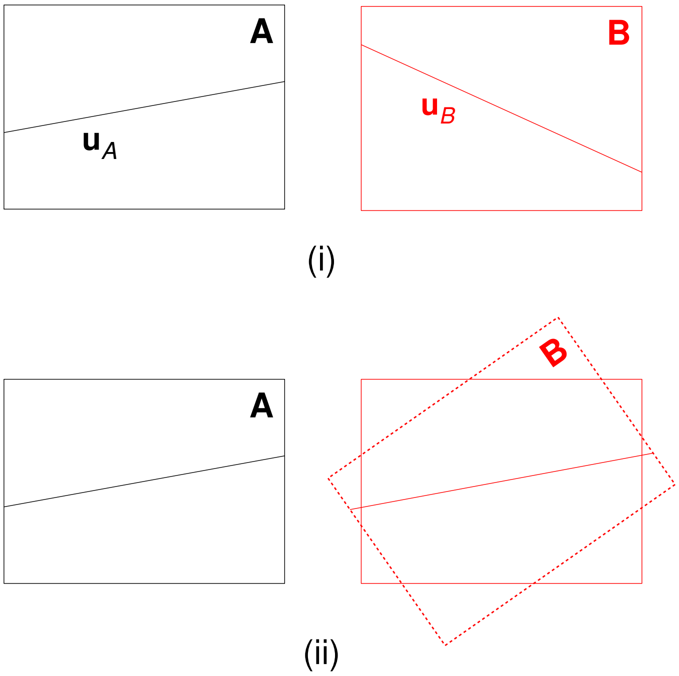

Figure 7: Illustration of direction alignment. *(i)* The image planes from Fig. 5. *(ii)* In order for the interpolation to produce straight-line motion, image B is first transformed (by a rotation in this case) so that the motion directions are aligned in both views.

Multiplying both sides of Eq. 9 by $\mathcal{T}_{VB}$ now yields:

$$\mathbf{u}(s)_B = \frac{s\lambda}{(1-s)k + s\lambda}\mathbf{u}_B \tag{12}$$

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.

One way to achieve straight-line motion is to rectify the two reference cameras with respect to the motion of Ω. Then $(\mathbf{u}_A)_y = (\mathbf{u}_B)_y = 0$ and hence $\mathbf{u}_A = \mathbf{u}_B$ up to a scalar. The straight-line trajectory will be parallel to the camera x-axis. When transformation $\mathcal{T}_{AU}$ is applied, the path is still a straight line in world coordinates, assuming the reference cameras share the same optical center.

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. By applying the earlier results to this special case, we derive the following corollary which forms the basis of the algorithm in Section 3.4:

If both camera matrices are equal and if $(\mathbf{u}_A)_z = 0$, 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.

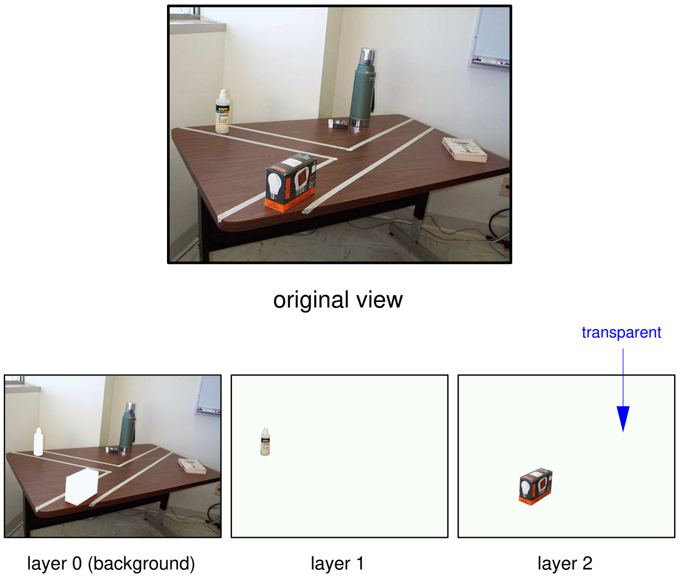

Figure 8: A view divided into layers. Each layer corresponds to a moving object. The single "background" object contains many different objects that all translate by the same amount.

3.4 Linear Motion Dynamic View Morphing Algorithm

We now present a dynamic view morphing algorithm that will portray linear motion. If all of the motion in the original scene was indeed linear then this algorithm can be used to accurately recreate the scene. In any event, linear motion is often a good "best guess" for portraying how a scene changed.

In order to use the corollary from Section 3.3, $\mathcal{T}_{AB}$ must be known or must be calculable. Determining $\mathcal{T}_{AB}$ is of major importance to this work and has been the focus of much research. Intuitively, if cameras A and B are at the same position in space, then $\mathcal{T}_{AB}$ represents how the view from camera B at a certain time can be determined given the view from camera A at that time. For this reason, we call $\mathcal{T}_{AB}$ the *camera-to-camera transformation.*

Assuming that $\mathcal{T}_{AB}$ is known and that each object has nonzero motion under the fixed-camera formulation, the dynamic view morphing algorithm is as follows. To handle objects that have no motion, see the discussion in Section 5.1.

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

We number the layers as follows: 0 is the background object layer, 1 is the next closest layer, and so on incrementally up to the nearest layer (Fig. 8).

The layers are depth ordered only for the sake of recompositing them to create the final view. Because the objects are in motion, the depth ordering of objects may change over time. Assuming a known $\mathcal{T}_{AB}$, the relative depth of the objects can be determined and the layers can be recomposited correctly for different times (perhaps by changing the original depth order during the reassembly process). In many cases, the objects never overlap and the depth ordering is not important.

It is conceivable that different parts of an object should have different depth orderings. This is especially true of the so-called background object because of its size. Such objects can be subdivided into several smaller objects, each on its own layer, and the resulting layers can be depth ordered appropriately. For example, if there was a second building in Fig. 2 that was closer to the cameras than the truck, this second building would be placed on its own layer and the new layer would be designated in the depth ordering as being closer than the truck layer.

Currently, we segment the views by hand. Depending on what information is available or what techniques are developed in the future, this process may eventually be automated.

(Step 2) Transform each layer of view B by $\mathcal{T}_{BA}$.

Each of these layers now becomes a view of its corresponding object as seen at time $t = 1$ through camera A. This compliments the views that already exist of each object at time $t = 0$ through camera A. At this point, we can forget about camera B and continue as if all views of the scene were captured by camera A.

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

To understand what step (3) does, consider the result of applying static view morphing to a particular layer i using a particular interpolation factor s. Both views are associated with the same camera so the same prewarp transformation $\mathcal{T}_{AR_i}$ is applied to each; R_i denotes the new camera basis. Note that it is only necessary in this case that the image *plane* of R_i be parallel to the displacement vector $\mathbf{u}_i$ of object i. This can be useful if the true direction of motion of object i is unknown.

A virtual view is then created through linear interpolation and morphing. By the corollary of Section 3.3, the virtual camera will also be R_i and the virtual view will be a physically correct view of the object i at time s following a constant-velocity straight-line trajectory.

Once the virtual view has been created, it is transformed by $\mathcal{T}_{R_iA}$ to create a virtual view through camera A. In this way, basis A becomes a common reference basis linking all the virtual views of the different layers.

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

The virtual camera for each layer is camera A, so compositing the layers creates a virtual view that is also through camera A. The composite view portrays all the virtual objects at time s following linear virtual trajectories.

Compositing the layers involves stacking the layers from closest to farthest away and then using alpha channel transparency values to determine which parts of the further-away layers are blocked out by the closer layers. The alpha channel is morphed just like the color channels. The compositing process is analogous to cel animation, where an animated figure is painted on a clear cel and then placed over a background.

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

This is done, for instance, to produce a smooth change in the bounding rectangles from view A to view B. A simple postwarp transformation is the linear interpolation of $\mathcal{I}$ and $\mathcal{T}_{AB}$ (i.e., $(1-s)\mathcal{I} + s\mathcal{T}_{AB}$). In our implementation, we use a postwarp method discussed in [SD96]: The four matching corners of each original view are prewarped and then linearly interpolated. The postwarp is chosen to map the four interpolated corners to the corners of a rectangle (a projective transformation can be determined by its behavior on four such points [Hec89]). The rectangle can be the linear interpolation of the two bounding rectangles of the original views.

Step 5 marks the end of the algorithm. We return now to a complication that can arise in step 2. If the direction of motion of object i, also called its vanishing point, is visible inside layer i then not all of the layer can be transformed to camera R_i and the original view morphing algorithm cannot be applied. Since most moving objects (except the background) are typically small, this restriction can be relaxed: the vanishing point need only be outside the object itself; it can be in the "transparent" region of the layer. A modification to the basic implementation is necessary to account for this relaxed condition.

Alternatively, the fully unrestricted "environment map morphing" technique described in [MD98] can be used. In particular, this solves the problem with respect to the background object or any other large object that can easily contain its own vanishing point. The background object is especially prone to containing its own vanishing point; this will happen whenever the camera is translating into or out of the scene being viewed.

In the next two sections we examine some special case scenarios for which dynamic view interpolations can be produced without knowledge of $\mathcal{T}_{AB}$.

3.5 Special Case: Parallel Motion

Assume a fixed-camera formulation and let $\mathbf{u}_i$ denote the displacement between the position of 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. In such scenes, due to the lack of distinct conjugate directions, it is difficult to use the techniques of Section 4 to find $\mathcal{T}_{AB}$. Without $\mathcal{T}_{AB}$ the linear motion algorithm of Section 3.4 cannot be applied. On the other hand, the fundamental matrix with respect to each object is the same, leading to the following algorithm:

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.*

In static view morphing, the prewarps are determined from the fundamental matrix. Since each object has the same fundamental matrix, the same prewarps are applied to each layer. Also, all conjugate points for all the layers can be used together to compute the one fundamental matrix.

If the reference cameras actually shared the same optical center, then the background object has no motion. In this case, static view morphing cannot be applied to the background layer and a panoramic mosaicing technique should be used instead (see Section 5.1).

The virtual objects will follow straight-line trajectories as measured in the camera frame. This is because, during the static view morphing step the reference views are transformed into rectified views; hence the conjugate directions of motion for all objects will be parallel to the x-axis in both views. Depending on what path of motion is chosen for the virtual camera, the virtual objects may or may not follow straight-line paths in world coordinates. However, it is typical to assume that the background object is stationary in world coordinates. In this case, the virtual camera motion is equal and opposite to the virtual motion of the background object and is therefore parallel to the motion of all the virtual objects. Under this interpretation, the virtual objects and the virtual camera will all follow straight-line paths in world coordinates, though perhaps at unknown rates of speed.

We now give an example of why these last comments are important. Imagine a truck is driving down a straight road and is viewed by a camera in an airplane that is flying parallel to the road. The parallel motion case applies to this scenario. By the previous comments, the resulting dynamic view morph will portray the truck driving straight down the road, although perhaps at an uneven rate of speed; it may even portray the truck as moving backwards at some point in the interpolation.

3.6 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.

The planar parallel motion case can arise in a situation such as the following: Two straight roads intersect on fairly flat terrain and several vehicles are traveling along the roads. The scene is viewed from an aerial camera that stays at relatively constant altitude.

As usual, if $\mathcal{T}_{AB}$ is known or can be estimated then the linear motion algorithm applies. Otherwise, the following observation can be used.

Recall that in Section 3.3 the only requirement for the virtual view to be a physically-accurate portrayal of an object Ω that translates by $\mathbf{u}$ is that the image planes of both reference views be parallel to $\mathbf{u}$ 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 vanishing point of each object to infinity, meaning it makes the z component of the transformed vanishing point equal to* 0. *For each layer: prewarp the layer, linearly interpolate conjugate points, and morph the layer based on the interpolated conjugate points. Recomposite all the morphed layers to make a new, physically-accurate virtual view.*

The prewarps needed for the algorithm can be found with knowledge of two conjugate directions that are not parallel to each other but are both parallel to the target plane (see Section 4.1 for the definition of conjugate directions). If the direction of displacement for two objects is known and the objects are not displaced parallel to each other, then this information is sufficient. Typically, a reliable fundamental matrix can be calculated for the background object due to its size. From this fundamental matrix, one pair of conjugate directions can be found. If a reliable fundamental matrix can be calculated for another object that is not moving parallel to the background object, then the two necessary conjugate directions are known. Alternatively, pairs of parallel lines can be used as discussed in Section 4.1. In the example given above, the edges of the straight roads can be used because we know the roads are parallel to the terrain surface, and all the objects are moving parallel to the surface. Tops of buildings can also be used in the same way.

The algorithm given above only guarantees physical correctness, not straight-line or linear virtual motion. Linear motion can be achieved by finding $\mathcal{T}_{AB}$. A straight-line trajectory in the camera frame can be achieved for any object whose conjugate direction of motion is known; the technique is the same as that used in Section 3.7 to achieve straight-line trajectories. The first step is to transform the reference views so as to align all known pairs of conjugate directions. "Alignment" means the conjugate directions are made to point in the same direction in each view. With four or more conjugate directions, this can be achieved by finding $\mathcal{T}_{AB}$ (see Section 4.1). With three or fewer, the method in Section 3.7 can be used (Fig. 11).

3.6.1 Producing the Appearance of Straight Motion Without $\mathcal{T}_{AB}$

We now give an example to illustrate the planar parallel algorithm and to make an important observation (Fig. 9). This example serves to illustrate the technique for producing straight-line trajectories with orthographic cameras as well. For the example, assume a car is driving down a straight road on level terrain and is photographed at two different times from an airplane that is not moving parallel to the car but is flying at constant altitude. The flat surface of the ground can be taken as the common parallel plane.

Assuming a fixed-camera formulation, let $\mathbf{u}_0$ denote the direction of motion of the background object and let $\mathbf{u}_1$ denote the direction the road is heading. That is, $\mathbf{u}_1$ is a vector parallel to the edges of the road. Note that $\mathbf{u}_1$ is not the direction of the car's motion under the fixed-camera formulation; the car's motion is the sum of the original translation of the car traveling down the road and the translation of the camera.

Since vectors $\mathbf{u}_0$ and $\mathbf{u}_1$ are both parallel to the ground, they can be used to transform the two

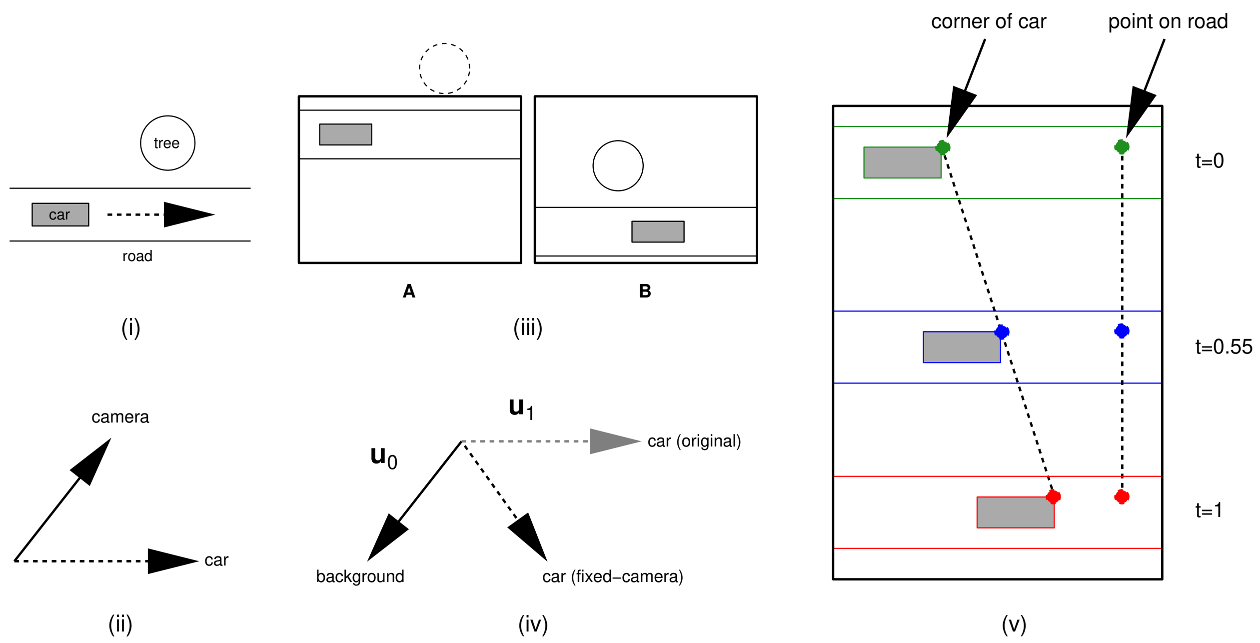

Figure 9: Illustration of how a virtual car object can appear to stay traveling down a road even when $\mathcal{T}_{AB}$ is unknown. *(i)* A dynamic scene in world coordinates. *(ii)* Direction of camera and car movement in world coordinates. *(iii)* Reference views as captured by the moving camera. *(iv)* Movement of objects under the fixed-camera formulation. *(v)* Assuming the background motion is purely vertical and the road is purely horizontal, linear interpolation of conjugate points on the car will produce a virtual car that stays in the same horizontal position on the interpolated road.

reference views to be parallel to the common plane. After this transformation, the views can be further transformed so that $\mathbf{u}_0$ points in the direction of the y-axis in both views and $\mathbf{u}_1$ points in the direction of the x-axis.

The car's translation at this point is the sum of two perpendicular vectors: one is the translation of the background, which is entirely along the y-axis, the other is the original translation of the car along the road, which is entirely along the x-axis. When the car object is interpolated, the interpolation of the y-components will exactly match the interpolation of the background object (which is entirely in the y direction). Since the road is part of the background object, this means that the car will stay on the road at every step of the interpolation, even as it translates in the x direction. In particular, the wheels of the car will stay on the correct horizontal line of the road throughout the interpolation; if the wheels had been touching one of the straight edges of the road, then they would stay touching that edge (Fig. 9(v)).

Producing a view interpolation in which the car appears to move straight down the road is crucial for realism. The previous observation shows that this can be accomplished in some cases even without knowledge of $\mathcal{T}_{AB}$.

Is it necessary that the road be made purely horizontal and the background motion be made purely vertical before the interpolation is performed in order for the interpolated car object to stay on the road? The answer is "no" as we now demonstrate.

Assume the two reference views have been transformed so that (1) the image planes of both cameras are parallel to the common plane of the ground, and (2) the conjugate directions $\mathbf{u}_0$ and $\mathbf{u}_1$ are aligned in the views. At this point a transformation $\mathcal{M}$ that preserves z-coordinates (i.e., an affine transformation) could be applied to both views so as to make the road parallel to the x-axis and the motion of the background object parallel to the y-axis. For instance, $\mathcal{M}$ could be a rotation followed by a shear, both in the x,y-plane.

By the discussion above, after $\mathcal{M}$ is applied the interpolated car object will always stay on the interpolated road. After the interpolation, $\mathcal{M}^{-1}$ could be applied to the virtual view to bring it into agreement with the reference views. The interpolated car will still be on the road after $\mathcal{M}^{-1}$ is applied. Mathematically, this process of interpolation is described by

$$\mathcal{M}^{-1}[(1-s)\frac{1}{Z}\mathcal{M}\mathcal{T}_{UA}\mathbf{q}_U + s\frac{\lambda}{Z}\mathcal{M}\mathcal{T}_{UB}(\mathbf{q}+\mathbf{u})_U] \tag{13}$$

The corresponding virtual camera is

$$\mathcal{M}^{-1}[(1-s)\mathcal{M}\mathcal{T}_{UA} + s\mathcal{M}\mathcal{T}_{UB}] \tag{14}$$

In both equations, the matrix $\mathcal{M}$ cancels out and the resulting equations exactly match those of Section 3.3. The net effect is that it was not necessary to apply $\mathcal{M}$ at all; straight-forward interpolation would have kept the car on the road just as well.

Note that the previous discussion applies to any invertible $\mathcal{M}$ that preserves the z component. This can be a powerful tool for generalizing view interpolation results. For instance, it could have been used to prove the results of Section 3.3.

3.7 Orthographic Cameras

Although the development for orthographic cameras is very similar to that for pinhole cameras, the underlying geometry of the two models is very different and warrants a separate treatment.

The orthographic camera case is extremely important in practice. In addition, the algorithms for orthography are simpler and more robust than those for pinhole cameras.

Like a pinhole camera, an orthographic camera has an orientation in space and can be associated with a system of basis vectors. However, projection onto the image plane consists entirely of setting the z-coordinate to 0. In effect, an orthographic camera has a z-axis corresponding to an infinitely long z basis vector. Using the fixed-camera formulation, in which the camera matrices are just 3×3, the third row of an orthographic camera is all 0's. Such a representation does not contain information about the camera's orientation and basis vectors. Hence, for the purposes of this section we will represent an orthographic camera A by the product of two matrices:

$$\widetilde{\mathcal{T}}_{UA} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathcal{T}_{UA} \tag{15}$$

Here $\widetilde{\mathcal{T}}_{UA}$ denotes the orthographic camera matrix for A while $\mathcal{T}_{UA}$ provides the underlying camera basis and orientation.

When cameras A and B share the same z-axis, there exists a camera-to-camera transformation. This transformation is given by the matrix

$$\widetilde{\mathcal{T}}_{AB} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathcal{T}_{AB} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{16}$$

When the camera-to-camera transformation $\widetilde{\mathcal{T}}_{AB}$ exists, an algorithm analogous to the linear motion algorithm in Section 3.4 can be used and the resulting virtual views will depict fully linear motion. $\widetilde{\mathcal{T}}_{AB}$ can be determined in this case from knowledge of just two conjugate directions: one conjugate direction is aligned with the x-axis of each view, the other with the y-axis, and then the x and y axes are scaled to place conjugate points on the same horizontal or vertical lines. Alternatively, a third conjugate direction could be used to determine scaling for the x and y axes (Fig. 11).

In general, except for when the z axes are aligned, no camera-to-camera transformation exists for orthographic cameras (Fig. 10). The following results lead to a general-purpose algorithm for orthographic cameras:

Let Ω denote an object in the scene and let $\mathbf{u}$ be the associated translation vector. Let ω be a point belonging to the object and let $\mathbf{q}$ and $\mathbf{q} + \mathbf{u}$ denote the position of ω at time $t = 0$ and $t = 1$, respectively.

Projecting ω into camera A at time $t = 0$ and into camera B at time $t = 1$ and then linearly interpolating the projected image points yields

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} [(1-s)\mathcal{T}_{UA}\mathbf{q}_U + s\mathcal{T}_{UB}(\mathbf{q}+\mathbf{u})_U] \tag{17}$$

Because the x,y-projection matrix can be factored to the left, the development for orthographic cameras continues exactly as with pinhole cameras in Section 3.3 with one notable exception: There are no conditions on the reference views because the z-coordinates will always be 0, and consequently there is no prepwarp step. As before, a virtual camera V exists such that the interpolation in Eq. 17 is the projection of scene point $\mathbf{q} + \mathbf{u}(s)$ into camera V. Camera V is defined by

$$\widetilde{\mathcal{T}}_{UV} = (1-s)\widetilde{\mathcal{T}}_{UA} + s\widetilde{\mathcal{T}}_{UB} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \mathcal{T}_{UV} \tag{18}$$

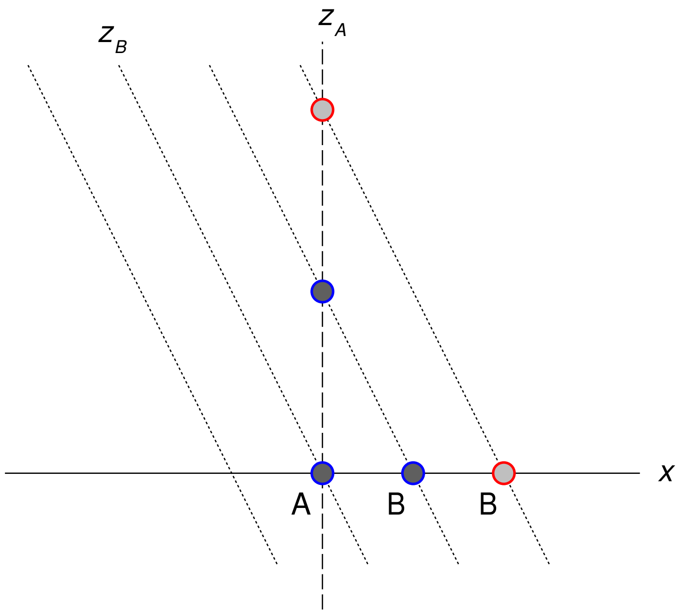

Figure 10: Two orthographic cameras share the same optical center, but not the same z-axis. The scene consists of two points lying on the z-axis of camera A. The points both project to the same location in camera A but to different locations in camera B, so no camera-to-camera transformation exists.

and $\mathbf{u}(s)$ is as given in Eq. 9. In Eq. 9, $\mathbf{u}(s)$ is given in terms of basis V, which is the basis implicit in $\mathcal{T}_{UV}$. Note that the z-coordinate of $\mathbf{u}(s)$ is arbitrary.

The results of Section 3.3 pertaining to straight-line and linear virtual motion also hold for orthographic cameras.

It is always possible to simultaneously align three or fewer conjugate directions by using a technique analogous to that described above for finding $\widetilde{\mathcal{T}}_{AB}$ (Fig. 11). Any object whose conjugate translation direction can be aligned in both views will be portrayed as moving along a straight-line path in camera coordinates.

With pinhole cameras, the camera-to-camera transformation always exists so it is always possible to align any number of conjugate directions. With orthographic cameras, the camera-to-camera transformation may not exist so it is not always possible to portray straight-line motion for four or more objects.

4 Calculating or Estimating $\mathcal{T}_{AB}$

The problem of determining $\mathcal{T}_{AB}$ is central to the linear motion algorithm of Section 3.4. This problem, similar in nature to the relative orientation problem of [Hor90], can be termed the *relative calibration* problem since it consists of calibrating camera B in terms of camera A's basis. We call $\mathcal{T}_{AB}$ the *camera-to-camera transformation.*

In order to use the linear motion algorithm it is only necessary to find $\mathcal{T}_{AB}$ up to a scalar. $\mathcal{T}_{AB}$ is used in Step 2 of the algorithm to prewarp the layers of view B. Since transforming a view involves transforming the colored rays that compose the view (and then resampling), it is irrelevant how the rays are scaled; all that matters is that each ray points in the correct direction. In terms of projective geometry, the scale factor cancels out after division by the z-coordinate.

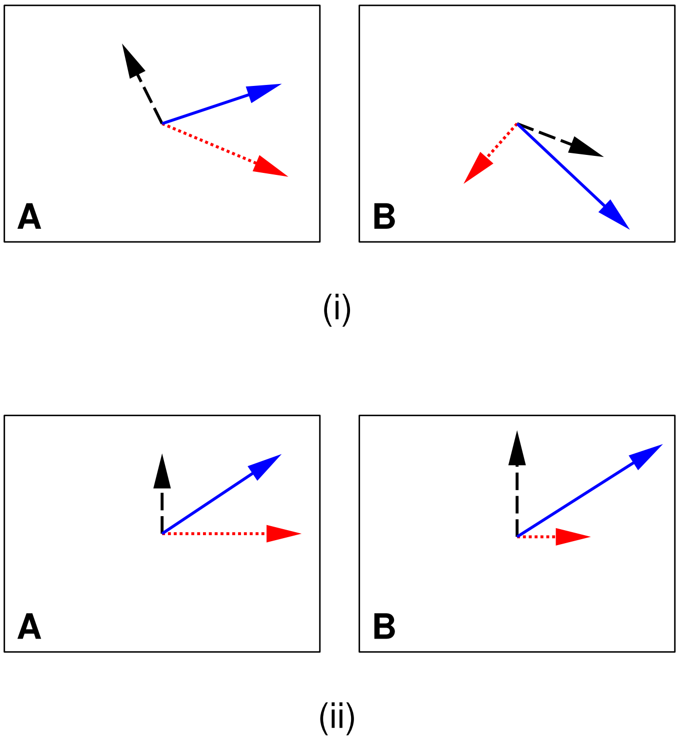

Figure 11: *(i)* Three conjugate directions as seen in view A and view B. *(ii)* It is possible to transform the views so that all three conjugate directions are aligned. In this case, a rotation is used to point the dotted vector along the x-axis, then a shear is used to point the dashed vector along the y-axis, and finally the x and y axes are scaled to align the solid vector.

$\mathcal{T}_{AB}$ can be determined if the cameras are internally calibrated and if enough conjugate points are known to determine the relative orientation. In [Hor90] it is shown that five conjugates points is a sufficient number for finding the relative orientation; we always assume many more correspondences than this are provided anyway. If the full camera matrices $\mathcal{T}_{UA}$ and $\mathcal{T}_{UB}$ are known, then $\mathcal{T}_{AB}$ is just $\mathcal{T}_{UB}\mathcal{T}_{UA}^{-1}$.

The two methods just mentioned require knowledge about the reference cameras. We now present four methods for calculating or estimating $\mathcal{T}_{AB}$ from information available solely in the reference views.

4.1 From Four Conjugate Directions

Recall that a point in view A and a point in view B are called conjugate if they are the projection of the same scene point. We have the following concept by analogy:

A direction $\mathbf{d}$ represented in basis A and a direction $\mathbf{e}$ represented in basis B are *conjugate directions* if they represent the same direction in world coordinates. A direction is just a vector in $\Re^3$, and two directions are the same if they are equal up to a positive scale factor.[1] Formally, $\mathbf{d}, \mathbf{e} \in \Re^3$ are conjugate directions iff $\mathcal{T}_{AB}\ \mathbf{d} = k\mathbf{e}$ for some $k > 0$.

Even if the cameras are moved to different positions by translational motion, conjugate directions will remain unchanged. Conjugate points, however, will change position.

Whenever four or more conjugate directions are known in the two camera bases, $\mathcal{T}_{AB}$ can be determined. This is easy to see if we think of the cameras as sharing the same optical center, in which case the conjugate directions become point correspondences in the two views. In the context of mosaicing, it is well known that by identifying four corresponding points in a pair of images, a projective mapping between the images can be found [Hec89, Sze94]. This projective mapping is just a transformation between the camera bases. Note that there is a precondition for calculating the projective transformation: all three-member subsets of the four conjugate directions must span $\Re^3$.

Each object's fundamental matrix contains a pair of epipoles and these epipoles are conjugate directions. This suggests that whenever four or more moving objects are present, the linear motion dynamic view morphing algorithm of Section 3.4 can be applied.

In practice, the fundamental matrix will only be reliable for objects that are relatively large and that are distinct enough for reliable conjugate points to be determined. Since most objects (e.g., a truck moving down a road) are relatively small in a given view, their fundamental matrices cannot be relied upon. Typically, only the background object yields a reliable fundamental matrix and a reliable conjugate direction.

Furthermore, it would be rare to find a scene with four or more objects that meets the precondition of rigid translational motion and such that the directions of any three of the objects span space.

One strength of using conjugate directions is that they can be determined by means other than fundamental matrices. For instance, if two lines are parallel in space and are viewed through a pinhole camera, the projections of these two lines will intersect at a *vanishing point*. The vanishing point in camera A and that in camera B form a pair of conjugate directions.[2]

It is possible that the two parallel lines are also parallel to the image plane of one of the cameras, in which case the projected lines will not intersect. In this case, *any* vector that is parallel to the projected lines will be in the direction of the vanishing point (and hence can be used as part of a

[1] In projective geometry, when a vector is considered a direction rather than a position in space it is termed a *point at infinity*.

[2] Although directions are vectors in $\Re^3$ and vanishing points are in $\Re^2$, we can think of the vanishing point as lying on the camera's image plane and take its z-coordinate to be 1.

conjugate direction pair). However, the sign of the direction vector will be indeterminate and this fact can cause some problems in an implementation.

Using conjugate directions to help find $\mathcal{T}_{AB}$ is completely natural for dynamic view morphing. First, each object is associated with a fundamental matrix and each fundamental matrix corresponds to a pair of conjugate directions. Even if the fundamental matrix for an object cannot be accurately determined, in many cases a direction of motion can still be extracted for both views, and thus a pair of conjugate directions. For instance, if an object is traveling down a straight road then the vanishing point of the road (i.e., the intersection of the edges of the road) reveals the direction of motion.[3] It may be possible to use the object itself to help find the road's vanishing point. For instance, if the object is a truck or car then a line connecting the bottoms of two same-sided tires will be parallel to the edges of the road and will intersect the vanishing point.

Many scenes contain instances of parallel lines, from whose intersections conjugate direction pairs can be determined. Along with straight roads and bridges, rectangular buildings are common. In theory, from a single rectangular building three conjugate direction pairs can be determined; the parallel edges of the building, as well as rows of windows, can be utilized.

In orthographic views, parallel lines will stay parallel and not intersect. Nonetheless, the directions of these parallel lines will form conjugate direction pairs.

4.2 From Two Conjugate Directions

A more reliable $\mathcal{T}_{AB}$ can be calculated by placing restrictions on the general pinhole camera model. With the right restrictions and perhaps some assistance from the user as suggested below, as few as two or three conjugate directions is sufficient.

The general pinhole camera model allows for cameras that have properties no real camera would normally have.[4] For instance, real cameras will not significantly skew images, nor stretch them in the x or y direction. The principle point will also be close to the image center.[5] Of course, these observations are only true to a certain extent; real cameras do have manufacturing flaws and for applications that need an exact camera model (such as applications that recover depth), it cannot be assumed that the principle point is zero and that the x and y axes are orthonormal.

We have experimented with the following simplified camera model, given below for camera A:

$$\mathcal{T}_{UA} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/f_A \end{bmatrix} \mathcal{R}_A \tag{19}$$

where f_A is the focal length and $\mathcal{R}_A$ is a rotation matrix. For another example of this model, see [Dav98].[6]

To the extent that this simplified camera matrix accurately models the reference cameras, the following theory can be used. If we take $\mathcal{R}_A$ to be $\mathcal{I}$, then world coordinates become associated with camera A and $\mathcal{T}_{AB}$ can be written:

$$\mathcal{T}_{AB} = \mathcal{T}_{UB}\mathcal{T}_{UA}^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/f_B \end{bmatrix} \mathcal{R}_B \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & f_A \end{bmatrix}$$

[3] However, the direction of motion under the fixed-camera assumption is often somewhat different; see Fig. 9.
[4] At least, not if the manufacturer wants to stay in business!
[5] However, if the original images are cropped this may not hold.
[6] This model is used commonly enough that it deserves a name, such as *ideal pinhole camera model.*

There is, however, another world coordinate system W for which $\mathcal{R}_B$ is $\mathcal{I}$, leading to the following:

$$\mathcal{T}_{AU} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & f_A \end{bmatrix}$$

$$\mathcal{T}_{BW} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & f_B \end{bmatrix}$$

If $\mathbf{d}$ and $\mathbf{e}$ are (nonparallel) conjugate directions then the angle between $\mathbf{d}$ and $\mathbf{e}$ is the same in basis U as it is in basis W (because these two bases differ only by a rotation $\mathcal{R}_B$). We can use this fact to relate f_A and f_B, regardless of what $\mathcal{R}_B$ is:

$$\widehat{\mathbf{d}_U} \cdot \widehat{\mathbf{e}_U} = \widehat{\mathbf{d}_W} \cdot \widehat{\mathbf{e}_W}$$

where $\hat{\mathbf{v}} = \mathbf{v}/\|\mathbf{v}\|$ denotes the unit vector operation and $\mathbf{d}_U = \mathcal{T}_{AU}\mathbf{d}_A = (\mathbf{d}_A^x, \mathbf{d}_A^y, f_A)$, etc. This leads to the following nonlinear equation:

$$\frac{\mathbf{d}_A^x \mathbf{e}_A^x + \mathbf{d}_A^y \mathbf{e}_A^y + (f_A)^2}{\sqrt{(\mathbf{d}_A^x)^2 + (\mathbf{d}_A^y)^2 + (f_A)^2}\sqrt{(\mathbf{e}_A^x)^2 + (\mathbf{e}_A^y)^2 + (f_A)^2}} = \frac{\mathbf{d}_B^x \mathbf{e}_B^x + \mathbf{d}_B^y \mathbf{e}_B^y + (f_B)^2}{\sqrt{(\mathbf{d}_B^x)^2 + (\mathbf{d}_B^y)^2 + (f_B)^2}\sqrt{(\mathbf{e}_B^x)^2 + (\mathbf{e}_B^y)^2 + (f_B)^2}} \tag{20}$$

All quantities in Eq. 20 will be known except for f_A and f_B. If camera A and camera B have the same focal length f (which will often happen in practice when a single camera is being used), Eq. 20 leads to a numerical solution for f. The following problem arises, however. If the z-coordinates of the directions are positive, then as $f \to \infty$, the angle between $\mathbf{d}$ and $\mathbf{e}$ in world space goes to 0 and Eq. 20 is satisfied. One solution to this problem is to place an upper bound on $f$corresponding to the largest known focal length that can be used in the application.

When f_A and f_B differ, Eq. 20 still constrains them. Additional information can be used to solve for f_A and f_B. For instance, if a third conjugate direction is known, it leads to two more equations like (20), which over-constrains the solution for f_A and f_B.

Notice that, in the method above, we are not just finding $\mathcal{T}_{AB}$ but also the camera matrices $\mathcal{T}_{UA}$ and $\mathcal{T}_{UB}$. From this information, it is possible to construct a three-dimensional model of the scene (e.g., by using conjugate points). The model will, of course, only be accurate up to a scale factor, and because of the crude camera approximation will not have completely accurate depth values. By guessing a value for f_A, we can instantly solve for f_B (using Eq. 20) and build a wireframe model of the scene (if conjugate line segments are supplied; see Section 7). By means of a slider control, the user could change the value of f_A and watch as new models are produced. Since the user can rapidly interpret the correctness of a model, an f_A that produces the best model could quickly be found. When f_A is incorrect, the models will typically be incomprehensible or wildy distorted.

Can the above process be automated even further? We give two suggestions:

- Real scenes that involve man-made structures and objects typically contain many right angles. An algorithm might search the space of f_A, building models as it goes, and then pick the f_A that produces the most right-angles or near right-angles in the resulting model.

- If three points are colinear in space, then they should be colinear in the resulting model. If the user can identify three conjugate points that correspond to colinear points in the scene, this will provide the algorithm with enough information to solve for f_A and f_B. For instance, identifying three conjugate points on the edge of a building or along a straight road.

Methods similar to the above are plentiful — it just depends on what information is readily available to the algorithm, which in turn depends on the circumstances in which the algorithm is used.

4.3 From Two Fundamental Matrices

Having exact fundamental matrices for just two objects provides sufficient information for finding $\mathcal{T}_{AB}$, provided the objects are not moving parallel to each other. In practice, the following method is extremely susceptible to noisy data; given the general problems with finding fundamental matrices for small objects, we have not experimented with this approach.

Let $\mathcal{F}_0$ and $\mathcal{F}_1$ be the fundamental matrices for two moving objects. Let $\mathbf{v}_0$ and $\mathbf{v}_1$ denote the direction of motion for these objects. Recall that

$$\mathcal{F}_i = [(\mathbf{v}_i)_B]_\times \mathcal{T}_{AB}$$

$\mathcal{T}_{AB}$ is a rank three invertible matrix, but $[(\mathbf{v}_i)_B]_\times$ is rank two, and consequently $\mathcal{F}_i$ is also rank two. Because of the rank deficiency in $[(\mathbf{v}_i)_B]_\times$, the following arises:

Let $S_i = \{\mathcal{M} \in \Re^{3\times 3} : \mathcal{F}_i = [(\mathbf{v}_i)_B]_\times \mathcal{M}\}$. Then S_i is a four-dimensional vector space over the real numbers. A basis for S_i is given by the matrices $\mathbf{b}_0^i$, $\mathbf{b}_1^i$, $\mathbf{b}_2^i$, and $\mathbf{b}_3^i$, where

$$\mathbf{b}_0^i = \mathcal{T}_{AB}, \mathbf{b}_1^i = [(\mathbf{v}_i)_B, 0, 0], \mathbf{b}_2^i = [0, (\mathbf{v}_i)_B, 0], \mathbf{b}_3^i = [0, 0, (\mathbf{v}_i)_B].$$

Because $\mathcal{T}_{AB}$ is in the basis of both S_0 and S_1 and because $\mathbf{v}_0$ and $\mathbf{v}_1$ are not parallel, $S_0 \cap S_1 = < \mathcal{T}_{AB} >$, where $< \cdot >$ denotes the subspace generated by a set of vectors. Since we only need to find $\mathcal{T}_{AB}$ up to a scalar, we only need to find one nonzero element in the intersection of S_0 and S_1. This is accomplished by first finding *any* two matrices $\mathbf{b}_4^i$ such that

$$\mathcal{F}_i = [(\mathbf{v}_i)_B]_\times \mathbf{b}_4^i. \tag{21}$$

Next, notice that S_i is spanned by $\mathbf{b}_1^i$, $\mathbf{b}_2^i$, $\mathbf{b}_3^i$, and $\mathbf{b}_4^i$ (because if $\mathbf{b}_4^i$ is in $< \mathbf{b}_1^i, \mathbf{b}_2^i, \mathbf{b}_3^i >$, then $[(\mathbf{v}_i)_B]_\times \mathbf{b}_4^i = 0$). Consequently, there exist scalars $k_1, ..., k_8$ such that

$$k_8 \mathcal{T}_{AB} = -k_1 \mathbf{b}_1^0 - k_2 \mathbf{b}_2^0 - k_3 \mathbf{b}_3^0 + \mathbf{b}_4^0 = k_4 \mathbf{b}_1^1 + k_5 \mathbf{b}_2^1 + k_6 \mathbf{b}_3^1 + k_7 \mathbf{b}_4^1 \tag{22}$$

The second equality means that

$$[\mathbf{b}_1^0\ \mathbf{b}_2^0\ \mathbf{b}_3^0\ \mathbf{b}_1^1\ \mathbf{b}_2^1\ \mathbf{b}_3^1\ \mathbf{b}_4^1][k_1\ k_2\ k_3\ k_4\ k_5\ k_6\ k_7]^+ = \mathbf{b}_4^0 \tag{23}$$

Here we treat the matrices $\mathbf{b}_j^i$ as column vectors in $\Re^9$. The above can be solved using standard techniques from linear algebra. Once the k_i's are found, we can find $\mathcal{T}_{AB}$ (up to a scalar) using Eq. 22.

Formally, we must show that the left-most matrix in Eq. 23 has rank 7. The vectors $\mathbf{b}_1^0$, $\mathbf{b}_2^0$, $\mathbf{b}_3^0$, $\mathbf{b}_1^1$, $\mathbf{b}_2^1$, $\mathbf{b}_3^1$ clearly form a linearly independent set because $\mathbf{v}_0$ and $\mathbf{v}_1$ are not parallel. If $\mathbf{b}_4^1 = h_1\mathbf{b}_1^0 + h_2\mathbf{b}_2^0 + h_3\mathbf{b}_3^0 + h_4\mathbf{b}_1^1 + h_5\mathbf{b}_2^1 + h_6\mathbf{b}_3^1$ for some scalars h_i, then by Eq. 21, $\mathcal{F}_1 = [h_1\mathbf{v}_2, h_2\mathbf{v}_2, h_3\mathbf{v}_2]$ where $\mathbf{v}_2 = \mathbf{v}_1 \otimes \mathbf{v}_0$. This is a contradiction since $\mathcal{F}_1$ has rank 2, not rank 1.

4.4 From Three Conjugate Directions

Finally, we present a simple, approximate method that produces good results for three moving objects.

Assume that three linearly independent conjugate directions $\mathbf{v}^0$, $\mathbf{v}^1$, and $\mathbf{v}^2$ have been identified in the two views, and assume they have been normalized to unit length. We have

$$\mathcal{T}_{AB}\mathbf{v}_A^i = k_i\mathbf{v}_B^i \tag{24}$$

for some scalars k_i. Consequently, $\mathcal{T}_{AB}$ has the form:

$$\mathcal{T}_{AB} = [k_0\mathbf{v}_B^0, k_1\mathbf{v}_B^1, k_2\mathbf{v}_B^2][\mathbf{v}_A^0, \mathbf{v}_A^1, \mathbf{v}_A^2]^{-1} \tag{25}$$

If a fourth conjugate direction $\mathbf{v}^3$ (linearly independent from the first three) happened to be known, then the equation above could be used directly to find $\mathcal{T}_{AB}$: First, since we are only interested in determining $\mathcal{T}_{AB}$ up to a scalar, notice that $\mathbf{v}_B^3 = \mathcal{T}_{AB}\mathbf{v}_A^3 = [k_0\mathbf{v}_B^0, k_1\mathbf{v}_B^1, k_2\mathbf{v}_B^2][\mathbf{v}_A^0, \mathbf{v}_A^1, \mathbf{v}_A^2]^{-1}\mathbf{v}_A^3$. Thus, $\mathbf{v}_B^3 = k_0\mathbf{u}_0 + k_1\mathbf{u}_1 + k_2\mathbf{u}_2$ for some known $\mathbf{u}_i$ and unknown k_i. This can be easily solved for the k_i.

Of course, if four conjugate directions are known, then the earlier technique of Section 4.1 can be used. However, the technique just presented might be more applicable if three reliable conjugate directions are known along with several less reliable ones.

Returning to Eq. 25, since it is only necessary to find $\mathcal{T}_{AB}$ up to a scalar, we can assume $k_0 = 1$. We approximate $\mathcal{T}_{AB}$ by also taking k_1 and k_2 to be 1:

$$\mathcal{T}_{AB} = [\mathbf{v}_B^0, \mathbf{v}_B^1, \mathbf{v}_B^2][\mathbf{v}_A^0, \mathbf{v}_A^1, \mathbf{v}_A^2]^{-1} \tag{26}$$

The above approximation works well in many cases. First, it works without error if $\mathcal{T}_{AB}$ is a rotation matrix. This happens, for instance, when camera A and camera B have identical camera matrices (i.e., when $\mathcal{T}_{AB}$ is $\mathcal{I}$). If the cameras are like those described by Eq. 19, then $\mathcal{T}_{AB}$ is also a rotation matrix when camera B equals camera A rotated around its z-axis. If the relationship between camera A and camera B is close to either of these situations (e.g., if camera A and camera B are almost identical) then Eq. 26 represents a good approximation.

The latter case happens more often than might be expected. This is due to two restrictions on static view morphing. First, the epipole (i.e., vanishing point) must not be visible in either view.[7] Second, there must be significant overlap in each view of each object, so that conjugate points can be determined. Consequently, the z axes of the two cameras tend to point in much the same direction (assuming the principal points are near the image centers).

Most importantly, the approximation given by Eq. 26 has a strength that can lead to good results regardless of how close it is to the true $\mathcal{T}_{AB}$. Assume the three conjugate directions $\mathbf{v}^i$ are determined from the apparent direction of motion for three objects before the fixed-camera formulation is made (i.e., if a vehicle is seen traveling down a straight road, then the direction for the vehicle is assumed to be the vanishing point for the road). Then the discussion in Section 3.6.1 is applicable and each object will be portrayed in the virtual views as continuing to travel in its apparent direction of motion (e.g., the vehicle will stay on the road). The motion may not be truly linear, but it will be close except in extreme cases.

Sequence (iii) of Fig. 14 was created using the approximation presented in this section. It is left for future work to quantify the statements given above, both in terms of theoretical error analysis and experimental trials on a wide variety of test cases.

5 Implementation Issues

So far, the results of this paper have applied to various special cases. In this section we combine the special-case results into a single algorithm for an arbitrary scenario. We then discuss several other

[7] This condition is eliminated, however, by environment map morphing

implementation issues and conclude by extending the methods to incorporate more than two reference views. The discussion is for pinhole cameras; see Section 3.7 for orthographic cameras.

5.1 General Algorithm

We assume that two views of a scene are given. For each object in the scene, the total amount of movement evident in the views must be equivalent to a rigid translation. We assume that a set of conjugate points has been determined for each object, and that the views have been segmented into layers, with each layer corresponding to an object.

(Phase I) Determine if any object is stationary relative to the fixed-camera frame.

If an object is stationary, then its conjugate points are also conjugate directions. Recall that knowledge of four conjugate directions is sufficient to determine $\mathcal{T}_{AB}$. Since many more than four conjugate points are provided, a RANSAC method [FB81] could be used to automatically calculate $\mathcal{T}_{AB}$: $\mathcal{T}_{AB}$ is repeatedly estimated from random subsets of four conjugate directions (or a similar small number) and each estimate is tested against the remaining conjugate directions. If some estimated $\mathcal{T}_{AB}$ represents a sufficiently perfect transformation between all the conjugate directions, then it can be deduced that the object has no translation relative to the cameras. When this occurs, a significant benefit is that $\mathcal{T}_{AB}$ has been determined also. Once $\mathcal{T}_{AB}$ has been found, it can be used to test whether other layers are stationary or to find the fundamental matrix for other layers through Eq. 3.

For stationary layers, a panoramic mosaicing technique can be used to create the view interpolation. First, using $\mathcal{T}_{AB}$ to incorporate the view from B with the view from A, an environment map [Gre86] is made relative to basis A. Then, during the process of creating a virtual view, the transformation $\mathcal{T}_{AV}$ is used to convert the environment map to basis V of the virtual camera. $\mathcal{T}_{AV}$ can be found by multiplying Eq. 10 on the right by $\mathcal{T}_{AB}$ yielding

$$\mathcal{T}_{AV} = (1-s)\mathcal{I} + s\lambda\mathcal{T}_{AB} \tag{27}$$

Notice that this equation relies only on known quantities. The transformed environment map is used to fill in sections of the virtual view in accordance with the depth-ordering and transparency values of the layers.

(Phase II) View-interpolate the non-stationary objects.

This phase can be complicated. If $\mathcal{T}_{AB}$ and the direction of motion for each object can be determined, then the linear motion algorithm of Section 3.4 can be used. Knowledge of $\mathcal{T}_{AB}$ can be useful in finding the direction of motion for each object (by helping to find the fundamental matrix for each object). Alternatively, knowledge of four or more conjugate directions (such as conjugate motion directions) can be used to find $\mathcal{T}_{AB}$.

More typically, $\mathcal{T}_{AB}$ cannot be conveniently found and the special case methods will apply. If the scene has only one non-stationary object, or has several objects all moving parallel to each other, the method of Section 3.5 can be used. If the motion directions for the various objects cannot be determined automatically, it may take user interaction to choose this case. Alternatively, if a single fundamental matrix is consistent with all conjugate points for all the non-stationary objects then the parallel motion case applies.

If the scene has just two non-stationary objects and they are not moving parallel to each other (i.e., they do not have the same fundamental matrix), then the planar parallel motion case of Section 3.6 can be applied. This case can also be used for more than two non-stationary objects if it can be determined that they all move parallel to the same plane.

In typical applications the original reference cameras are not co-centered and so there is a large, non-stationary background object. A reliable fundamental matrix can be determined for this large object, and from the fundamental matrix a conjugate direction. If one or more additional conjugate directions can be found, either from other moving objects or from parallel lines visible in the views, then the methods in Section 4 can be used to estimate $\mathcal{T}_{AB}$. The resulting virtual motion will be linear, which can make this approach preferable to the planar parallel algorithm.

If the scene has three or more non-stationary objects and the planar parallel motion case does not apply, then it should be possible to determine $\mathcal{T}_{AB}$ using the methods of Section 4. That is, either there should be enough distinct features in the scene or enough parallel lines (such as road edges and building edges) to determine $\mathcal{T}_{AB}$, or else the scene is so indistinct that it will not matter because the correctness of the virtual motion will be judged in terms of reference elements in the scene. For example, a car driving down a straight road should appear to stay on the road. If there is no road evident, then there is no way to judge the correctness of the virtual motion for the car.

5.2 Uncertainty Channel

When an object in a scene moves, what was once hidden behind the object from a particular viewpoint becomes visible. Unfortunately, in working from still images, the object initially obstructs from view what is behind it, leaving a major problem of how to fill in the unseen areas that get revealed during a view interpolation of a dynamic scene.

Since interpolation involves two views, the second view often contains information about what was hidden behind an object in the first view, especially when the reference views are widely separated. To make use of this information, we use an *uncertainty channel* similar to using an alpha channel for transparency. The uncertainty channel for each pixel on each layer of each view stores whether the pixel's color and transparency are known quantities or not. A pixel's color might be known, for instance, because it is visible in the view. Pixels that are unknown are those hidden behind moving objects. For pixels along the edge of an object, a partial uncertainty can be stored. The uncertainty channel is basically the same as the object mask used to segment each view into object layers; its interpretation is different, however. Note that all pixels outside the boundary of a view should be considered completely uncertain.

During the morphing process, the uncertainty channel is morphed like the other channels, but the blending of uncertainty values is performed differently and can affect the blending of the other channels. For instance, when determining the color for a pixel in the virtual view, if the source color from image A is marked 0% certain and the source color from image B is marked 100% certain, the virtual pixel *color* should be made exactly equal to the source color from image B. The virtual pixel *uncertainty value* is related to the interpolation factor: If the interpolation factor is near 1, then the interpolated image is close to image B and the uncertainty value should be close to that for image B also (i.e., 100% certain). If the interpolation factor is close to 0, the new uncertainty value should be close to that for image A (i.e., 0% certain).

5.3 Using More than Two Reference Views

As presented, view interpolation uses two reference views of a scene to create a third, physically-correct view. When three reference views are available, view interpolation can be applied in a pairwise manner to make use of all three reference views: An interpolation can be performed on two of the views, and the resulting virtual view can be interpolated with the third reference view. This process can be continued to incorporate an arbitrary number of reference views.

If the relative camera calibration between each pair of reference views is known, then the repeated

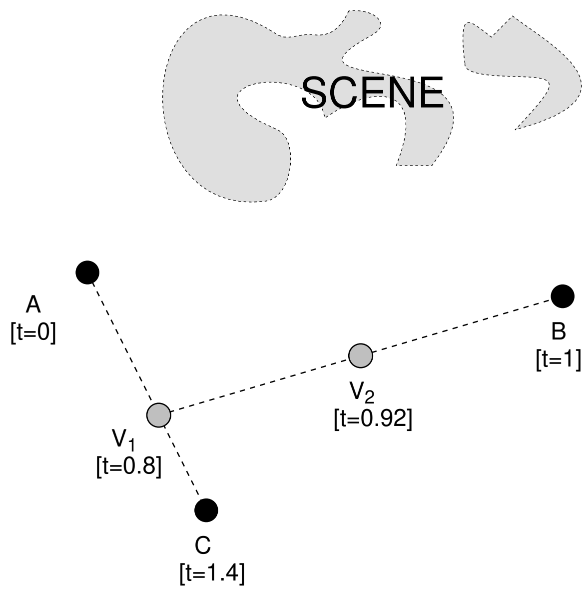

Figure 12: Virtual views can be created from multiple reference views by combining them in a pairwise manner. Here the reference views are A, B, and C, and the pairwise relative camera calibrations are known.

pairwise interpolation will always portray linear virtual motion (Fig. 12). The order in which the pairwise interpolations are performed will not matter, and the final virtual image will represent a "weighted sum" of reference-view differences analogous to a weighted sum of Euclidean vectors (Fig. 13).

Note that in general, except when linear virtual motion can be achieved, the order in which the pairwise interpolations are performed will determine the final virtual view (i.e., the process is not commutative).

6 Applications

Filling a gap in a movie. There are many ways for a movie of a scene to have a missing section, as the following examples demonstrate:

- The camera may have been pointed the wrong way for a period of time, perhaps because it was tracking something else.
- The camera may have undergone severe shaking (e.g., due to turbulence if the camera was in an airplane).
- The camera's view may have been temporarily obstructed, or the lens of the camera may have been quickly changed during filming.
- A segment of film may have been edited and lost, or physically damaged by storage conditions or poor manufacturing.

When such a film is viewed, the missing gap will represent a disorienting jump-cut for the audience. By using the two frames that bound the gap as reference views, the techniques of this paper can be

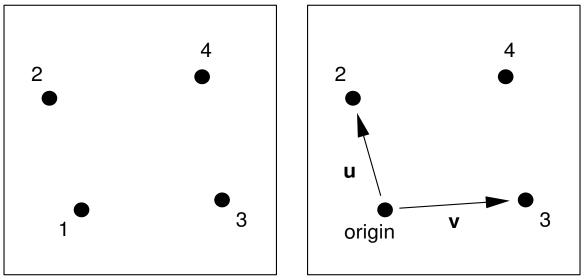

Figure 13: Reference view differences. The left-hand figure shows four points on a plane. By formally defining **u** to be the *difference* between point 2 and point 1, formally defining **v** to be the difference between point 3 and point 1, and formally taking point 1 to be the *origin*, point 4 can be represented as a linear combination of the quantities **u** and **v**. The same development exists if the points represent reference views acquired at various positions and times, assuming that linear motion can be achieved. Thus we can create a vector space of virtual views out of the purely formal concept of *reference view differences*. Note that each virtual view will have a time component as well as a space component.

used to create a smooth transition in the film, provided that the original dynamic scene meets the required conditions of translational motion. Thus the audience will better understand how the two reference viewpoints are related spacially and temporally.

Camera hand-off. Two cameras may be viewing a dynamic scene, and it may be desired to switch from one camera view to the other. Dynamic view morphing can create a smooth bridge between the views, continuing to portray the scene motion during the interpolation.

Creating virtual views. Since view interpolation involves creating a continuous sequence of virtual views between two reference views, the process can be frozen at any of the individual virtual views.

Removing obstructions or moving objects. Any layer can simply be left out during the view interpolation process.

Projecting motion into future or past. The factor of interpolation can be any real number. Thus a dynamic scene can be interpolated into the future or the past; the final sequence is not restricted to the time between when the reference views were captured. This could be used to estimate the position of a moving object that has become obstructed; for instance, a vehicle that has driven behind a building or a stand of trees, or into a tunnel. It can also be used to predict the future position of an object that is being tracked.

A complication arises in using a morph algorithm to fill in the remaining pixels since the virtual view is no longer between two reference views, and consequently it is unclear how the blending step should proceed. Assuming a lambertian scene and a dense correspondence, it would only be necessary to use the colors from one of the reference views (except for obstructed areas) to create any virtual view, and no blending step would be needed. Without these conditions, the nearest reference view can be used to provide the source colors; the morphing step then becomes a "warping" step.

Stabilization. A severely jittery segment of film can be replaced by a dynamic view morph.

Compression. A film could be condensed to a series of keyframes, and then reconstructed by using dynamic view morphing to link the keyframes.

Creating movies from still images. A series of still images of a dynamic scene might be taken from different locations and times. Dynamic view morphing can then interpolate between the still images

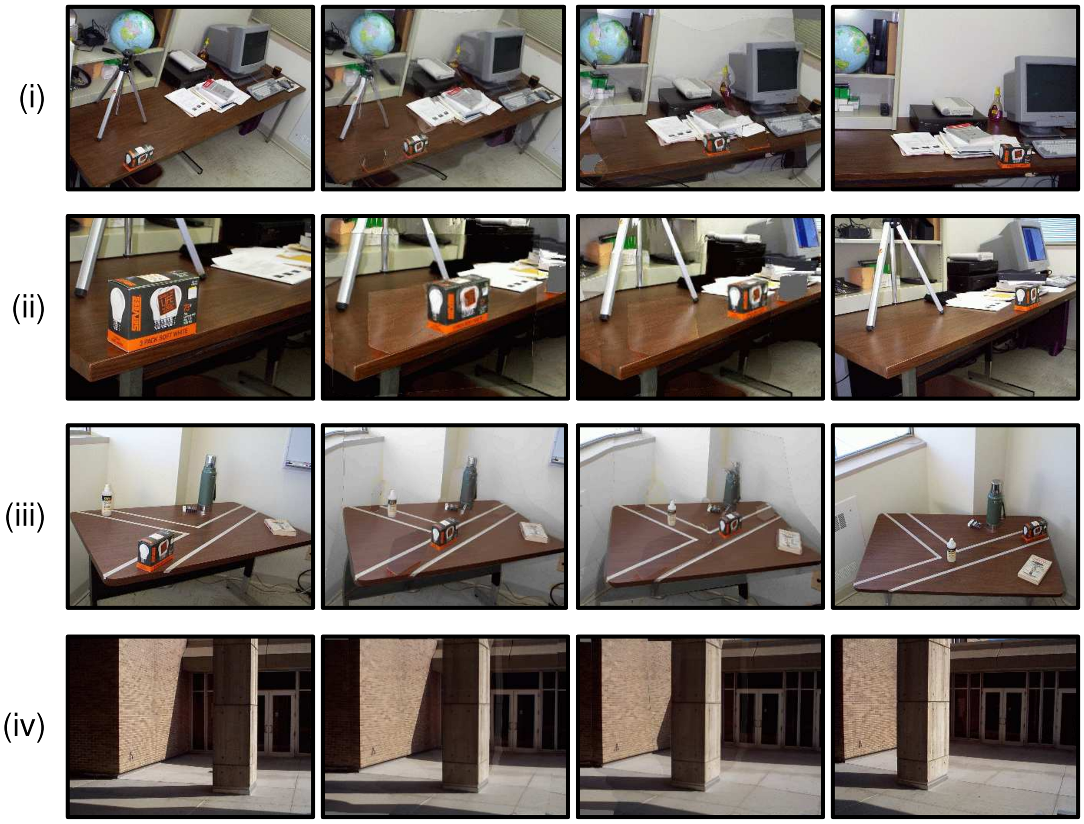

Figure 14: Experimental results.

to create a movie showing the relationship between the original viewpoints. Similar techniques have been used for artistic effect in the mass media [GB95].

Tracking from a single fixed camera: If views are captured by a single camera whose position, orientation, and internal parameters all stay fixed, then the camera-to-camera transformation $\mathcal{T}_{AB}$ between any pair of views is just $\mathcal{I}$. Elements of the scene that are static will be unchanging in the views; this helps to identify moving objects in the scene. For any object in the scene that moves at constant velocity along a straight-line path, the theory presented in Section 3.3 can be used to calculate the future or past position of the object. Dynamic view morphing can also be used to portray the object during periods of occlusion, when it would otherwise not be seen in the view. These techniques are particularly applicable when a wide-field-of-view pinhole camera such as OmniCamera [Nay97] is used to acquire the views. This is because, even when such a camera is completely unchanging, it still acquires much of the available scene information. Two identical, wide-field-of-view cameras placed back-to-back could conceivable acquire all of the visual scene information available from a single position in space.

7 Experimental Results

We have tested the concepts of this paper on a variety of scenarios. Fig. 14 shows the results of four tests, each as a series of still frames from a dynamic 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; for example, Fig. 8 shows the layers for sequence (iii). 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. A line-segment correspondence consists of two pairs of endpoints, which are conjugate points, and a pair of line segments, each of which runs along a continuous straight edge or over a continuous flat surface in the appropriate layer. For each sequence, between 30 and 50 line-segment correspondences were used (counting every layer).

For all the sequences, the camera calibration was completely unknown and the focal lengths were different. Except in sequence (ii), the reference cameras were widely separated in space.

Sequence (i): This sequence involves two moving objects and was created using the planar parallel motion algorithm (Section 3.6). The first moving object is the box in the foreground. Since the reference cameras were located at different locations and the remaining elements of the scene were static, the second object consists of everything in the scene except for the moving box. If desired, this single large "background" object could have been subdivided into a series of smaller objects, each on its own layer. This extra layering might have been used to increase realism, for instance.

Since the moving box is flush with the edge of the table in both reference views, it is natural to portray the box as sliding along the edge of the table during the missing time interval, as is shown in the middle two frames. The discussion of Section 3.6.1 explains how the virtual box appears to move along the table's edge even though $\mathcal{T}_{AB}$ was not calculated.

Notice in this example that the reference views are very different: the cameras are widely spaced (relative to the scene's size), different focal lengths were used, and the left reference view contains many scene elements not visible in the right view. Consequently, the tripod, only visible in the first reference view, appears to dissolve during the course of the sequence. The globe and the bookshelf hold together fairly well, however. We are developing tools for improving the appearance of regions that are only visible in one reference view.

Sequence (ii): This sequence is similar to sequence (i). However, the first reference view is a close-up of the box, and the second was created by moving the camera slightly and zooming out. The resulting dynamic view interpolation shows the virtual camera zooming-out continuously while the box slides along the edge of the table.

Although the uncertainty channel technique (see Section 5.2) was used in all of the experiments, this sequence presents a good demonstration of it. In the first reference view, the box covers a large portion of the background. When the box moves this region must be filled in, which is possible because the obstructed area is visible in the second reference view. The uncertainty channel technique was used to automatically fill-in the obstructed region. The filled-in area is visible as a light patch in the middle two frames. This is because the surface of the table is reflective, causing the table to be brighter in the right-hand reference view than in the left-hand one because the camera was in a different position. Since the missing region is filled using information from the right-hand view, the patch is initially brighter than the surrounding table surface, but gradually blends in as the sequence progresses.

Also notice how the edges of the left-hand reference view are visible in the middle frames as the first reference view is slowly deformed to fit into the second. Using the uncertainty channel, areas outside the first reference view are filled in entirely from the second view.

Sequence (iii): Here is a three-moving-object sequence: The background is one object, while the box and the bottle are the other two. Tape has been placed on a table surface to outline "roads," and it is desired that the box and the bottle should travel straight down their respective roads. The sequence shows this to be the case.

The two reference views were taken from widely separated cameras with different focal lengths. The top of the tall thermos bottle is distorted during the interpolation through a violation of the Copernican assumption: although it is visibly near an edge on the wall behind it, the thermos and the wall are completely unrelated and should not be influencing each other. This problem can be solved by subdividing the background object into separate layers, with unrelated objects on different layers.

The sequence was created by approximating $\mathcal{T}_{AB}$ through the technique of Section 4.4. Two of the conjugate directions were derived from the vanishing points of the "roads" and the third from the fundamental matrix of the background. Note that the edges of the tape roads were placed on the table by hand and no steps were taken to guarantee parallelism.

Sequence (iv): This sequence shows a view interpolation for a static scene that violates the monotonicity assumption discussed in [SD96, Sei97]. The monotonicity assumption was circumvented by the use of layering: the pillar in the foreground was placed on one layer and the remaining scene elements formed the background layer. Basically, the scene contains one large moving object which gets subdivided into two objects that both move in the same direction. Hence the parallel motion algorithm of Section 3.5 was applicable.

8 Conclusion

We have presented 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 view and the second 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 accurate. With more conditions on the reference cameras, the virtual 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 that 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 distinct fundamental matrices, the camera-to-camera transformation can be determined.

References

[AS97] Shai Avidan and Amnon Shashua. Novel view synthesis in tensor space. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 1034–1040, 1997.

[BCS97] Christopher Bregler, Michele Covell, and Malcolm Slaney. Video rewrite: Driving visual specch with audio. In *Proc. SIGGRAPH 97*, pages 353–360, 1997.

[BN92] Thaddeus Beier and Shawn Neely. Feature-based image metamorphosis. In *Proc. SIGGRAPH 92*, pages 35–42, 1992.

[Che95] Shenchang Eric Chen. Quicktime VR — An image-based approach to virtual environment navigation. In *Proc. SIGGRAPH 95*, pages 29–38, 1995.

[CW93] Shenchang Eric Chen and Lance Williams. View interpolation for image synthesis. In *Proc. SIGGRAPH 93*, pages 279–288, 1993.

[Dav98] James Davis. Mosaics of scenes with moving objects. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 354–360, 1998.

[FB81] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, June 1981.

[GB95] Michel Gondry and Pierre Buffin. Like a rolling stone. Music Video, 1995.

[Gre86] Ned Greene. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications*, 6(11):21–29, November 1986.

[Hec89] Paul S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, University of California, Dept. CS, Berkeley, CA, May 1989.

[Hor90] B. K. P. Horn. Relative orientation. *Int. J. Computer Vision*, 4:59–78, January 1990.

[IAH95] Michal Irani, P. Anandan, and Steve Hsu. Mosaic based representations of video sequences and their applications. In *Proc. Fifth Int. Conf. on Computer Vision*, pages 605–611, 1995.

[Int97] Interactive Pictures Corporation, Inc. IPIX, version 1.0, 1997.

[MB95] Leonard McMillan and Gary Bishop. Plenoptic modeling. In *Proc. SIGGRAPH 95*, pages 39–46, 1995.

[MD98] Russell A. Manning and Charles R. Dyer. Environment map morphing. Technical report, Computer Sciences Department, University of Wisconsin-Madison, 1998. To appear.

[Nay97] Shree K. Nayar. Catadioptric omnidirectional camera. In *Proc. Computer Vision and Pattern Recognition Conf.*, pages 482–488, 1997.

[RPFRA98] B. Rousso, S. Peleg, I. Finci, and A. Rav-Acha. Universal mosaicing using pipe projection. In *Proc. 6th Int. Conf. on Computer Vision*, pages 945–952, 1998.

[SD96] Steven M. Seitz and Charles R. Dyer. View morphing. In *Proc. SIGGRAPH 96*, pages 21–30, 1996.

[SD97] Steven M. Seitz and Charles R. Dyer. Photorealistic scene reconstruction by voxel coloring. In *Proc. Image Understanding Workshop*, pages 1067–1073, 1997.

[Sei97] Steven M. Seitz. *Image-Based Transformation of Viewpoint and Scene Appearance*. PhD thesis, University of Wisconsin, Madison, WI, 1997.

[SS97] Richard Szeliski and Heung-Yeung Shum. Creating full view panoramic image mosaics and environment maps. In *Proc. SIGGRAPH 97*, pages 251–258, 1997.

[Sze94] Richard Szeliski. Image mosaicing for tele-reality applications. In *Proc. Workshop Applications of Computer Vision*, pages 44–53, 1994.

[Sze96] Richard Szeliski. Video mosaics for virtual environments. *IEEE Computer Graphics and Applications*, 16(2):22–30, 1996.

[TK92] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: A factorization method. *Int. J. of Computer Vision*, 9(2):137–154, 1992.

[TK96] Jay Torborg and James T. Kajiya. Talisman: Commodity realtime 3D graphics for the PC. In *Proc. SIGGRAPH 96*, pages 353–363, 1996.

[WA94] John Y. A. Wang and Edward H. Adelson. Representing moving images with layers. *IEEE Trans. Image Processing*, 3(5):625–638, 1994.

[ZDFL95] Z. Zhang, R. Deriche, O. Faugeras, and Q.-T. Luong. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artificial Intelligence*, 78:87–119, 1995.