Spatio-Temporal View Interpolation

Sundar Vedula, Simon Baker, and Takeo Kanade

CMU-RI-TR-01-35

The Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

September 2001

Abstract

We propose an algorithm for creating novel views of a non-rigidly varying dynamic event by combining images captured from different positions, at different times. The algorithm operates by combining images captured across space and time to compute voxel models of the scene shape at each time instant, and dense 3D scene flow between the voxel models (the non-rigid motion of every point in the scene). To interpolate in time the voxel models are "flowed" using the appropriate scene flow and a smooth surface fit to the result. The novel image is then computed by ray-casting to the surface at the intermediate time, following the scene flow to the neighboring time instants, projecting into the input images at those times, and finally blending the results. We use the algorithm to create re-timed slow-motion fly-by movies of real-world events.

Keywords: Image Based Rendering, View Synthesis, Scene Flow, 3D Modeling

Contents

List of Figures

1 Introduction

We describe an algorithm for interpolating images of a non-rigidly varying dynamic event across space and time. While there has been a large amount of research on image-based interpolation of static scenes across space (see, for example, [Chen and Williams, 1993, Seitz and Dyer, 1996, Gortler *et al.*, 1996, Levoy and Hanrahan, 1996, Sato *et al.*, 1997, Narayanan *et al.*, 1998]), there has been almost no research on re-rendering a dynamic event across time. What work there has been has assumed a very restricted motion model. Either the event consists of rigidly moving objects [Manning and Dyer, 1999] or point features moving along straight lines with constant velocity [Wexler and Shashua, 2000]. Our algorithm is applicable to completely non-rigid events and uses no scene or object specific models.

Figure 1 presents an illustrative example of this task which we call *Spatio-Temporal View Interpolation.* The figure contains 4 images captured by 2 cameras at 2 different time instants. The images on the left are captured by camera C_1, those on the right by camera C_2. The bottom 2 images are captured at the first time instant and the top 2 at the second. Spatio-temporal view interpolation consists of combining these 4 views into a novel image of the event at an arbitrary viewpoint and time. Although we have described spatio-temporal view interpolation in terms of 2 images taken at 2 time instants, our algorithm applies to an arbitrary number of images taken from an arbitrary collection of cameras spread over an extended period of time.

Our algorithm is based on the explicit recovery of 3D scene properties. We use the 3D *voxel coloring* algorithm [Seitz and Dyer, 1999] to recover a voxel model of the scene at each time instant, and a 3D *scene flow* algorithm [Vedula *et al.*, 1999] to recover the non-rigid motion of the scene between consecutive time instants. The voxel models and scene flow then form part of the input to our algorithm.

To generate a novel image at an intermediate viewpoint and time, the 3D voxel models at the neighboring times are first "flowed" to estimate an interpolated scene shape at that time. After a smooth surface has been fit to the flowed voxel model, the novel image is generated by ray casting. Rays are projected into the scene and intersected with the interpolated scene shape. The points at which these rays intersect the surface are used to find the corresponding points at the neighboring times by following the scene flow forwards and backwards. The known geometry of the scene at those times is then used to project the corresponding points into the input images. The input images are sampled at the appropriate locations and the results blended to generate the novel image at the intermediate space and time.

2 Inputs to the Algorithm

2.1 Explicit 3D Models Vs. Correspondences

To generate novel views of the event we need to know how the pixels in the input images are geometrically related to each other. In the various approaches to imaged-based

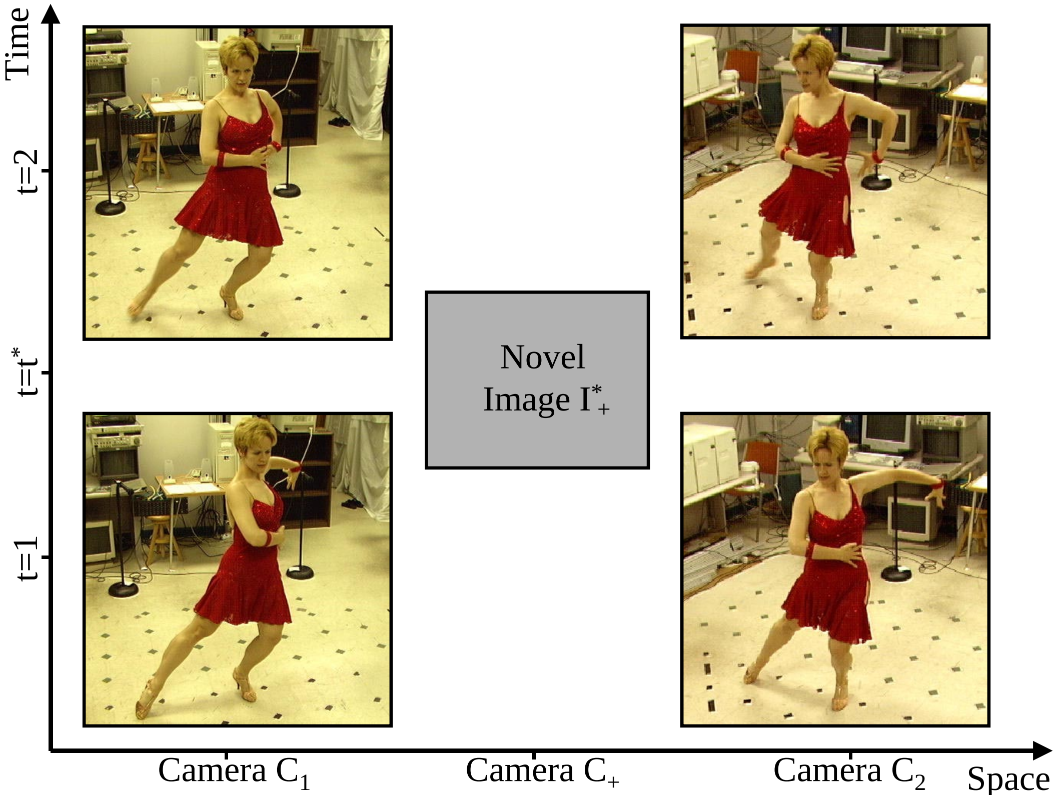


Figure 1: Spatio-temporal view interpolation consists of taking a collection of images of an event captured with multiple cameras at different times and re-rendering the event at an arbitrary viewpoint and time. In this illustrative figure, the 2 images on the left are captured with the same camera at 2 different times, and the 2 images on the right with a different camera at the same 2 time instants. The novel image and time are shown as halfway between the cameras and time instants but are arbitrary in our algorithm.

interpolation of static scenes across space, there are 2 common ways in which this information is provided. First, there are algorithms that use *implicit* geometric information in the form of *feature correspondences* [Chen and Williams, 1993, Seitz and Dyer, 1996]. Second, there are approaches which use *explicit* 3D models of the scene [Sato *et al.*, 1997, Narayanan *et al.*, 1998]. (Note that these references are meant to be representative rather than comprehensive.)

We decided to base our algorithm on explicit 3D models of the scene. The primary reason for this decision is that we would like our algorithms to be fully automatic. The correspondences that are used in implicit rendering algorithms are generally specified by hand. While hand-marking (sparse) correspondences might be possible in a pair of images, it becomes an enormous task when images of a dynamic sequence are captured over time, and from a multitude of viewing directions.

The relationship between pixels *across time* can be described by how the points on the surface of the scene move across time. Assuming that the scene can move in an arbitrarily

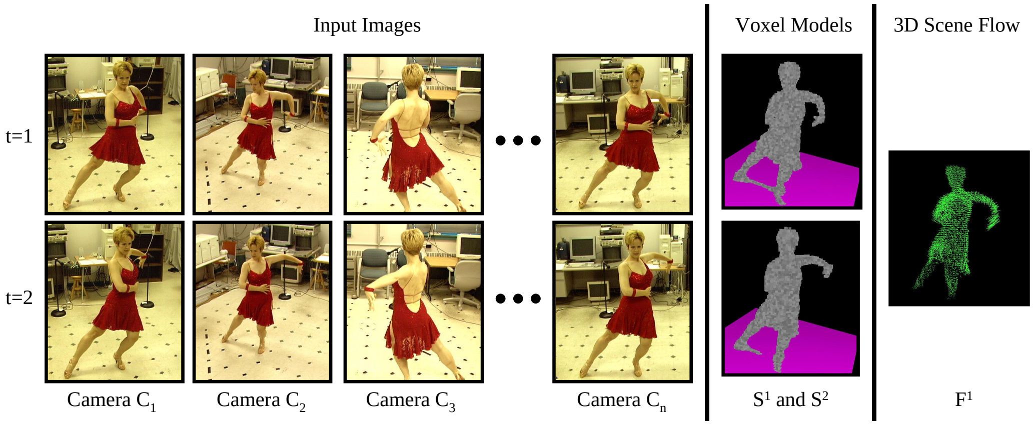


Figure 2: The input to the spatio-temporal view interpolation algorithm is a set of calibrated images at 2 or more consecutive time instants. From these input images, 3D voxel models are computed at each time instant using the voxel coloring algorithm [Seitz and Dyer, 1999]. We then compute the dense non-rigid 3D motion of points between the models using a scene flow algorithm [Vedula *et al.*, 1999].

non-rigid way, the 3D motion of points is the *scene flow* [Vedula *et al.*, 1999]. We use the combination of scene shape (represented as 3D voxel models) and 3D scene flow to relate the pixels in the input images.

2.2 3D Voxel Models and 3D Scene Flow

Denote the time-varying scene S^t where $t = 1, \ldots, T$ is a set of time instants. Suppose that the scene is imaged by N fully calibrated cameras with synchronized shutters. The input to the algorithm is the set of images I_i^t captured by cameras C_i, where $i = 1, \ldots, N$ and $t = 1, \ldots, T$. (See Figure 2 for an example set of input images.) We compute a 3D voxel model of the scene from these images:

$$S^t \;=\; \{X_i^t \mid i = 1, \ldots, V^t\} \tag{1}$$

for $t = 1, \ldots, N$ and where $X_i^t = (x_i^t, y_i^t, z_i^t)$ is one of the V^t surface voxels at time t. We compute the set of surface voxels S^t at each time instant t independently using a *voxel coloring* algorithm [Seitz and Dyer, 1999]. Figure 2 illustrates the voxel models computed for $t = 1$ and $t = 2$.

The scene flow of a voxel describes how it moves from time t to time $t + 1$. If the 3D voxel $X_i^t = (x_i^t, y_i^t, z_i^t)$ at time t moves to:

$$X_i^t + F_i^t \;=\; (x_i^t + f_i^t, y_i^t + g_i^t, z_i^t + h_i^t) \tag{2}$$

at time $t + 1$ its scene flow at time t is then $F_i^t = (f_i^t, g_i^t, h_i^t)$. We compute the scene flow F_i^t for every voxel in the model S^t at each time t using the algorithm [Vedula *et al.*, 1999]. Figure 2 contains the result of computing the scene flow from $t = 1$ to $t = 2$. The inputs to

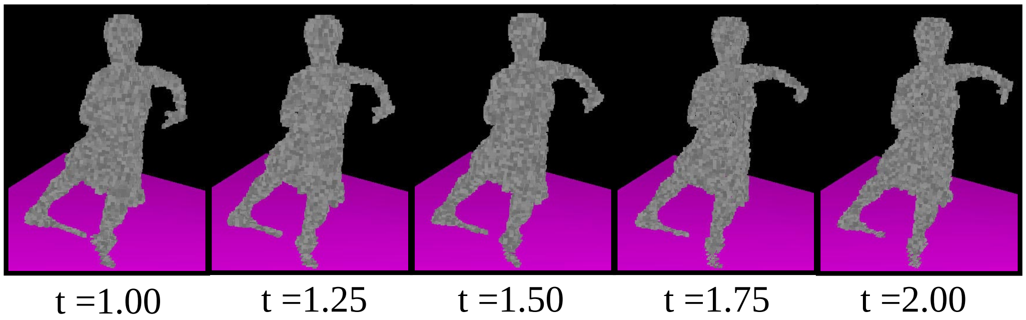


Figure 3: The scene shape between neighboring time instants can be interpolated by flowing the voxels at time t forwards. Note how the arm of the dancer flows smoothly from $t = 1$ to $t = 2$.

our spatio-temporal view interpolation algorithm consist of the images I_i^t, the cameras C_i, the 3D voxel models S^t, and the 3D scene flows F_i^t. (Although we do not use them in this paper, note that algorithms have been proposed to compute voxel models and scene flow simultaneously [Vedula *et al.*, 2000].)

3 Spatio-Temporal View Interpolation

3.1 High-Level Overview of the Algorithm

Suppose we want to generate a novel image I_+^* from virtual camera C_+ at time t^*, where $t \leq t^* \leq t+1$. The first step is to "flow" the voxel models S^t and S^{t+1} using the scene flow to estimate an interpolated voxel model S^*. The second step consists of fitting a smooth surface to the flowed voxel model S^*. The third step consists of ray-casting across space and time. For each pixel (u, v) in I_+^* a ray is cast into the scene and intersected with the interpolated scene shape (the smooth surface). The scene flow is then followed forwards and backwards in time to the neighboring time instants. The corresponding points at those times are projected into the input images, the images sampled at the appropriate locations, and the results blended to give the novel image pixel $I_+^*(u, v)$. Spatio-temporal view interpolation can therefore be summarized as:

1. Flow the voxel models to estimate S^*.

2. Fit a smooth surface to S^*.

3. Ray-cast across space and time.

We now describe these 3 steps in detail starting with Step 1. Since Step 3. is the most important step and can be explained more easily without the complications of surface fitting, we describe it before explaining how intersecting with a surface rather than a set of voxels modifies the algorithm.

3.2 Flowing the Voxel Models

The scene shape is described by the voxels S^t at time t and the voxels S^{t+1} at time $t+1$. The motion of the scene is defined by the scene flow F_i^t for each voxel X_i^t in S^t. We now describe how to interpolate the shapes S^t and S^{t+1} using the scene flow. By comparison, previous work on shape interpolation [Turk and O'Brien, 1999, Alexa *et al.*, 2000] is based solely on the shapes themselves rather than on a flow field connecting them. We assume that the voxels move at constant speed in straight lines and so flow the voxels with the appropriate multiple of the scene flow. If t^* is an intermediate time ($t \leq t^* \leq t+1$), we interpolate the shape of the scene at time t^* as:

$$S^* \ = \ \{X_i^t + (t^* - t) \times F_i^t \mid i = 1, \ldots, V^t\} \tag{3}$$

i.e. we flow the voxels forwards from time t. Figure 3 contains an illustration of voxels being flowed in this way.

Equation (3) defines S^* in an asymmetric way; the voxel model at time $t+1$ is not even used. Symmetry and other desirable properties of the scene flow are discussed in Section 4 after we have presented the ray-casting algorithm.

3.3 Ray-Casting Across Space and Time

Once we have interpolated the scene shape we can ray-cast across space and time to generate the novel image I_+^*. As illustrated in Figure 4, we shoot a ray out into the scene for each pixel (u, v) in I_+^* at time t^* using the known geometry of camera C_+. We find the intersection of this ray with the flowed voxel model. Suppose for now that the first voxel intersected is $X_i^{t^*} = X_i^t + (t^* - t) \times F_i^t$. (Note that we will describe a refinement of this step in Section 3.4.)

We need to find a color for the novel pixel $I_+^*(u, v)$. We cannot project the voxel $X_i^{t^*}$ directly into an image because there are no images at time t^*. We can find the corresponding voxels X_i^t at time t and $X_j^{t+1} = X_i^t + F_i^t$ at time $t+1$, however. We take these voxels and project them into the images at time t and $t+1$ respectively (using the known geometry of the cameras C_i) to get multiple estimates of the color of $I_+^*(u, v)$. This projection must respect the visibility of the voxels X_i^t at time t and X_j^{t+1} at time $t+1$ with respect to the cameras at the respective times.

Once the multiple estimates of $I_+^*(u, v)$ have been obtained, they are *blended*. We just have to decide how to weight the samples in the blend. Ideally we would like the weighting function to satisfy the property that if the novel camera C_+ is one of the input cameras C_i and the time is one of the time instants $t^* = t$, the algorithm should generate the input image I_i^t, *exactly.* We refer to this requirement as the *same-view-same-image* principle.

There are 2 components in the weighting function, space and time. The temporal aspect is the simpler case. We just have to ensure that when $t^* = t$ the weight of the pixels at time t is 1 and the weight at time $t+1$ is 0. We weight the pixels at time t by $(t+1) - t^*$ and those at time $t+1$ so that the total weight is 1; i.e. we weight the later time $t^* - t$.

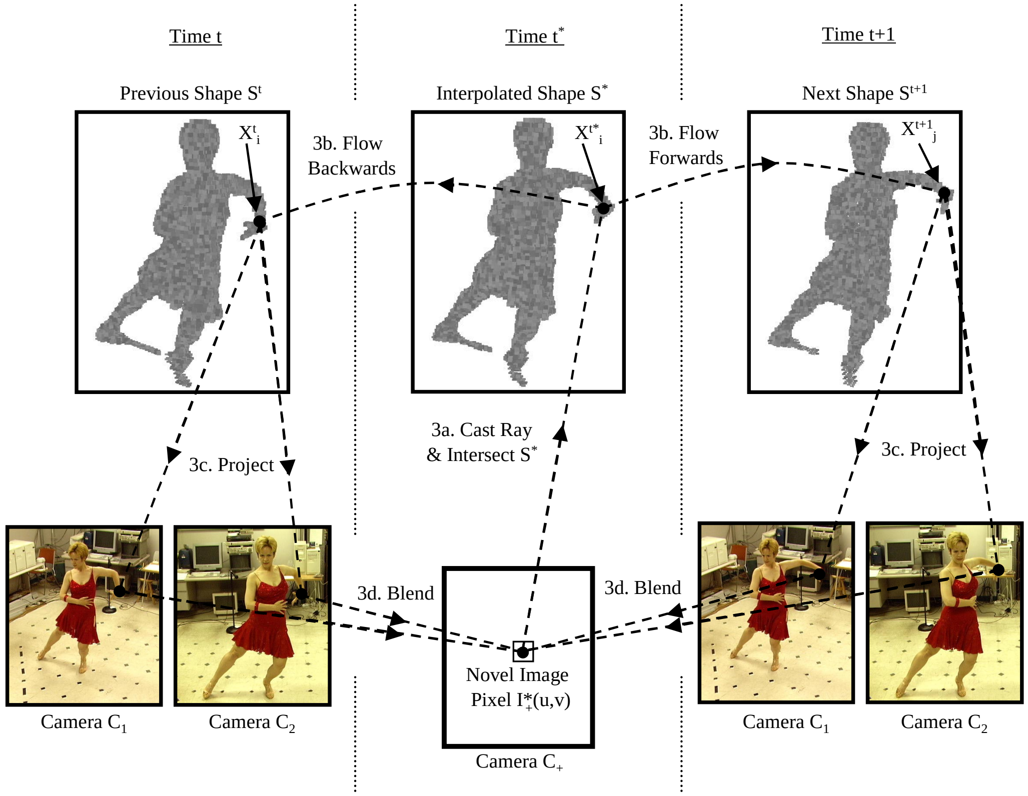


Figure 4: Ray-casting across space and time. 3a. A ray is shot out into the scene at time $t = t^*$ and intersected with the flowed voxel model. (In Section 3.4 we generalize this to an intersection with a smooth surface fit to the flowed voxels.) 3b. The scene flow is then followed forwards and backwards in time to the neighboring time instants. 3c. The voxels at these time instants are then projected into the images and the images sub-sampled at the appropriate locations. 3d. The resulting samples are finally blended to give $I^*_+(u, v)$.

The spatial component is slightly more complex because there may be an arbitrary number of cameras. The major requirement to satisfy the principle, however, is that when $C_+ = C_i$ the weight of the other cameras is zero. This can be achieved for time t as follows. Let $\theta_i(u, v)$ be the angle between the rays from C_+ and C_i to the flowed voxel $X_i^{t^*}$ at time t^*. The weight of pixel (u, v) for camera C_i is then:

$$\frac{1/(1 - \cos\theta_i(u, v))}{\sum_{j=1}^{\text{Vis}(t,u,v)} 1/(1 - \cos\theta_j(u, v))} \tag{4}$$

where $\text{Vis}(t, u, v)$ is the set of cameras for which the voxel X_i^t is visible at time t. This function ensures that the weight of the other cameras tends to zero as C_+ approaches one of the input cameras. It is also normalized correctly so that the total weight of all of the visible cameras is 1.0. An equivalent definition is used for the weights at time $t + 1$.

In summary (see also Figure 4), ray-casting across space and time consists of the following four steps:

3a. Intersect the (u, v) ray with S^{t^*} to get voxel $X_i^{t^*}$.

3b. Follow the flows to voxels X_i^t and X_j^{t+1}.

3c. Project X_i^t & X_j^{t+1} into the images at times t & $t+1$.

3d. Blend the estimates as a weighted average.

For simplicity, the description of Steps 3a. and 3b. above is in terms of voxels. We now describe the details of these steps when we fit a smooth surface through these voxels, and ray-cast onto it.

3.4 Ray-Casting to a Smooth Surface

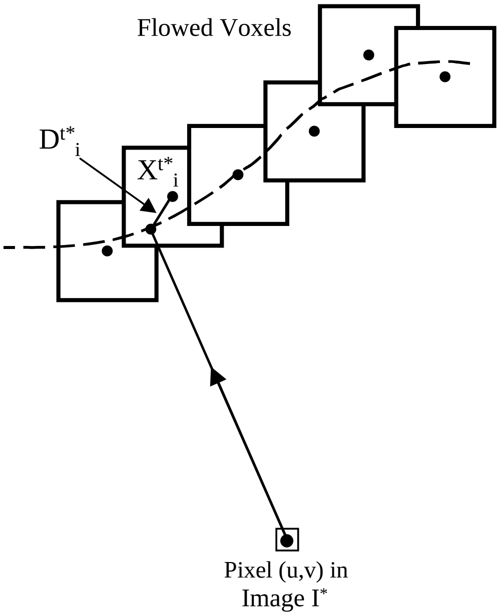


Figure 5: Ray-casting to a smooth surface. We intersect each cast ray with a smooth surface interpolated through the voxel centers (rather than requiring the intersection point to be one of the voxel centers, or boundaries.) Once the ray is intersected with the surface, the perturbation to the point of intersection D_i^{t*} can be transferred to the previous and subsequent time steps.

The ray-casting algorithm described above casts rays from the novel image onto the model at the novel time t^*, finds the corresponding voxels at time t and time $t+1$, and then projects those points into the images to find a color. However, the reality is that voxels are

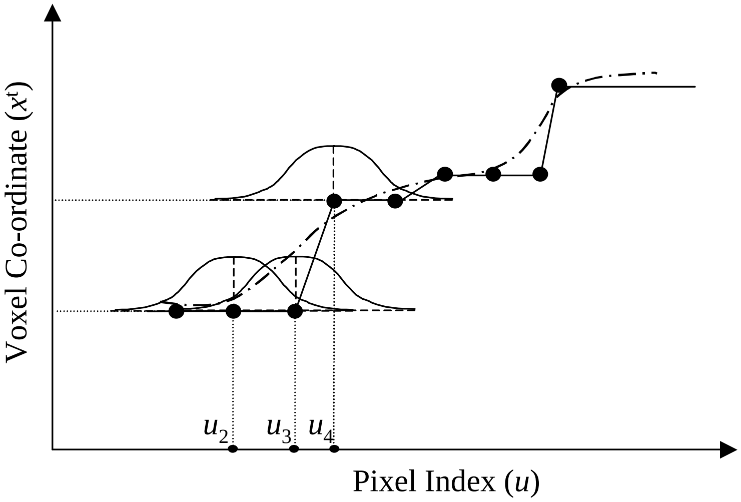


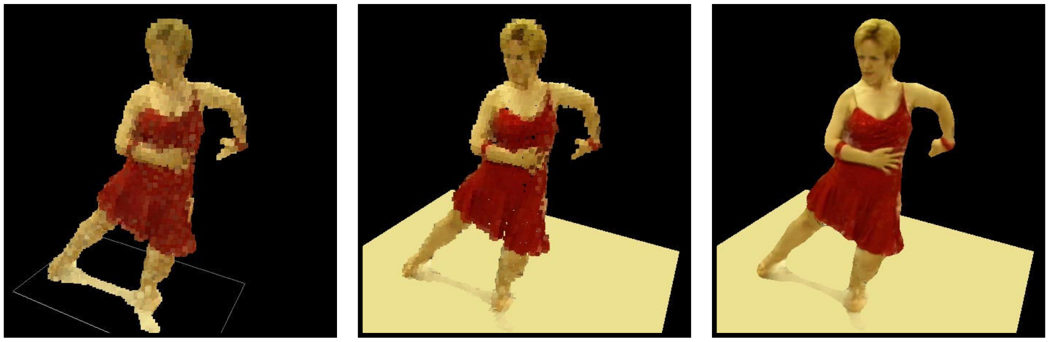
Figure 6: The voxel coordinate changes in an abrupt manner for each pixel in the novel image. Convolution with a simple Gaussian kernel centered on each pixel changes its corresponding 3-D coordinate to approximate a smoothly fit surface.

just point samples of an underlying smooth surface. If we just use voxel centers, we are bound to see cubic voxel artifacts in the final image, unless the voxels are extremely small.

The situation is illustrated in Figure 5. When a ray is cast from the pixel in the novel image, it intersects one of the voxels. The algorithm, as described above, simply takes this point of intersection to the be center of the voxel X_i^{t*}. If, instead, we fit a smooth surface to the voxel centers and intersect the cast ray with that surface, we get a slightly perturbed point $X_i^{t*} + D_i^{t*}$. Assuming that the scene flow is constant within each voxel, the corresponding point at time t is $X_i^t + D_i^{t*}$. Similarly, the corresponding point at $t+1$ is $X_j^{t+1} + D_i^{t*} = X_i^t + F_i^t + D_i^{t*}$. If we simply use the centers of the voxels as the intersection points rather than the modified points, a collection of rays shot from neighboring pixels will all end up projecting to the same points in the images, resulting in obvious box-like artifacts.

Fitting a surface through a set of voxel centers in 3-D is complicated. However, the main contribution of a fit surface in our case would be that it prevents the discrete jump while moving from one voxel to a neighbor. What is really important is that the interpolation between the coordinates of the voxels be smooth. Hence, we propose the following simple algorithm to approximate the true surface fit. For simplicity, we explain in terms of time t^* and time t, the same arguments hold for time $t+1$.

For each pixel u_i in the novel image that intersects the voxel X^{t^*}, the coordinates of the corresponding voxel at time t, $X^t = (x^t, y^t, z^t)$ (which then get projected into the input images) are stored. We therefore have a 2-D array of (x, y, z) values. Figure 6 shows the typical variation of the x component of X^t with u_i. Because of the discrete nature of



(a) Colored Voxel Model

(b) Ray-Casting With Cubic Voxels

(c) Ray-Casting With Surface Fit

Figure 7: The importance of fitting a smooth surface. (a) The voxel model rendered as a collection of voxels, where the color of each voxel is the average of the pixels that it projects to. (b) The result of ray-casting without surface fitting. showing that the voxel model is a coarse approximation. (c) The result of intersecting the cast ray with a surface fit through the voxel centers.

the voxels, this changes abruptly at the voxel centers, whereas, we really want it to vary smoothly like the dotted line. Therefore, we apply a simple Gaussian operator centered at each pixel (shown for u_2, u_3, and u_4) to the function $x^t(u)$ to get a new value of $x^{'t}$ for each pixel u_i (and similarly for $y^t(u)$ and $z^t(u)$), that approximates the true fit surface. These perturbed values $X^{'t} = (x^{'t}, y^{'t}, z^{'t})$ for each pixel in the novel image are projected into the input images as described earlier. [Bloomenthal and Shoemake, 1991] suggest the use of convolution as a way to generate smooth potential surfaces from point skeletons, although their intent is more to generate a useful representation for solid modeling operations.

Figure 7 illustrates the importance of this surface fitting step. Figure 7(a) shows the voxel model rendered as a collection of voxels. The voxels are colored with the average of the colors of the pixels that they project to. Figure 7(b) shows the result of ray-casting by just using the voxel centers directly. Figure 7(c) shows the result after intersecting the cast ray with the smooth surface. As can be seen, without the surface fitting step the rendered images contain substantial voxel artifacts.

4 Ideal Properties of the Scene Flow

In Section 3.2 we described how to flow the voxel model forward to estimate the interpolated voxel model S^*. In particular, Equation (3) defines S^* in an asymmetric way; the voxel model S^{t+1} at time $t+1$ is not even used. A related question is whether the interpolated shape is continuous as $t^* \to t+1$; i.e. in this limit, does S^* tend to S^{t+1}? Ideally we want this property to hold, but how do we enforce it?

One suggestion might be that the scene flow should map *one-to-one* from S^t to S^{t+1}. Then, the interpolated scene shape will definitely be continuous. The problem with this requirement, however, is that it implies that the voxel models must contain the same number of voxels at times t and $t+1$. It is therefore too restrictive to be useful. For example, it outlaws motions that cause the shape to expand or contract. The properties that we really need are:

Inclusion: Every voxel at time t flows to a voxel at time $t+1$: i.e. $\forall t, i \; X_i^t + F_i^t \in S^{t+1}$.

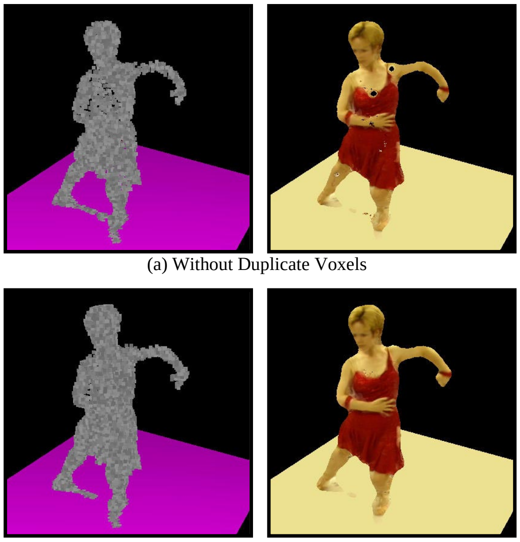
Onto: Every voxel at $t+1$ has a voxel at t that flows to it: $\forall t, i, \; \exists j$ s.t. $X_j^t + F_j^t = X_i^{t+1}$.

These properties immediately imply that the voxel model at time t flowed forward to time $t+1$ is *exactly* the voxel model at time $t+1$:

$$\{X_i^t + F_i^t \mid i = 1, \ldots, V^t\} \; = \; S^{t+1}. \tag{5}$$

This means that the scene shape will be continuous at $t+1$ as we flow the voxel model forwards using Equation (3).

4.1 Duplicate Voxels


(a) Without Duplicate Voxels

(b) With Duplicate Voxels

Figure 8: A rendered view at an intermediate time, with and without duplicate voxels. Without the duplicate voxels, the model at the first time does not flow *onto* the model at the second time. Holes appear where the missing voxels should be. The artifacts disappear when the duplicate voxels are added.

Is it possible to enforce these 2 conditions without the scene flow being one-to-one? It may seem impossible because the second condition seems to imply that the number of voxels cannot get larger as t increases. It is possible to satisfy both properties, however, if we introduce what we call *duplicate voxels*. Duplicate voxels are additional voxels at time t which flow to different points in the model at $t+1$; i.e. we allow 2 voxels X_i^t and X_j^t

$(i \neq j)$ where $(x_i^t, y_i^t, z_i^t) = (x_j^t, y_j^t, z_j^t)$ but yet $F_i^t \neq F_j^t$. We can then still think of a voxel model as just a set of voxels and satisfy the 2 desirable properties above. There may just be a number of duplicate voxels with different scene flows.

Duplicate voxels also make the formulation more symmetric. If the 2 properties *inclusion* and *onto* hold, the flow can be inverted in the following way. For each voxel at the second time instant there are a number of voxels at the first time instant that flow to it. For each such voxel we can add a duplicate voxel at the second time instant with the inverse of the flow. Since there is always at least one such voxel (onto) and every voxel flows to some voxel at the second time (inclusion), when the flow is inverted in this way the two properties hold for the inverse flow as well.

So, given forwards scene flow where *inclusion* and *onto* hold, we can invert it using duplicate voxels to get a backwards scene flow for which the properties hold also. Moreover, the result of flowing the voxel model forwards from time t to t^* with the forwards flow field is the same as flowing the voxel model at time $t + 1$ backwards with the inverse flow. We can then formulate shape interpolation symmetrically as flowing either forwards and backwards. Whichever way the flow is performed, the result will be the same.

The scene flow algorithm [Vedula *et al.*, 1999] unfortunately does not guarantee either of the 2 properties. (Developing such an algorithm is outside the scope of this paper and is left for future research.) Therefore, we take the scene flow and modify it as little as possible to to ensure that the 2 properties hold. First, for each voxel X_i^t we find the closest voxel in S^{t+1} to $X_i^t + F_i^t$ and change the flow F_i^t so that X_i^t flows there. Second, we take each voxel X_i^{t+1} at time $t + 1$ that does not have a voxel flowing to it and add a duplicate voxel at time t that flows to it by averaging the flows in neighboring voxels at $t + 1$.

4.2 Results With and Without Duplicate Voxels

The importance of the duplicate voxels is illustrated in Figure 8. This figure contains 2 rendered views at an intermediate time, one with duplicate voxels and one without. Without the duplicate voxels the model at the first time instant does not flow *onto* the model at the second time. When the shape is flowed forwards holes appear in the voxel model (left) and in the rendered view (right). With the duplicate voxels the voxel model at the first time does flow *onto* the model at the second time and the artifacts disappear.

The need for duplicate voxels to enforce the continuity of the scene shape is illustrated in the movie version of Figure 8 “duplicatevoxels.mpg” available from the first author’s website `http://www.cs.cmu.edu/~srv/stvi/results.html`. This movie consists of a sequence of frames generated using our algorithm to interpolate across time only. (Results interpolating across space are included later.) The movie contains a side-by-side comparison with and without duplicate voxels. Without the duplicate voxels (right) the movie is jerky because the interpolated shape is discontinuous. With the duplicate voxels (left) the movie is very smooth.

The best way to observe this effect is the play the movie several times. The first time concentrate on the left hand side, with the duplicate voxels. The second time concentrate on

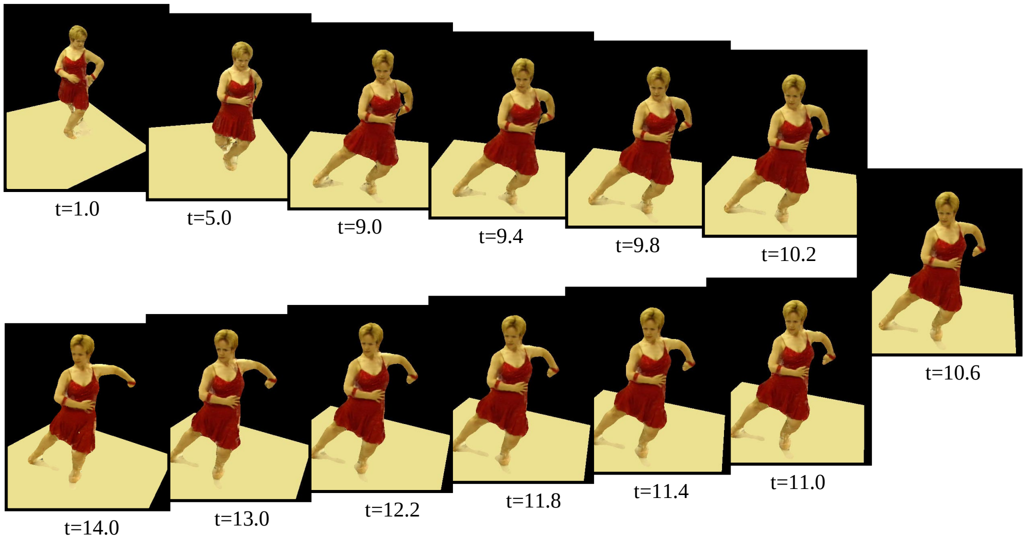


Figure 9: A collection of frames from a slow-motion fly-by movie of the “dance sequence.” This movie was created using our interpolation algorithm. Some of the inputs are included in Figure 2. The novel camera moves along a path that first takes it towards the scene, then rotates it around the scene, and the takes it away from the dancer. The new sequence is also re-timed to be 10 times slower than the original camera speed. In the movie “dance_flyby.mpg”, we include a side by side comparison with the closest input image in terms of both space and time. This comparison makes the inputs appear like a collection of snap-shots compared to the slow-motion movie.

the right hand side. Finally, play the movie one last time and study both sides at the same time for comparison.

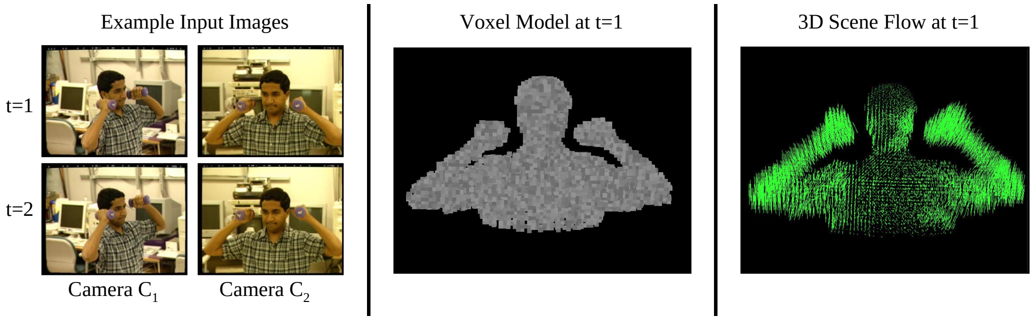
5 Re-Timed Fly-By Movies

We have described a spatio-temporal view interpolation algorithm that can be used to create novel views of a non-rigidly varying dynamic event from arbitrary viewpoints and at any time during the event. We have used this algorithm to generate re-timed slow-motion fly-by movies of real events. The camera can be moved on an arbitrary path through the scene and the event slowed down to any speed.

The input images, computed shapes and scene flows, and fly-by movies created using the spatio-temporal view interpolation algorithm are available online from the first authors website: `http://www.cs.cmu.edu/~srv/stvi/results.html`.

The first movie, “dance_flyby.mpg” is a fly-by movie of the “dancer sequence” that we have been using throughout this paper. Some of the inputs are included in Figure 2 as well as two voxel models and one of the scene flows.

In the sequence a dancer turns, uncrossing her legs, and raising her arm. The input to the algorithm consists of 15 frames from each of 17 cameras. The path of the camera is initially



Example Input Images

Voxel Model at t=1

3D Scene Flow at t=1

Figure 10: Some of the input images, and an example voxel model and scene flow used for the "dumbell" sequence.

towards the scene, then rotates around the dancer and then moves away. Watch the floor to get a good idea of the camera motion. We interpolate 9 times between each neighboring pair of frames. In the movie "dance_flyby.mpg" we show both the slow-motion fly-by and, beside it, the closest input frame in terms of both space and time. Notice how the inputs look like a collection of snap-shots, whereas the fly-by is a smooth re-creation of the event. Figure 9 shows a few sample frames from this movie.

The movie "dumbell_retimed.mpg" shows a re-timed slow-motion movie of a man lifting a pair of dumbells. Figure 10 shows some input images (9 frames from 14 cameras were used), and also an example voxel model and scene flow. Some of the motion in this sequence is highly non-rigid, notice the shirt in particular. To better illustrate this motion in the movie, we leave the novel viewpoint fixed in space. Figure 11 shows some of the sample frames from this re-timed movie sequence.

6 Discussion

We have described a spatio-temporal view interpolation algorithm for creating novel images of a non-rigidly varying dynamic event across both space and time. We have demonstrated how this algorithm can be used to generate very smooth, slow-motion fly-by movies of the event.

We have also addressed the question of what the desirable properties of a 3-D scene flow field are.We have shown that by introducing duplicate voxels we can enforce the constraint that the result of flowing the model forward from time t is exactly the model at $t+1$, but yet without any constraints on the number of voxels at the two time instants. At present there is no scene flow algorithm that sets out to compute a flow field with the 2 properties introduced in Section 4. Developing such an algorithm is a suitable topic for future research.

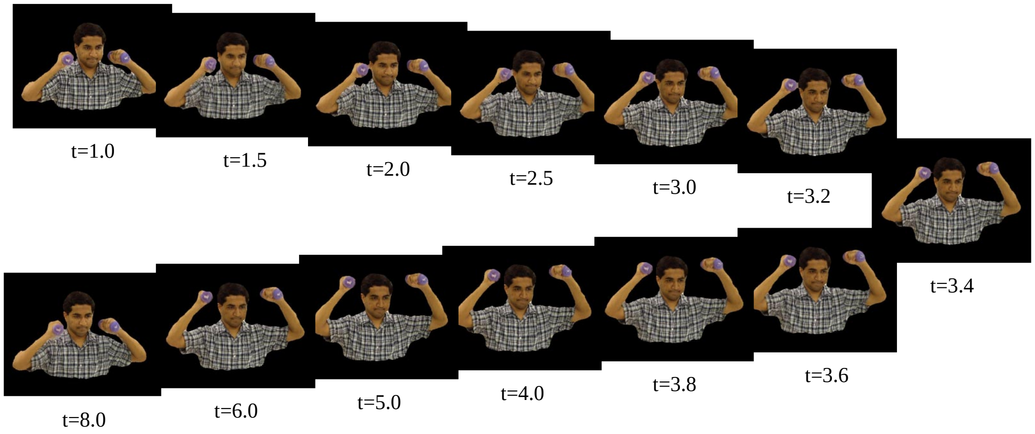


Figure 11: A collection of frames from the re-timed slow motion movie "dumbell_retimed.mpg" of the "dumbell" sequence. Notice the complex non-rigid motion of the shirt and the articulated motion of the arms.

References

[Alexa *et al.*, 2000] M. Alexa, D. Cohen-Or, and D. Levin. As-rigid-as-possible shape interpolation. In *Proc. of SIGGRAPH*, pages 157–164, 2000.

[Bloomenthal and Shoemake, 1991] Jules Bloomenthal and Ken Shoemake. Convolution surfaces. In *Computer Graphics, Annual Conference Series (Proc. SIGGRAPH)*, pages 251–256, 1991.

[Chen and Williams, 1993] S.E. Chen and L. Williams. View interpolation for image synthesis. In *Proc. of SIGGRAPH*, pages 279–288, 1993.

[Gortler *et al.*, 1996] S.J. Gortler, R. Grzeszczuk, R. Szeliski, and M.F. Cohen. The lumigraph. In *Proc. of SIGGRAPH*, pages 43–54, 1996.

[Levoy and Hanrahan, 1996] M. Levoy and M. Hanrahan. Light field rendering. In *Proc. of SIGGRAPH*, 1996.

[Manning and Dyer, 1999] R.A. Manning and C.R. Dyer. Interpolating view and scene motion by dynamic view morphing. In *Proc. of CVPR*, pages 388–394, 1999.

[Narayanan *et al.*, 1998] P.J Narayanan, P.W. Rander, and T. Kanade. Constructing virtual worlds using dense stereo. In *Proc. of ICCV*, pages 3–10, 1998.

[Sato *et al.*, 1997] Y. Sato, M. Wheeler, and K. Ikeuchi. Object shape and reflectance modeling from observation. In *Proc. of SIGGRAPH*, pages 379 – 387, 1997.

[Seitz and Dyer, 1996] S.M. Seitz and C.R. Dyer. View morphing. In *Proc. of SIGGRAPH*, pages 21–30, 1996.

[Seitz and Dyer, 1999] S.M. Seitz and C.R. Dyer. Photorealistic scene reconstruction by voxel coloring. *Intl. Journal of Computer Vision*, 35(2):151–173, 1999.

[Turk and O'Brien, 1999] G. Turk and J.F. O'Brien. Shape transformation using variational implicit functions. In *Proc. of SIGGRAPH*, pages 335–342, 1999.

[Vedula *et al.*, 1999] S. Vedula, S. Baker, P. Rander, R. Collins, and T. Kanade. Three-dimensional scene flow. In *Proc. of ICCV*, volume 2, pages 722–729, 1999.

[Vedula *et al.*, 2000] S. Vedula, S. Baker, S.M. Seitz, and T. Kanade. Shape and motion carving in 6D. In *Proc. of CVPR*, volume 2, pages 592–598, 2000.

[Wexler and Shashua, 2000] Y. Wexler and A. Shashua. On the synthesis of dynamic scenes from reference views. In *Proc. of CVPR*, volume 1, pages 576–581, 2000.