Thirteenth Eurographics Workshop on Rendering (2002)
P. Debevec and S. Gibson (Editors)

Spatio-Temporal View Interpolation

Sundar Vedula[†‡], Simon Baker[‡], and Takeo Kanade[‡]

† Sony Electronics, San Jose, California, USA
‡ The Robotics Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

Abstract

We propose a fully automatic algorithm for view interpolation of a completely non-rigid dynamic event across both space and time. The algorithm operates by combining images captured across space to compute voxel models of the scene shape at each time instant, and images captured across time to compute the "scene flow" between the voxel models. The scene-flow is the non-rigid 3D motion of every point in the scene. To interpolate in time, the voxel models are "flowed" using an appropriate multiple of the 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 instant, following the scene flow to the neighboring time instants, projecting into the input images at those times, and finally blending the results. We use our algorithm to create re-timed slow-motion fly-by movies of dynamic real-world events.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Ray-tracing and Virtual Reality

1. Introduction

We propose an algorithm for view interpolation of a dynamic event across space and time. While there has been a large amount of research on image-based interpolation of static scenes across space [3, 5, 9, 12, 13, 15], there has been almost no research on re-rendering a dynamic event across time. What work there has been has assumed a restricted motion model. Either the scene consists of rigidly moving objects [11, 14] or point features moving along straight lines with constant velocity [22]. Our algorithm is applicable to non-rigid events and uses no scene or object specific models. Our algorithm is also automatic, requiring no user input.

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.

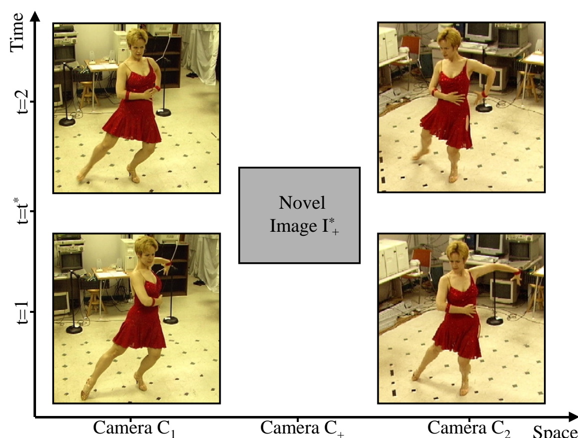


Figure 1: *Spatio-temporal view interpolation consists of taking a collection of images of an event captured with multiple cameras at different times and re-rendering the event at an arbitrary viewpoint and time. In this illustrative figure, the 2 images on the left are captured with the same camera 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 completely arbitrary in our algorithm.*

Our algorithm is based on the explicit recovery of 3D scene properties. We use the *voxel coloring* algorithm [16] to

© The Eurographics Association 2002.

delivered by EUROGRAPHICS DIGITAL LIBRARY www.eg.org diglib.eg.org

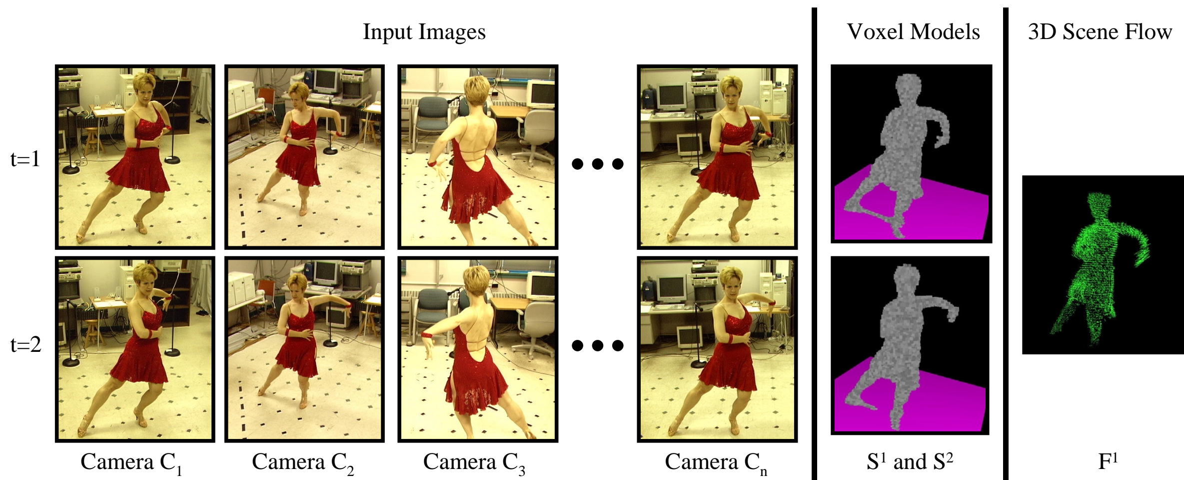


Figure 2: *The input to our spatio-temporal view interpolation algorithm is a set of calibrated images at 2 or more consecutive time instants. From these images, 3D voxel models are computed at each time instant using the voxel coloring algorithm*[16]*. After we have computed the 3D voxel models, we then compute the dense non-rigid 3D motion or "scene flow" between these models using our scene flow algorithm*[20].

recover a 3D voxel model of the scene at each time instant, and our 3D *scene flow* algorithm [20] to recover the 3D non-rigid motion of the scene between consecutive time instants. The voxel models and scene flow then become additional inputs to our spatio-temporal view interpolation 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 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.

To obtain high image quality results, there are a number of technical issues that have to be dealt with. First, we require that the 3D scene flow, and the 3D voxel models it relates, all be consistent in a way that when the models are flowed no holes (or other artifacts) are generated. Second, a surface must be fit to the flowed voxel models to avoid artifacts introduced by the cubic voxels. Finally, the blend of the sampled pixels should be chosen to ensure that the algorithm gives exactly the input image when the novel image parameters (and time) match one of the actual cameras.

The remainder of this paper is organized as follows. We begin in Section 2 by describing the inputs to our algorithm, the input images, and the voxel models and scene flow computed from them. We proceed in Section 3 to outline our spatio-temporal view interpolation algorithm. In Section 4 we describe the 2 ideal properties that the voxel models and the scene flow should obey. We also demonstrate the effect that these properties have on the image quality of the interpolated images. In Section 5 we describe how standard graphics hardware can be used to improve the efficiency of our algorithm. We present experimental results in Section 6 and end with a conclusion in Section 7.

2. Inputs to the STVI Algorithm

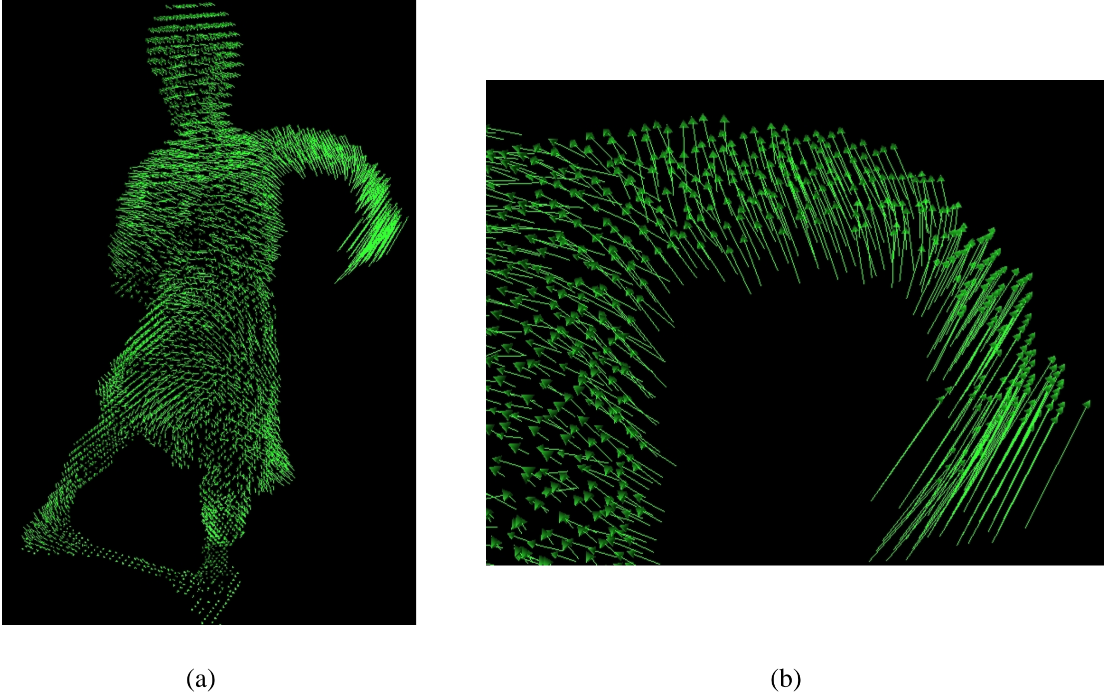
2.1. Explicit 3D Models Vs. Correspondences

To generate novel views we need to know how the pixels in the input images are geometrically related to each other. In the various approaches to view interpolation of static scenes across space there are 2 common ways in which this geometric information is provided. First, there are algorithms that use *implicit* geometric information in the form of *point correspondences* [3, 15]. Second, there are approaches which use *explicit* 3D models of the scene [4, 12, 13].

Although either choice is theoretically possible, we decided to base our spatio-temporal view interpolation 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 event are captured over an extended period of time, and from multiple viewing directions.

The relationship between pixels *across time* can be described by how points in the scene move across time. Since in general the scene can move in an arbitrarily non-rigid way, the 3D motion of points is the *scene flow* [20]. We use the com-

© The Eurographics Association 2002.



(a) (b)

Figure 3: *Two enlarged views of the scene flow in Figure 2. Notice how the motion of the dancer's arm is highly non-rigid.*

bination of scene shape (represented by 3D voxel models) and 3D scene flow to relate the pixels in the input images.

2.2. 3D Voxel Models

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 surface voxels 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 voxels S^t at each time instant t independently using *voxel coloring* [16]. Figure 2 illustrates the voxel models for $t = 1$ and $t = 2$.

2.3. 3D Scene Flow

The scene flow of a voxel describes how it moves across time. 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 $F_i^t = (f_i^t, g_i^t, h_i^t)$. We compute the scene flow F_i^t for every voxel in the model S^t at each time instant t using our scene flow algorithm [20]. Figure 2 contains the result of computing the scene flow from $t = 1$ to $t = 2$. Figure 3 also shows two enlarged snapshots of the computed scene flow for two different viewpoints. Scene flow is computed as a dense flow field, so there is a distinct motion vector computed for every single voxel in the scene. The close-up snapshot shows the highly non-rigid motion of the voxels as the dancer raises and stretches out her arm.

In summary, the inputs to spatio-temporal view interpolation 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. See Figure 2 for an illustration of the inputs and [19] for more details of how the voxel models and scene flow are computed.

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 $t \le t^* \le t+1$. 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,

© The Eurographics Association 2002.

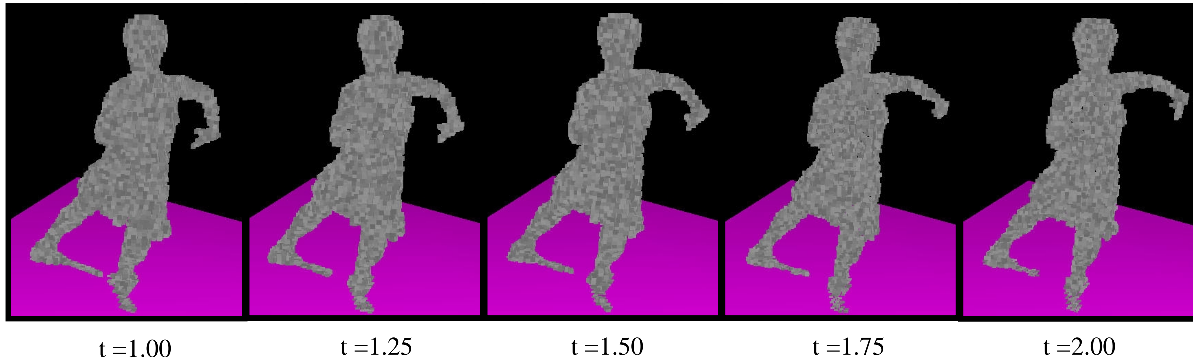


Figure 4: *The scene shape can be interpolated between neighboring time instants by flowing the voxels at time t forwards with an appropriate multiple of the scene flow. Notice how the arm of the dancer flows smoothly upwards and outwards from* $t = 1.00$ *to* $t = 2.00$*.*

and the results blended to give the novel image pixel $I^*_+(u,v)$. Our algorithm 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 next, 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 described by the scene flow F^t_i for each voxel X^t_i 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 is based solely on the shapes themselves rather than on a flow field connecting them or on interpolating between manually selected feature points [1, 2, 8, 18]. 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. (In making this constant linear motion assumption we are assuming that the motion is *temporally* "smooth" enough. This assumption does not impose *spatial* smoothness or rigidity on the motion.) If t^* is an intermediate time ($t \leq t^* \leq t+1$), we interpolate the shape of the scene at time t^* as:

$$S^* \;=\; \{X^t_i + (t^* - t) \times F^t_i \mid i = 1, \ldots, V^t\} \qquad (3)$$

i.e. we flow the voxels forwards from time t. Figure 4 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 5, 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 $X^{t^*}_i = X^t_i + (t^* - t) \times F^t_i$. (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 $X^{t^*}_i$ directly into an image because there are no images at time t^*. We can find the corresponding voxels X^t_i at time t and $X^{t+1}_j = X^t_i + F^t_i$ 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^t_i at time t and X^{t+1}_j at time $t+1$ with respect to the cameras at the respective times. See Section 5.2 for details.

Once the multiple estimates of $I^*_+(u,v)$ have been obtained, they are *blended*. Ideally we would like the weighting function in the blend 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^t_i, *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 $t^* - t$.

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. One way this can be achieved is as follows. Let $\theta_i(u,v)$ be the angle

© The Eurographics Association 2002.

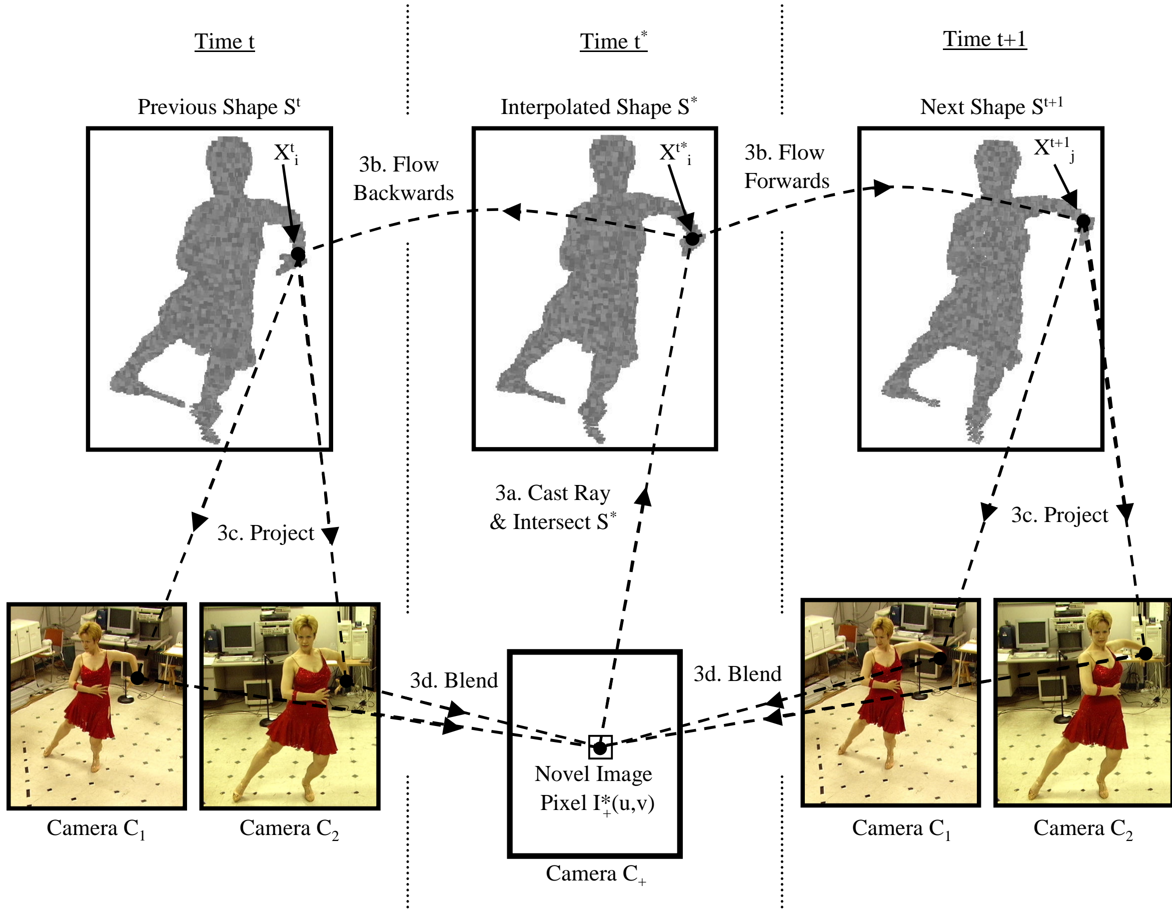


Figure 5: *Ray-casting across space and time. 3a. A ray is shot out into the scene at time $t = t^*$ and intersected with the flowed voxel model. (In Section 3.4 we generalize this to an intersection with a smooth surface fit to the flowed voxels.) 3b. The scene flow is then followed forwards and backwards 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 $I^*_+(u,v)$.*

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:

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

where $\mathrm{Vis}(t,u,v)$ is the set of cameras for which the voxel X_i^t is visible at time t. This function, a variation of *view dependent texture mapping* [4], 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 $t+1$. More sophisticated weighting functions could be used that, for example, take into account how frontal the surface is or attempt to estimate the parameters of more complex BRDF functions [17]. The investigation of such approaches is left as future work.

In summary (see also Figure 5), 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.

3.4. Ray-Casting to a Smooth Surface

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 just point samples of an

© The Eurographics Association 2002.

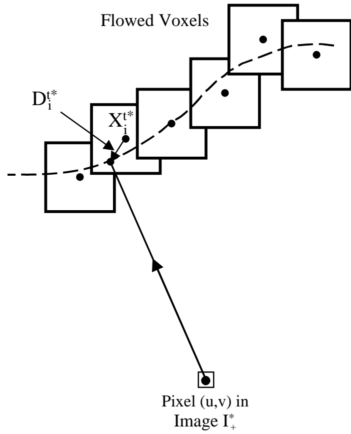


Figure 6: *Ray-casting to a smooth surface. We intersect each cast ray with a smooth surface interpolated through the voxel centers. The perturbation to the point of intersection D_i^{t*} can then be transferred to the previous and subsequent time instants.*

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

The situation is illustrated in Figure 6. 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 an arbitrary set of voxel centers in 3-D is a well studied problem [6, 10]. Most algorithms only operate on regular grids, however. Fitting a 3D surface through the voxel centers of an irregular grid is a much harder problem. However, the main requirement of the fit surface in our case is just that it prevents the discrete jump while moving from one voxel to a neighbor. What is important is that the interpolation between the coordinates of the voxels be smooth. We propose the following simple algorithm to approximate the surface fit.

For each pixel u_i in the novel image, the 3D coordinates of the corresponding voxel at time t, $X^t = (x^t, y^t, z^t)$ are stored to give a 2-D array of (x,y,z) values. Figure 7 shows the typical variation of the x component of X^t with the image coordinate u_i. Because of the discrete nature of the voxels, this function changes abruptly at the voxel centers, whereas, we really want it to vary smoothly, say like the dotted line.

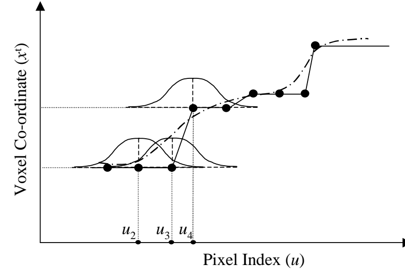


Figure 7: *The voxel coordinate changes in an abrupt manner for each pixel in the novel image. Convolution with a simple Gaussian kernel centered on each pixel changes its corresponding 3-D coordinate to approximate a smoothly fit surface.*

We apply a Gaussian smoothing 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 $\overline{x}^t$ (and similarly for $y^t(u)$ and $z^t(u)$). The smoothed 3D vector function $\overline{X}^t = (\overline{x}^t, \overline{y}^t, \overline{z}^t)$ is then used in place of $X^t = (x^t, y^t, z^t)$; i.e. the perturbation $D_i^{t*} = \overline{X}^t - X^t$.

Figure 8 illustrates the importance of surface fitting. Figure 8(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 8(b) shows the result of ray-casting, but just using the voxels. Figure 8(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:

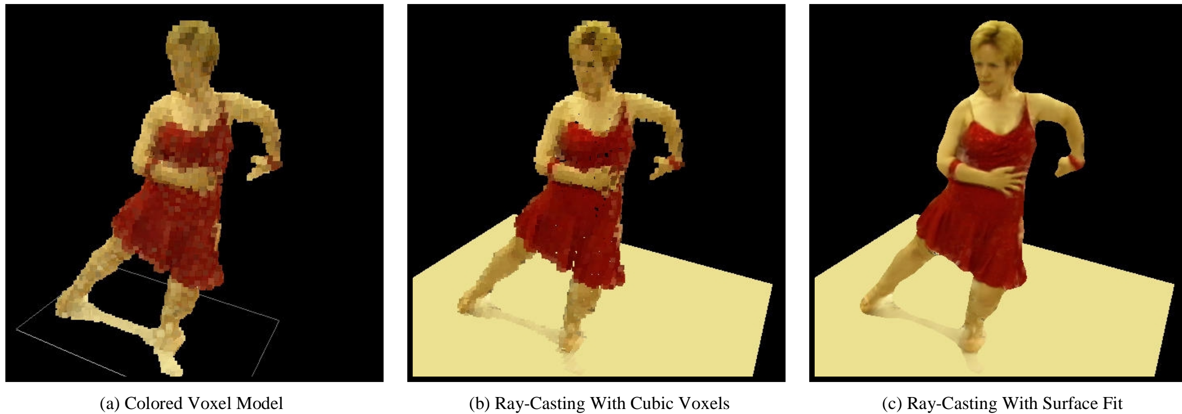
Inclusion: Every voxel at time t should flow to a voxel at time $t+1$: i.e. $\forall t, i \; X_i^t + F_i^t \in S^{t+1}$.

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

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

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

© The Eurographics Association 2002.



(a) Colored Voxel Model (b) Ray-Casting With Cubic Voxels (c) Ray-Casting With Surface Fit

Figure 8: *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 results in a far better rendering.*

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

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 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 $F_i^t \neq F_j^t$. A voxel model is then still just a set of voxels and but can 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 [20] unfortunately does not guarantee either of the 2 desirable properties. Therefore, we take the scene flow computed with [20] 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 $t+1$.

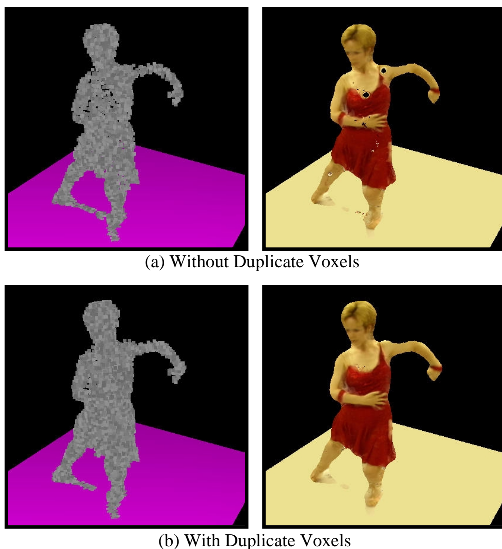
4.2. Results With and Without Duplicate Voxels

The importance of the duplicate voxels is illustrated in Figure 9. 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 continuity is illustrated in the movie duplicate_voxels.mpg available from http://www.ri.cmu.edu/projects/project_464.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 to play the movie several times. The first time concentrate on the left hand side with the duplicate voxels. The second time concentrate on the right hand side. Finally, play the movie one last time and study both sides at the same time for comparison.

© The Eurographics Association 2002.



(a) Without Duplicate Voxels

(b) With Duplicate Voxels

Figure 9: *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.*

5. Optimization Using Graphics Hardware

Spatio-temporal view interpolation involves 2 fairly computationally expensive operations. It is possible to optimize these using standard graphics hardware as we now discuss.

5.1. Intersection of Ray with Voxel Model

Steps 3a. and 3b. of our algorithm involve casting a ray from pixel (u,v) in the novel image, finding the voxel $X_i^{t^*}$ that this ray intersects, and then finding the corresponding voxels X_i^t and X_i^{t+1} at the neighboring time instants. Finding the first point of intersection of a ray with a voxel model is potentially an expensive step, since the naive algorithm involves an exhaustive search over all voxels. In addition, extra bookkeeping is necessary to determine the corresponding voxels at times t and $t+1$ for each flowed voxel $X_i^{t^*}$.

We implement this step as follows. Each voxel in the model S^t is given a unique ID, which is encoded as a unique (r,g,b) triplet. This voxel model is then flowed as discussed in Section 3.2 to give the voxel model S^* at time t^*. This voxel model S^* (see Equation (3)) is then rendered as a collection of little cubes, one for each voxel, colored with that voxel's unique ID. In particular, the voxel model S^* is rendered from the viewpoint of the novel camera using standard OpenGL. Lighting is turned off (to retain the base color of the cubes), and z-buffering turned on, to ensure that only the closest voxel along the ray corresponding to any pixel is visible. Immediately after the rendering, the color buffers are read and saved. Indexing the rendered image at the pixel (u,v) gives the ID (the (r,g,b) value) of the corresponding voxel at time t (and hence at time $t+1$.)

This method of using color to encode a unique ID for each geometric entity is similar to the *item buffer* [21] which is used for visibility computation in ray tracing.

5.2. Determining Visibility of the Cameras

Step 3c. of our algorithm involves projecting X_i^t and X_j^{t+1} into the input images. But how do we compute whether the X_i^t was actually visible in camera C_k?

Again, we use a z-buffer approach similar to the previous case, except this time, we don't need to encode any sort of information in the color buffers (that is, there are no voxel IDs to resolve). The occlusion test for Camera C_k runs as follows. Let $\mathbf{R}_k$ and $\mathbf{t}_k$ be the rotation matrix and translation vector for camera C_k relative to the world coordinate system. Then, X_i^t is first transformed to camera coordinates:

$$\begin{Bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{Bmatrix} = \begin{bmatrix} \mathbf{R}_k & \mathbf{t}_k \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{X}_i^t. \quad (6)$$

The image coordinates of the projection $\mathbf{u} = (u,v)$ are obtained by multiplying by the 3×4 camera matrix P_k

$$\mathbf{u} = P_k \mathbf{x} \quad (7)$$

The voxel model is then rendered, with the camera transformation matrix set to be exactly that corresponding to the calibration parameters of camera C_k. After the rendering, the hardware z-buffer is read. This z-buffer now gives the depth to the nearest point on the shape for any particular pixel in the rendered image, and therefore any pixel in the real image as well, since the viewpoints are identical for both.

In reality, the value of the hardware z-buffer is between zero and one, since the true depth is transformed by the perspective projection that is defined by the near and far clipping planes of the viewing frustum. However, since these near and far clipping planes are user-specified, the transformation is easily invertible and the true depth-map can be recovered from the value of the z-buffer at any pixel.

Let (u,v) be the image coordinates in image I_k, as computed from Equation (7). The value of the z-buffer at that pixel, $z_k(u,v)$ is compared against the value of x_3 (which is the distance to the voxel $\mathbf{X}(i,j)$ from the camera). If $x_3 = z_k(u,v)$, then the voxel $\mathbf{X}(i,j)$ is visible in the camera. Instead if $x_3 > z_k(u,v)$ the voxel $\mathbf{X}(i,j)$ is occluded by another part of the scene.

6. Experimental Results

We have applied our spatio-temporal view interpolation algorithm to two highly non-rigid dynamic events: a "Paso Doble Dance Sequence" and a "Person Lifting Dumbbells." These events were both imaged in the CMU 3D Room [7].

© The Eurographics Association 2002.

6.1. Sequence 1: Paso Doble Dance Sequence

The first event is a "Paso Doble Dance Sequence". The images are those used in the figures throughout this paper. See Figures 1, 2, 5, 6, and 9. In the sequence, the dancer turns as she un-crosses her legs and raises her left arm.

The input to the algorithm consists of 15 frames from each of 17 cameras, in total 255 images. Handmarking point correspondences in 255 images is clearly impossible. Hence it is important that our algorithm is fully automatic. The input frames for each of the cameras are captured 1/10 of a second apart, so the entire sequence is 1.5 seconds long. The 17 cameras are distributed all around the dancer, with 12 of the cameras on the sides, and 5 cameras overhead.

Figure 10 shows a collection of frames from a virtual slow-motion fly-through of this dance sequence. The path of the camera is initially towards the scene, then rotates around the dancer, and finally moves away. Watch the floor (which is fixed) to get a good idea of the camera motion. We interpolate 9 times between each neighboring pair of input frames.

The movie dance_flyby.mpg which is available from our website at http://www.ri.cmu.edu/projects/project_464.html was created by assembling the spatio-temporal interpolated images. Also shown is a comparison with what would have been obtained had we just switched between the closest input images, measured both in space and time. Note that the movie created in this manner looks like a collection of snap-shots, whereas the spatio-temporal fly-by is a much smoother and natural looking re-rendering of the event.

There are some visible artifacts in the fly-by movie, such as slight blurring, and occasional discontinuities. The blurring occurs because our shape estimation is imperfect. Therefore corresponding points from neighboring cameras are slightly misaligned. The discontinuities are because of imperfect scene flow - a few voxels have erroneous flows and flow to the wrong place at the next time instant.

6.2. Sequence 2: Person Lifting Dumbbells

The second event consists of a "Person Lifting Dumbbells". The person pushes up a pair of dumbbells from their shoulder to full arm extension, and then brings them down again. The input to the algorithm consists of 9 frames from each of 14 cameras for a total of 126 images. Again, handmarking point correspondences in so many images would be impossible. Two images at consecutive time steps from each of two cameras are shown in Figure 11. This figure also contains the voxel model computed at the first time instant and the scene flow computed between the those two time steps.

From these input images, voxel models, and scene flows we used our spatio-temporal view interpolation algorithm to generate a re-timed slow-motion movie of this sequence. Again, we interpolated 9 frames between each pair in the input. To better illustrate the motion, we also left the novel viewpoint fixed in space. The novel viewpoint doesn't correspond to any of the camera views and could easily be changed. Figure 12 shows a number of sample frames from the re-timed movie dumbbell_slowmo.mpg available from http://www.ri.cmu.edu/projects/project_464.html. Notice the complex non-rigid motion on the shirt as the person flexes their muscles and the articulated motion of their arms.

Just like for the dance sequence, we also include a side-by-side comparison with the closest input image. In this movie, the viewpoint doesn't change and so the closest image always has the same pose. The closest image in time just steps through the original sequence from that camera.

6.3. Efficiency of the Algorithm

The computation time of spatio-temporal view interpolation is linear in the number of pixels in the output image, irrespective of the complexity of the model, as for most image based rendering algorithms. It is also linear in the number of input images that are used to contribute to each pixel of the output. In our examples we compute a 640×480 novel image, using the six closest images (3 closest cameras at each of 2 time instants). The algorithm takes about 5 seconds to run for each output frame on an SGI 02, using graphics hardware to compute the ray-voxel intersection and the visibility. See [19] for more details of the experimental set-up and the efficiency of the voxel carving and scene flow algorithms.

7. Conclusion

7.1. Summary

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

7.2. Future Work

Perhaps the major limitation of our algorithm is that it assumes that the cameras are fully calibrated. One possible direction for future work is to develop a projective version of the algorithm that can operate with un-calibrated data. Another possible extension is to investigate more sophisticated blending functions that, for example, take into account how frontal the surface of the scene is [17]. More complex BRDF functions could also be used. Finally, work could be performed to characterize the errors in 3D scene flow and the effect that they have on the rendered images.

References

1. M. Alexa, D. Cohen-Or, and D. Levin. As-rigid-as-possible shape interpolation. In *Computer Graphics Annual Conference Series (SIGGRAPH)*, 2000. 4

© The Eurographics Association 2002.

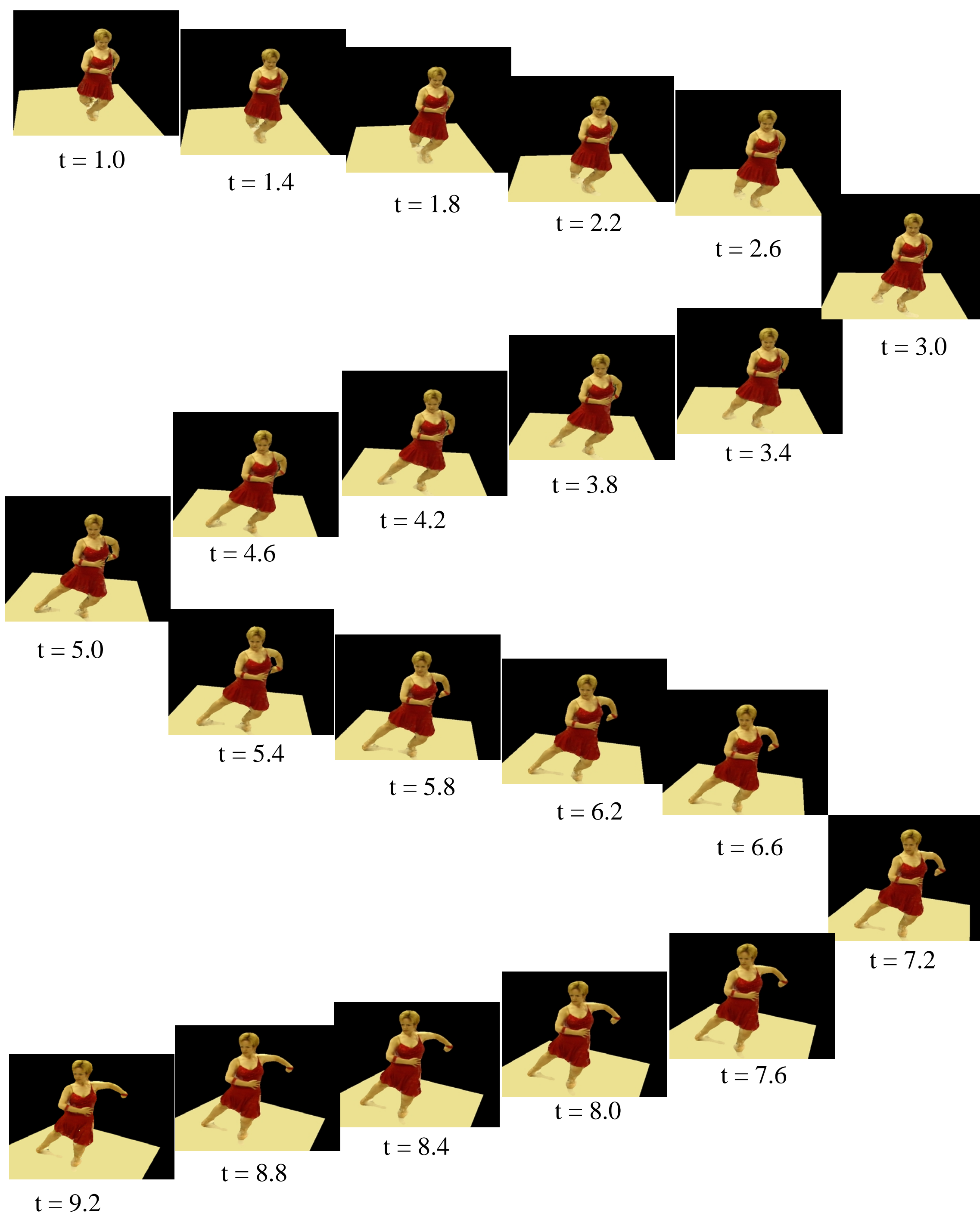


Figure 10: *A collection of frames from the movie dance_flyby.mpg available from http://www.ri.cmu.edu/projects/project_464.html created from the "Paso Doble Dance Sequence." Some of the inputs are included in Figure2. The virtual camera moves along a path that first takes it towards the scene, then rotates it around the scene, and finally 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, 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 our spatio-temporally interpolated movie.*

© The Eurographics Association 2002.

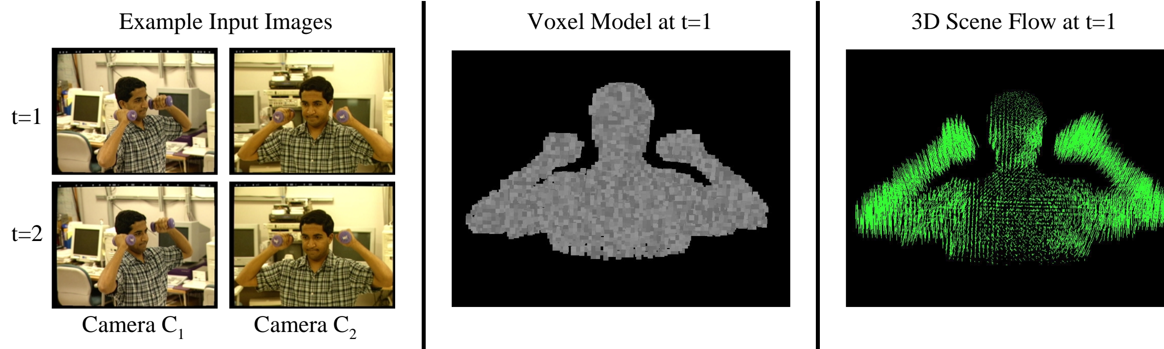


Figure 11: *Some of the input images, an example voxel model, and one of the scene flows used for the "dumbbell" sequence. The complete sequence consists of 9 frames from each of 14 cameras for a total of 126 images. Handmarking point correspondences in this many images would be very time consuming. From these images we compute 9 voxel models, one at each time instant, and 8 pairs of scene flows between them. One voxel model and one scene flow are shown. We then use all this information to generate a re-timed slow-motion movie of the event. See Figure 12. The complete movie dumbbell_slowmo.mpg is available from http://www.ri.cmu.edu/projects/project_464.html.*

2. D. Breen and R. Whitaker. A level-set approach for the metamorphosis of solid models. *IEEE Trans. on Visualization and Computer Graphics*, 7(2):173–192, 2001. 4

3. S.E. Chen and L. Williams. View interpolation for image synthesis. In *Computer Graphics Annual Conference Series (SIGGRAPH)*, pages 279–288, 1993. 1, 2

4. P.E. Debevec, C.J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Comp. Graphics Annual Conf. Series (SIGGRAPH)*, 1996. 2, 5

5. S.J. Gortler, R. Grzeszczuk, R. Szeliski, and M.F. Cohen. The lumigraph. In *Computer Graphics Annual Conference Series (SIGGRAPH)*, pages 43–54, 1996. 1

6. A. Kadosh, D. Cohen-Or, and R. Yagel. Tricubic interpolation of discrete surfaces for binary volumes. *IEEE Trans. on Visualization and Computer Graphics*, 2000. 6

7. T. Kanade, H. Saito, and S. Vedula. The 3D room: Digitizing time-varying 3D events by synchronized multiple video streams. Technical Report CMU-RI-TR-98-34, Robotics Institute, Carnegie Mellon University, 1998. 8

8. A. Lerios, C.D. Garfinkle, and M. Levoy. Feature-based volume metamorphosis. In *Computer Graphics Annual Conference Series (SIGGRAPH)*, 1995. 4

9. M. Levoy and M. Hanrahan. Light field rendering. In *Computer Graphics Annual Conference Series (SIGGRAPH)*, 1996. 1

10. W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3D surface reconstruction algorithm. *Computer Graphics*, 21(4):163–169, 1992. 6

11. R.A. Manning and C.R. Dyer. Interpolating view and scene motion by dynamic view morphing. In *Proc. of the Conf. on Comp. Vis. and Pattern Recognition*, 1999. 1

12. P.J Narayanan, P.W. Rander, and T. Kanade. Constructing virtual worlds using dense stereo. In *Proc. of the 6th Intl. Conf. on Computer Vision*, 1998. 1, 2

13. Y. Sato, M. Wheeler, and K. Ikeuchi. Object shape and reflectance modeling from observation. In *Comp. Graphics Annual Conf. Series (SIGGRAPH)*, 1997. 1, 2

14. E. Schechtman, Y. Capsi, and M. Irani. Increasing space-time resolution in video. In *Proceedings of the 7th European Conference on Computer Vision*, 2002. 1

15. S.M. Seitz and C.R. Dyer. View morphing. In *Computer Graphics Annual Conference Series (SIGGRAPH)*, pages 21–30, 1996. 1, 2

16. S.M. Seitz and C.R. Dyer. Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 35(2):151–173, 1999. 1, 2, 3

17. G. Turk and M. Levoy. Zippered polygon meshes from range images. In *Computer Graphics Annual Conference Series (SIGGRAPH)*, 1994. 5, 9

18. G. Turk and J.F. O'Brien. Shape transformation using variational implicit functions. In *Computer Graphics Annual Conference Series (SIGGRAPH)*, 1999. 4

19. S. Vedula. *Image Based Spatio-Temporal Modeling and View Interpolation of Dynamic Events*. PhD thesis, Robotics Institute, Carnegie Mellon University, 2001. 3, 9

© The Eurographics Association 2002.

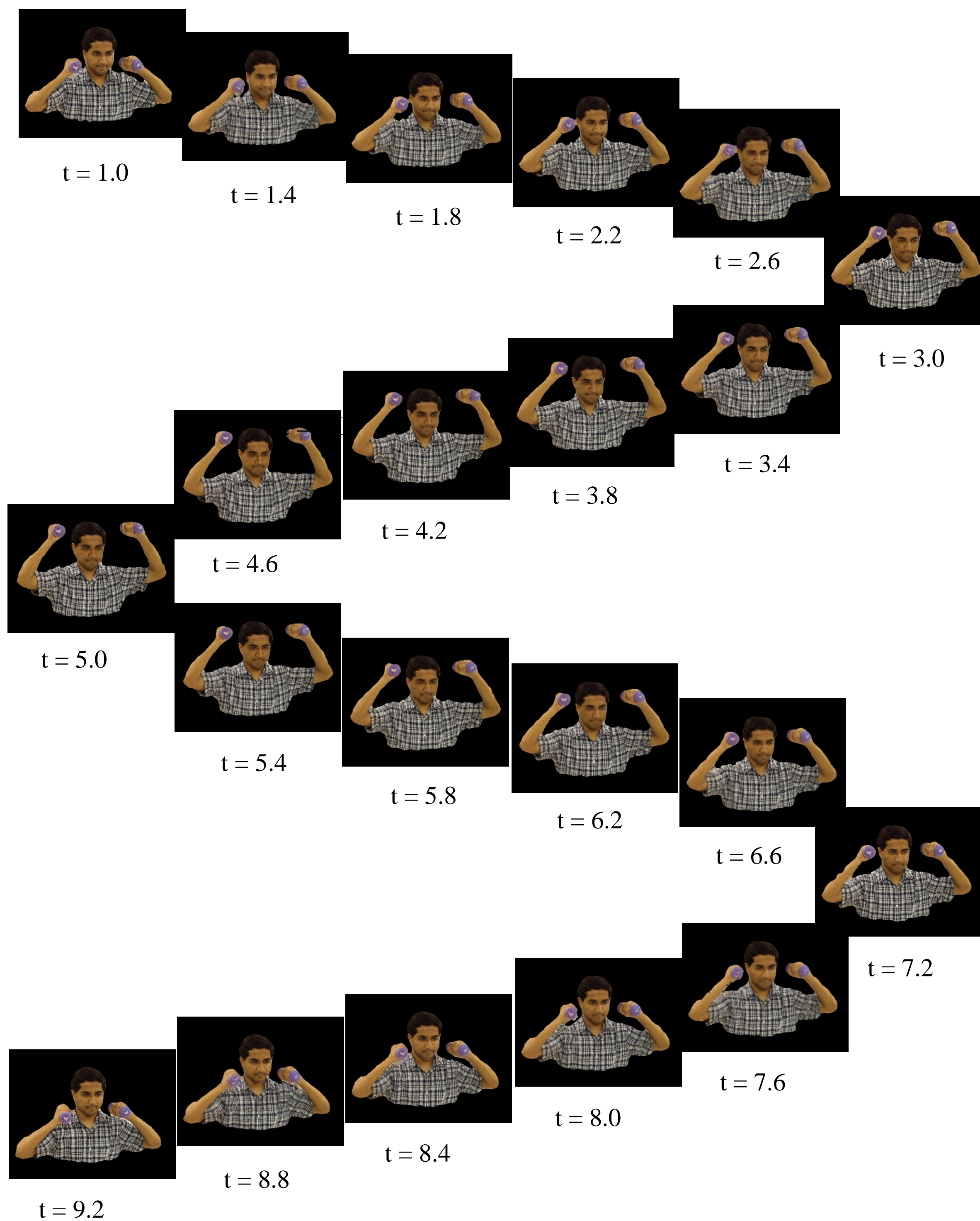


Figure 12: *A collection of frames from the re-timed slow motion movie (dumbbell_slowmo.mpg) of the "dumbbell" sequence. See Figure11 for an illustration of the inputs. Notice the complex non-rigid motion of the shirt and the articulated motion of the arms. The complete movie dumbbell_slowmo.mpg is available from our website at URL http://www.ri.cmu.edu/projects/project_464.html.*

© The Eurographics Association 2002.

20. S. Vedula, S. Baker, P. Rander, R. Collins, and T. Kanade. Three-dimensional scene flow. In *Proc. of the 7th IEEE Intl. Conf. on Computer Vision*, 1999. 2, 3, 7

21. H. Weghorst, G. Hooper, and D.P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, 1984. 8

22. Y. Wexler and A. Shashua. On the synthesis of dynamic scenes from reference views. In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 2000. 1

© The Eurographics Association 2002.