Dynamosaicing: Mosaicing of Dynamic Scenes

Alex Rav-Acha, *Student Member*, *IEEE*, Yael Pritch,
Dani Lischinski, and Shmuel Peleg, *Member*, *IEEE*

Abstract—This paper explores the manipulation of time in video editing, which allows us to control the chronological time of events. These time manipulations include slowing down (or postponing) some dynamic events while speeding up (or advancing) others. When a video camera scans a scene, aligning all the events to a single time interval will result in a panoramic movie. Time manipulations are obtained by first constructing an aligned space-time volume from the input video, and then sweeping a continuous 2D slice (*time front*) through that volume, generating a new sequence of images. For dynamic scenes, aligning the input video frames poses an important challenge. We propose to align dynamic scenes using a new notion of "dynamics constancy," which is more appropriate for this task than the traditional assumption of "brightness constancy." Another challenge is to avoid visual seams inside moving objects and other visual artifacts resulting from sweeping the space-time volumes with time fronts of arbitrary geometry. To avoid such artifacts, we formulate the problem of finding optimal time front geometry as one of finding a minimal cut in a 4D graph, and solve it using max-flow methods.

Index Terms—Video mosaicing, dynamic scene, video editing, graph cuts, panoramic mosaicing, time manipulations, space-time volume.

✦

---

1 INTRODUCTION

IN traditional video mosaicing, a panoramic still image is created from video captured by a camera scanning a scene. The resulting panoramic image shows simultaneously objects that were photographed at different times. The observation that traditional mosaicing does not keep the original time of events helps us to generate richer representations of scenes.

Imagine a person standing in the middle of a crowded square looking around. When requested to describe his dynamic surroundings, he will usually describe ongoing actions. For example, "some people are talking in the southern corner, others are eating in the north," etc. This kind of description ignores the chronological order in which each activity was observed, focusing on the activities themselves instead.

The same principle of manipulating the progression of time while relaxing the chronological constrains may be used to obtain a flexible representation of dynamic scenes. It allows us not only to postpone or advance some activities, but also to manipulate their speed. Dynamic panoramas are indeed the most natural extension of panoramic mosaicing. But, dynamic mosaicing can be used also with a video taken from a static camera where we present a scheme to control the time progress for individual objects. We will start the description of temporal video manipulations in the case of a static camera, before we will continue to the case of dynamic panorama.

In our framework, the input video is represented as an aligned space-time volume. The time manipulations we explore are those that can be obtained by sweeping a 2D slice

(*time front*) through the space-time volume, generating a new sequence of images.

In order to analyze and manipulate videos of dynamic scenes, several challenging problems must be addressed: The first one is the stabilization of the input video sequence. In many cases, the field of view of the camera includes mostly dynamic regions, when even robust alignment methods fail. The second problem is that time slices in the space-time volume may pass through moving objects. As a result, visual seams and other visual artifacts may occur in the resulting movie. To reduce such artifacts, we use image-based optimization of the time fronts which favors seamless stitching. This optimization problem is formulated as one of finding the minimal cut in a 4D graph.

1.1 Related Work

The most popular approach for the mosaicing of dynamic scenes is to compress all the scene information into a single static mosaic image. There are numerous methods for dealing with scene dynamics in the static mosaic. Some approaches eliminate all dynamic information from the scene, as dynamic changes between images are undesired [25]. Other methods encapsulate the dynamics of the scene by overlaying several snapshots of the moving objects into the static mosaic, resulting in a "stroboscopic" effect [15], [12], [1]. In contrast to these methods that generate a single still mosaic image, we use mosaicing to generate a dynamic video sequence having a desired time manipulation.

The mechanism of slicing through a stack of images (which is essentially the space-time volume) is similar to video-cubes [16], which produces composite still images, and to panoramic stereo [20], [30]. Unlike these methods, dynamosaics are generated by coupling the scene dynamics, the motion of the camera, and the shape and the motion of the time front.

In [18], [9], two videos of dynamic textures (or the same video with two different temporal shifts) are being stitched seamlessly side by side, yielding a movie with a larger field of view. In this work we are interested in more general time

---

● *The authors are with the School of Computer Science and Engineering, The Hebrew University of Jerusalem, Ross Building, Giva Ram, 91904 Jerusalem, Israel. E-mail: {alexis, yaelpri, danix, peleg}@cs.huji.ac.il.*

manipulations, in which the edited movies combine information from many frames of the input sequence.

The basic idea of dynamosaicing was presented in an earlier paper [22]. Since dynamosaicing is concerned with dynamic scenes and since dynamic scenes present challenges both in alignment and in stitching, these topics are expanded substantially in this paper. A different approach toward seamless stitching in the case of dynamic textures (with the ability to produce infinite loops) was suggested in [2]. A discussion on the differences between the two approaches appears in Section 4.

1.2 An Overview of Dynamosaicing

Given a sequence of input video frames $I_1, \ldots, I_N$, they are first registered and aligned to a global spatial coordinate system. A specialized alignment scheme for sequences of dynamic scenes is described in Section 2, but other stabilization methods can sometimes be used (e.g., [5], [24]).

Stacking the aligned video frames along the time axis results in a 3D space-time volume $V(x, y, t)$. Fig. 2 shows two examples of 2D space-time volumes. For a static camera the volume is a rectangular box, while a moving camera defines a more general swept volume. In either case, planar slices perpendicular to the t axis correspond to the original video frames. A static scene point traces a line parallel to the t axis (for a static or panning camera), while a moving point traces a more general trajectory.

Sweeping the aligned space-time volume with various evolving time fronts can be used to manipulate the time flow of the input video in a variety of ways. A particularly interesting case is the one of creating dynamic panoramas with the time front shown in Fig. 6b. The time manipulations that may be obtained with the proposed scheme are discussed in Section 3.

Images are generated from a time front sweeping the space-time volume by interpolation. Simple interpolation as commonly used in mosaicing [21], [13] can produce visually appealing results in many cases, but the existence of moving objects in the scene (such as walking people) requires a special care to avoid visible seams in the output videos. This is done by modifying the time front to avoid seams inside moving objects in accordance to the minimization of an appropriate cost function. This stage is described in Section 4.

2 VIDEO ALIGNMENT USING VIDEO EXTRAPOLATION

An initial task that must be carried out before mosaicing is motion analysis for the alignment of the input video frames. Many motion analysis methods exist, some even offer robust motion computation that overcomes the presence of moving objects in the scene [5], [24], [7]. However, scenes which consist mostly of dynamic scenes are still problematic for existing methods. There are a few methods that address the stabilization of dynamic scenes [11], [26], but they address stochastic textures and cannot handle moving objects.

Unlike computer motion analysis, the human eye can easily distinguish between the motion of the camera and the internal dynamics in the scene. For example, when viewing a video of a sea, we can easily distinguish between the motion of the camera and the dynamics of the waves. The key to this human ability is an assumption regarding the simplicity and consistency of the scenes and of their

dynamics: It is assumed that when a video is aligned, the dynamics in the scene become smoother and more predictable. This allows humans to track the motion of the camera even when no apparent registration information exists. We therefore try to replace the "brightness constancy assumption" with a "dynamics constancy assumption."

This dynamics constancy assumption is used as a basis for our registration algorithm: Given a new frame of the sequence, it is aligned to best fit the extrapolation of the preceding frames. The extrapolation is done using video synthesis techniques [28], [10], [18], and the alignment is done using traditional methods for parametric motion computation [5], [14]. Alternating between video extrapolation and image alignment results in a registration algorithm which can handle complex dynamic scenes, having both dynamic textures and moving objects.

2.1 Dynamics Constancy Assumption

Let $V(x, y, t)$ be a space-time volume, consisting of frames $I_1, \ldots, I_N$. The "dynamics constancy" assumption implies that when the volume is aligned (e.g., when the camera is static), we can estimate a large portion of each image $I_n = V(x, y, n)$ from the preceding frames $I_1, \ldots, I_{n-1}$. We will denote the space-time volume constructed by all the frames up to the kth frame by $V(x, y, \overrightarrow{k})$. The "dynamics constancy" assumption states we can obtain the nth frame by extrapolating from the preceding $n - 1$ frames,

$$I_n(x, y) = V(x, y, n) \approx Extrapolate\left(V\left(x, y, \overrightarrow{n-1}\right)\right). \quad (1)$$

$Extrapolate$ is a nonparametric extrapolation function, estimating the value of each pixel in the new frame given the preceding space-time volume.

When the camera is moving, the image transformation induced by the camera motion should be added to this equation. Assuming that all frames in the space time volume $V(x, y, \overrightarrow{n-1})$ are aligned to the coordinate system of I_{n-1}, the new frame I_n can be approximated by

$$I_n \approx T_n\left(Extrapolate\left(V\left(x, y, \overrightarrow{n-1}\right)\right)\right). \quad (2)$$

T_n is a 2D image transformation between frames I_{n-1} and I_n and is applied on the extrapolated image. Applying the inverse transformation T_n^{-1} on both sides of the equation gives

$$T_n^{-1}(I_n) \approx Extrapolate\left(V\left(x, y, \overrightarrow{n-1}\right)\right). \quad (3)$$

This relation is used in our registration scheme.

2.2 Video Extrapolation

Our video extrapolation is closely related to dynamic texture synthesis [8], [3]. However, dynamic textures are characterized by repetitive stochastic processes and do not apply to more structured dynamic scenes, such as walking people. We therefore prefer to use nonparametric video extrapolation methods [28], [10], [18]. These methods assume that each small space-time block has likely appeared in the past and, thus, the video can be extrapolated using similar blocks from earlier video portions. This is demonstrated in Fig. 3.

Assume that the aligned space time volume $V(x, y, \overrightarrow{n-1})$ is given, and a new frame I_n^{pred} is to be estimated. For each pair

of space-time blocks W_p and W_q, we define the SSD (sum of square differences) to be:

$$d(W_p, W_q) = \sum_{(x,y,t)} (W_p(x,y,t) - W_q(x,y,t))^2. \quad (4)$$

As shown in Fig. 3, for each pixel (x,y) in frame I_{n-1} we define a 3D space-time block $W_{x,y,n-1}$ whose spatial center is at pixel (x,y) and whose temporal boundary is at time $n-1$ (frames which were not aligned yet can not be used). We then search in the space time volume $V(x,y,\overrightarrow{n-2})$ for a space-time block with the minimal SSD to block $W_{x,y,n-1}$. Let $W_p = W(x_p, y_p, t_p)$ be the most similar block, spatially centered at pixel (x_p, y_p) and temporally bounded by t_p. The value of the extrapolated pixel $I_n^{pred}(x,y)$ will be taken from $V(x_p, y_p, t_p+1)$, the pixel that appeared immediately after the most similar block. This scheme follows the "dynamics constancy" assumption: Given that two different space time blocks are similar, we assume that their continuations are also similar. While a naive search for each pixel may be exhaustive, the scheme can be significantly accelerated by focusing on a smaller set of image features. Additional modifications can further accelerate the process [23].

We used the SSD (sum of squared differences) as a distance measure between two space-time blocks, but other distance measures can be used such as the sum of absolute differences or more sophisticated measures [28]. We did not notice a substantial difference in registration results when changing the distance measure.

2.3 Alignment with Video Extrapolation

Alignment with video extrapolation can be described by the following steps:

1. Assume that the motion of the first K frames has already been computed, and let $n = K + 1$.
2. Align all frames in the space time volume $V(x,y,\overrightarrow{(n-1)})$ to the coordinate system of Frame I_{n-1}.
3. Estimate the next new frame by extrapolation from the previous frames

$$I_n^{pred} = Extrapolate\left(V\left(x,y,\overrightarrow{(n-1)}\right)\right).$$

4. Compute the motion parameters (the global 2D image transformation T_n^{-1}) by aligning the new input frame I_n to the extrapolated frame I_n^{pred}.
5. Increase n by 1 and return to Step 2. Repeat until reaching the last frame of the sequence.

The global 2D image alignment in Step 2, as well as the initialization step, are performed using direct methods for parametric motion computation [5], [14]. We usually used a motion model having image rotation and translation, which gave good results in the case of rotating cameras. Objects with depth parallax can be treated as moving objects when the camera motion varies slowly.

The initialization, in which the motion of the first K frames is computed, is done as follows: The entire video sequence is scanned to find K consecutive frames which are best suited for traditional alignment methods, e.g., frames where motion computation converges and having the smallest residual error. We used Lucas-Kanade alignment on blurred frames [5]. From these K frames, video extrapolation continues in the positive and negative time directions.

2.4 Masking Unpredictable Regions

Real scenes always have a few regions that can not be predicted. For example, people walking in the street often change their behavior in an unpredictable way, e.g., raising their hands or changing their direction. In these cases, the video extrapolation will fail, resulting in outliers. The alignment can be improved by masking out unpredictable regions.

This is done as follows: After the new input image I_n is aligned with the extrapolated image I_n^{pred} which estimated it, the color difference between the two images is computed. Each pixel (x,y) is masked out if the color difference in its neighborhood is higher than some threshold r (We usually used $r = 1$.)

$$\frac{\sum (I_n - I_n^{pred})^2}{\sum I_x^2 + I_y^2} > r. \quad (5)$$

The predictability mask is used in the alignment of frame I_{n+1} to frame I_{n+1}^{pred}.

2.5 Fuzzy Estimation

The alignment may be further improved by using fuzzy estimation. This is done by keeping not only the best candidate for extrapolating each pixel, but the best S candidates (we used up to five candidates for each pixel). The multiple estimations for extrapolating each pixel can be combined using a summation of the error terms

$$T_n = \arg\min_T \left\{ \sum_{x,y,s} \lambda_{x,y,s} (T_n^{-1}(I_n)(x,y) - I_n^{pred}(x,y,s))^2 \right\}, \quad (6)$$

where $I_n^{pred}(x,y,s)$ is the sth candidate for the value of the pixel $I_n(x,y)$. The weight $\lambda_{x,y,s}$ of each candidate is based on the difference of its corresponding space-time cube from the current one as defined in (4) and is given by

$$\lambda_{x,y,s} = e^{\frac{-d(W_p,W_q)^2}{2\sigma^2}}.$$

We almost always used $7 \times 7 \times 7$ space-time cubes and $\sigma = 1/255$ to reflect the noise in the image gray levels. Note that the weights for each pixel do not necessarily sum to one and, therefore, the registration mostly relies on the predictable regions. Also, other ways to combine different predictions are also possible.

2.6 Handling Alignment Drift

Alignment based on video extrapolation follows Newton's First Law: An object in uniform motion tends to remain in that state. If we initialize our registration algorithm with a small motion relative to the real camera motion, our method will continue this motion for the entire video. In this case, the background will be handled as a slowly moving object. This is not a bug in the algorithm, but rather a degree of freedom resulting from the "dynamics constancy" assumption.

This degree of freedom can be eliminated by incorporating a prior bias, assuming that part of the scene is static. This is done by adding a new predictive static candidate $S + 1$ at every pixel (by simply copying the value of the previous frame). In our experiments, we gave a small weight of 0.1 to the static candidate relative to the total weight of the pixel. In this way, we have prevented the drift without effecting the accuracy of the motion computations.

Fig. 1. Dynamosaicing can create dynamic panoramic movies of a scene. This figure shows a single frame in a panoramic movie, generated from a video taken by a panning camera (420 frames). When the movie is played (see www.vision.huji.ac.il/dynmos), the entire scene comes to life, and all water flows down simultaneously.

2.7 Examples

The sequence shown in Fig. 4 was used by [26] and by [11] as an example for their registration of dynamic textures. The global motion in this sequence is a horizontal translation and the true displacement can be computed from the motion of one of the flowers. The displacement error reported by [26] was 29.4 percent of the total displacement between the first and last frames, while the error of our methods was only 1.7 percent.

Fig. 5 shows an examples of video registration using extrapolation in a challenging scene. In this scene, masking out the unpredictable regions (parts of the falls and the fumes), as described in Section 2.4, was important for obtaining a good registration.

3 EVOLVING TIME FRONTS

3.1 Mosaicing by an Evolving Time Front

Image mosaicing is the process of creating novel images by selecting patches from the frames of the input sequence and combining them to form a new image ([21], [13], [1] are just a few examples of the wide literature on mosaicing). It can be described by a function $M(x, y)$ that maps each pixel (x, y) in the output image S to the input frame from which this pixel is taken and its location in that frame. In this work, we focus only on temporal warping, that is $S(x, y) = V(x, y, M(x, y))$,

Fig. 2. Two-dimensional space-time volumes: Each frame is represented by a 1D row and the frames are aligned along the global x axis. (a) A static camera defines a rectangular space-time region, while (b) a moving camera defines a more general swept volume.

where $V(x, y, t)$ is the aligned space-time volume. This function can be represented by a continuous slice (*time slice*) in the space-time volume as illustrated in Fig. 6. A time slice determines the mosaic patches by its intersection with the frames of the original sequence at the original discrete time values (shown as dashed lines in Fig. 6).

To get a desired time manipulation we specify an *evolving time front*: A free-form surface that deforms as it sweeps through the space-time volume. Taking snapshots of this surface at different times results in a sequence of time slices that are represented by temporal-shift functions $S_k(x, y) = V(x, y, M_k(x, y))$.

3.2 What Time Manipulations Can Be Obtained?

In this section, we describe the manipulation of chronological time versus local time using dynamosaicing. We first describe the dynamic panoramas, where the chronological time is eliminated. This application inspired this work. We then show other applications where a video should be edited in a way that changes the chronological order of events in the scene. The realistic appearance of the movie is kept by preserving the time flow locally, even when the global chronological time is being changed.

3.2.1 Panoramic Dynamosaicing

Panoramic dynamosaics may be generated using the approach described above with the time slices shown in Fig. 6b. Assuming that the camera is scanning the scene from left to right, the first mosaic in the sequence will be constructed from strips taken from the right side of each input frame, showing regions as they first appear in the field of view (see Fig. 7). The last mosaic in the resulting sequence will be the mosaic image generated from the strips on the left, just before a region disappears from the field of view. Between these two marginal slices of the space-time volume, we take intermediate slices, smoothly showing regions from their appearance to their disappearance. Each of the mosaic images is a panoramic image and the resulting movie is a dynamic panorama in which local time is preserved. Fig. 1 shows a single panorama from such a movie.

Panoramic dynamosaics represent the elimination of the chronological time of the scanning camera. Instead, all regions appear simultaneously according to the local time of their visibility period: From their first appearance to their disappearance. But, there is more to time manipulation than eliminating the chronological time.
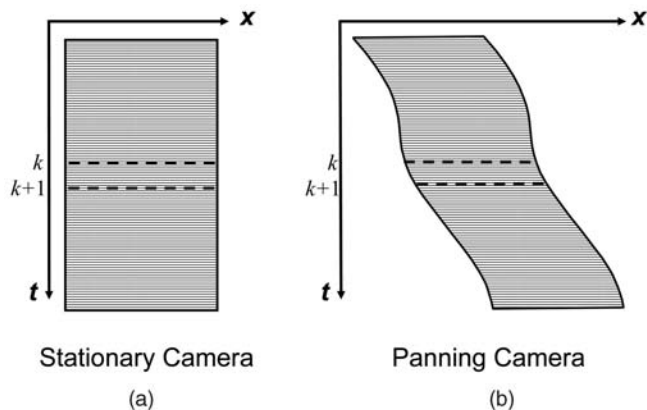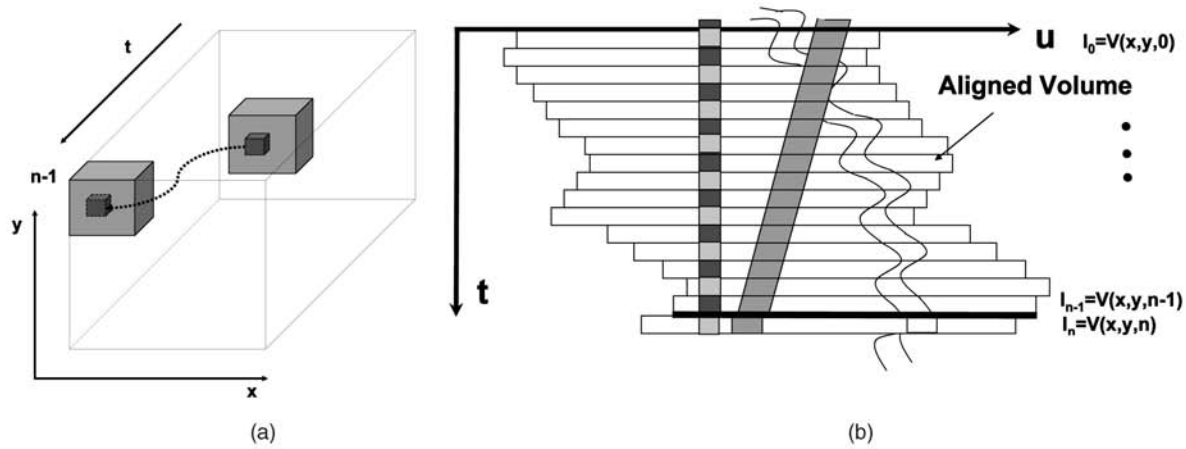
Fig. 3. Video extrapolation using a space-time block search. Both motion and intensity variation are accounted for. (a) For all blocks bordering on time $(n-1)$, we search for the best matching block in the space-time volume. Once such a block is found, the pixel in front of this block is copied to the corresponding position in the extrapolated frame $I_n^p(x,y)$. (b) The new frame I_n is not aligned to Frame I_{n-1}, but to the frame that has been extrapolated from the preceding space-time volume. This extrapolation is based on image features with repetitive behavior, such as the ones shown in this figure.

Figs. 1 and 8 show examples of panoramic dynamosaics for different scenes. To generate the panoramic movies corresponding to Figs. 1 and 8, simple slices, as the one demonstrated in Fig. 6b, were used. Since it is impossible to visualize the dynamics effects in these static images, we urge the reader to examine the video clips at www.vision.huji.ac.il/dynmos.

3.2.2 Advancing Backward in Time

This effect is best demonstrated with the waterfalls sequence (Fig. 1), which was scanned from left to right by a video camera. If we want to reverse the scanning direction, we can

Fig. 4. A sequence of moving flowers taken by a panning camera. See http://www.robots.ox.ac.uk/~awf/iccv01/. Our motion computation with video extrapolation gave an accumulated translation error of 1.7 percent between the first and last frames, while [26] reported an accumulated error of 29.4 percent.

Fig. 5. (a) This waterfall sequence poses a challenging task for registration, as most of the scene is covered with falling water. The video was stabilized using video extrapolation (using a rotation and translation motion model). (b) An average of 40 frames in the stabilized video is shown to evaluate the quality of the stabilization. The dynamic regions are blurred only in the flow direction, while the static regions remain relatively sharp after averaging.

simply play the movie backward. However, playing the movie backward will result in the water flowing upward.

At first glance, it seems impossible to play a movie backward without reversing its dynamics. Yet, this can also be achieved by manipulating the chronological time, while preserving the local dynamics. Looking at panoramic dynamosaics, one can claim that all objects are moving simultaneously and the scanning direction does not have any role. Thus, there must be some kind of symmetry, which enables us to convert the panoramic movie into a scanning sequence in which the scanning is at any desired direction and speed.

Indeed, the simple slicing scheme shown in Fig. 9 reverses the scanning direction while keeping the dynamics of the objects in the scene. In the water falls example, the scanning direction is reversed, but the water continues to flow down!

3.2.3 Time Manipulations with Planar Time Fronts

The different types of time manipulations that can be obtained with planar time fronts are described in Fig. 10. The time fronts always sweep "downward" in the direction of positive time at the original speed to preserve the original local time.
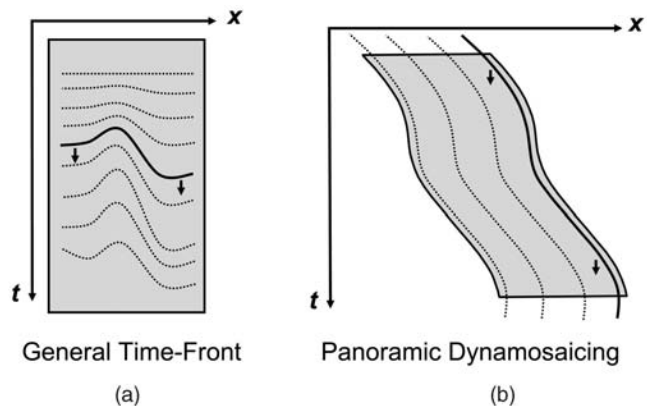

Fig. 6. Slicing the space-time volume: (a) Snapshots of an evolving time front surface produce a sequence of time slices; each time slice is mapped to produce a single output video frame. (b) The particular time flow for generating dynamic panoramas from a panning camera.
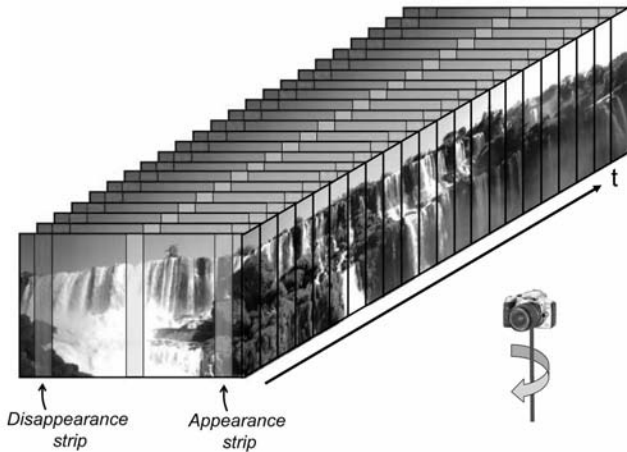
Fig. 7. Input frames are stacked along the time axis to form a space-time volume. Given frames captured with a video camera panning clockwise, panoramic mosaics can be obtained by pasting together vertical strips taken from each image. Pasting together strips from the right side of the images will generate a panoramic image where all regions appear as they first enter the sequence, regardless of their chronological time.
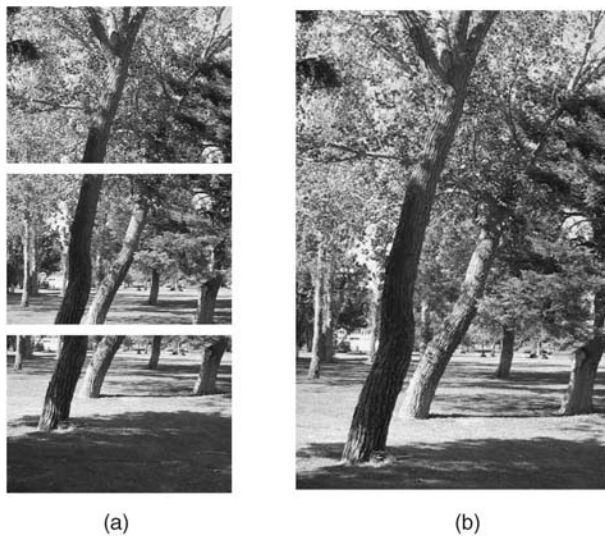

Fig. 8. A dynamic panorama of a tree whose leaves are blowing in the wind. (a) Three frames from the sequence (out of 300 frames), scanning the tree from the bottom up. (b) A single frame from the resulting dynamosaic movie.

The different time fronts, as shown in Fig. 10, can vary both in their angles relative to the x axis and in their lengths. Different angles result in different scanning speeds of the scene. For example, maximum scanning speed is achieved with the panoramic slices. Indeed, in this case the resulting movie is very short, as all regions are played simultaneously. (The scanning speed should not be confused with the dynamics of each object, which preserve the original speed and direction.)

The field of view of the resulting dynamosaic frames may be controlled by cropping each time slice as necessary. This can be useful, for example, when icreasing the scanning speed of the scene while preserving the original field of view.

3.3 Temporal Video Editing

Consider a space-time volume generated from a video of a dynamic scene captured by a static camera (as in Fig. 2a).
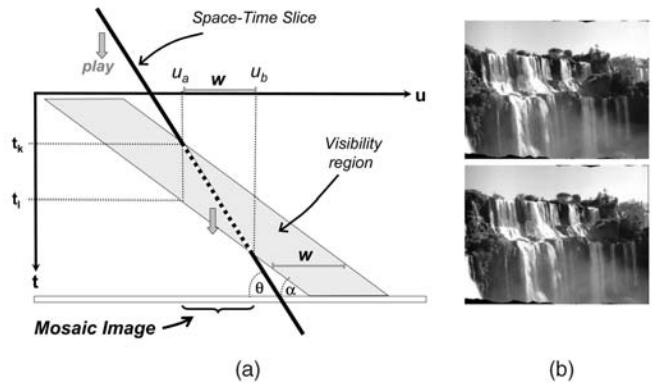

Fig. 9. (a) A slicing scheme that reverses the scanning direction using a time front whose slope is twice the slope of the occupied space-time region $(\tan \theta = 2 \tan \alpha)$. The width of the generated mosaic image is w, the same as that of the original image. Sweeping this time front in the positive time direction (down) moves the mosaic image to the left, in the opposite direction to the original scan. However, each region appears in the same relative order as in the original sequence: u_a first appears in time t_k, and ends in time t_l. (b) Two frames from an edited movie. The scanning direction of the camera was reversed, but the water continues to flow down. The entire video appears at www.vision.huji.ac.il/dynmos.
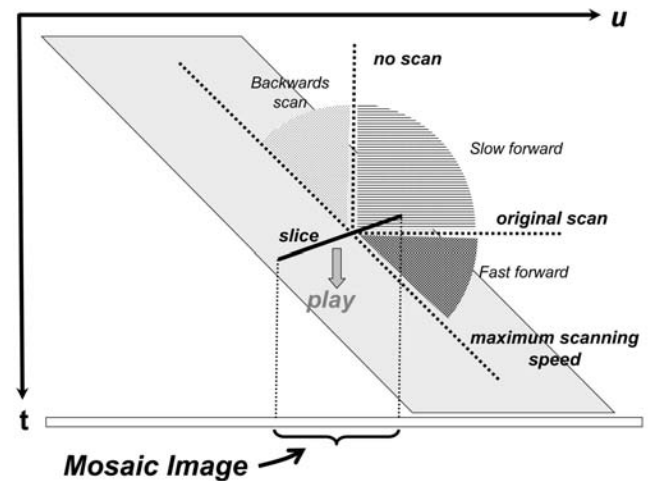

Fig. 10. The effects of various planar time fronts. While the time front always sweeps in a constant speed in the positive time direction, various time front angles will have different effects on the resulting video.

The original video may be reconstructed from this volume by sweeping forward in time with a planar time front perpendicular to the time axis. We can manipulate dynamic events in the video by varying the shape and speed of the time front as it sweeps through the space-time volume.

Fig. 11 demonstrates two different manipulations of a video clip capturing the demolition of a stadium. In the original clip the entire stadium collapses almost uniformly. By sweeping the time front as shown in Fig. 11c the output frames use points ahead in time towards the sides of the frame, causing the sides of the stadium to collapse before the center (Fig. 11a). Using the time front evolution in Fig. 11d produces a clip where the collapse begins at the dome and spreads outward, as points in the center of the frame are taken ahead in time. It should be noted that Agarwala et al. [1] used the very same input clip to produce still *time-lapse mosaic* images where time appears to flow in different directions (e.g., left-to-right or top-to-bottom). In contrast, our approach generates entire new dynamic video clips.
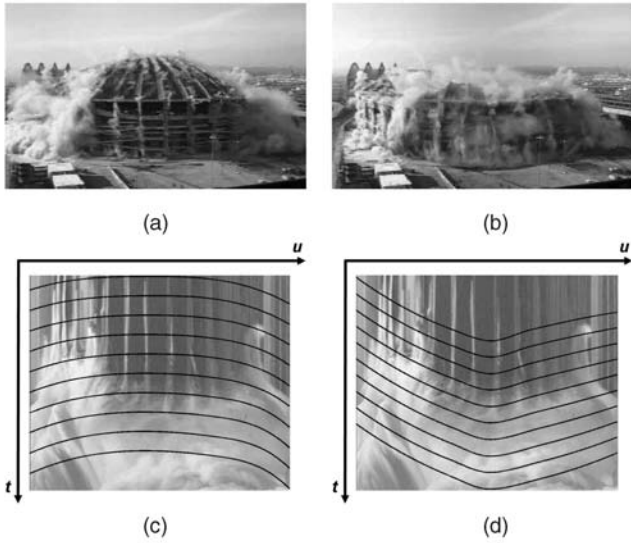
(a)

(b)

(c)

(d)

Fig. 11. (a) and (b) are frames from two video clips, generated from the same original video sequence with different time flow patterns. (c) and (d) show several time slices superimposed over a $x - t$ slice passing through the center of the space-time volume. The full video clips are available at www.vision.huji.ac.il/dynmos.
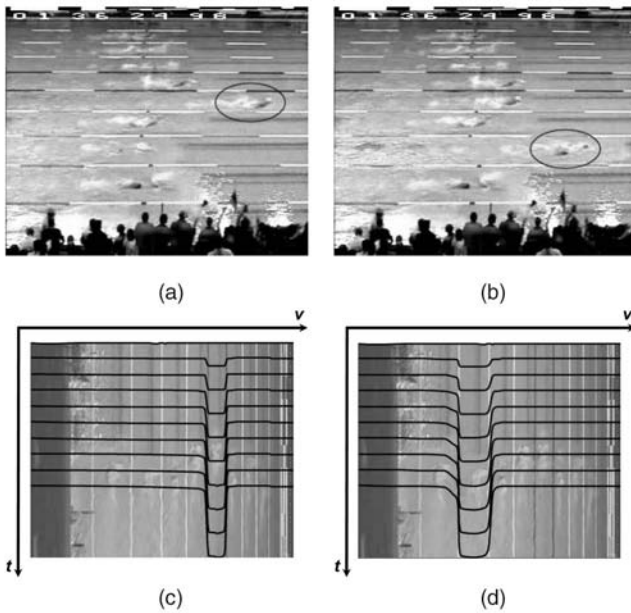

(a)

(b)

(c)

(d)

Fig. 12. Who is the winner of this swimming competition? Temporal editing enables time to flow differently at different locations in the video, creating new videos with any desired winner, as shown in (a) and (b). (c) and (d) show several time slices superimposed over a $y - t$ slice passing through the center of the space-time volume. In each case, the time front is offset forward over a different lane, resulting in two different "winners." The full video clips are available at www.vision.huji.ac.il/dynmos.

Another example is shown in Fig. 12. Here, the input is a video clip of a swimming competition, taken by a stationary camera. By offsetting the time front at regions of the space-time volume corresponding to a particular lane one can speed up or slow down the corresponding swimmer, thus altering the outcome of the competition at will. The shape of the time slices used to produce this effect is shown as well.

In this example, we took advantage of the fact that the trajectories of the swimmers are parallel. In general, it is not necessary for the trajectories to be parallel, or even linear, but
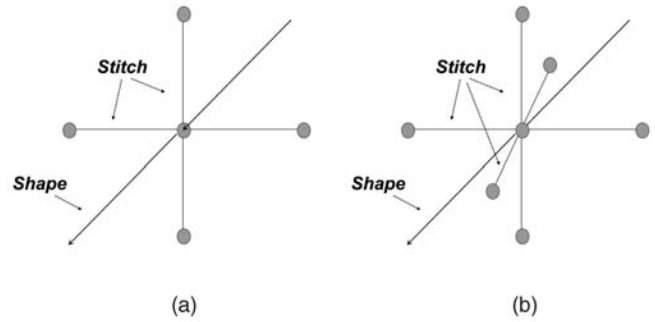

(a)

(b)

Fig. 13. Two types of edges are used in the graph-cut formulation of the mosaicing problem: "Shape" edges and "Stitching" edges. The "Shape" edges penalize deviations from the ideal shape of the time front, while (a) the "Stitching" edges (marked with circles) encourage spatial consistency for the case of a single time front, and (b) both spatial and temporal consistency in the case of an evolving time front.
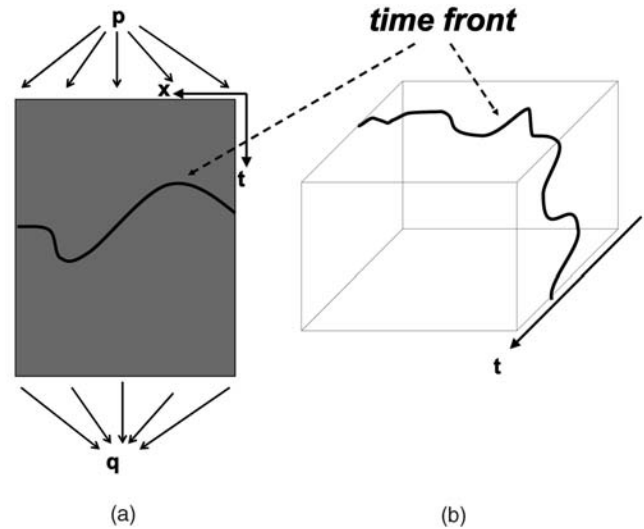

(a)

(b)

Fig. 14. (b) A single time slice in the space-time volume can be represented as a cut in a 3D graph corresponding to this volume. (a) A top view of such a cut is shown. With the proposed cost, the cut can be minimized by maximizing the flow from p (Source vertex) to q (Sink vertex).

it is important that the tube-like swept volumes that correspond to the moving objects in space time do not intersect. If they do, various anomalies, such as duplication of objects, may arise.

4 SEAMLESS DYNAMOSAICING USING GRAPH-CUTS

The mosaicing described in the previous section may result in visual artifacts from seams at the middle of moving objects. An analysis of "Doppler Effect" distortions for objects moving in the same direction of camera motion is described in [22]. In our experiments, we found out that for dynamic textures (such as flowing water or trees), the simple slicing scheme described in the previous section was sufficient to create impressive time manipulations without noticeable artifacts. Yet, when longer objects appeared in the scene, such as walking people, the artifacts became much more apparent. Even minor time manipulations may cause distortions due to stitching inside moving objects, resulting in a visually disturbing seams. See Fig. 17a.

The distortions of objects and the visual seams can be significantly reduced by taking into consideration that stitching inside dynamic regions should be avoided. In order to do so, we define a cost function whose minimization determines the optimal time slice surface. This cost function balances between the minimization of a "stitching" cost and the maximization of the similarity of the time front to its desired ideal shape.

A common way to represent and solve such problems is by multilabel graphs, where the labels of each vertex denote all possible time shifts for that pixel [1], [2]. Unfortunately, the general formulation results in an NP-hard problem, whose approximation requires intensive computations, such as using loopy belief propagation [27] or iterative graph-cuts [6]. This becomes prohibitive for the case of video editing, where the cost is minimized over a 3D graph. In [2], [29], the computational time was reduced by assuming that all the time fronts and all the pixels in a single column have the same time-shift, resulting in a 1D problem which can be solved using dynamic programming. In [2], the solution was further enhanced by passing several candidates for each pixel and applying the full optimization to those candidates. This approach can handle dynamic textures or objects that move horizontally but may fail for objects having a more general motion.

We take a different approach which can be implemented without reducing the dimensionality of the problem. In our method, we assume that the desired time front is continuous in the space-time volume, i.e., neighboring pixels have similar temporal shifts. Based on this assumption, we formulate the problem as the one of minimizing a cost function defined on a 4D graph. With this formulation, the problem can be solved in polynomial time as shown in the next section.

It is interesting to compare dynamosaicing with the PVT approach [2]. The PVT approach, with its ability to have discrete jumps in time, is most effective with repetitive stochastic textures and with its ability to generate infinite dynamics. When the scene has moving objects each having a given structure, e.g., moving people, discrete time jumps may result in unacceptable distortions and discontinuities. In this case, dynamosaicing with the continuous time fronts is more applicable. Continuous time fronts are also more robust in cases of error in camera motion. In addition, while the PVT approach perform best with camera that jumps from one stable position to another stable position, dynamosaicing works best with smooth camera motion.

4.1 A Single Time Front

In this section, we will examine the creation of a single seamless image, while keeping the general shape of the ideal time front that corresponds to the desired time-manipulation. Movie generation will be addressed later.

We assume that the input sequence has already been aligned to a single reference frame and stacked together along the time axis to form an aligned space-time volume $V(x, y, t)$. For simplicity, we will also assume that all the frames after alignment are of the same size. Pixels outside the field of view of the camera will be marked as impossible.

The output image S is created from the input movie according to a time front which is represented by a function $M(x, y)$. The value of each pixel $S(x, y)$ is taken from $V(x, y, M(x, y))$ in the aligned space-time volume. To produce a seamless mosaic, we modify its ideal shape (e.g.,

as computed in Section 3) qaccording to the moving objects in the scene. We define a cost function on the time shifts $M(x, y)$. The general form of this cost function is

$$E(M) = E_{shape}(M) + \alpha E_{stitch}(M). \qquad (7)$$

The term E_{shape} attracts the time front to follow its predefined shape, the term E_{stitch} works to minimize the stitching artifacts, and α balances between the two. (We used $\alpha = 0.3$ when gray values were between 0-255.) When the image dynamics is only a dynamic texture, such as water or smoke, α should be small.

To create panoramas, E_{shape} can constrain the time front to pass through the entire sequence, yielding a panoramic image. For more general time manipulations, we can use the time fronts described in Section 3.2 as shape priors. Let M' be the ideal time front (for example, a time front determined by the user) then E_{shape} may be defined as

$$E_{shape}(M) = \sum_{x,y} \|M(x, y) - M'(x, y)\|^r, \qquad (8)$$

where $r \geq 1$ can be any norm. We usually used the l_1 norm and not the l_2 norm in order to obtain a robust behavior of the time front, making it follow the original time front unless it cuts off parts of a moving object.

The second term, E_{stitch} addresses the minimization of the stitching artifacts. It is based on the cost between each pair of neighboring output pixels (x, y) and (x', y'). Without loss of generality, we assume that $M(x, y) \leq M(x', y')$

$$E_{spatial}(x, y, x', y') = \sum_{k=M(x,y)}^{M(x',y')-1} \frac{1}{2} \|V(x, y, k) - V(x, y, k+1)\|^2$$
$$+ \frac{1}{2} \|V(x', y', k) - V(x', y', k+1)\|^2. \qquad (9)$$

This cost is zero when the two adjacent points (x, y) and (x', y') come from the same frame $(M(x, y) = M(x', y'))$. When $M(x, y) \neq M(x', y')$, this cost is zero when the colors of (x, y) and (x', y') do not change and it increases based on the dynamics at those pixels.

The global stitching cost for the time front M is given by

$$E_{stitch}(M) = \sum_{(x,y)} \sum_{(x',y') \in N(x,y)} E_{spatial}(x, y, x', y'), \qquad (10)$$

where:

- $N(x, y)$ are the pixels in the neighborhood of (x, y).
- $E_{spatial}(x, y, x', y')$ is the stitching cost for each pair of spatially neighboring pixels (x, y) and (x', y'), as described in (9).

Note that the cost in (9) differs from traditional stitching costs (for example, [1]) where there is no summation, but only the two time-shifts $M(x, y)$ and $M(x', y')$ are used. The cost in (9) is reasonable when the time front is continuous, which means that if (x, y) and (x', y') are neighboring pixels, their source frames $M(x, y)$ and $M(x', y')$ are close in time. The main advantage of the cost in (9) is that its global minimum can be found in polynomial time using a min-cut as will be described below.

When the camera is moving, some pixels in the space-time volume $v(x, y, t)$ may not be in the field of view. We do
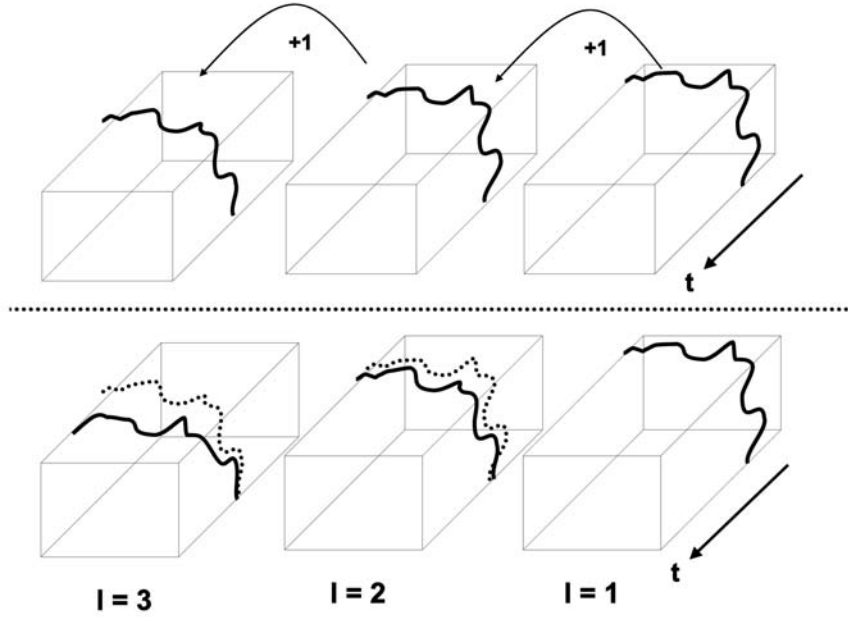
Fig. 15. An evolving time front is computed using a 4D graph which consists of L instances of the 3D graph used to compute a single time slice. (a) Sweeping the space-time volume with a stationary time front is equivalent to setting a shift of 1 between consecutive time slices. (b) When the time front evolves, the shift between consecutive time slices varies. Temporal edges between the 3D graphs may be added to enforce temporal consistency.

not assign vertices to such pixels, therefore, only pixels in the field of view are used in the panoramic image.

4.2 A Single Time Front as 3D Min-Cut

A 3D directed graph G is constructed from the aligned space-time volume, such that each location (x, y) at frame k is represented by a vertex (x, y, k). A cut in this graph is a partitioning of the nodes into two sets P and Q. Further, assume that two vertices, the source $p \in P$ and the sink $q \in Q$, have been distinguished. Given a time front $M(x, y)$, the corresponding cut in G is defined as follows:

$$\begin{cases} (x, y, k) \in P & if M(x, y) \leq k \\ (x, y, k) \in Q & otherwise. \end{cases} \quad (11)$$

In the other direction, given a cut $\{P, Q\}$, we define $M(x, y)$ as $\min_k\{(x, y, k) \in Q\}$. A cost of a cut $\{P, Q\}$ is defined to be $\sum_{u \in P, v \in Q} w(u, v)$, where $w(u, v)$ is the weight of the edge $u \to v$. We will assign weights to edges between neighboring vertices in G such that the cost of each cut in G will be equal to the cost of the corresponding time front $M(x, y)$.

Before we turn to describing the edge weights reflecting E_{stitch} and E_{shape}, we need to ensure that there is a 1:1 correspondence between cuts in G and assignment of M. To do so, we set infinite weights to the edges $(x, y, k + 1) \to (x, y, k)$, and to the edges $(x, y, N) \to q$. These edges prevents cuts in which $(x, y, k + 1) \in Q$ but $(x, y, k) \in P$, which are the only cuts that do not correspond to assignments of M.

4.2.1 Assigning Weights to the Graph Edges

The cost term E_{shape} measures the distance to the ideal shape of the time front. As seen in (8), this cost consists of terms which depend on the assignment of single variables $(M(x, y))$. To reflect this cost term, we add directed edges from (x, y, k) to $(x, y, k + 1)$ with the weights $\|k + 1 - M'(x, y)\|^r$ (M' corresponds to the prior time front, $r \geq 1$ is a

norm). We also add edges from the Source p to $(x, y, 1)$ with the weights $\|1 - M'(x, y)\|^r$. The sum of weights in the each cut gives E_{shape} for the corresponds assignment of time front.

To take into account the stitching cost E_{stitch}, we add edges (in both directions) between each adjacent pair of pixels (x, y) and (x', y') and each k with the following weight:

$$\frac{1}{2}\|V(x, y, k) - V(x, y, k + 1)\|^2$$
$$+ \frac{1}{2}\|V(x', y', k) - V(x', y', k + 1)\|^2.$$

These edges are shown in Fig. 13. It can be seen that the given a cut, the sum of weights of these edges equals to the stitching cost given in (9). Note, however, that this equivalence is not true for traditional stitching costs (used, for example, in [2], [1]) but only for our cost function.

4.2.2 Computing the Best Assignment M

The minimal cut in G can be computed in polynomial time using min-cut [17]. From the construction of the graph, the cost of a cut in G equals to the corresponding cost defined on the original 2D graph. Therefore, the best assignment of time slice M can be found efficiently using a min-cut, as shown in Fig. 14.

It should be noted that although it seems as if the complexity of the problem was increased by the conversion from a 2D problem to a 3D one, the total number of labels in the original 2D formulation equals to the number of vertices in G.

4.3 Evolving Time Front as a 4D Min-Cut

To create a new movie (of length L), we have to sweep the space-time volume with an evolving time front, defining a sequence of time-slices $M_1, \ldots, M_L$. This is shown in Fig. 15. One way to control the time slices is using the ideal shape of the time front as a shape prior. In this case, each time slice M_l is computed independently according to the ideal shape of the
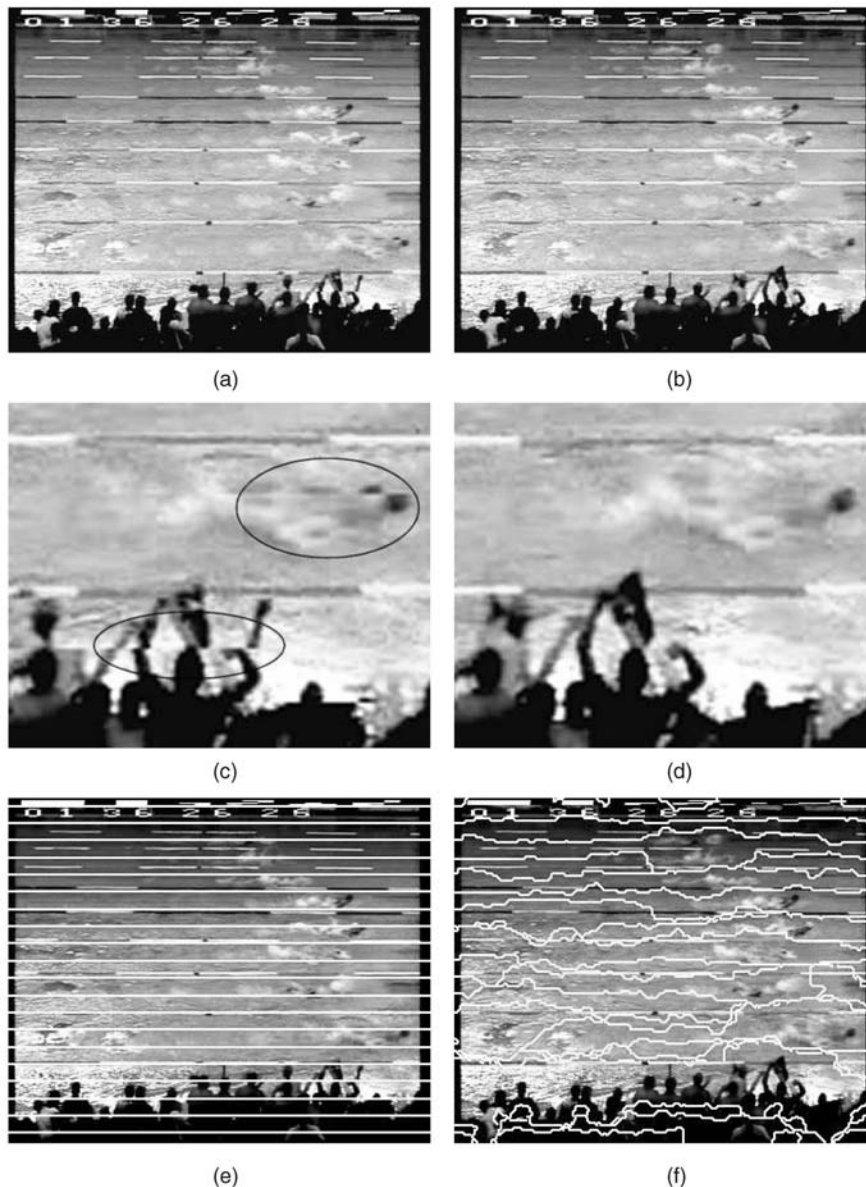
Fig. 16. Time manipulations using min-cut: The shape of the ideal time front was roughly determined so as to accelerate close-by regions. (a) One frame from the output movie when using the ideal time front. (b) Minimizing the energy function in (7) using a min-cut, made most visible seams disappear. (c) and (d) The differences are better seen in the magnified versions. Some cut-offs are circled. (e) and (f) The seams of the time slices used to generate the frames shown in (a) and (b) are marked on top of the frames. The seams in the left figure are uniformly distributed, as the ideal shape of each time slice was set to be linear. Note how the min-cut avoids stitching inside the swimmers. (See www.vision.huji.ac.il/dynmos for the full video clip.)

corresponding time slice and according to the stitching constraints, as was described in the previous section. An example for a time manipulation that can be obtained in this way is shown in Fig. 16. The ideal time front evolved in a way that made nearby regions move faster, while the exact shape of the each time slice was determined using a min-cut to avoid visible seams.

When the time manipulation aims to preserve the dynamics of the original movie (as is the case in producing panoramic movies), a better control can be obtained by adding temporal consistency constraints that avoids "jumps" in the output sequence, and minimizing the cost for all the time-slices at once. We first describe the modified stitching cost that involves also temporal consistency and later show how it may be solved using min-cut.

4.3.1 Preserving Temporal Consistency

Temporal consistency can be encouraged by setting for each pair of temporally neighboring pixels (x, y, t) and $(x, y, t + 1)$ the following cost (assuming that $M_t(x, y) \leq M_{t+1}(x, y)$):

$$E_{temporal}(x, y, t) = \sum_{k=M_t(x,y)+1}^{M_{t+1}(x,y)-1} \frac{1}{2} \|V(x, y, k) - V(x, y, k+1)\|^2 \qquad (12)$$
$$+ \frac{1}{2} \|V(x, y, k+1) - V(x, y, k+2)\|^2.$$

This formulation reflects temporal consistency in both past and future. This cost is zero for $M_{t+1}(x, y) = M_t(x, y) + 1$, an assignment which preserves the temporal relations of the

(a)

(b)

(c)

Fig. 17. A dynamic panorama of a crowd looking at a street performer. The performer was swaying quickly forward and backward. (a) Therefore, linear time front resulted in a distorted dynamic panorama. The distortions disappear using the 4D min-cut as shown in (b). The seams for that image are marked in (c).

original movie. The global stitching cost for the time front M is now given by

$$E_{stitch}(M) = \sum_l E_{stitch}(M_l) + \sum_{(x,y,t)} E_{temporal}(x,y,t), \quad (13)$$

where $E_{stitch}(M_l)$ is the global spatial stitching cost defined in (10).

4.3.2 Computing the Evolving Time Front Using Min-Cut

As was done for the case of a single time front, the cost defined for the evolving time front can be formulated as a cut in a directed graph G'. The 4D graph $G'(x,y,k,l)$ is constructed from the aligned space-time volume, such that each location (x,y) at input frame k and output frame l is represented by a vertex. A cut $\{P,Q\}$ that corresponds to the set of time slices $M_1(x,y),\ldots,M_L(x,y)$ is defined as follows:

$$\begin{cases} (x,y,k,l) \in P & if\, M_l(x,y) \leq k \\ (x,y,k,l) \in Q & otherwise. \end{cases} \quad (14)$$

As the 4D graph G' is very similar to L instances of the 3D graph G (described in the previous section), we describe only the modifications that should be done to obtain G'. To reflect the modified stitching cost given in (13), the edges (in

both directions) between each pair of temporal neighbors (x,y,k,l) and $(x,y,k+1,l+1)$ are assigned with the following weights:

$$\frac{1}{2}\|V(x,y,k) - V(x,y,k+1)\|^2$$
$$+ \frac{1}{2}\|V(x,y,k+1) - V(x,y,k+2)\|^2. \quad (15)$$

The minimal cut of this graph corresponds to a set of time-slices $M_1,\ldots,M_L$ which implement the desired time manipulation while keeping a seamless movie.

4.3.3 Flow-Based Temporal Consistency

A variant of this algorithm is to enforce temporal consistency by assigning weights to edges between pixels according to the optical flow at that pixels, instead of using temporal consecutive pixels. (See [4] regarding methods to compute optical flow).

Let (x,y) be a pixel at the kth frame. Let (x',y') be the corresponding location at frame $k-1$ according to the flow from frame k to frame $k-1$, and let (x'',y'') be the corresponding location at frame $k+1$ according to the flow from frame k to frame $k+1$.
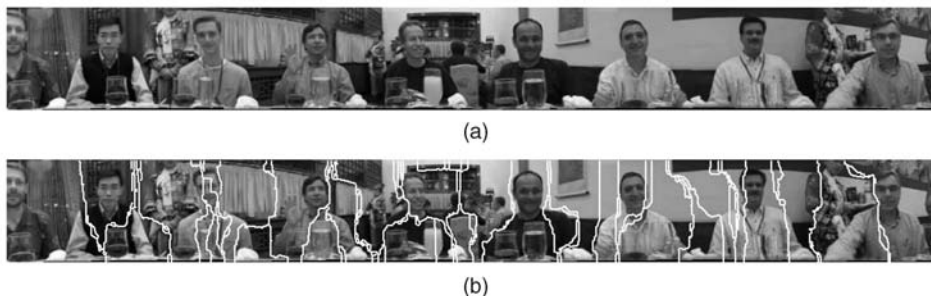
Fig. 18. When the camera is translating, the dynamics in the scene consists of both moving objects and the parallax. Both are treated in the same manner using the 4D min-cut. Gradient domain composition [1] handled variations in illumination. (a) A frame from panoramic movie (the entire video clip is available at www.vision.huji.ac.il/dynmos). (b) The min-cut avoids stitching inside moving objects or inside foreground objects (which have high disparity due to parallax).

To enforce temporal consistency we do the following:

- Edges $(x, y, k, l) \rightarrow (x', y', k-1, l-1)$ are assigned with the weights:

$$\frac{1}{2} \|V(x', y', k+1) - V(x', y', k)\|^2.$$

- Edges $(x, y, k, l) \rightarrow (x'', y'', k+1, l+1)$ are assigned with the weights:

$$\frac{1}{2} \|V(x'', y'', k+2) - V(x'', y'', k+1)\|^2.$$

The reason we had to separate between the two directions ("past" edges versus "future" edges) is that the forward flow and the inverse flow are not necessarily the same.

The advantage of the flow-based temporal consistency over the simpler approach is that the older one encourages the time fronts to remain static unless necessary, while the optical-flow-based approach encourages the time fronts to evolve in a more natural way according to the flow in the scene.

4.4 Accelerations

The memory required for saving the 4D graph may be too large. For example, the input movie that was used to create the panoramic movie shown in Fig. 18 consists of 1,000 frames, each of size 320×240. Constructing the graph would require prohibitive computer resources. We therefore suggest several modifications that reduce both the memory requirements and the runtime of the algorithm:

- We solve only for a sampled set of time slices, giving a sparser output movie, and interpolate the stitching function between them. (This acceleration is possible when the motion in the scene is not very large.)
- We can constrain each pixel to come only from a partial set of input frames. This is very reasonable for sequences taken from a video, where their is a lot of redundancy between consecutive frames. (It is important though to sample the source-frames in a consistent way. For example, if the frame k is a candidate source for pixel (x, y) in the one output frame, then the frame $k+1$ should be a candidate for pixel (x, y) in the successive output frame.)

- We use an hierarchical framework, where a coarse solution is found for low resolution images, and the solution is refined at higher resolution levels only along the boundaries. Similar accelerations were also used in [2], and are discussed in [19].

5 CONCLUDING REMARKS

It was shown that by relaxing the chronological constraints of time, a flexible representation of dynamic videos can be obtained. Specifically, when the chronological order of events is no longer considered a hard restriction, a wide range of time manipulations can be applied. An interesting example is creating dynamic panoramas where all events occur simultaneously and the same principles hold even for videos taken by a static camera.

Manipulating the time in movies is performed by sweeping an evolving time front through the aligned space-time volume. The strength of this approach is that accurate segmentation and recognition of objects are not needed. This fact significantly simplifies and increases the robustness of the method. This robustness comes at a cost of limiting the time manipulations that can be applied on a given video. Assume that one moving object occludes another moving object. With our method, the concurrency of the occlusion must be preserved for both objects.

In order to overcome this limitation and allow independent time manipulations even for objects that occlude each other, very good object segmentation and tracking is needed. In addition, methods for video completion should be used.

It is interesting to compare dynamosaicing with the PVT approach [2]. The PVT approach, with its ability to have discrete jumps in time, is most effective with repetitive stochastic textures and with its ability to generate infinite dynamics. When the scene has moving objects each having a given structure, e.g., moving people, dynamosaicing with the continuous time fronts is more applicable. Continuous time fronts are also more robust in cases of error in camera motion. In addition, while the PVT approach perform best with camera that jumps from one stable position to another stable position, dynamosaicing works best with smooth camera motion.

REFERENCES

[1] A. Agarwala, M. Dontcheva, M. Agrawala, S. Drucker, A. Colburn, B. Curless, D. Salesin, and M. Cohen, "Interactive Digital Photo-montage," *Proc. ACM SIGGRAPH '04,* pp. 294-302, Aug. 2004.

[2] A. Agarwala, C. Zheng, C. Pal, M. Agrawala, M. Cohen, B. Curless, D. Salesin, and R. Szeliski, "Panoramic Video Textures," *Proc. ACM SIGGRAPH '05,* pp. 821-827, July 2005.

[3] Z. Bar-Joseph, R. El-Yaniv, D. Lischinski, and M. Werman, "Texture Mixing and Texture Movie Synthesis Using Statistical Learning," *IEEE Trans. Visualization and Computer Graphics,* vol. 7, no. 2, pp. 120-135, Apr.-June 2001.

[4] J.L. Barron, D.J. Fleet, S.S. Beauchemin, and T.A. Burkitt, "Performance of Optical Flow Techniques," *Proc. Computer Vision and Pattern Recognition,* pp. 236-242, 1992.

[5] J.R. Bergen, P. Anandan, K.J. Hanna, and R. Hingorani, "Hierarchical Model-Based Motion Estimation," *Proc. European Conf. Computer Vision,* pp. 237-252, May 1992.

[6] Y. Boykov, O. Veksler, and R. Zabih, "Fast Approximate Energy Minimization via Graph Cuts," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 23, no. 11, pp. 1222-1239, Nov. 2001.

[7] L.G. Brown, "Survey of Image Registration Techniques," *ACM Computing Surveys,* vol. 24, no. 4, pp. 325-376, Dec. 1992.

[8] G. Doretto, A. Chiuso, S. Soatto, and Y.N. Wu, "Dynamic Textures," *Int'l J. Computer Vision,* vol. 51, no. 2, pp. 91-109, Feb. 2003.

[9] G. Doretto and S. Soatto, "Towards Plenoptic Dynamic Textures," *Proc. Workshop Textures,* pp. 25-30, Oct. 2003.

[10] A. Efros and T. Leung, "Texture Synthesis by Non-Parametric Sampling," *Proc. Int'l Conf. Computer Vision,* vol. 2, pp. 1033-1038, Sept. 1999.

[11] A.W. Fitzgibbon, "Stochastic Rigidity: Image Registration for Nowhere-Static Scenes," *Proc. Int'l Conf. Computer Vision,* pp. 662-669, July 2001.

[12] W.T. Freeman and H. Zhang, "Shape-Time Photography," *Proc. Conf. Computer Vision and Pattern Recognition,* vol. 2, pp. 151-157, 2003.

[13] S. Hsu, H.S. Sawhney, and R. Kumar, "Automated Mosaics via Topology Inference," *IEEE Trans. Computer Graphics and Applications,* vol. 22, no. 2, pp. 44-54, 2002.

[14] M. Irani and P. Anandan, "Robust Multi-Sensor Image Alignment," *Proc. Int'l Conf. Computer Vision,* pp. 959-966, Jan. 1998.

[15] M. Irani, P. Anandan, J. Bergen, R. Kumar, and S. Hsu, "Mosaic Representations of Video Sequences and Their Applications," *Signal Processing: Image Comm.,* vol. 8, no. 4, pp. 327-351, May 1996.

[16] A. Klein, P. Sloan, A. Colburn, A. Finkelstein, and M. Cohen, "Video Cubism," Technical Report MSR-TR-2001-45, Microsoft Research, 2001.

[17] V. Kolmogorov and R. Zabih, "What Energy Functions can be Minimized via Graph Cuts?" *Proc. European Conf. Computer Vision,* pp. 65-81, May 2002.

[18] V. Kwatra, A. Schödl, I. Essa, G. Turk, and A. Bobick, "Graphcut Textures: Image and Video Synthesis using Graph Cuts," *ACM Trans. Graphics, SIGGRAPH '03,* vol. 22, no. 3, pp. 277-286, July 2003.

[19] H. Lombaert, Y. Sun, L. Grady, and C. Xu, "A Multilevel Banded Graph Cuts Method for Fast Image Segmentation," *Proc. Int'l Conf. Computer Vision,* pp. 259-265, Oct. 2005.

[20] S. Peleg, M. Ben-Ezra, and Y. Pritch, "Omnistereo: Panoramic Stereo Imaging," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 23, no. 3, pp. 279-290, Mar. 2001.

[21] S. Peleg, B. Rousso, A. Rav-Acha, and A. Zomet, "Mosaicing on Adaptive Manifolds," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 22, no. 10, pp. 1144-1154, Oct. 2000.

[22] A. Rav-Acha, Y. Pritch, D. Lischinski, and S. Peleg, "Dynamosaicing: Video Mosaics with Non-Chronological Time," *Proc. Conf. Computer Vision and Pattern Recognition,* June 2005.

[23] A. Rav-Acha, Y. Pritch, and S. Peleg, "Online Registration of Dynamic Scenes Using Video Extrapolation," *Proc. Workshop Dynamical Vision at ICCV '05,* Oct. 2005.

[24] P.H.S. Torr and A. Zisserman, "MLESAC: A New Robust Estimator with Application to Estimating Image Geometry," *J. Computer Vision and Image Understanding,* vol. 78, no. 1, pp. 138-156, 2000.

[25] M. Uyttendaele, A. Eden, and R. Szeliski, "Eliminating Ghosting and Exposure Artifacts in Image Mosaics," *Proc. Conf. Computer Vision and Pattern Recognition,* vol. II, pp. 509-516, Dec. 2001.

[26] R. Vidal and A. Ravichandran, "Optical Flow Estimation and Segmentation of Multiple Moving Dynamic Textures," *Proc. Computer Vision and Pattern Recognition,* pp. 516-521, 2005.

[27] Y. Weiss and W.T. Freeman, "On the Optimality of Solutions of the Max-Product Belief Propagation Algorithm in Arbitrary Graphs," *IEEE Trans. Information Theory,* vol. 47, no. 2, pp. 723-735, 2001.

[28] Y. Wexler, E. Shechtman, and M. Irani, "Space-Time Video Completion," *Proc. Conf. Computer Vision and Pattern Recognition,* vol. 1, pp. 120-127, June 2004.

[29] Y. Wexler and D. Simakov, "Space-Time Scene Manifolds," *Proc. Int'l Conf. Computer Vision,* Oct. 2005.

[30] A. Zomet, D. Feldman, S. Peleg, and D. Weinshall, "Mosaicing New Views: The Crossed-Slits Projection," *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 25, no. 6, pp. 741-754, June 2003.

Alex Rav-Acha received the BSc and MSc degrees in computer science from the Hebrew University of Jerusalem, Israel, in 1997 and 2001, respectively. He is currently a PhD student in computer science at the Hebrew University of Jerusalem. His research interests include video summary, video editing, image deblurring, and motion analysis. He is a student member of the IEEE.

Yael Pritch received the BSc and MSc degrees in computer science from the Hebrew University of Jerusalem, Israel, in 1998 and 2000, respectively. For the last several years, she has been working at HumanEyes Technologies. She is now a PhD student in computer science at the Hebrew University of Jerusalem. Her research interests are in computer vision with emphasis on motion analysis and video manipulations.

Dani Lischinski received the BSc and MSc degrees in computer science from the Hebrew University of Jerusalem in 1987 and 1989 and the PhD degree in computer science from Cornell University in 1994. He is currently on the faculty of the School of Engineering and Computer Science at the Hebrew University of Jerusalem. Professor Lischinski's areas of interest include computer graphics and image and video processing. In particular, he has worked on algorithms for photorealistic image synthesis, image-based modeling and rendering, texture synthesis, and editing and manipulation of images and video. For more information see http://www.cs.huji.ac.il/~danix.

Shmuel Peleg received the BSc degree in mathematics from The Hebrew University of Jerusalem, Israel, in 1976 and the MSc and PhD degrees in computer science from the University of Maryland, College Park, in 1978 and 1979, respectively. He has been a faculty member at the Hebrew University of Jerusalem since 1980 and has held visiting positions at the University of Maryland, New York University, and the Sarnoff Corporation. He is a member of the IEEE.

▷ **For more information on this or any other computing topic, please visit our Digital Library at** www.computer.org/publications/dlib.