High Resolution Animated Scenes from Stills

Zhouchen Lin, *Member*, *IEEE*, Lifeng Wang, *Member*, *IEEE*, Yunbo Wang, Sing Bing Kang, *Senior Member*, *IEEE*, and Tian Fang

Abstract—Current techniques for generating animated scenes involve either videos (whose resolution is limited) or a single image (which requires a significant amount of user interaction). In this paper, we describe a system that allows the user to quickly and easily produce a compelling-looking animation from a small collection of *high resolution* stills. Our system has two unique features. First, it applies an automatic partial temporal order recovery algorithm to the stills in order to approximate the original scene dynamics. The output sequence is subsequently extracted using a second-order Markov Chain model. Second, a region with large motion variation can be automatically decomposed into semiautonomous regions such that their temporal orderings are softly constrained. This is to ensure motion smoothness throughout the original region. The final animation is obtained by frame interpolation and feathering. Our system also provides a simple-to-use interface to help the user to fine-tune the motion of the animated scene. Using our system, an animated scene can be generated in minutes. We show results for a variety of scenes.

Index Terms—Texture synthesis, animation.

1 Introduction

A single picture conveys a lot of information about the scene, but it rarely conveys the scene's true dynamic nature. A video effectively does both but is limited in resolution. Off-the-shelf camcorders can capture videos with a resolution of 720×480 at 30 fps, but this resolution pales in comparison to those for consumer digital cameras, whose resolution can be as high as 16 MPixels.

What if we wish to produce a high resolution animated scene that reasonably reflects the true dynamic nature of the scene? Video textures [15] is the perfect solution for producing arbitrarily long video sequences—if only very high resolution camcorders exist. Chuang et al.'s system [6] is capable of generating compelling-looking animated scenes, but there is a major drawback: Their system requires a considerable amount of manual input. Furthermore, since the animation is specified completely manually, it might not reflect the true scene dynamics.

We use a different tack that bridges video textures and Chuang et al.'s system: We use as input a small collection of high resolution stills that (under-)samples the dynamic scene. This collection has both the benefit of the high resolution and some indication of the dynamic nature of the scene (assuming that the scene has some degree of regularity in motion). We are also motivated by a need for a more practical solution that allows the user to easily generate the animated scene.

In this paper, we describe a scene animation system that can easily generate a video or video texture from a small collection of stills (typically, 10 to 20 stills are captured within 1 to 2 minutes, depending on the complexity of the scene motion). Our system first builds a graph that links similar images. It then recovers partial temporal orders among the input images and uses a second-order Markov Chain model to generate an image sequence of the video or video texture (Fig. 1). Our system is designed to allow the user to easily fine-tune the animation. For example, the user has the option to manually specify regions where animation occurs independently (which we term *independent animated regions* (IAR)) so that different time instances of each IAR can be used independently. An IAR with large motion variation can further be automatically decomposed into *semi-independent animated regions* (SIARs) in order to make the motion appear more natural. The user also has the option to modify the dynamics (e.g., speed up or slow down the motion, or choose different motion parameters) through a simple interface. Finally, all regions are frame interpolated and feathered at their boundaries to produce the final animation. The user needs only a few minutes of interaction to finish the whole process. In our work, we limit our scope to quasi-periodic motion, i.e., dynamic textures.

There are two key features of our system. One is the automatic partial temporal order recovery. This recovery algorithm is critical because the original capture order typically does not reflect the true dynamics due to temporal undersampling. As a result, the input images would typically have to be sorted. The recovery algorithm automatically suggests orders for subsets of stills. These recovered partial orders provide reference dynamics to the animation. The other feature is its ability to automatically decompose an IAR into SIARs when the user requests and treat the interdependence among the SIARs. IAR decomposition can greatly reduce the dependence among the

- *Z. Lin and L. Wang are with Microsoft Research, Asia, 5th Floor, Sigma Building, Zhichun Road 49#, Haidian District, Beijing, China, 100080. E-mail: {zhoulin, lfwang}@microsoft.com.*
- *Y. Wang is with the University of Science and Technology of China, Hefei, Anhui, China, 230027. E-mail: ybwang@ustc.edu.*
- *S.B. Kang is with Microsoft Research, Microsoft Corporation, One Microsoft Way, Redmond, WA 98052. E-mail: sbkang@microsoft.com.*
- *T. Fang is with the School of Computer Science and Engineering, South China University of Technology, Wushan, Guangzhou, China, 510604. Email: tian.ft@gmail.com.*

Manuscript received 14 Apr. 2006; revised 25 Aug. 2006; accepted 20 Oct. 2006; published online 3 Jan. 2007.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-0045-0406.
Digital Object Identifier no. 10.1109/TVCG.2007.1005.

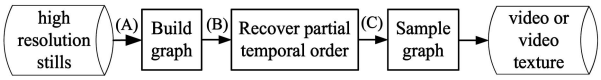


Fig. 1. Outline of our system. All these steps are automatic; user-specified operations (A), (B), and (C) may be added to improve the visual quality of the video.

temporal orderings of local samples if the IAR has significant motion variation that results in unsatisfactory animation. Our system then finds the optimal processing order among the SIARs and imposes soft constraints to maintain motion smoothness among the SIARs.

2 Prior Work

There are many approaches to synthesizing videos of dynamic scenes. One approach that has garnered a lot of attention is video texture [15], which reuses frames to generate a seamless video of arbitrary length. Video textures work by figuring out frames in the original video that are temporally apart but visually close enough, so that jumping between such frames (via a first-order Markov Chain model) appears seamless. This work was extended to produce video sprites [14], which permit high-level control of moving objects in the synthesized video. Unlike videos, the ordering of our input stills may not be 1D. Thus, we can only use partial orders as reference dynamics. In addition, we adopt a second-order Markov Chain model for generating image sequences, rather than the first-order Markov Chain model in [15]. While the video texture paper [15] mentioned used independent regions (IARs in our case), it did not address the issue of local region decomposition. As we demonstrate in our paper, local (IAR) decomposition is another important operation required to produce seamless animation.

Kwatra et al. [8] further extended video textures by recomposing different frames with graph cuts instead of reshuffling the complete frames. Agarwala et al. [1] created panoramic video textures using min-cut optimization to select fragments of video that can be stitched together both spatially and temporally. They also manually partitioned the scene into static and dynamic regions. Sun et al. [18] developed a video-input driven animation system to extract physical parameters such as wind speed from real videos. These parameters are then used to drive the physical simulation of synthetic objects.

Many approaches rely on user input to specify motion in the synthesized video. Bhat et al. [4], for instance, synthesized flow-based videos by analyzing the motion of textured particles in the input video along user-specified flow lines and synthesizing seamless video of arbitrary length by enforcing temporal continuity along other user-specified flow lines. Litwinowicz and Williams [12] used keyframe line drawings to deform images to create 2D animation. Treuille et al. [20] also used keyframes to control the smoke simulation. In [5], video sequences of a person's mouth were extracted from a training sequence of the person speaking and then reordered in order to match the phoneme sequence of a new audio track. Aoki et al. [2] combined physically-based animation and image morphing techniques to simulate and synthesize plants. Chuang et al.'s [6] system allows the user to animate a *single* image. Here, all motion is assumed to be caused by wind. In addition, the user has to manually segment the image layers and specify the motion model for each layer.

Some approaches for synthesizing dynamic scenes are based on more mathematically rigorous analysis. For example, Wang and Zhu [21] modeled the motion of texture particles in video using a second-order Markov chain. Soatto et al. [17] applied nonlinear dynamic systems to model dynamic textures and borrowed tools of system identification to capture the essence of the dynamic textures. Szummer and Picard [19] built a spatio-temporal autoregressive model for temporal textures.

Human perception has also been considered in producing dynamic textures. Freeman et al. [10] applied quadrature pairs of oriented filters to vary the local phase in an image to give the illusion of motion. Paintings can also be illuminated by sequentially timed lights to create the illusion of motion, e.g., the kinetic waterfall [11].

3 Scene Animation System

The basic pipeline of our system, shown in Fig. 1, is fully automatic. The system first builds a graph that links similar images, then recovers partial temporal orders among the stills. Finally, the video or video texture is generated by sampling the graph. However, a fully automatic process may not result in satisfactory videos or video textures as the computer does not have high-level understanding of the scene. The user has the option of modifying or fixing the dynamics of the animated scene through simple interfaces. The optional procedures (described later) may be added at places labeled with (A), (B), and (C) in Fig. 1.

3.1 Building a Graph of Image Similarity

The system first builds a graph that connects similar images. Each image is a node in the graph and the weight of the edge connecting two nodes is the similarity measure of the corresponding two images. To speed up computation, the L_2 norm is used as the distance measure.

Next, the edges with weights larger than twice the mean weight of all the edges indicate that the corresponding image pairs are dissimilar and are deleted. The distances between nodes with their edges deleted are recomputed as the minimum sum of weights along a path linking them [13]. The minimal weight path can be efficiently computed using Floyd's algorithm [16], [13].

Once the graph is built and distances computed, the system decimates the stills to about 800×600 and computes low resolution optical flow between adjacent pairs of nodes (images) hierarchically [3]. The optical flow fields are stored and used for fast video preview as a feedback mechanism.

Unfortunately, the capture order may not reflect the true dynamics. The River example in the submitted video[1] (Fig. 6a) shows that an unusual effect was produced using the original capture order. However, in many cases, it is hard to sort the input images manually (the River data set in

1. All videos can also be found at http://research.microsoft.com/~zhoulin/Animation.htm.

Fig. 2. Close-ups of images of a river (4 out of 12, shown in their capture order). Their temporal order is hard to determine manually. If they are interpolated directly in the same order they were captured, an unnatural-looking effect results. (Please see the video submission.)

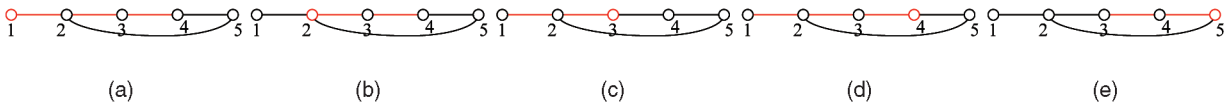


Fig. 3. Finding extremal paths in the graph. Each image is a node in the graph, with the numbers indicating the capture order. The edges indicate the corresponding images they connect are similar, with the weights being the similarity (the lengths of edges drawn here are not proportional to the similarities). (a)-(e) We start from every node to find the extremal paths. The red dots represent the starting nodes to find the extreme paths. For each node, the farthest node is found to construct an extremal path, which is represented by red edges. The rest edges are shown in black. The reverse of an extreme path is also an extreme path, but extremal paths that are part of other extremal paths are deleted. In this example, the extremal paths in (b) and (c) are deleted, leaving four extremal paths (a), (d), (e), and the reverse of (e).

Fig. 2 is a good example). In addition, it may be impossible to arrange the stills into a 1D sequence without *repeating* some images, which makes manual sorting even more difficult. Independent local motions (such as in the Candles data set in Fig. 6b) also complicates manual sorting. To handle these problems, we propose a partial temporal order recovering algorithm that discovers orderings for subsets of stills automatically. These partial temporal orders are critical to providing reference dynamics for the output image sequence (regular video or video texture).

3.2 Recovering Partial Temporal Orders

To recover the partial orders, we find, for each node of the graph, the node farthest from it (Fig. 3). This generates two paths connecting these nodes (in opposite directions), which we call *extremal paths*. Extremal paths are recorded as they are generated; they are important because the end nodes correspond to two extreme scene appearances and the intermediate nodes correspond to the sequential scene appearances between the two extremes. For example, if the scene consists of a pendulum swinging sideways, the two end nodes would correspond to times when the pendulum is to the far left and to the far right, respectively. Also, the intermediate nodes would correspond to the action of the pendulum swinging from the far left and to the far right or from the far right to the far left.

However, an extremal path may be part of another extremal path. This is not desirable, because traversal along one extremal path could end up in another, possibly disrupting the overall dynamics. Hence, we remove extremal paths that are either identical to or subsets of other extremal paths. The nodes (images) on the remaining extremal paths should be in the correct temporal order (forward or backward in time). Fig. 4 shows two examples as validation of our ordering algorithm. After partial order recovery, the animated River (Fig. 6a) appears much more natural.

3.3 Sampling the Graph to Create Image Sequences

We formulate the image sequence selection process as a problem of sampling a second-order Markov Chain on the graph. Given the previous and the current frames f_{n-1} and f_n, respectively, the next frame is computed as follows: First, we compute the penalties over all paths and nodes, with the penalty defined as

$$w_{s,k} = \mathrm{Dist}(\mathrm{Img}(s, k-1), f_{n-1}) + \mathrm{Dist}(\mathrm{Img}(s, k), f_n), \quad (1)$$

where $\mathrm{Img}(s, k)$ is the kth node (image) on the sth path. The value of Dist() can be directly taken from the precomputed distance lookup table. The distribution of $w_{s,k}$ determines the probability of choosing $\mathrm{Img}(s, k+1)$ as the next frame in the following way:

$$P(\mathrm{Img}(s, k+1)|f_{n-1}, f_n) \sim \exp\left(-w_{s,k}/\left(\sigma \sum_{s,k} w_{s,k}\right)\right), \quad (2)$$

where σ is a user controllable parameter and its default value is 0.1. This is a sampling scheme of a second-order Markov Chain. Intuitively, (1) and (2) jointly imply that if the $(k-1)$th and the kth images of the sth path are close to f_{n-1} and f_n, respectively, then it is likely to choose the $(k+1)$th images of the sth path as the next frame f_{n+1}. In the above sampling scheme, the image subsequence specified by an extremal path provides a reference of the scene dynamics.

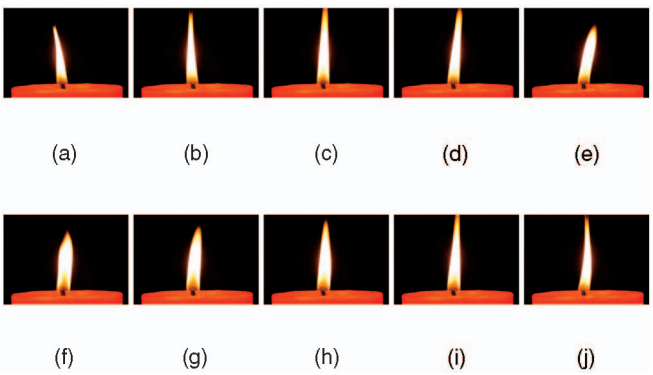


Fig. 4. Examples of extremal paths. In path (a)-(e), the flame is swaying from left to right. In path (f)-(j), the flame gradually thins and lengthens. The ordering looks visually plausible.

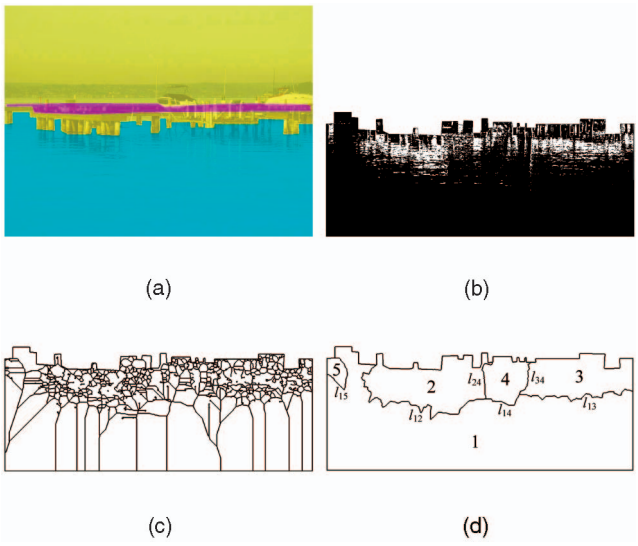


Fig. 5. Image decomposition and the creation of SIARs from an IAR for the Lake data set (Fig. 6c). (a) The IAR (blue), the stationary region (pink), and the slow moving region (yellow). (b) The common low-frequency area A_{LF} (black) of the IAR. (c) The initially skeletonized A_{LF}. (d) The final boundaries selected to produce SIARs. Here, we have five SIARs, numbered 1 to 5. The length of the mutual boundary between SIARs i and j is l_{ij}. The dynamics incompatibility among the SIARs is measured based on l_{ij}.

Note that this frame selection framework is quite different from that in [15], where the model is a first-order Markov chain and the dynamics are respected by diagonally filtering the distance matrix. We cannot filter the distance matrix in the same way because it is possible that *all* the images are not in the correct order. As a result, we cannot reduce the second-order Markov Chain to a first-order Markov Chain.

Because many extremal paths share the same subpaths so that $w_{s,k}$s in (1) are the same for these paths, our system can choose among different extremal paths smoothly. A video texture can be generated if the random loop feature in our system is enabled. Otherwise, a video of finite length is generated instead. The user can adjust the relative global speed of motion using a scroll bar. Depending on the speed, fewer or more intermediate frames are generated. Note that the preview optic flow fields are refined at the original resolution for the final output.

4 Fine-Tuning by Image Decomposition

The image decomposition process is labeled (A) in Fig. 1.

Breaking up images can make the temporal sorting of different regions relatively independent, hence can make better use of the very limited number of available stills. A similar idea has appeared in video texture [15] as "motion factorization."

In our system, there are three kinds of regions. The first is IARs (Fig. 5a), each of which contains an independently moving object. Each IAR is roughly the union of the area that an object occupies in each image. The second is stationary regions (Fig. 5a). For such regions, it is sufficient to sample them from any one image and replicate them throughout the synthesized video in order to remove random motion that is of no interest. The user marks the first two kinds of regions on any one of the image samples. The third is the remaining unmarked regions that are considered to contain small movements (Fig. 5a), so that their temporal ordering is inconsequential. Each region is dilated by a few pixels. Once the videos for the IARs have been synthesized in the same way as described in Section 3,

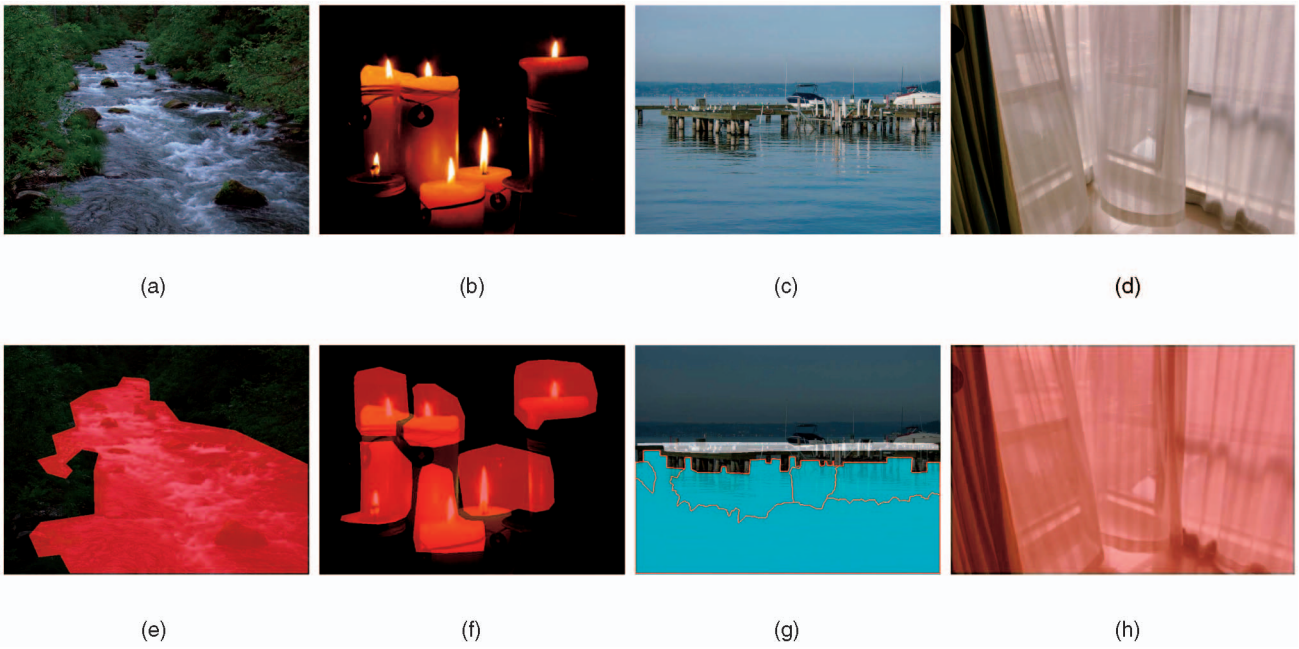


Fig. 6. Examples of animated scenes from still images. (a), (e) River. (b), (f) Candles. (c), (g) Lake. (d), (h), Curtain. (a), (b), (c), and (d) each show one of the frames of the animated scene, while (e), (f), (g), and (h) show the IARs (marked in red) or SIARs (marked in blue, only in (g)), the slow moving regions (marked in black), and the stationary regions (marked in white, only in the central rows of (g)). In (g), the red curves indicate the boundaries of the SIARs. In (h), the whole image is an IAR. (Please see the submitted video for the animations.)

all regions are feathered linearly at the boundaries per frame to produce a seamless-looking video.

Note that although there has been some research on automatic region segmentation (e.g., [7]), in our case, the input stills are temporally undersampled. As a result, the dynamic texture is unknown. In our work, we attempt to discover this by estimating the temporal order first. The problem of segmentation with unknown temporal order and unknown motion cannot be solved with prior techniques (that we know of). Consequently, having the user quickly specify the regions seems like a good compromise.

4.1 Handling IARs with Large Motion Variation

When an IAR with a relatively large area has significant motion variation, the number of time instances will still be too sparse to realize the true dynamics in the IAR. The Lake animation (Fig. 6c) in our video submission shows that when the whole water surface is a single IAR, the resulting motion of the wave appears rather peculiar.

In such a case, the user can request that the IAR be automatically decomposed into SIARs for further analysis and reanimation, where weak constraints between adjacent SIARs are enforced to ensure that their overall flow fields are similar. This is a feature not seen in other proposed animation systems (that we are aware of).

4.1.1 Decomposing an IAR into SIARs

The boundaries between SIARs should ideally lie within low frequency areas of the scene in order to not cause significant blur in the final result. Manually partitioning the IAR is difficult; instead, our system does this automatically.

The system first identifies the common low-frequency areas A_{LF} of all time instances of the IAR (Fig. 5b), then skeletonizes it using the technique of [9] (Fig. 5c). This usually produces too many small regions that typically do not preserve the global visual effect when animated simultaneously. Consequently, our system only selects optimal boundaries among the skeleton of A_{LF} to break the IAR into several SIARs (typically 5) (Fig. 5d).

To do so, a graph is first built such that each junction in the skeleton is a node. The weight between directly connected junctions is the sum of the gradient magnitudes along the skeleton connecting the junctions and over all time instances. The system then computes the minimum spanning tree [16] of the graph. Finally, the system finds several paths on the tree sequentially to separate the IAR into SIARs. The procedure is as follows. Starting with the IAR, at each pass the system finds the longest path that is completely inside (the endpoints being the exceptions) one of the existing regions. This longest path separates the region containing it into two subregions. Typically, the procedure stops when five SIARs are found. Take Fig. 5d for example: The original single IAR is separated into two subregions $1 \cup 5$ and $2 \cup 4 \cup 3$ by the skeleton $l_{12} \cup l_{14} \cup l_{13}$. Then l_{24}, l_{34}, and l_{15} further separate the subregions into the final five SIARs sequentially. We deem that five is roughly the optimal value for the number of the SIARs because fewer SIARs may not decouple the large motion in the IAR sufficiently, while more SIARs may not preserve the global visual effect well when animated simultaneously.

After an IAR is broken into SIARs, for each SIAR the steps of building a graph and a list of extremal paths are exactly the same as those for IARs. In addition, the average optical flow directions between every two similar time instances of an SIAR are computed and stored. The average flow directions will be used as a soft constraint between spatially neighboring SIARs when creating videos for SIARs.

4.1.2 Finding Optimal Processing Order for SIARs

Taking into account the dynamics compatibility among all the SIARs simultaneously is complex. Particularly, the input stills are often temporally undersampled, making the dynamics within each SIAR incorrect without appropriate sorting of the time instances. As a middle ground, we choose to process the SIARs sequentially. The amount of dynamics incompatibility among the SIARs is dependent on the processing order of the SIARs. Therefore, we have to find the optimal processing order of SIARs so that such incompatibility among them is minimized. The analysis is detailed below.

Except the first processed SIAR, the motion in each of the SIARs S_k is dependent on the *previously processed* SIARs: S_r, $S_{i_1}, S_{i_2}, \cdots, S_{i_m}$, that are *spatially neighbor* to S_k, where S_r has the longest common boundary with S_k (Fig. 5d). We use S_r as the *reference* of S_k and make the motion in S_k compatible with that in S_r. Then the dynamics incompatibility between S_k and $S_{i_1}, S_{i_2}, \cdots, S_{i_m}$ can be measured by the sum of the length of the common boundary between S_k and $S_{i_1}, S_{i_2}, \cdots, S_{i_m}$.[2] Minimizing the total dynamics incompatibility, however, is a combinatorial problem. As the number of SIARs is usually small (which we limit to five), a brute force search is adequate to find the optimal order. While this optimal order may not be unique, it does not cause a problem for the examples presented in this paper.

4.1.3 Creating Videos for SIARs

The video for the first processed SIAR is generated in exactly the same way as described in Section 3. However, the other SIARs cannot be generated independently. Each of them has to use the dynamics of its reference SIAR as the constraint. To account for their weak dependence on their reference SIARs that have been processed, we add an extra term

$$\lambda \left\| \bar{\mathbf{v}}(\mathrm{Img}(s,k), \mathrm{Img}(s,k+1)) - \bar{\mathbf{v}}\left(\hat{f}_n, \hat{f}_{n+1}\right) \right\| \qquad (3)$$

to $w_{s,k}$ in (1), and sample the graph as before using (2). Here, λ is a value that we set to be 0.05 times the average weight of the graph that connects similar time instances of the SIAR. $\bar{\mathbf{v}}(x,y)$ is the average optical flow direction between x and y. Note that $\hat{f}_n$ refers to the nth frame of the image sequence previously generated for the *reference* SIAR (*not* the current SIAR). The extra term specified in (3) forces the average flow direction from the current frame to the next frame to be close to that in the reference SIAR.

2. Since our system is designed to choose the boundaries between SIARs in the common low-frequency areas, the motion along these boundaries is small. As a result, the amount of incompatibility should be roughly proportional to the length of the boundaries.

5 Other User Options

It is possible that the default video output may not be satisfactory to the user (say, due to incorrectly interpreted dynamics). Our system was designed to provide the user some flexibility to fix or fine-tune the video through simple interfaces. More specifically, the user has the option of reordering the time instances of a region and editing the motion dynamics.

5.1 Temporal Reordering

This process is labeled (B) in Fig. 1. The system may ask the user to verify if the images that it deemed similar are actually perceptually close; if they are not deemed perceptually close, the appropriate edges are deleted from the graph. The user can also manually sort part of the images. The path connecting the images in this new order is added to the path list. After temporal reordering, the distance matrix and the path list are updated accordingly as described in Sections 3.1 and 3.2.

5.2 Editing Motion Dynamics

This process is labeled (C) in Fig. 1. The user can verify if the motion direction in an extremal path is acceptable, e.g., the user can reject a path that corresponds to a car moving backward.

The user can also specify the properties of motion for each IAR.[3] Unlike Chuang's approach [6], in which the user specifies the physics of motion, our system only asks the user to provide several parameters. The first is the value of σ in (2), which controls the motion smoothness. The second is a number between 0 and 1 that is a measure of motion irregularity, which gives the probability of accepting f_{n+1} sampled in Section 3.3 if f_{n-1}, f_n, and f_{n+1} are on the same extremal path and the direction from f_n to f_{n+1} is the reverse of that from f_{n-1} to f_n. The third is an option of whether the motion speed has to decelerate or accelerate when the motion direction reverses. Although, in theory, these parameters alone cannot fully depict the complex motion in the scene, considering that the collection of stills has already captured some characteristics of the motion, changing these parameters can effectively fine-tune the dynamics of the scene.

6 Results

The video submission shows the videos created using our system on a variety of scenes, including River, Candles, Lake, and Curtain (Figs. 6a, 6b, 6c, and 6d. More demos can be found at http://research.microsoft.com/~zhoulin/Animation.htm). Note that Curtain is a very difficult scene to animate using traditional techniques. The reader is strongly encouraged to view the videos as the generated effects are hard to show in the paper. Some statistics about the data sets and performance are listed in Table 1. In the experiments, user interaction took just a few minutes. (Note that the interaction time excludes "offline" processes indicated by CT1, CT2, and CT3 in Table 1.)

Our system was implemented on an Intel Pentium 4 3.0 GHz PC with 1 GB RAM. The timing information for the animations is listed in Table 1. The times to compute low resolution optical flows are relatively short. However, refining the optical flow took a lot longer, with the times depending on the sizes of (S)IARs. The timings for frame interpolations also vary significantly due to the different numbers of frames interpolated. (These timings could be improved by optimizing the code.) Note that the optical flow refinement and high resolution frame interpolation need only be done *after* the user is satisfied with the previewed video. These operations do not impact the real-timeness of user interaction.

TABLE 1
Statistics for the Examples

	River	Candles	Lake	Curtain
# Stills	12	14	10	20
Resolution	1280 ×960	2048 ×1536	2048 ×1536	2544 ×1696
CI (sec)	1	5	5	5
# IARs	1	6	1	1
# SIARs	0	0	5	0
CT1 (sec)	10	45	20	30
CT2 (sec)	40	400	160	240
CT3 (sec)	360	810	810	2610
UT (min)	1	3.5	2.5	1

CI is the approximate capture interval of the stills. CT1 is the time to compute the low resolution optical flows in (S)IARs. CT2 is the time for refinement to high resolution optical flows. CT3 is the frame interpolation time. UT is the time a user took using the interface.

7 Discussion and Future Work

Our technique works best for scenes with spatially separable moving objects so that a few images are enough to capture different phases of the motion of each object. Scenes with superimposed moving objects may require significantly more image samples. We currently do not consider the issue of layer separation.

Unlike video textures [15], we do not deliberately avoid dead ends because our primary goal is to animate a scene. Dead ends can be easily avoided by deleting the "dead end" branches. However, this would reduce the number of usable images, particularly when some "dead end" branches are long. A long "dead end" branch may still be used at the end of the output video if looping ends.

Our system relies on optical flow estimation [3] for frame interpolation, which typically fails in cases with large object motion or finely textured areas. It remains a challenge to design a robust optical flow estimation technique, and adding an interface for user-specified hints and correction is an option we are currently exploring.

There is also the question as to how many input images are adequate. Intuitively, having more stills is better because with more stills it is easier to find similar images and the optical flow can be more accurate. Unfortunately, as for the minimum number of stills, there does not appear to be a clear-cut answer, because the number should depend on the magnitude and the complexity of the motion and when the snapshots were taken relative to the motion period. Too few stills may result in no similarity between any two images, and the large motion between the two images simply makes the optical flow computation impossible.

3. The motion properties of SIARs are inherited from the IAR.

8 Conclusion

We have presented a system that is capable of generating high resolution animated videos from a small number of stills with very little user assistance. There are two key ideas that made this possible. For one, we have proposed a new framework that involves graph generation with partial temporal orders and a second-order Markov Chain model to produce plausible dynamics for each (S)IAR. Second, to handle an IAR with a large motion variation, we have proposed an automatic algorithm that further decomposes it into SIARs and imposes soft constraints to ensure the motion compatibility among the SIARs. Our system also features user-friendly interfaces to allow the user to fine-tune the motion quickly and effectively.

Acknowledgments

Yunbo Wang and Tian Fang were visiting students to Microsoft Research, Asia, when this work was done.

References

[1] A. Agarwala, K.C. Zheng, C. Pal, M. Agrawala, M. Cohen, B. Curless, D. Salesin, and R. Szeliski, "Panoramic Video Textures," *ACM Trans. Graphics,* vol. 24, no. 3, pp. 821-827, 2005.

[2] M. Aoki, M. Shinya, K. Tsutsuguchi, and N. Kotani, "Dynamic Texture: Physically-Based 2D Animation," *Proc. ACM SIGGRAPH Conf. Sketches and Applications,* 1999.

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

[4] K.S. Bhat, S.M. Seitz, J.K. Hodgins, and P.K. Khosla, "Flow-Based Video Synthesis and Editing," *ACM Trans. Graphics,* vol. 23, no. 3, pp. 360-363, 2004.

[5] C. Bregler, M. Covell, and M. Slaney, "Video Rewrite: Driving Visual Speech with Audio," *Proc. ACM SIGGRAPH '97,* pp. 353-360, 1997.

[6] Y.-Y. Chuang et al. "Animating Pictures with Stochastic Motion Textures," *ACM Trans. Graphics,* vol. 24, no. 3, pp. 853-860, 2005.

[7] G. Doretto, D. Cremers, P. Favaro, and S. Soatto, "Dynamic Texture Segmentation," *Proc. Int'l Conf. Computer Vision,* pp. 1236-1242, 2003.

[8] V. Kwatra et al. "Graphcut Textures: Image and Video Synthesis Using Graph Cuts," *ACM Trans. Graphics,* vol. 22, no. 3, pp. 277-286, 2003.

[9] B. Fisher, S. Perkins, A. Walker, and E. Wolfart, "Thinning," http://www.cee.hw.ac.uk/hipr/html/thin.html#1, 1994.

[10] W.T. Freeman, E.H. Adelson, and D.J. Heeger, "Motion without Movement," *Proc. SIGGRAPH '91,* pp. 27-30, 1991.

[11] T. Hathaway, D. Bowers, D. Pease, and S. Wendel, "Philipps Model 40 (Celesta) Pianella," *Mechanical Music Press,* http://www.mechanicalmusicpress.com/history/pianella/p40.htm, 2003.

[12] P. Litwinowicz and L. Williams, "Animating Images with Drawings," *Proc. ACM SIGGRAPH '94,* pp. 409-412, 1994.

[13] R. Pless, "Image Spaces and Video Trajectories: Using Isomap to Explore Video Sequences," *Proc. Int'l Conf. Computer Vision,* pp. 1433-1440, 2003.

[14] A. Schödl and I.A. Essa, "Controlled Animation of Video Sprites," *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation,* pp. 121-127, 2002.

[15] A. Schödl, R. Szeliski, D.H. Salesin, and I. Essa, "Video Textures," *Proc. ACM SIGGRAPH '00,* pp. 489-498, 2000.

[16] R. Sedgewick, *Algorithms in C++, Part 5: Graph Algorithms.* Pearson Education North Asia Limited, 2002.

[17] S. Soatto, G. Doretto, and Y.N. Wu, "Dynamic Textures," *Proc. Int'l Conf. Computer Vision,* pp. 439-446, 2001.

[18] M. Sun, A.D. Jepson, and E. Fiume, "Video Input Driven Animation (VIDA)," *Proc. Int'l Conf. Computer Vision,* pp. 96-103, 2003.

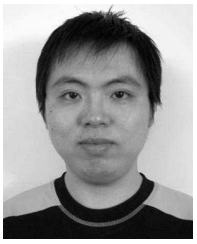
[19] M. Szummer and R.W. Picard, "Temporal Texture Modeling," *Proc. Int'l Conf. Image Processing,* pp. 823-826, 1996.

[20] A. Treuille, A. McNamara, C. Popović, and J. Stam, "Keyframe Control of Smoke Simulations," *ACM Trans. Graphics,* vol. 22, no. 3, pp. 716-723, 2003.

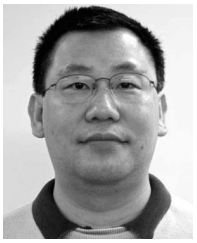
[21] Y. Wang and S.C. Zhu, "Modeling Textured Motion: Particle, Wave and Sketch," *Proc. Int'l Conf. Computer Vision,* pp. 213-220, 2003.

Zhouchen Lin received the PhD degree in applied mathematics from Peking University, China, in July 2000. He is currently a researcher in Visual Computing Group, Microsoft Research, Asia. His research interests include computer vision, computer graphics, pattern recognition, statistical learning, document processing, and human computer interaction. He is a member of the IEEE and Siggraph.

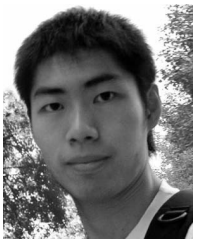
Lifeng Wang received the PhD degree in computer graphics from CAD&CG Lab of Zhejiang University, Hangzhou, China, in July 1999. After that he joined Microsoft Research Asia as a researcher from 1999 to 2006 in the Internet Graphics Group. He is now a research and development manager of the Media and Entertainment Division of Autodesk Inc., Shanghai. His research interests focus on interactive 3D graphics, realistic modeling and rendering algorithms, image-based rendering, and computer vision. He is a member of the IEEE.



Yunbo Wang received the Master of Sciences degree in electronic engineering from the University of Science and Technology of China in 2006. He is now a PhD student in computer science at the University of Nebraska-Lincoln. His research interests include computer vision, embedded systems, and wireless sensor networks.



Sing Bing Kang received the PhD degree in robotics from Carnegie Mellon University in 1994. He is currently a senior researcher at Microsoft Corporation working on environment modeling from images as well as image and video enhancement. He has coedited two books in computer vision (*Panoramic Vision* and *Emerging Topics in Computer Vision*), and recently coauthored a book on image-based rendering. He has also served as area chair and a member of the technical committee for ICCV, CVPR, and ECCV, and is currently a program cochair for ACCV 2007. He is a senior member of the IEEE.



Tian Fang received the bachelor of engineering and master of engineering degrees in computer science from South China University of Technology in 2003 and 2006, respectively. He is now a PhD student at the Hong Kong University of Science and Technology. His research interests include real-time rendering, precomputed radiance transfer, and image-based modeling and rendering.