Sophisticated Construction and Search of 2D Motion Graphs for Synthesizing Videos

Jun Kobayashi*, Chongke Bi†, and Shigeo Takahashi†
*Graduate School of Information Science and Technology, The University of Tokyo, Tokyo 113-8656, Japan
†Graduate School of Frontier Sciences, The University of Tokyo, Chiba 277-8561, Japan

***Abstract*—This paper presents an intuitive method for synthesizing videos by directly manipulating video objects without using 3D models. The proposed method extracts a video object from each video frame and creates locally consistent video sequences using a 2D motion graph, where its node corresponds to the extracted video object and its edge represents a motion transition between a pair of nodes. Our primary contribution lies in a sophisticated construction of the 2D motion graph using shape matching techniques, and its search that allows us to intuitively synthesize a new video sequence by manipulating feature points extracted from the video objects through the 2D screen space. The method further employs a deformation technique to interpolate between video objects with relatively different shapes, and thus can increase available motion transitions by inserting intervening video objects into the 2D motion graph. Several examples have been generated to demonstrate that this approach can create the user-intended motions of the video objects easily by clicking and dragging the feature points.**

***Keywords*-2D motion graphs; video synthesis; feature point tracking;**

I. Introduction

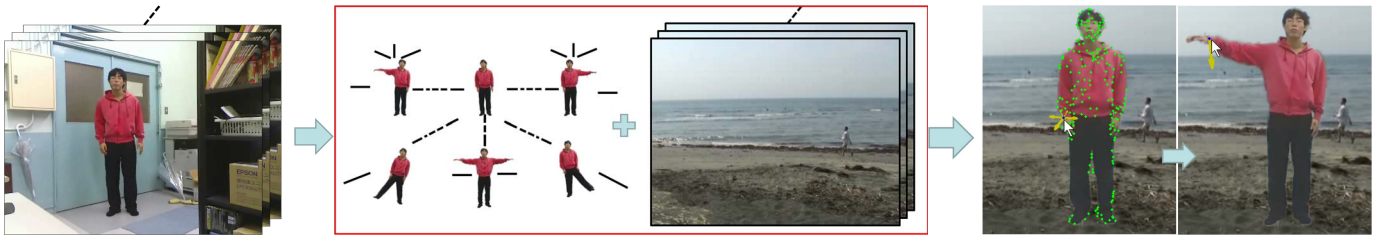
Recent advances in computer hardware allow us to conduct heavy tasks such as video editing and synthesis in our daily life, while generating visually plausible human motions is still difficult with currently available techniques. Example-based approaches can synthesize such realistic and consistent motions by taking advantage of the 3D character models, while the 3D example motions must be obtained beforehand using capture devices that are still expensive, and the process for synthesizing new motions through 3D character models may require laborious software implementation to enhance interactive motion control. On the other hand, another available approach is to segment 2D video objects from a sequence of video frames and to traverse them for generating a new sequence of animated video objects. Although this approach does not require 3D models, it cannot fully guarantee consistent and plausible control over the complex behavior of articulated figures such as humans. This is because the conventional techniques just track existing temporal changes in 2D position and shape of the video objects, or rather randomly jump between pairs of video objects that are not next to each other in the original video, without paying careful attention to the temporal coherence in the resulting motions.

The goal of this paper is to synthesize consistent and visually plausible complex motions such as human behaviors by taking as input ordinary 2D videos only. Our basic idea is to extend the video editing method by direct manipulation [1], and the associated technical contributions are: (1) a sophisticated approach for constructing 2D motion graphs by tracking the 2D video objects and identifying their shape similarities in the given sequence of video frames; (2) an algorithm for traversing a specific sequence of nodes in the constructed motion graph by manipulating the feature points over the video objects through the 2D screen space; (3) an image deformation technique for interpolating between video objects having relatively different shapes using the feature points, in order to increase the repertoire of the available motions. We implemented an interface for synthesizing new motions of video objects from the given video sequences through the screen space manipulation (Figure 1), and demonstrate the capability of the proposed approach together with several examples.

This paper is organized as follows: Section II provides a brief survey on researches related to this work. Section III describes an algorithm for constructing 2D motion graphs from input video sequences. Section IV introduces an algorithm for seeking visually plausible motion by dragging the feature points associated with the extracted video objects. Section V shows how to interpolate between video objects that are relatively different in shape and its application to our video synthesis approach. After presenting several video examples synthesized using our method together with some discussions in Section VI, we conclude this paper in Section VII.

II. Related Work

Motion synthesis has been tackled intensively in the computer graphics community in order to facilitate editing of realistic motions with minimal user intervention. Kover et al. [2] proposed an example-based approach called *motion graphs*, which allow us to synthesize new motions by taking advantage of example motion data together with 3D articulated models. However, managing such 3D motion data requires efficient interactive interfaces [3], and necessary motion data must be recorded beforehand using motion capture devices on demand, which are still expensive and thus usually inaccessible to ordinary users. Schödl et al. [4] developed a technique for constructing 2D motion graphs from captured videos. However, this technique allows us to control the trajectories or positions of video objects only,



(a) (b) (c)

Figure 1. Overview of the proposed method. (a) An input video for constructing a 2D motion graph. (b) The resulting motion graph and a background video into which synthesized motions will be embedded. (c) Designing the new motions by dragging feature points extracted from the video object to seek specific motions in the constructed motion graph.

because it was not equipped with an appropriate algorithm for extracting complex behaviors of the video objects from the motion graphs. Another method has been proposed by Xu et al. [5], in which realistic motions are synthesized from a captured still image of an animal flock again using 2D motion graphs. Nonetheless, this method cannot be applied to complex behaviors of articulated figures since its motion synthesis is still limited to simple cyclic motions such as walking and running. Flagg et al. [6] alleviated this problem by first recording human motions with both a 3D motion capture device and a video camera simultaneously, and then synchronizing the 3D motion data with the 2D projections of the 3D marker trajectories. This again requires special setting of the motion recording system and cannot allow us to make the best use of existing videos that were recorded beforehand.

On the other hand, morphing and deformation techniques can be employed to synthesize new motions especially in 2D video sequence. Cobzaş et al. [7] and Celly et al. [8] employed warped patches and feature-based morphing for animating articulated figures, respectively, while the corresponding motion synthesis is rather limited to cases where the object silhouettes can be easily outlined and have no overlaps with other parts. Igarashi et al. [9] presented a technique for interactively animating 2D objects using preserving shape deformation. Weng et al. [10] proposed a method to synthesizing new motions by extracting 2D video objects from each frame and deforming their shapes. Although these methods can synthesize various motions, they still leave an excessive numbers of degrees of freedom to the shape deformation and cannot successfully constrain it to generate visually plausible motions of articulated figures.

Interactively editing realistic motions of 2D video objects by seeking the node connectivities of the motion graphs is another challenging issue. Dragicevic et al. [11] proposed a method for browsing videos by manipulating the feature points associated with the target video objects. This method provides the sequential access to the given video contents so that users can traverse the video frames back and forth by interactively specifying the movements of the feature

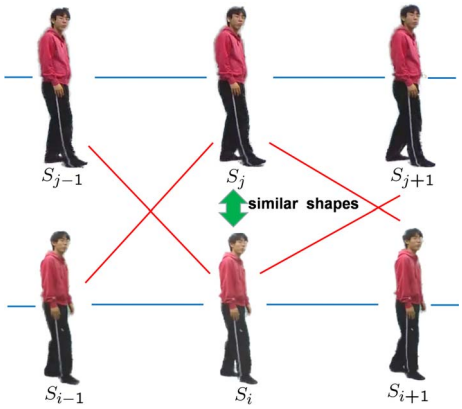


Figure 2. Constructing edge connectivities in the 2D motion graph. Sequential edges (in blue) connect video sprites in temporally adjacent frames while crossover edges (in red) bridge pairs of video sprites having similar shapes.

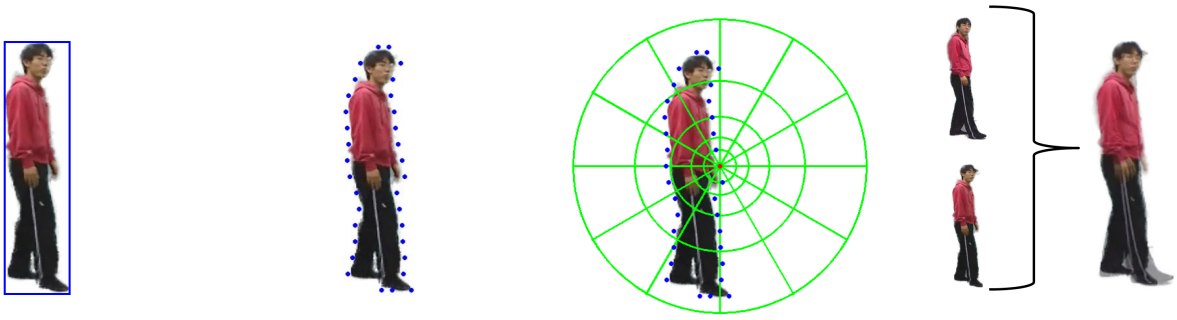
points. Goldman et al. [1] extended this work to provide random access to the video frames so that we can edit the input videos through the direct manipulation of the feature points. However, the method cannot allow us to synthesize realistic motions because it cannot appropriately constrain consistent motion transitions between video objects. We resolve this problem by intensively examining the spatial displacements and shape similarities between video objects and aggressively generating smooth motion transition in between to construct appropriate 2D motion graphs.

III. Construction of 2D Motion Graphs

This section describes our first contribution: 2D motion graph construction from the given video sequence.

A. Extracting Video Sprites

Our algorithm for constructing 2D motion graphs begins by extracting a video object from each frame as a *video sprite*. Among available approaches [12], [13], [14], [15], we employ the method proposed in [14], which extracts video sprites from each frame using the graphcut algorithm [16]. This is because its implementation is relatively simple while we can extract video sprites semi-automatically from each



(a) (b) (c) (d)

Figure 3. Shape matching for seeking crossover edges. Our coarse-to-fine scheme evaluates the shape similarity between pairs of video sprites by (a) first comparing their bounding boxes, then plotting (b) contour points (in blue) in (c) the log-polar coordinate system (in green), and (d) finally performing pixel-wise comparison of colors between the aligned video sprites.

frame of the input video. Note that we denote the video sprite in the i-th frame as S_i ($1 \leq i \leq n$).

B. Constructing Edge Connectivities

Having extracted the video sprites from the video sequence, we insert these sprites into the 2D motion graph as its nodes, and then try to establish edge connectivities between them. In this scheme, we introduce two types of edges in the motion graph, *sequential* edges that represent temporal transition between a pair of immediately adjacent frames in the given video sequence, and *crossover* edges obtained by analyzing the shape similarity between the corresponding end video sprites. In practice, we connect a video sprite S_i with its temporal neighbors S_{i-1} and S_{i+1} by sequential edges first, and then evaluate the possibility of crossover edge connectivity between the other pairs of video sprites by evaluating the shape similarity between them. Figure 2 shows such edge connectivities in the 2D motion graph where the sequential edges are indicated in blue while the crossover edges are drawn in red.

For accelerating the shape similarity comparison between the video sprites, we employ a coarse-to-fine approach, by first filtering out ineligible pairs of video sprites having large difference in their shape using two-steps shape comparison, and then applying the rigorous shape matching to the remaining pairs of video sprites for more precise evaluation. In the first shape comparison, we estimate the rough shape of a video sprite using its bounding box (Figure 3(a)). Let h_i and w_i denote the height and width of the bounding box for the video sprite S_i, and a_i denote the corresponding aspect ratio w_i/h_i. We evaluate the shape dissimilarity D_a between the video sprites S_i and S_j, as follows:

$$D_a(S_i, S_j) = \frac{\| a_i - a_j \|}{a_i} + \frac{\| a_i - a_j \|}{a_j}, \qquad (1)$$

under the assumption that the video sprites are similar in shape if the corresponding aspect ratios are close to each other. We filter out a pair of video sprites if its shape dissimilarity D_a is more than some given threshold in this stage.

The second shape comparison is conducted by plotting their object contour lines as proposed in [17]. This method provides stable results, and has been used also to locate joint positions of the articulated figures [18]. The actual shape comparison in this stage proceeds by uniformly sampling *contour points* on the boundary of each video sprite, which has been normalized with respect to the height of its bounding box beforehand (Figure 3(b)). This is followed by evaluating the feature vector $f_{i,k}$ of the k-th contour point for the video sprite S_i, where each vector element is defined to be the number of other contour points that fall into the corresponding bin arranged in the log-polar coordinate system, whose center is fixed at that k-th contour point (drawn in red Figure 3(c)). Letting N_i be the number of contour points on S_i, the dissimilarity D_c between S_i and S_j in this comparison stage is defined as:

$$D_c(S_i, S_j) = \sum_{k=1}^{N_i} \frac{\|f_{i,k} - f_{j,m^*}\|}{N_i} + \sum_{k=1}^{N_j} \frac{\|f_{j,k} - f_{i,n^*}\|}{N_j}, \qquad (2)$$

where,

$$m^* = \underset{m}{\operatorname{argmin}} \|f_{i,k} - f_{j,m}\| \text{ and } n^* = \underset{n}{\operatorname{argmin}} \|f_{i,k} - f_{j,n}\|. \qquad (3)$$

This means that, for each pair of video sprites, we match a feature vector of one video sprite with the most similar feature vector of the other video sprite, and calculate the average difference between the pairs of the feature vectors.

Finally, we conduct the rigorous shape matching between the remaining pairs of video sprites, by taking into account the interior region of each video sprite. Here, we use a template matching algorithm to align the video sprites so that they have the largest overlap (Figure 3(d)), and evaluate the difference in each pixel value between them. Suppose that the pixel value of S_i at the position (x, y) is defined as $p_i(x, y)$, and the number of pixels on S_i as $|S_i|$. The

dissimilarity D_t between S_i and S_j can be given by

$$D_t(S_i, S_j) = \frac{\sum_{(x,y)\in S_i \cup S_j} \| \, p_i(x,y) - p_j(x,y) \, \|}{|S_i| + |S_j|}. \quad (4)$$

Note that each pixel value is defined as a 3D vector consisting of R, G, and B color components. This shape matching stage enables us to insert crossover edges between the video sprites S_i and S_j, if they have sufficiently similar shapes. However, we do not connect S_i and S_j directly. Instead we connect S_i with any video sprites incident to S_j in the 2D motion graph, and vice versa (Figure 2). In this paper, we will denote the edge between S_α and S_β as $e_{\alpha,\beta} \in E$, where E represents the set of edges in the motion graph.

C. Spatial Displacements and Scale Ratios

Having constructed the 2D motion graph, our next task is to calculate, for the edge between S_i and S_j, attribute values such as the spatial displacement $d_{i,j}$ and scale ratio $s_{i,j}$, while its actual calculation differs according to the type of the corresponding edge. For an sequential edge, we define $d_{i,j}$ by evaluating the spatial displacement in the pixel coordinates between S_i and S_j in the video frames adjacent to each other, and set $s_{i,j}$ to 1 for simplicity. On the other hand, for a crossover edge, we refer to the corresponding bounding boxes (Figure 3(a)) and set $s_{i,j}$ to be the ratio of the heights h_i/h_j, because the sizes of the video sprites have already been normalized with respect to the heights as described earlier. For finding $d_{i,j}$, we first scale the size of S_i by $1/s_{i,j}$, and then seek the optimal overlap between the two video sprites using the template matching (Figure 3(d)). The spatial displacement $d_{i,j}$ is finally obtained as the distance between the centers of S_i and S_j.

In this framework, we can create new motions by traversing the sequence of edges in the constructed 2D motion graph, together with these two attribute values of the edges we are going to pass through (Figure 4(a)). Suppose that the video sprite S_i is scaled by the current scale factor s_i and embedded at P_i in the 2D screen coordinate system (Figure 4(b)). After traversing an edge between S_i and S_j, we can transform S_i into S_j by scaling the size of the current video sprite by $s_j = s_{i,j} \cdot s_i$ and translating it to the new position $P_j = P_i + s_i \cdot d_{i,j}$ (Figure 4(c)). In practice, these two attribute values successfully allow us to synthesize new motions of video sprites frame by frame in the 2D screen space, especially when we traverse crossover edges.

IV. Motion Graph Traversal

Although we can obtain the 2D motion graph as described in Section III, it is still troublesome to synthesize new motions by traversing the graph edges as intended. This is because, as shown in Figure 5, we may have multiple edges for our next visit from the current node (circled with green), and often cannot specify the right choice intuitively through the 2D screen space, even when each node is annotated by its corresponding thumbnail image. To alleviate this problem, we extend an algorithm proposed by Goldman et al. [1], where they traverse the sequence of video sprites randomly by handling the associated feature points. Inspired by their method, we employ an approach to dragging such feature points to synthesize complex motions in a visually consistent manner through the traversal of the constructed 2D motion graph, which is the second technical contribution of this work. This section consists of two parts; we first describe how to track the temporal transitions of such feature points through the input video sequence, and then explain how to control the traversal of the 2D motion graph by manipulating the extracted feature points.

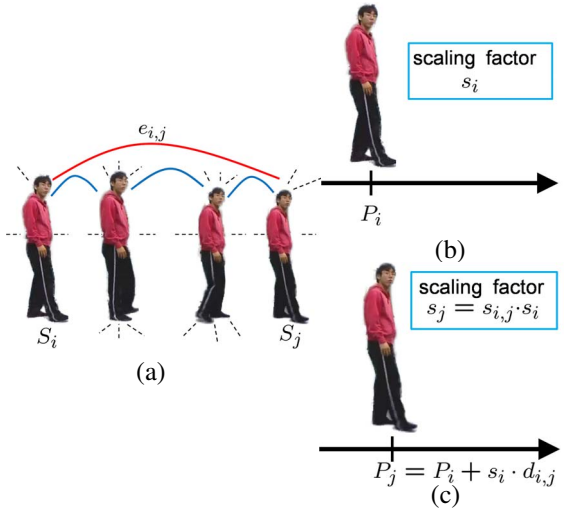


Figure 4. Transforming video sprites while referring to the edge attributes in the motion graph. Here, positions of the video sprites are represented as 1D coordinates for simplicity. (a) S_i and S_j are joined by a crossover edge $e_{i,j}$ (in red). (b) S_i is positioned at P_i after scaling its normalized version by s_i. (c) The corresponding video sprite is moved to P_j while being scaled by $s_{i,j} \cdot s_i$ and translated by $s_i \cdot d_{i,j}$.

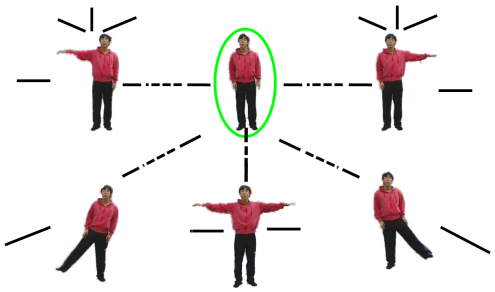


Figure 5. Edge connectivity around the current node (circled with green) in the motion graph. The proposed scheme allows us to traverse specific edges to synthesize new motions.

A. Feature Point Tracking

Our first requirement is to establish the correspondence between two sets of feature points extracted from video sprites if they are next to each other via a sequential edge in

the constructed motion graph. For traversing such sequential edges, we can determine the correspondences by simply tracking the temporal transitions of feature points using conventional techniques such as optical flows. However, we still cannot avoid missing some of the existing feature points when the corresponding video sprites have image noises or self-occlusions (Figure 6). We resolve this problem by updating the arrangement of the feature points scattered over the video sprites every N frames, where $N = 30$ (1 second) by default in our implementation. This means that we have to establish a mapping between the sets of feature points before and after each of these updates. Suppose that the feature points are distributed uniformly over the original video sprites at the initial stage as shown in Figure 6(a). The distribution may be spatially distorted or cannot cover previously occluded regions if the corresponding video sprite S_j has self-occlusions in the legs through a walking motion as shown in Figure 6(b). This problem is solved by replacing the old distribution we have tracked T_j^{track} (in green), with the new updated one T_j^{update} (in blue), which is directly extracted from S_j as shown in Figure 6(c). We then try to establish the mapping M_j from T_j^{track} to T_j^{update} as:

$$M_j(t_j^{\mathrm{track}}(k)) = \underset{t \in T_j^{\mathrm{update}}}{\operatorname{argmin}} \parallel t - t_j^{\mathrm{track}}(k) \parallel, \tag{5}$$

where the position of the k-th feature point of T_j is denoted by $t_j(k)(\in T_j)$. This implies that we identify each feature point in T_j^{track} as its nearest one in T_j^{update} through the mapping. Similarly, we can define the inverse mapping from T_j^{update} to T_j^{track} so that we can introduce bidirectional correspondence between them.

Even when passing through a crossover edge, we can carry out the similar process. Here, before making the correspondence, we overlap the end two video sprites S_i and S_j in the 2D screen space with the template matching, by taking into account the spatial displacement and scale factor that have already been assigned to that crossover edge. We then transform the feature points T_i into T_j^{track} by referring to the optical flows from S_i to S_j, which is followed by the process of updating T_j^{track} with T_j^{update} using Eq. (5). Note that we use the Harris corner detection algorithm [19] to extract the feature points from each video sprite, while more sophisticated algorithms could be used as well.

B. Feature-Point-Driven Motion Graph Traversal

We are now ready to search for the desirable motions by traversing a specific sequence of video sprites in the constructed motion graph. In our framework, this is accomplished by directly manipulating the feature points with the mouse to specify the spatiotemporal transition of object parts in the video sprite. Suppose that we are currently focusing on the video sprite S_i (Figure 7(a)). Our system first requires the user to click a specific position of the video sprite and then identifies the corresponding feature point that is closest

Figure 6. Updating the distribution of feature points. (a) We extract feature points T_i (in green) from the video sprite S_i. (b) After having tracked the feature point transition frame by frame from S_i to S_j, the distribution T_j^{track} on S_j is spatially distorted. (c) New feature points T_j^{update} (in blue) is extracted directly from S_j to establish a mapping from T_j^{track}.

to that clicked point (Figure 7(b)). In addition, the system collects feature points of all the video sprites that are within the path length L from S_i in the motion graph, and indicates the corresponding transitions from the current feature point using yellow arrows, as used in [1]. Note that the path length L is defined to be the number of edges we have to pass through, and $L = 7$ by default in our setting.

When the user moves the k-th feature point $t_i(k) \in T_i$ on S_i by the displacement vector m with the mouse, the system searches for the next video sprite S_j by:

$$j = \underset{\{j | e_{i,j} \in E\}}{\operatorname{argmin}} \parallel (t_j(k) - t_i(k)) - m \parallel. \tag{6}$$

This equation implies that we seek the adjacent video sprites S_j in the motion graph where the displacement of the k-th feature point $t_j(k) - t_i(k)$ is closest to that of the dragged mouse pointer m. Finally, the system shows S_j and updates the status of the corresponding feature point repeatedly, in a way that we can visually confirm the video sprites we have explored in the motion graph (Figure 7(c)). However, only dragging a single feature point will not discriminate a specific motion from other available motions effectively, for example, when we like to raise the both arms of the video sprite simultaneously (Figure 8(a)). Our approach avoids this problem by allowing the user to move and fix multiple feature points at the same time (Figure 8(b)).

V. Morphing Video Sprites

Now we can synthesize intended motions by dragging feature points of video sprites. However, adjusting the thresholds for evaluating the shape similarity in Eqs. (1), (2), and (4) leads to a trade-off between the variety of available motions and frame consistency in the synthesized videos. Furthermore, the above scheme does not fully take into account the velocities of the object local motions, which may result in visual flickering effects especially when we traverse crossover edges in the motion graph.

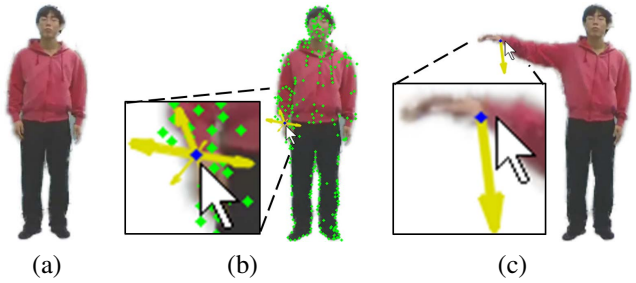


Figure 7. Motion graph traversal by dragging feature points. (a) An initial video sprite. (b) A specific feature point (in blue) is selected among the available feature points (in green). while the possible transitions are indicated by yellow arrows. (c) The feature point is dragged for traversing the intended path in the motion graph.

Figure 8. Dragging and fixing multiple feature points. The points in blue and green represent the feature points to be moved and fixed, respectively. (a) An initial video sprite, and (b) a video sprite specified by multiple feature points.

To alleviate these problems, we introduced a shape morphing technique into our scheme in order to automatically generate interpolated video sprites, which is the third technical contribution of this work. In practice, these interpolated video sprites successfully allow us not only to improve the visual quality of the synthesized video, but also to increase the repertoire of available motions in the motion graph. For this purpose, our approach employs the moving least squares based deformation technique [20], which facilitates the rigidity-preserving shape deformation without any preprocessing such as triangulation over the video sprites. However, this technique usually requires appropriate sets of control points over the video sprites, which are expected to be manually adjusted by users. In this approach, we introduce the features points, which have already been extracted from the video for traversing the motion graph, as the control points for the shape deformation instead. For this purpose, our system retrieves the spatiotemporal displacement of each feature point as its optical flow at every pair of adjacent video frames (Figure 9). Nonetheless, we do not use optical flows of all the feature points because some of them may contain computational errors due to image noises and self-occlusions as described earlier. In our implementation, we eliminate optical flow vectors having the sizes (i.e., norms) larger than some specified threshold as erroneous ones, and use the remaining set of feature points together with their

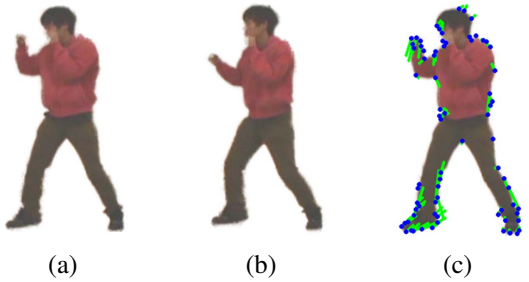


Figure 9. Calculation of optical flows. The optical flows between the video sprites (a) and (b) can be obtained as (c), where the blue points represent the feature points in (a) and the green segments their corresponding optical flows.

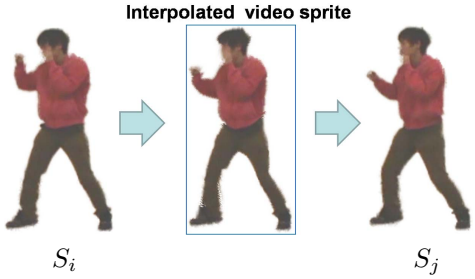


Figure 10. Interpolation between the video sprites S_i and S_j.

Table I
STATISTICS FOR THE EXPERIMENTAL RESULTS

input scene	# of frames	preprocessing	# of sprites
side to side	589	2,443 sec	1,178
back to front	630	3,773 sec	1,260
shadow boxing	885	3,838 sec	1,770

optical flow vectors to automatically interpolate between the video sprites. In this way, our interpolation scheme provides us with smooth transition between the video sprites (Figure 10), and thus enhances the usability of our motion synthesis framework.

VI. Results and Discussions

In our approach, the motion synthesis is conducted as follows (Figure 1): We first construct the 2D motion graph from the input video (Section III), and track the feature points associated with the extracted video stripes (Section IV-A). We then embed the video stripes into another background video and drag the associated feature points to synthesize the new motion by traversing the constructed motion graph as intended (Section IV-B). We also have an option to morph between the video sprites even when they are relatively different in shape (Section V). In practice we take into account the camera motion of the background video in animating extracted video sprites, and further confirm the quality of the synthesized video by freely playbacking the motion of the video sprites and background video simultaneously.

Our system has been implemented on a PC with an Intel Core2Duo 2.66GHz processor and 3.2GB RAM using the C++ programming language and OpenCV library. Table I

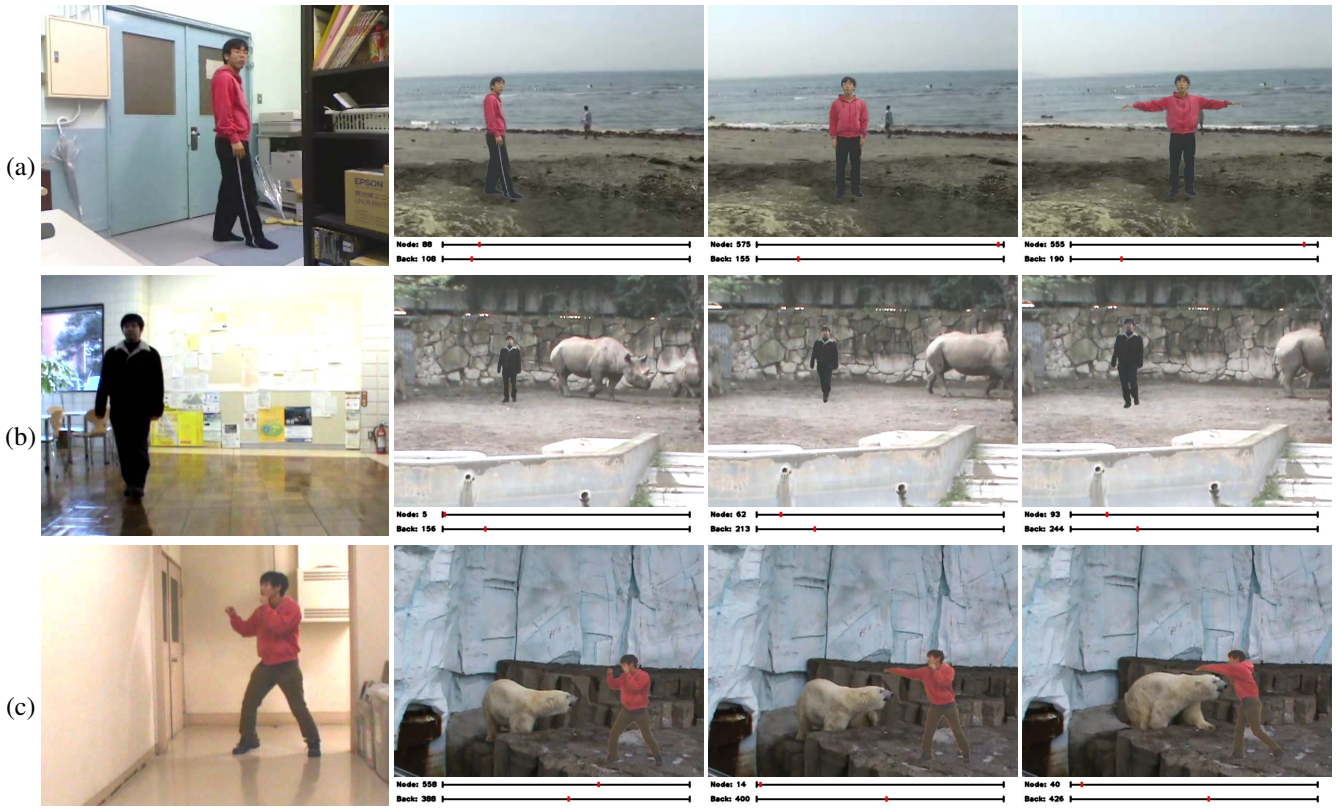


Figure 11. New videos synthesized using our method. (a) Walking from side to side. (b) Walking from back to front. (c) Shadow boxing. The left column shows a representative frame of each input video for constructing the 2D motion graph. The other columns are snapshots of the synthesized videos.

shows the number of original video frames, computation time for constructing the 2D motion graph and tracking feature points, and number of video sprites used as the graph nodes, for each of the synthesized videos in Figure 11. Note that the number of video sprites was doubled by employing the mirror images of the original video frames.

As shown in Figure 11(a), the system takes as input the video scene where the person is walking from side to side while sitting down, standing up, raising arms, etc., and synthesizes a new video by embedding the extracted video sprites into the background seashore scene. Note that, at the bottom of each synthesized video scene, the frame number of the input video that corresponds to the extracted video sprite is indicated as "Node," and that of the background video as "Back." The change in the "Node" number lets us confirm that our algorithm randomly accesses the original video sequence while retaining the consistency in the synthesized motion. Furthermore, the camera motion of the background video has been incorporated to adjust the transformation of the foreground video sprite in this example.

Figure 11(b) corresponds to a case where the person is walking from back to front in the input video. In this case, the size of the video sprite changes according to the depth of the person. We can incorporate this perspective effect into our approach by adjusting the position and size of the video sprite according to the spatial displacements and scale ratios associated with the edges in the motion graph. Figure 11(c) shows a case where the person is doing shadow boxing. This case is the most involved case among the three input videos because the video contains complicated and speedy motions. Nonetheless, our algorithm still synthesizes visually plausible behavior of the person in the new video. In particular, we apply the shape morphing techniques to this case for achieving temporally coherent transition of the boxing styles. See the accompanying video for more details.

Although these experimental results show the feasibility of the proposed approach, there are still limitations in our current implementation. It is troublesome to cope with the difference in lighting environments between the foreground and background videos, especially when we blend the colors around the boundary of the synthesized video objects with that of the given background scene. This problem can be resolved, for example, by seeking some extension of the Poisson image editing method [21] to videos. The repertoire of the available motions depends on the contents of the input video. We may able to enhance existing image deformation techniques to make non-existent behaviors available in our 2D motion graphs, while increasing the number of video sprites results in the time-consuming preprocessing for the shape similarity matching between video sprites. Balancing

the trade-off between the number of available video sprites and required computation time is an important issue. This scalability issue can also be alleviated by classifying the available motions into several categories according to their semantics, which will significantly reduce the number of video sprite pairs to be compared in the shape matching phase. Furthermore, we assume that the camera motion for the background video is basically parallel to the screen, and try to detect the corresponding motion by applying the optical flow techniques to the background scene. Inferring the camera motion along the line of sight effectively from the given background video is still left as our future research theme. Editing the ordering of depth of the foreground and background video sprites together with their mutual occlusion relationships should also be tackled while this will be still an intricate problem even when we can segment important video sprites also from the background video.

VII. Conclusion

We have presented an intuitive and straightforward method for synthesizing new motions from the input videos. Our method yields complex motions of articulated figures such as humans in a visually plausible manner by taking advantage of the 2D motion graphs, without any special hardware such as motion capture devices. The method also facilitates the interactive traversal in the constructed 2D motion graph by dragging the feature points of the video objects. Shape deformation has also been incorporated to enhance our framework so that we can automatically generate intervening shapes of video objects to increase the repertoire of the available motions. The accompanying experimental results demonstrate that we can synthesize new videos while efficiently controlling the complex behaviors of the video objects through the screen space.

Acknowledgment

This work has been partially supported by Japan Society of the Promotion of Science under Grants-in-Aid for Scientific Research (B) (No. 20300033 and No. 22300037) and Challenging Exploratory Researches (No. 21650019).

References

[1] D. B. Goldman, C. Gonterman, B. Curless, D. Salesin, and S. M. Seitz, "Video object annotation, navigation, and composition." in *Proc. 21st annual ACM symposium on User interface software and technology (UIST2008)*, 2008, pp. 3–12.

[2] L. Kovar, M. Gleicher, and F. H. Pighin, "Motion graphs," *ACM Trans. Graphics*, vol. 21, no. 3, pp. 473–482, 2002.

[3] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard, "Interactive control of avatars animated with human motion data," *ACM Trans. Graphics*, vol. 21, no. 3, pp. 491–500, 2002.

[4] A. Schödl and I. A. Essa, "Controlled animation of video sprites," in *Proc. ACM SIGGRAPH/Eurographics Symposium on Computer Animation 2002 (SCA2002)*, 2002, pp. 121–127.

[5] X. Xu, L. Wan, X. Liu, T.-T. Wong, L. Wang, and C.-S. Leung, "Animating animal motion from still," *ACM Trans. Graphics*, vol. 27, no. 5, 2008, article No. 117.

[6] M. Flagg, A. Nakazawa, Q. Zhang, S. B. Kang, Y. K. Ryu, I. A. Essa, and J. M. Rehg, "Human video textures," in *Proc. Symposium on Interactive 3D Graphics and Games 2009 (I3D'09)*, 2009, pp. 199–206.

[7] D. Cobzaş, K. Kerex, and M. Jägersand, "Dynamic textures for image-based rendering of fine-scale 3D structure and animation of non-rigid motion," *Computer Graphics Forum*, vol. 21, no. 3, pp. 493–502, 2002.

[8] B. Celly and V. B. Zordan, "Animated people textures," in *Proc. 17th International Conference on Computer Animation and Social Agents (CASA2004)*, 2004.

[9] T. Igarashi, T. Moscovich, and J. F. Hughes, "As-rigid-as-possible shape manipulation," *ACM Trans. Graphics*, vol. 24, no. 3, pp. 1134–1141, 2005.

[10] Y. Weng, W. Xu, S. Hu, J. Zhang, and B. Guo, "Keyframe based video object deformation," in *Proc. International Conference on Cyberworlds 2008 (CW'08)*, 2008, pp. 142–149.

[11] P. Dragicevic, G. Ramos, J. Bibliowitcz, D. Nowrouzezahrai, B. Ravin, and K. Singh, "Video browsing by direct manipulation," in *Proc. 26th Annual SIGCHI Conference on Human Factors in Computing Systems (CHI2008)*, 2008, pp. 237–246.

[12] I. Drori, T. Leyvand, D. Cohen-Or, and H. Yeshurun, "Interactive object segmentation in video by fitting splines to graph cuts," in *Proc. ACM SIGGRAPH 2004 Posters*, 2004, p. 59.

[13] J. Wang, P. Bhat, R. A. Colburn, M. Agrawala, and M. F. Cohen, "Interactive video cutout," *ACM Trans. Graphics*, vol. 24, no. 3, pp. 585–594, 2005.

[14] Y. Li, J. Sun, and H.-Y. Shum, "Video object cut and paste," *ACM Trans. Graphics*, vol. 24, no. 3, pp. 595–600, 2005.

[15] X. Bai, J. Wang, D. Simons, and G. Sapiro, "Video snapcut: robust video object cutout using localized classifiers," *ACM Trans. Graphics*, vol. 28, no. 3, 2009, article No. 70.

[16] Y. Y. Boykov and M.-P. Jolly, "Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images," in *Proc. 8th IEEE International Conference on Computer Vision (ICCV2001)*, vol. 1, 2001, pp. 105–112.

[17] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 24, no. 4, pp. 509–522, 2002.

[18] G. Mori and J. Malik, "Recovering 3D human body configuration using shape contexts," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 28, no. 7, pp. 1052–1062, 2006.

[19] C. Harris and M. Stephens, "A combined corner and edge detection," in *Proc. 4th Alvey Vision Conference*, 1988, pp. 147–151.

[20] S. Schaefer, T. McPhail, and J. Warren, "Image deformation using moving least squares," *ACM Trans. Graphics*, vol. 25, no. 3, pp. 533–540, 2006.

[21] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," *ACM Trans. Graphics*, vol. 22, no. 3, pp. 313–318, 2003.