Interactive Dynamic Background Scene Generator Based on Limited Input Motions*

TSE-HSIEN WANG[1], BING-YU CHEN[1] AND RUNG-HUEI LIANG[2]
[1]*National Taiwan University*
[2] *National Taiwan University of Science and Technology*

In this paper, an interactive dynamic background scene generating and editing system is proposed based on improved motion graph. By analyzing the input motions with limited frame length and their metadata, our system could synthesize a large amount of various motions to yield a composing dynamic background scene with unlimited frame length by connecting the motion pieces through smooth transitions based on their motion graph layers. The motion pieces are generated by segmenting the input motions and corresponding deforming meshes spatially and temporally, while the smooth transitions connected the motion pieces are obtained by searching the best path in the motion graph layers according to the specified circumstances. Finally, the result motions are optimized by repeatedly substituting the motion sub-sequences. To design the dynamic background scene, users can interactively specify some physical constraints of the environment on the keyframes, such as wind direction or velocity of flow, or even some simple paths for characters to follow, and the system can automatically generate a continuous and natural dynamic background scene in accordance with the user-specified environment constraints.

Keywords: motion graph, motion synthesis, motion analysis, background animation

1. INTRODUCTION

Making complicated animations is a tedious work. Even though several techniques, such as motion capture, can help a lot for generating realistic character motions, to build a dynamic background scene with a lot of various animation objects is still a time-consuming task. The dynamic background scene in this paper is defined as a set of animation objects in one scene except the leading roles (characters) , it usually contains a lot of animation models with similar but different motions, such as flying birds, running horses, or some environment parameters, like a huge amount of flags or trees (formed a forest). The amount of the background animation objects is usually very huge, but it is not necessary to fine tune their motions in details for mass production.

Because of the characteristics of the large quantity of the background animation objects, the physical conditions express much clearer on them rather than a few of the leading roles. For example, the wind direction and wind speed in the scene maybe observed much easier from a grassland or a forest than from one dog walking through the field. Therefore, the problem of making a natural and smooth dynamic background scene is important in animation production. While making a dynamic background scene, we always want to have similar background objects act according to the same physical conditions, but behave variously at the same time. For example, we may use duplications

* This paper was partially supported by the National Science Council of Taiwan under the numbers: NSC98-2221-E-002-140-MY2.

of trees to make a forest, and manipulate those copies to perform different motions at the same time. However, to generate such similar but different motions for a huge amount of animation objects is a tedious task by using current editing tools. Sometimes physical-based simulation is a possible solution, but not all background animation objects can be generated by it and some simulation methods are time-consuming.

Another possible approach is to make a loop animation and let each instance play it at different starting frames. However, people should notice that a simple loop may still reveal the monotonous of the animation, and it is also difficult to set some constraints for these instances, such as wind direction or wind speed. To repeatedly play an existed animation, motion graph [11] provides a suitable technique for motion synthesis. It finds the transitions between the motion pieces of the original motions as the edges to construct a directed graph and easily generates new motions by building walks on the graph. Compared with other motion synthesis methods, motion graph has a higher potential to handle high-level constraints, since we could simply add or modify certain constraints to generate new motions through the synthesis phase of the motion graph. However, the main drawback of the traditional motion graph approaches based methods is that if the input data is not so sufficient, it may generate non-smooth results. Furthermore, without considering the environment constraints, the generated motions may not be consistent with the scene settings and are hardly to be controlled.

Hence, in this paper, we present an improved motion graph method to generate asynchronous transitions for each kind of background animation objects, therefore effectively solve the main problem of motion graph which produces weak results when source data is insufficient. We also use a hill-climbing optimization technique to smooth the synthesized animation when the constraints are modified. Furthermore, we provide an interactive high-level keyframe-based editing tool for users to quickly and easily specify the global environment constraints of the scene and control a large amount of background animation objects in an easy way. To make our method as general as possible, in our system, we take the deforming meshes as the input, so that most kind of motions can be imported into our system, although some information might be lost, such as the skeleton of articulated characters.

The rest of this paper is organized as the follows. Sec. 2 and Sec. 3 introduces some related work and the overview of our system, respectively. The major components of our system are described in Sec. 4, Sec. 5, and Sec. 6. Finally, Sec. 7 and Sec. 8 show some results, conclusions, and future work.

2. RELATED WORK

Our approach for dynamic background scene generation is related to the data-driven animation techniques of motion synthesis. Motion synthesis has been widely discussed and many different approaches have been presented. Li *et al.* [13] constructed a two-level statistical model to represent character motions. They used a linear dynamic system (LDS) to model the local dynamics and used the distribution of the LDS to model the global dynamics. Chai and Hodgins [6] used a continuous control state to control the dynamics instead of using discrete hidden states such like LDS. However, the statistical model is not really suitable for our target background animation models, such like flags or plants,

because of their stochastic motions.

Motion graph [11] is one of the data-driven techniques of motion synthesis. The similar idea is also provided for 2D videos [19], while later comes with many interesting researches concerning video control [18], panorama composition [1] or other video applications. There are also many methods applying motion graph for character animation [2], or to invent new motion structures [5, 12]. Our work is also inspired by the idea of the motion graph dealing with the background animation models.

Some methods are also proposed for constrained motion synthesis [6, 17, 20]. Arikan *et al.*'s method [3] searches the motion database to find suitable frames to synthesize motions that match the annotations. It requires users to annotate the database before synthesis and there are many labels for human actions. Even though our system also requires users to annotate the motion features, the background animation objects are relatively simple compared to human characters and the annotation task is also much easier.

Oshita [14] developed a system to synthesize new motions by automatically selecting different methods to connect the pieces of motions, whereas we use asynchronous transitions to reduce the transition errors. James and Fatahalian's method [9] precomputed the dynamics to build the parameterizations of the deformed shapes and interactively synthesize the deformable scenes. There are also some methods focused on the analysis and evaluation of the motion transitions [21, 16]. Most of the motion synthesis methods emphasize the motion capture data, while we attempt to draw on these methods to generate a dynamic background scene.

There are also some approaches discussing about large outdoor scenes. Perbet and Cani [15] used mesh primitives to animate large-scale prairies in real-time. Beaudoin and Keyser [4] discussed how to improve the simulation time by configuring level of details (LODs). Diener *et al.* [7] utilized 2D motion fields extracted from a video to animate 3D plant models. Zhang *et al.* [22] also provided a similar method to ours but only considered tree models.

In this paper, we propose a method to generate a large amount of animation objects and use motion graph to raise the variances between the duplicated objects, which is effective for a large dynamic background scene generation. Our strategy drawing on the motion graph layers to increase the usability of the short source motions is similar to James *et al.*'s work [10], which also achieved asynchronous transitions. The major difference is that in our method, the animation models are segmented semi-automatically while we also provide an interactive scheme for users to control the whole environment settings.

3. SYSTEM OVERVIEW

Fig. 1 shows our system flowchart. There are three major stages in our system. They are motion analysis (left part), scene design (right upper part), and motion synthesis (right lower part).

In the motion analysis stage (Sec. 4), we take an animation model M with N vertices and F frames as the input. Since the data may not be so sufficient to provide possible smooth transitions, we first segment the model M to be S components (Sec. 4.1) according to its vertices' spatial-temporal variations. Then, for further control pur-

poses, each frame $i = 1...F$ is annotated with metadata $c_i \in C$ (Sec. 4.2). The metadata here mean the physical characteristics, e.g., wind direction, which influence the motion of the animation model, and the metadata annotation is basically performed by the user with an interactive and interpolation scheme, like the user interface shown in Fig. 6. However, in some physical-based animation cases, the metadata c_i can be automatically assigned with the physical parameters when the physical-based simulator generated the original animation model M. Finally, the motion graph $G_j \in G$ is constructed for each segmented component $j = 1...S$ (Sec. 0).

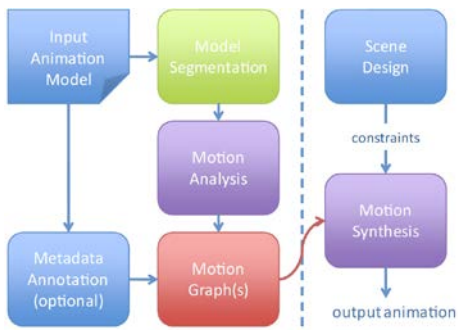


Fig. 1. The system flowchart.

In the scene design stage (Sec. 5), the animation model M can be imported into our system as many instances $\hat{M}$ and users can set the environment layout and constraints $\hat{C}$ through an easy-to-use user interface. Finally, in the motion synthesis stage (Sec. 6), the background animation scene can be synthesized by using the motion graph layer(s) G while considering the annotated metadata C and the environment constraints $\hat{C}$ with a user-specified arbitrary frame length $\hat{F}$. The output animation model $\hat{M}$ will have the same vertex number, connectivity, and segments as the original animation model M, but their frame lengths are different (in most cases, $\hat{F} > F$).

4. MOTION ANALYSIS

The first phase of our system is motion analysis. In this phase, the system requires the user to import a source animation model M with annotated physical characteristics (metadata) C, e.g., wind speed, for further control purposes. Before analyzing the input animation model M, it will be segmented first according to its vertices' spatial-temporal variations. Hence, there are three components in this phase: model segmentation, metadata annotation, and motion graph construction.

4.1. Model Segmentation

In order to increase the usability of the input animation model's motion and raise the quality of the generated result, the input model M is first segmented into several components. The segmentation process is performed on the geometry of the model first, and then the over-segmented segments are merged based on the spatial-temporal analysis.

The input animation model M is first segmented by using randomized cuts [8]. Since it is a pose sensitive algorithm, we take all frames of the input model into the calculation stage in order to derive the best segmentation result for overall sequence. That is, the partition function of the input model is obtained by combining the result of every frame of the input data.

After the spatial cuts, the model is over-segmented as shown in Fig. 2. Since we only considered the geometry characteristics of the input animation model in the spatial segmentation process, some segmented segments have to be grouped due to their temporal characteristics. According to our observation, there are two types of segments should be grouped after the over-segmentation. The first-type segments are those which show high dependencies to each other, such as the dove's two legs (Fig. 2 (right)), and the second-type ones are those generated from useless or too detailed cuts such as the disordered pieces on the flag (Fig. 2 (left)). Hence, we evaluate the motion variance of each segment for comparing their motion differences. For each segment j, we first calculate its centroid p_j^i at each frame i to form a motion vector $\mathbf{P}_j$, which is used to describe the movement trend of the segment and defined as:

$$\mathbf{P}_j = (|\, p_j^1 - \overline{p}_j \,|, |\, p_j^2 - \overline{p}_j \,|, ..., |\, p_j^F - \overline{p}_j \,|)\,,$$

where $\overline{p}_j$ is the average of p_j^i for all frames $i = 1...F$.

Then, we use *k*-means clustering to combine the over-segmented segments into S clusters (components), where S is determined according to the type of the input data. In our strategy, we want to have as many clusters as possible, but meanwhile each group must be distinct to others. Therefore, we iteratively change *k* from the number of the over-segmented segments to 1 and stop when the differences between the new clusters are all larger than a threshold, which is defined as αD, where D stands for the maximum difference between each pair of the over-segmented segments and α is a model-dependent constant (in our experiences, its range is set between 0.7 ~ 0.9).

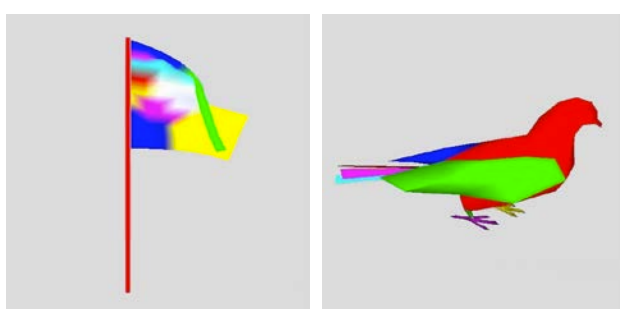


Fig. 2. The result of performing randomized cuts. The segmentation result of the flag (left) looks much more disorderly because we set an over-segmented parameter but the model has no plenty of trivial boundaries.

After the above segmentation and grouping processes, each vertex could be assigned to one of the S groups as shown in Fig. 3. By the two-pass analysis, we can not only semi-automatically separate the original animation model into several components ac-

cording to their natural structures over sequences, but also can cluster its vertices which show similar variation during the whole motion sequence. This can help us to construct the motion graph for each cluster later. Fig. 4 shows other two segmentation results.

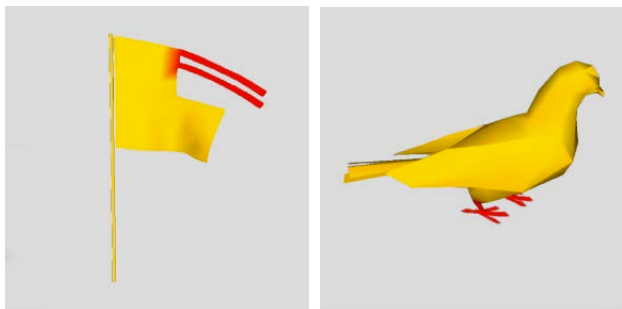


Fig. 3. After performing the model segmentation, the input animation model is segmented. The vertices of the flag (left) and dove (right) models are both separated into two groups. Vertices belong to the red cluster have more intense movement than those in the yellow one.

4.2. Metadata Annotation

For further control purposes, it is better to annotate each frame of the original animation model M, so that the user can use the annotated metadata to control the synthesized instances $\hat{M}$ of the animation model. If the original animation model M is animated by a physical based simulator, such as the flag shown in Fig. 3 (left), the metadata $c_i \in C$ of each frame i is assigned automatically with the physical parameters of the physical based simulator. However, if the annotation cannot be obtained automatically, we also provide a user interface for users to annotate the metadata.

If a user tends to annotate the metadata semi-manually, the user can use an arrow to indicate the velocity and direction of the external forces of the environment like the user interface shown in Fig. 6. For stationary objects, like plants or flags, as shown in Fig. 3 (left), the arrow represents the flow effect such as the wind; for dynamic objects, like animals or humans, as shown in Fig. 3 (right), it stands for their moving speeds and facing directions. We define the characteristics annotated by arrows as the metadata of that motion. A user can just specify the metadata for some keyframes, and then the system will automatically assign the metadata to the rest frames by simply interpolating the user-annotated metadata.

Hence, for each frame i of the input animation model M, its metadata are defined as:

$$c_i = (\tau_i, \phi_i, \theta_i),$$

which is a vector of Cartesian spherical coordinate system, where τ_i is determined by the length of the arrow, and ϕ_i and θ_i are set according to the arrow's direction. The maximum length of the arrow is set to be fixed, and we normalize τ_i to make $0 \le \tau_i \le 1$ to prevent the ambiguity between different motions and make further computation easier.

4.3. Motion Graph Construction

Once we have a segmented animation model M with S groups and annotated metadata C, we could define the transition cost of every pair of frames to generate the motion graph layer G_j ($j = 1...S$). Since the metadata C may be roughly assigned by the user, the cost function does not take the metadata into consideration (we will discuss how we handle the metadata in later sections).

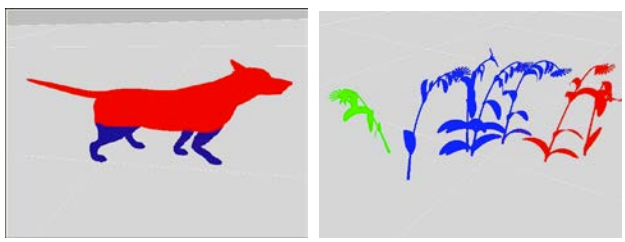


Fig. 4. Other segmentation results. The dog (left) and the plant (right) models are separated into two and three groups, respectively.

If the input animation model M is composed of F frames, the transition cost $D(m_p, m_q)$ between any two frames $m_p, m_q \in M$ is defined as:

$$\sum_{k=-n}^{n} \left(\alpha D_v(m_{p+k}, m_{q+k}) + \beta D_{\hat{v}}(m_{p+k}, m_{q+k}) \right),$$

where $D_x(m_p, m_q) = \left\| x_p - x_q \right\|^2$, $x \in \{v, \hat{v}\}$, n is the window size for calculating the difference between the two frames p and q, D_v represents the position difference of all vertices between them, and $D_{\hat{v}}$ stands for the velocity difference calculated by the same strategy. That means, to preserve the dynamics of the motion, we will compare two subsequences instead of directly comparing two frames as [19].

In addition, we prune the transitions by selecting only the local minimum with a specified threshold. Moreover, we also drop some useless transitions when $|p - q| \le n$. Limiting the frames to not jump to their near neighbors would bring us more reliable and continuous result as shown in Fig. 5.

Besides, we also compute the average playing costs before and after the transition, which are denoted by $D_{pre}(m_p, m_q)$ and $D_{post}(m_p, m_q)$. These two matrices are calculated by almost the same equation as Eq. (1). The difference is that $D_{pre}(m_p, m_q)$ only concerns about the frames from $p - n$ to p, and $D_{post}(m_p, m_q)$ relatively computes from q to $q + n$. By comparing the differences between $D_{pre}(m_p, m_q)$, $D_{post}(m_p, m_q)$, and $D(m_p, m_q)$, we can determine the number of blending frames when transition from frame p to frame q as:

$$B_{\max} \times \frac{|D_{pre}(m_p, m_q) - D(m_p, m_q)|}{\max\left(D_{pre}(m_p, m_q), D(m_p, m_q)\right)}, \tag{1}$$

where $B_{\max}$ is the maximum blending frame number and defined as $B_{\max} = 0.1F$ in our experiment, and F is the original frame number of the input motion.

All the above calculations are performed for each group. Therefore, we would have a motion graph G_j corresponding to each group j after these processes and thus composite the motion graph layer G of the animation model M.

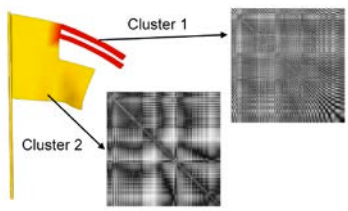


Fig. 5. The motion graphs are constructed for each segment (cluster). Note that the graph patterns are obviously different for each segment, which means it is reasonable for the segmentation.

5. SCENE DESIGN

After the motion analysis, the motion graph layer for each input animation model is constructed. Before using the motion graph layers to synthesize the output animation, we can now think about the issues of environment control and settings. In this phase, we will determine the structure of the whole background scene, for instance, how many models should be placed, where should we put them, and when and what constraints should be set.

5.1. Scene Setting

Because our goal is to generate a big dynamic background scene, it would be very possible that we demand for a large quantity of animation models, and there comes the problem of choosing models, making duplications and locating them. In our system, each input animation model is stored in a motion template bank when we load it, and every model imported to the scene from the template bank becomes an individual instance, which can be translated, rotated, scaled, and animated independently with other instances.

To simplify the laborious work of putting all objects one after another, our system let the user first draw an enclosed plane through the user interface as shown in Fig. 6, and then choose the items from the motion template bank and set the amount of them. Afterwards, the system simply uses uniform distribution to place each instance. In order to prevent penetration artifacts between the instances, the bounding boxes of them are calculated to avoid overlapping.

After placing the instances and specifying the frame length $\hat{F}$ of the output animation, the system would synthesize $\hat{F}$-frame animation as an initial output with initial environment constraints. The user can edit the environment constraints to modify the output animation interactively. The details of synthesizing the output animation will be discussed in Sec. 6.

5.2. Constraint Specification

The second part of the scene design is to set the constraints of the whole environment through the arrows shown in Fig. 7. These constraints are used to guide the instances to find the best synthesis paths on their motion graph layers. Like the metadata annotation described in Sec. 4.2, we also use arrows to specify the overall environment constraints of the background scene.

There are two types of constraints. The first one is physical force constraint. The user can draw some arrows at any point of view to form a flow field to control the motion of the stationary objects. The second type is self-movement constraint, which denotes a moving style on the ground for character motion (dynamic objects). The lengths and directions of the arrows determine the moving speeds and orientations, respectively.

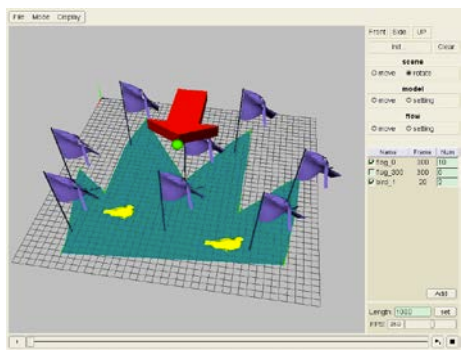


Fig. 6. The user interface of our system. The user can select animation models from the motion template bank and assign the amount of them, and then the instances will be placed in the specified area (the cyan plane). The red arrow shows the initial environment constraints of the scene. In this example, there are two kinds of animation models, flag and dove.

The difference between the metadata of one animation model and the constraints of the environment is that every animation model has only one metadata during a smalltime slot (i.e., one frame), no matter what kind of model it is. This means each animation model during the small time slot is only affected by only one constraint, like a flag blown by the wind with a specific wind speed and direction or a dog walking to ward a specific direction with a specific walking speed. However, for the background scene, every position could have different physical forces which will form an environment flow field. Then, the animation models will be affected by the flow field according to their positions. The weight of an arrow a_k to affect a model (instance) m^i in the environment is decided by the inverse of their distance d_{ik}.

6. MOTION SYNTHESIS

After completing the scene settings, the final phase of our system is motion synthesis. The first pass of motion synthesis is executed when we import the animation models (instances) into the scene. At this time, an initial output animation is generated temporarily. For dynamic objects, the initial frame is simply set to the first frame of their source be-

cause the initial physical force constraints have no effects on them. However, for stationary objects, we scan the frames of the animation models with a particular range of frames ($0.1F$ in our experiment, where F is the original frame number of the input motion) at a time to pick up a motion piece which relatively meets the environment constraints $\hat{C}$, and take it as the initial motion segment. Use an initial section instead of one initial frame could make the motion fit the initial constraints much more smoothly.

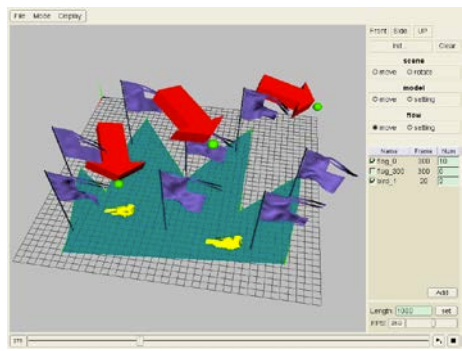


Fig. 7. We can add, remove, or modify the control arrows to specify the environment constraints of the background scene.

Since the environment constraints $\hat{C}$ may be different according to the positions in the scene, the above process should be performed for each instance individually. In the following description, we will only focus on how to synthesize the motion of one instance $\hat{M}$. For the other instances, the process is the same. After deciding the initial frame, for each segmented group $j = 1...S$ belongs to $\hat{M}$, we detect several edges of motion graph G_j to yield a path connecting the pieces of motions, and then combine the groups to output $\hat{M}$ with more than $\hat{F}$ frames.

After that, the user can specify new constraints or modify existing ones. Once the constraints have been specified or changed, the next step is to reproduce a revised version which corresponds to those constraints $\hat{C}$. We use a repeated subsequence replacing method similar to [18] to generate the new version. The error function is defined as:

$$E = \sum D(\hat{m}_p, \hat{m}_q) + \alpha \sum | c_q - \hat{c}_q | + \beta T , \tag{2}$$

where the first and second terms denote the total transition and constraint costs of motion sequence, T is the number of transitions, $\hat{m}_p$ and $\hat{m}_q$ stand for the original frame and replacing frame respectively, c_q and $\hat{c}_q$ are the metadata and environment constraints of frame q, α determines the relative weight between the motion smoothness and constraint effectiveness, and β is used to control the number of transitions. Too many transitions would slightly decrease the continuity but lack of transitions may result in a monotonous animation. To find the best path, we continuously select a segment of the motion, and then try to find other paths which could nearly connect the same source frame and destination frame. The system calculates the error value between paths, if the error of new path is smaller than exist one, we then attempt to replace it by switching

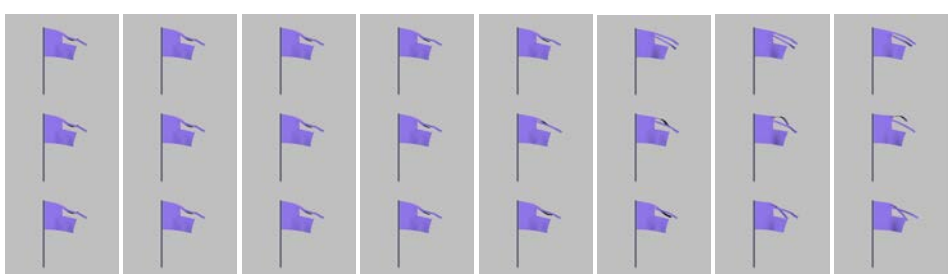
transitions.

The final task is to optimize our result. So far, we only do processing on the single motion, however, some global optimization could enhance the quality of the result animation. We examine the neighboring duplications to check if there are pieces of motions that have too many frames overlapped, and replace the subsequences of those motions to make more natural motions. This optimization will be done only for some input motions. If the input motion is too short, trivially the generated duplications would have many overlapped sections, even if we reproduce the animation again and again. For one generated animation which connected by many motion pieces, we compare the pieces longer than 100 frames with the nearest duplication. If there are at least 100 continuous frames overlapped, we would reproduce the animation.

7. RESULT

All results in this paper are generated on a desktop PC with an AMD Athlon 64 3500+ CPU with 1G memory, and the statistics of models and computation time are listed in Table 1. The Analysis Time includes model segmentation and motion graph construction and Synthesis Time shows the time for synthesizing 1,000 frames, which is the same as traditional motion graph approaches.

Fig. 8 shows the comparison of the original animation data, traditional motion graph's and our results (from top to down). From the left to right is the 295th to the 302nd frames. Since the original animation data has only 300 frames, there is an obvious discontinuity between the 299th and the 300th frames in the upper (source) sequence. Because the source data is insufficient, by using the traditional motion graph, the result shown in the middle sequence still has the discontinuity problem between the 298th and the 299th frames, which is the transition point of both of the traditional motiongraph method and ours.



(a)295 (b) 296 (c) 297 (d) 298 (e) 299 (f) 300(0) (g) 301(1) (h) 302(2)

Fig. 8. From top to down: source data, flag animation synthesized by traditional motion graph, and flag animation synthesized by our method. The original flag animation consists of 300 frames, we pick the pieces of the animation starting from the 295th to the 302nd frames.

Fig. 9 shows a scene with some bamboos and a dove. There are two kinds of bamboos in the scene as listed in Table 1. The arrow adhering to the dove controls the walking direction and speed of it, while another arrow gives the environmental physical force. People may notice that the movement of the synthesized bamboos in the video may not

obey the arrows shown in Fig. 9. That is because when the artist made the original (source) bamboo animation, although there is a specified global wind direction, the bamboo is adjusted to not only follow the wind direction but with a little bit ambient motion to make the bamboo look much more real. Hence, in our system, even a user specifies an extreme wind speed or wind direction, if there is no such data in the input motion, we cannot synthesize the desired result. Instead, the most similar motion pieces are selected and used to meet the requirement as close as possible.

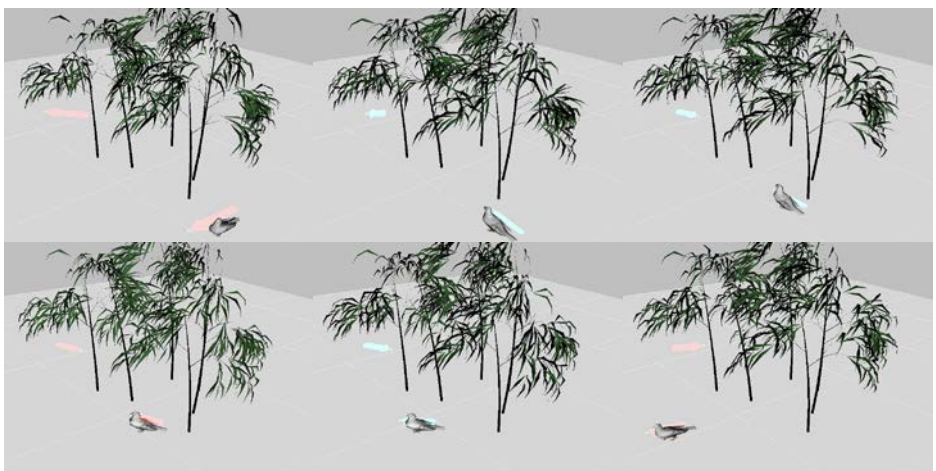


Fig. 9. A composite scene which contains synthesized bamboos (stationary objects) and a synthesized dove (dynamic objects). The red arrows indicate the keyframes. The images are captured every 100 frames.

Table 1 The source data information. The Groups# indicates the number of segmented components after processing the model segmentation, and Analysis Time and Synthesis Time show the time for motion analysis (includes model segmentation and motion graph construction) and motion synthesis (for 1,000 frames) of each input motion, respectively.

Models	Vertex#	Triangle#	Frame#	Group#	Analysis Time (sec.)	Synthesis Time (sec.)
Dove	215	338	20	2	5.7	0.1
Flag	115	92	300	2	4.4	1.8
Bamboo1	8188	7962	300	4	940.1	4.4
Bamboo2	24732	23052	400	5	2414.8	6.7

8. CONCLUSIONS AND FUTURE WORK

Our contribution could be mainly explained from two aspects. For motion graph algorithms, we introduce a hybrid segmentation method to enhance its flexibility by separating the original animation model into several groups according to its vertices' spatial-temporal variations. This method overcomes the bothersome problem of insufficient source for data-driven motion synthesis techniques. Furthermore, we innovate a scene

design interface to take the advantage of motion graph's convenience of high-level control. Our tool lets the user quickly generate a big scene with a large amount of animation objects, and could easily dominate the environment by setting some parameters simply through a pen-drawing user interface.

There are several limitations and future works for this system. In the segmentation part, we now semi-manually set the number of clusters by changing threshold variables. The number of clusters affects the result directly and intensely. It may be possible to perform some analysis to determine these parameters automatically. Besides, since we do not take skeleton motion into consideration, we could not benefit from the convenience of controlling skeleton based models by keyframe editing framework. However, there are many researches that focus on character motion synthesis. Hence, there is a potential to merge the character animation with our background scene animation to perform a complete scene design system.

REFERENCES

1. A. Agarwala, K. C. Zheng, C. Pal, M. Agrawala, M. Cohen, B. Curless, D. Salesin, and R. Szeliski, "Panoramic video textures", *ACM Transactions on Graphics*, Vol. 24, 2005, pp. 821-827.
2. O. Arikan and D. A. Forsyth, "Interactive motion generation from examples," *ACM Transactions on Graphics*, Vol. 21, 2002, pp. 483-490.
3. O. Arikan, D. A. Forsyth, and J. F. O'Brien, "Motion synthesis from annotations," *ACM Transactions on Graphics*, Vol. 22, 2003, pp. 402-408.
4. J. Beaudoin and J. Keyser, "Simulation levels of detail for plant motion," in *Proceedings of SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004, pp. 297-304.
5. P. Beaudoin, M. van de Panne, P. Poulin, and S. Coros, "Motion-motif graphs," in *Proceedings of SIGGRAPH/Eurographics Symposium on Computer Animation*, 2008, pp. 117-126.
6. J. Chai and J. K. Hodgins, "Constraint-based motion optimization using a statistical dynamic model," *ACM Transactions on Graphics*, Vol. 26, 2007, Article No.: 8.
7. J. Diener, L. Reveret, and E. Fiume, "Hierarchical retargetting of 2D motion fields to the animation of 3D plant models," in *Proceedings of SIGGRAPH/Eurographics Symposium on Computer Animation*, 2006, pp. 187-195.
8. A. Golovinskiy and T. Funkhouser, "Randomized cuts for 3D mesh analysis," *ACM Transactions on Graphics*, Vol. 27, 2008, Article No.: 145.
9. D. L. James and K. Fatahalian, "Precomputing interactive dynamic deformable scenes," *ACM Transactions on Graphics*, Vol. 22, 2003, pp. 879-887.
10. D. L. James, C. D. Twigg, A. Cove, and R. Y. Wang, "Mesh ensemble motion graphs: Data-driven mesh animation with constraints," *ACM Transactions on Graphics*, Vol. 26, 2007, Article No.: 17.
11. L. Kovar, M. Gleicher, and F. Pighin, "Motion graphs," *ACM Transactions on Graphics*, Vol. 21, 2002, pp. 473-482.
12. Y.-C. Lai, S. Chenney, and S. Fan, "Precomputing interactive dynamic deformable scenes," in *Proceedings of SIGGRAPH/Eurographics Symposium on Computer An-*

imation, 2005, pp. 281-290.
13. Y. Li, T. Wang, and H.-Y. Shum, "Motion texture: a two-level statistical model for character motion synthesis," *ACM Transactions on Graphics*, Vol. 21, 2002, pp. 465-472.
14. M. Oshita, "Smart motion synthesis," *Computer Graphics Forum*, Vol. 27, 2008, pp. 1909-1918.
15. F. Perbet and M.-P. Cani, "Animating prairies in real-time," in *Proceedings of the 2001 Symposium on Interactive 3D graphics*, 2001, pp. 103-110.
16. P. S. A. Reitsma and N. S. Pollard, "Evaluating motion graphs for character animation," *ACM Transactions on Graphics*, Vol. 26, 2007, Article No.: 18.
17. A. Safonova and J. K. Hodgins, "Construction and optimal search of interpolated motion graphs," *ACM Transactions on Graphics*, Vol. 26, 2007, Article No.: 106.
18. A. Schodl and I. A. Essa, "Controlled animation of video sprites," in *Proceedings of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2002, pp. 121-127.
19. A. Schodl, R. Szeliski, D. H. Salesin, and I. Essa, "Video textures," in *Proceedings of SIGGRAPH*, 2000, pp. 489-498.
20. A. Treuille, Y. Lee, and Z. Popovi´c, "Near-optimal character animation with continuous control," *ACM Transactions on Graphics*, Vol. 26, 2007, Article No.: 7.
21. J. Wang and B. Bodenheimer, "Synthesis and evaluation of linear motion transitions," *ACM Transactions on Graphics*, Vol. 27, 2008, Article No.: 1.
22. L. Zhang, Y. Zhang, Z. Jiang, L. Li, W. Chen, and Q. Peng, "Precomputing data-driven tree animation," *Computer Animation and Virtual Worlds*, Vol. 18, 2007, pp. 371-382.

Tse-Hsien Wang (汪澤先) received the B.S. degree in Computer Science and Information Engineering from the National Taiwan University in 2007, and received the M.S. degree in Networking and Multimedia also from the National Taiwan University in 2009. His research interests are mainly for computer graphics.

Bing-Yu Chen (陳炳宇) received the B.S. and M.S. degrees in Computer Science and Information Engineering from the National Taiwan University, Taipei, in 1995 and 1997, respectively, and received the Ph.D. in Information Science from The University of Tokyo, Japan, in 2003. He is currently a professor in the Department of Information Management, Department of Computer Science and Information Engineering, and Graduate Institute of Networking and Multimedia of the National Taiwan University.

He is also a visiting researcher and professor in the Department of Computer Science and Department of Complexity Science and Engineering of The University of Tokyo since 2008. His research interests are mainly for Computer Graphics, Video and Image Processing, and Human-Computer Interaction. He is a member of the ACM, ACM SIGGRAPH, Eurographics, IEEE, IEICE, IICM, and IPPR, the secretary of the Taipei ACM SIGGRAPH since 2010, and a steering committee member of the Pacific Graphics since 2011.

Rung-Huei Liang (梁容輝) received the Ph.D. in the Department of Computer Science and Information Engineering, National Taiwan University in 1997, Taipei, Taiwan, R.O.C. He is now an assistant Professor in the Department of Industrial and Commercial Design, National Taiwan University of Science and Technology, Taipei, Taiwan, R.O.C. His research interest includes interaction design, computer graphics and multimedia.