Background Animation Generator: Interactive Scene Design based on Motion Graph

Tse-Hsien Wang
National Taiwan University
starshine@cmlab.csie.ntu.edu.tw

Bin-Yu Chen
National Taiwan University
robin@ntu.edu.tw

ABSTRACT

In this paper, an interactive background scene generation and editing system is proposed based on improved motion graph. By analyzing the motion of an input animation with limited length, our system could synthesize large amount of various motions to yield a composting scene animation with unlimited length by connecting the input motion pieces through smooth transitions based on a motion graph layer, which is generated by using independent component analysis (ICA). The smooth connected motions are obtained by searching the best path according to specified circumstances. And finally, the result is optimized by repeatedly substituting animation subsequences. To construct an ideal scene, user can interactively specify some physical constraints of the environment on keyframes, such as wind direction or velocity of flow, even one simple path for a character to follow, and the system would automatically generate continuous and natural motion in accordance with them.

Categories and Subject Descriptors

I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*; H.5.2 [**Information Interfaces and Presentation (e.g., HCI)**]: User Interfaces—*Graphical user interfaces (GUI)*;

1. INTRODUCTION

Making complicated animations is a tedious work. Even though techniques such as motion capture help a lot for generating realistic character motions, a scene of general animation however consists of more background details. We define the background scene as a set which include all objects except the leading roles, it contains not only secondary roles but also environment parameters, such as a huge amount of flags or forest.

Because of the characteristic of large quantity of objects, the physical conditions express much stronger on background than on few of main roles. For example, the wind direction and velocity of the scene may be observed much easily from a grassland or a forest than from one lion walking through the field. Therefore, the problem of making a natural and smooth background motion is considerable in animation production. While making a background scene, we always want similar background objects act according to the same physical conditions, but behave variously at the same time. For instance, we may use duplications of trees to make a forest, and manipulate those copies to perform different motions at the same time. One approach is to make a loop animation and let each duplication play it at different starting frames. However, we should notice that a simple loop may still reveal the monotonous of animation, and it is difficult to set constraints for these models.

Current 3D modeling software, such as Maya, lets user importing static models and alternating them frame by frame to generate an smooth animation. This method has several drawbacks: First of all, it is hard to master great amount of objects elaborately at one time. Even if by the help of scripts, it would also be a challenge to control all objects in an environment with varying physical conditions. Second, it is inconvenient to reuse or merge existing motion clips into a new animation production.

To increase the reusability of accomplished animations, motion graph [10] provides a suitable technique for motion synthesis. It finds transitions between pieces of original motions as edges to construct a directed graph and easily generates new animations 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 take the graph walk extraction as an optimization problem, we could simply add or modify certain constraints to generate new motions corresponding to the new path. This data-driven method is also appropriate for most background scene which composed of excessive mesh objects.

In this paper, we present an improved motion graph algorithm to generate asynchronous transitions for each object, therefore effectively solve the main problem of motion graph which producing weak results when source data is insufficient. We also use a hill-climbing optimization to smooth the synthesized animation when the constraints are modified. Furthermore, we provide a high level keyframe-based editing tool for user to quickly and easily specify the global constraints of the scene and control large amount of objects

in an easy way.

2. RELATED WORK

Our approach for scene animation generation is related to the data-driven animation techniques of motion synthesis.

Motion synthesis has been widely discussed and many different approaches have also been presented. Li et al. [12] constructed a two-level statistical model to represent character motion. It uses linear dynamic system (LDS) to model local dynamics and the distribution of LDS to model global dynamics. Chai and Hodgins [6] used a continuous control state to control the dynamics instead of using discrete hidden state such like LDS. However, statistical model is not really suitable for our target meshes such like flags or plants because of their stochastic motions.

Motion graph [10] is one of data-driven techniques of motion synthesis. The similar idea is implemented on 2D videos which called video textures [20], while later comes with many interesting researches concerning video control [19], panorama composition [1] or other video applications. There are many methods applying motion graph for character animation [2], or to invent new motion structures [5, 11]. Our work tries to take the idea of motion graph for generating background mesh animations.

Some methods are also proposed for constrained motion synthesis [6, 18, 22]. Arikan etal's method [3] searches the motion database to find suitable frames to synthesis motions that match 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 motion features, the background object is relatively simple compared to human character and the annotation task is also much easier. Oshita [13] developed a system to synthesize new motions by automatically selecting different methods to connect pieces of motions, whereas we use asynchronous transitions to reduce transition errors. James and Fatahalian's method [8] precomputes the state space dynamics to build parameterizations of deformed shapes and interactively synthesize deformable scenes. There are also some methods focused on the analysis and evaluation of motion transitions [23, 16]. Most of motion synthesis methods put their attentions on the motion capture data, while we attempt to make use of these methods to generate a big background scene.

There are some approaches discussing about large outdoor scenes. Perbet and Cani [14] used mesh primitive to animate large scale prairies in real-time. Beaudoin and Keyser [4] discussed how to improve simulation times by configuring level of details (LODs). Zhang et al. [24] also provided a similar method to ours but only consider tree models. In this paper, we propose a method to generate large amount of objects and use motion graph to raise variances between duplicated objects, which is effective for large scene generation.

Our strategy to take advantage of motion group layer to increase the usability of short source motions is similar to [9], which proposed an algorithm to make asynchronous transitions. The main difference is that we present a method to segment models automatically and provide an interactive scheme for user to control the whole environment.

3. SYSTEM OVERVIEW

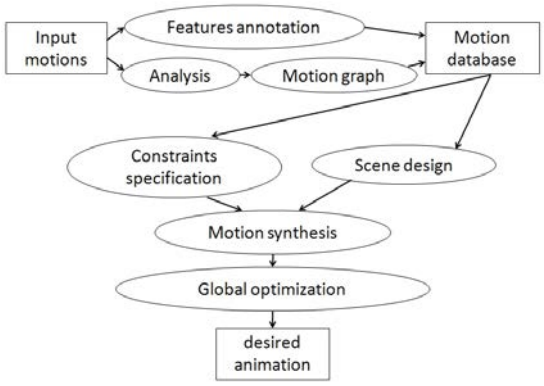

Figure 1: The system flow chart.

3.1 Problem definition

The input for our system is N_S short object motions $S = \{s_1, s_2, ..., s_{N_S}\}$. Each motion $s_i \in S$ originally has $L(s_i)$ frames. During pre-processing, we define the feature C_i and construct a motion graph layer G_i for s_i. Based on G_i, we want to generate a new motion $\hat{s}_i$ according to $N_{\hat{C}}$ constraints $\hat{C}$, which are given by user. And finally we use these new motions to composite a rich background scene animation A with user-specified length $L(A)$.

3.2 System flow

The key idea behind our approach is that the technique of reusing existing motions such like motion graph is especially suitable for background scene animation. The important observation is that one background scene always requires lots of similar or duplicated objects to reduce the cost of sketching new objects. While at the same time, we must be aware to handle duplications carefully to avoid making our results monotonous.

Our system involves automatic analysis, synthesis and interactive design. As shown in Figure 1, we divide the overall procedure into three major stages from input motions to motion database and finally desired animation:

- **Motion Analysis:** At this stage, we get the source data s_i, construct the motion graph layer G_i, and assign the feature set C_i.
- **Scene Design:** We import models to the scene from database, set animation preference such like $L(A)$, specify the control conditions $\hat{C}$, and design the appearance of the scene.
- **Animation Output:** Here we obtain the final animation A by a two-pass motion synthesis procedure and repeatedly optimizations.

4. MOTION ANALYSIS

The first phase of our production pipeline is motion analysis. In this section, the system requires the user to import source motions and annotate the physical characteristics, e.g. wind velocity, for further analysis. We divide this stage into three components: model segmentation, feature annotation, and motion graph construction.

4.1 Model segmentation

In order to increase the usability of source motions and raise the quality of generated result, we use independent component analysis (ICA) to first segment the input model. For an input motion s_i, we define a matrix V_i consists of the positions of all vertices of s_i. The number of rows is the amount of vertices and the number of columns shows how many frames s_i has. We also form the velocity matrix $\hat{V}_i$ which has the same dimension with V_i by calculating the velocity (e.g. the position deviation) of each vertex.

The element of $\hat{V}_i$ is defined as:

$$(\hat{v}_i)_{s,t} = \sum_{k=-n}^{n-1} \|(v_i)_{s,t+k+1} - (v_i)_{s,t+k}\|^2,$$

where n determines the window size. Generally, we just set $n = 1$. Here we introduce the basic form of the ICA model:

$$X = AW,$$

where X is the observed data, and A and W represent the mixing matrix and independent components, respectively. We now substitute $\hat{V}_i$ for X and perform ICA to get corresponding A and W. Finally we use k-means clustering to separate A into N_{Ri} clusters, where N_{Ri} is determined according to the type of input data.

Since each row of A maps to one vertex of s_i, after segmentation, each vertex would be assigned to one of N_{Ri} groups, The processed result is shown in Figure 2. By running ICA, we automatically separate the model into several parts according to their movement and could cluster the vertices which have similar variation during the whole motion sequence. This helps us to construct the motion graph for each cluster later.

4.2 Feature annotation

In our system, we use an arrow to indicate the velocity and direction of force. For static objects like plants and flags, the arrow represents the flow effect such as wind, and for dynamic models like animals or humans, it stands for their moving speed and facing direction. We define the characteristics annotated by arrows as the features of that motion. Each frame p of motion s_i has feature

$$c_{ip} = (\tau_{ip}, \phi_{ip}, \theta_{ip}),$$

which is a vector of Cartesian spherical coordinate system. τ_{ip} is determined by the length of arrow, and ϕ_{ip} and θ_{ip} are set according to the arrow's direction. The maximum length of arrow is fixed, and we normalize τ_{ip} so that $0 \leq \tau_{ip} \leq 1$ to prevent from ambiguities between different motions and make further computation easier.

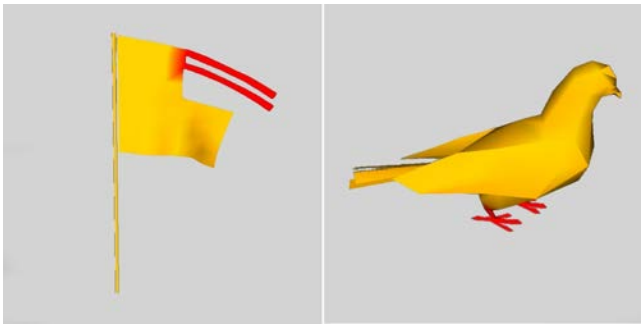

Figure 2: After performing ICA and k-means clustering, we have a segmented model. The flag and the bird are both separated into two groups. Vertices belong to the red cluster have more intense movement.

The system let user specify the features for a few of frames and then simply get others by interpolation. Every motion is required to be annotated when it is first loaded into the system.

4.3 Motion graph construction

Once we have a segmented motion s_i with N_{Ri} groups and annotated features C_i, we could define the transition cost of every pair of frames to generate the motion graph layer G_i. Since the features are roughly set by user, the cost function does not take them into consideration (We will discuss how we handle the features in later sections). The system calculates the transition cost only on the basis of pure source data.

If the input motion s_i is composed of $L(s_i)$ frames, e.g., $s_i = \{f_1, f_2, ..., f_{L(s_i)}\}$, the transition cost between two frames f_p and f_q is defined as:

$$D(f_p, f_q) = \alpha \sum_{k=-n}^{n} D_v(f_{p+k}, f_{q+k}) + \beta \sum_{k=-n}^{n} D_{\hat{v}}(f_{p+k}, f_{q+k}),$$

$$D_v(f_p, f_q) = \sum \|V_{Si}(colp) - V_{Si}(colq)\|^2,$$

$$D_{\hat{v}}(f_p, f_q) = \sum \|\hat{V}_{Si}(colp) - \hat{V}_{Si}(colq)\|^2,$$

where D_v represents the sum of position difference of all vertices between two frame sequences. That is, to preserve dynamics of motion, we compare two subsequences instead of directly comparing two frames [20]. $D_{\hat{v}}$ stands for velocity difference calculated by the same strategy.

We prune the transitions by selecting only local minimum and specifying a threshold. Moreover, we examine $\|p - q\|^2$ to drop some useless transitions. Limiting the frames to not jump to their near neighbor would bring us more reliable and continuous result.

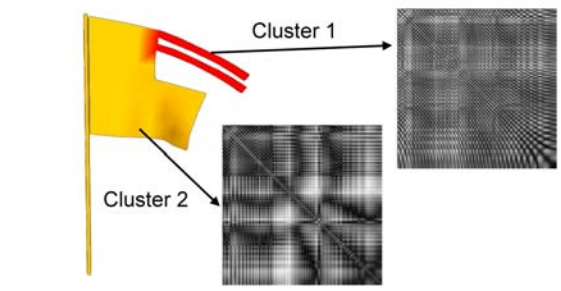

Figure 3: We construct motion graph for each group and thus form a motion graph layer. Notice that the graph patterns are obviously different, which means it is reasonable for the segmentation.

We also compute the average playing costs before and after the transition, which denoted by $D_{pre}(f_p, f_q)$ and $D_{post}(f_p, f_q)$. These two matrices are calculated almost the same as we described in Section 4.1, the difference is that $D_{pre}(f_p, f_q)$ only concerns about frames from $p-n$ to p, and $D_{post}(f_p, f_q)$ relatively computes from q to $q+n$. By comparing the differences between $D_{pre}(f_p, f_q)$, $D_{post}(f_p, f_q)$, and $D(f_p, f_q)$, we can determine the number of blending frames of transition $t(p, q)$.

All above calculations are performed for each group $r_{ik} \in R_i$. Therefore, we would have a motion graph g_{ik} corresponding to each group r_{ik} after these processes and thus composite the motion graph layer G_i.

5. SCENE DESIGN

After the pre-computation is completed, we can now think about the issues of environment control and settings. In this section, we determine the structure of whole background scene, for instance, how many models should be placed, where should we put them, and what constraints would occur at specific time. We separate the design procedure into two parts and discuss each below.

5.1 Background scene design

Because our goal is to generate a big background scene, it would be very possible that we demand for large quantity of models, and there comes the problem of choosing models, making duplications and locating them. Each input motion is stored in the motion template bank when we load it, and every motion imported to the scene from template bank becomes an independent duplication. It can be translated, rotated and scaled at will.

To simplify the laborious work of putting all objects one after another, our system let the user to first draw an enclosed plane, and then choose the items from motion template bank and set amounts of them, afterwards, the system simply uses uniform sampling to place each motion. In order to prevent penetration artifacts between motions, we calculate the bounding box of motions and avoid their overlapping at initial position. The system would automatically produce an arrow to represent the initial environment constraint if we have not specify any one.

After we finish placing the models and specifying the length of output animation, the system would synthesis N_A-frame animation as an initial output. We will discuss this part in details in Section 6.1.

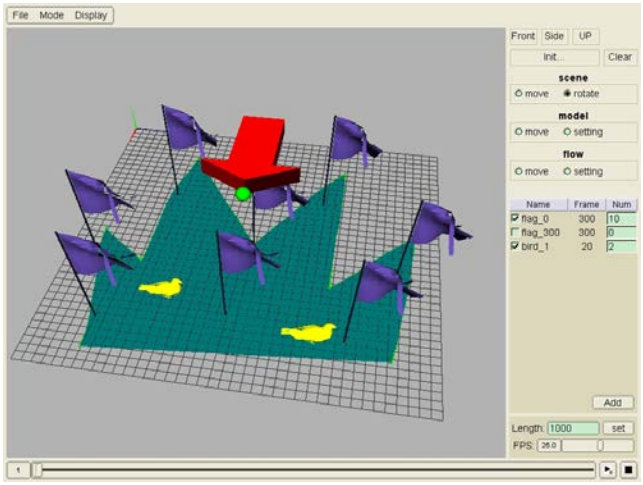

Figure 4: User selects models from template bank and assigns the amount, and then the motions would be placed in the specified area (The cyan plane). The system has automatically generated an initial arrow for the scene.

5.2 Constraints specification

The second part of scene design is to set the constraints of whole environment. These constraints instruct the motions to find the best synthesis paths on their graph layers. Like the features annotation we have described in Section 4.2, we use arrows to specify overall constraints of the scene.

There are two types of constraints for use. The first one is physical force constraint. The user can draw arrows at any point of view to form a flow field to control the motion of static object. The second type is self-movement constraint, which denotes a moving style on the ground for character motion. The length and direction of arrow determine the moving speed and orientation, respectively.

The major difference between features and constraints is that every motion template has only one fixed arrow to represent its feature, no matter the model is static or dynamic. When we design the scene, every self-movement constraint would adhere to a moving object. Comparatively, the physical force constraint arrows are free for translation, and the distance between arrow and model would affect the strength of constraint.

Suppose there are N_W physical force arrows in the environment, let's denote the distance between model s_i and arrow w_j as d_{ij}, and the strength weight of arrow w_k for s_i would be:

$$\frac{\frac{1}{d_{ik}} \times \sum_{j=0}^{N_W} d_{ij}}{\sum_{t=0}^{N_W} (\frac{1}{d_{it}} \times \sum_{j=0}^{N_W} d_{ij})}$$

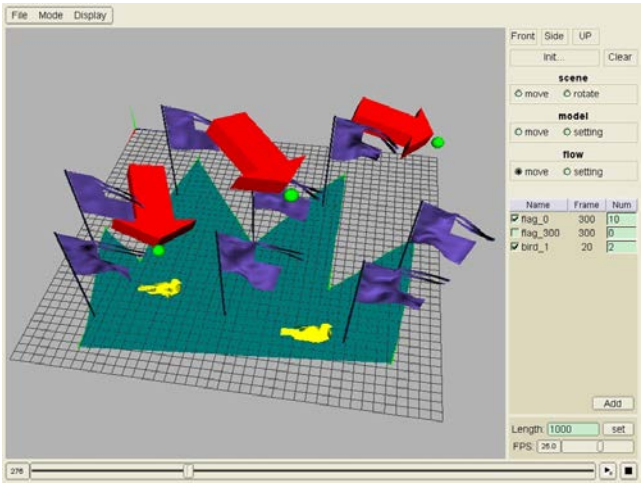

Figure 5: We can add control arrows or modify them at any time we want.

6. ANIMATION OUTPUT

In above sections we have done all pre-computations and settings of the scene. We now describe how the system generate the final result in this last stage.

6.1 Motion synthesis

The first pass of motion synthesis is executed when we import motions to the scene. At this time, an initial output $\hat{A}$ is generated temporarily as we mentioned in Section 5.1. At first we scan the frames of motion and specify a range that relatively meet the constraint, and then randomly choose one frame as initial frame. Next, for each group r_{ik} belongs to s_i, we detect several edges of g_{ik} to yield a path connecting pieces of motion, and then combine the groups to output $\hat{s}_i$ with $L(\hat{s}_i) \geq L(A)$. 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 animation which corresponds to those constraints. We use a repeated subsequence replacing method similar to [19] to generate new motions. The error function of each synthesized motion $\hat{s}_i$ is defined as:

$$E_i = C_T + \alpha C_C + \beta N_{T_i},$$

$$C_T = \sum D_{fp,fq},$$

$$C_C = \sum (c_k - \hat{c}_k),$$

where C_T denotes transition cost and C_C is constraint cost, α determines the relative weight between 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 motions and calculate E_i. If the error exceeds a threshold, we then attempt to replace it by switching transitions.

6.2 Optimization

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 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.

7. RESULTS

We present results in Fig.6 and Fig.7. Fig.6 shows three sequences of motions generated by different methods. The upper one is original source motion. The middle one is resulting motion synthesized by our method, and the nether one is generated by traditional motion graph. We can see apparent discontinuities both in upper and nether sequences between 300th frame and 301th frame. Fig.7 shows a scene with numerous flags. We could see the reactions of flags to the flow constraint. The directions gradually changed from upper left to bottom right. The red arrow indicates its keyframe status.

8. CONCLUSIONS AND FUTURE WORK

Our contribution could be mainly explained from two aspects. For motion graph algorithms, we introduce ICA to enhance its flexibility by separating motions into several groups. 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, the tool lets the user quickly generate a big scene with large amount of objects, and could easily dominate the environment by setting some parameters simply through pen drawing.

There are several limitations and future works for this system. In the segmentation part, we now manually set the numbers of clusters and independent components (IC). The numbers of clusters and ICs affect the result directly and intensely. It is possible to do some analysis to determine these parameters.

Besides, since we do not take skeleton motion into consideration, we could not benefit from the convenience of controlling skeleton based models. However, there are many researches that focus on character motion synthesis using motion graph, There is a potential to merge the character animation with our background scene animation to perform a complete big scene.

9. 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*, 24(3):821–827, 2005. (SIGGRAPH 2005 Conference Proceedings).

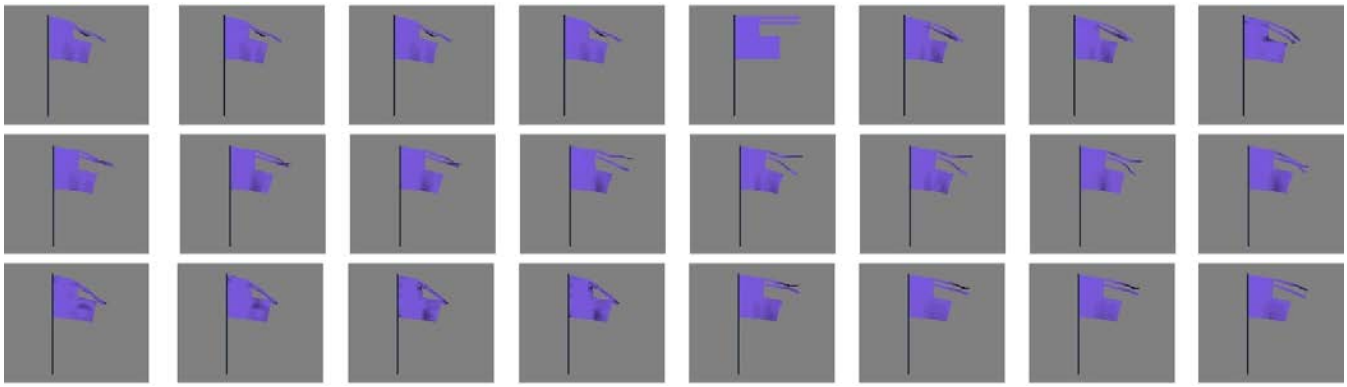

Figure 6: From up to down: Source data, motion synthesized by our method, motion synthesized by traditional motion graph. The original motion consists of 300 frames, we pick the piece of motion starting from 297th frame.

[2] O. Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Transactions on Graphics*, 21(3):483–490, 2002. (SIGGRAPH 2002 Conference Proceedings).

[3] O. Arikan, D. A. Forsyth, and J. F. O'Brien. Motion synthesis from annotations. *ACM Transactions on Graphics*, 22(3):402–408, 2003. (SIGGRAPH 2003 Conference Proceedings).

[4] J. Beaudoin and J. Keyser. Simulation levels of detail for plant motion. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 297–304, 2004.

[5] P. Beaudoin, M. van de Panne, P. Poulin, and S. Coros. Motion-motif graphs. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2008.

[6] J. Chai and J. K. Hodgins. Constraint-based motion optimization using a statistical dynamic model. *ACM Transactions on Graphics*, 26(3), 2007. (SIGGRAPH 2007 Conference Proceedings).

[7] S. Coros, P. Beaudoin, K. K. Yin, and M. van de Pann. Synthesis of constrained walking skills. *ACM Transactions on Graphics*, 27(5), 2008. (SIGGRAPH Asia 2008 Conference Proceedings).

[8] D. James and K. Fatahalian. Precomputing interactive dynamic deformable scenes. *ACM Transactions on Graphics*, 22(3):879–887, 2003. (SIGGRAPH 2003 Conference Proceedings).

[9] 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*, 26(3), 2007. (SIGGRAPH 2007 Conference Proceedings).

[10] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. *ACM Transactions on Graphics*, 21(3):473–482, 2002. (SIGGRAPH 2002 Conference Proceedings).

[11] Y.-C. Lai, S. Chenney, and S. Fan. Precomputing interactive dynamic deformable scenes. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 281–290, 2005.

[12] Y. Li, T. Wang, and H.-Y. Shum. Motion texture: a two-level statistical model for character motion synthesis. *ACM Transactions on Graphics*, 21(3):465–472, 2002. (SIGGRAPH 2002 Conference Proceedings).

[13] M. Oshita. Smart motion synthesis. *Computer Graphics Forum*, 27(7), 2008. (Pacific Graphics 2008 Conference Proceedings).

[14] F. Perbet and M.-P. Cani. Animating prairies in real-time. In *Proceedings of the 2001 Symposium on Interactive 3D graphics*, pages 103–110, 2001.

[15] K. Pullen and C. Bregler. Motion capture assisted animation: texturing and synthesis. *ACM Transactions on Graphics*, 21(3):501–508, 2002. (SIGGRAPH 2002 Conference Proceedings).

[16] P. S. A. Reitsma and N. S. Pollard. Evaluating motion graphs for character animation. *ACM Transactions on Graphics*, 26(3), 2007. (SIGGRAPH 2007 Conference Proceedings).

[17] C. Rose, M. Cohen, and B. Bodenheimer. Verbs and adverbs multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18(5):32–40, 1998.

[18] A. Safonova and J. K. Hodgins. Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics*, 26(3), 2007. (SIGGRAPH 2007 Conference Proceedings).

[19] A. Schodl and I. A. Essa. Controlled animation of video sprites. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 121–127, 2002.

[20] A. Schodl, R. Szeliski, D. H. Salesin, and I. Essa. Video textures. In *ACM SIGGRAPH 2000 Conference Proceedings*, pages 489–498, 2000.

[21] H.-P. Seidel. Polar forms for geometrically continuous spline curves of arbitrary degree. *ACM Transactions on Graphics*, 12(1):1–34, Jan. 1993.

[22] A. Treuille, Y. Lee, and Z. Popovic. Near-optimal character animation with continuous control. *ACM Transactions on Graphics*, 26(3), 2007. (SIGGRAPH 2007 Conference Proceedings).

[23] J. Wang and B. Bodenheimer. Synthesis and evaluation of linear motion transitions. *ACM Transactions on Graphics*, 27(1), 2008.

[24] L. Zhang, Y. Zhang, Z. Jiang, L. Li, W. Chen, and Q. Peng. Precomputing data-driven tree animation. *Computer Animation and Virtual Worlds*, 18(4-5):371–382, 2007. (Comptuer Animation and Social Agents 2007 Conference Proceedings).

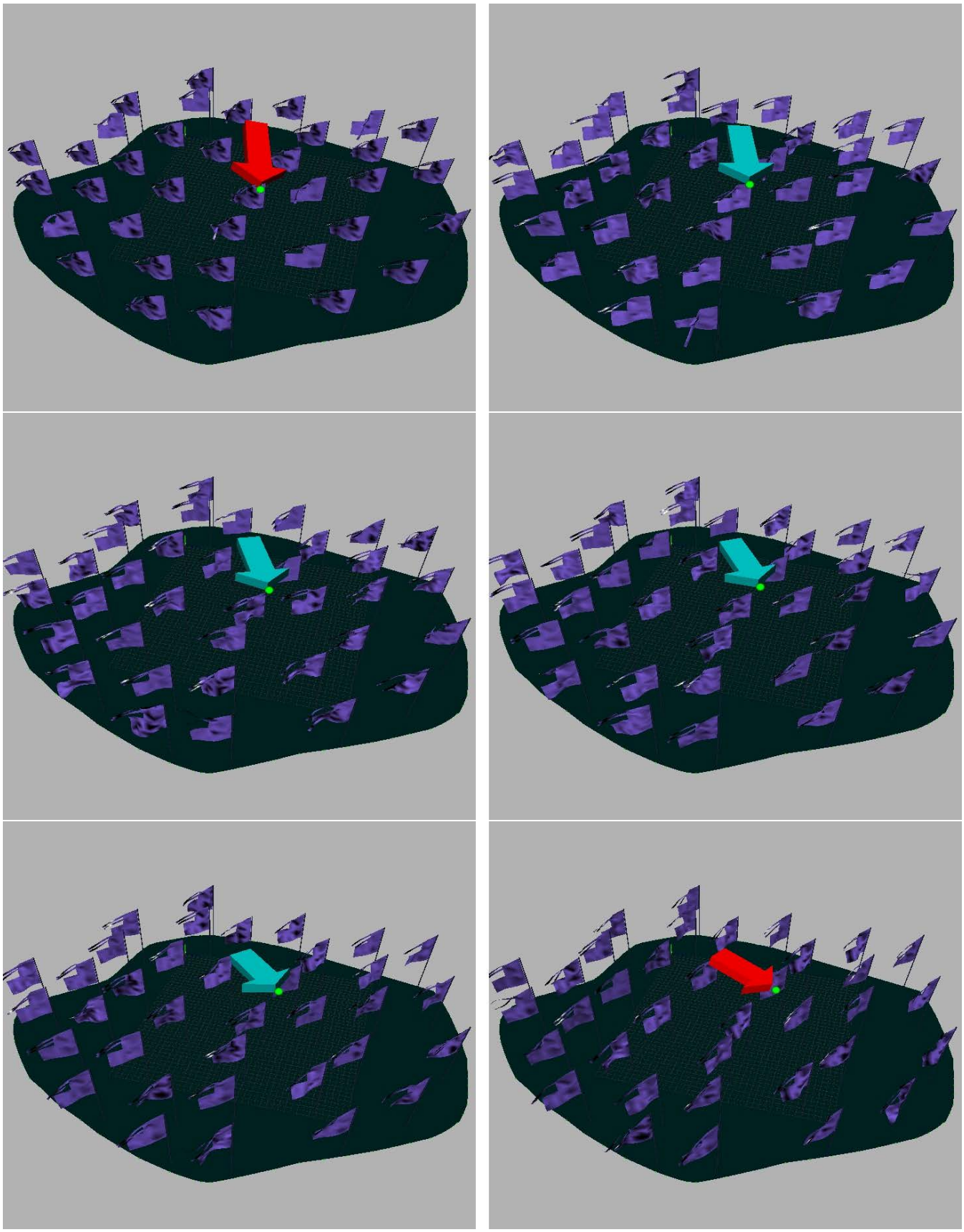

Figure 7: The flag scene is composed of two types of flags, each consist of only 300 frames.