Interactive and Flexible Motion Transition

Jen-Yu Peng
Dept. of Computer Science, National Chiao Tung University, Taiwan.

I-Chen Lin
Dept. of Computer Science, National Chiao Tung University, Taiwan.

Jui-Shiang Chao
Institute Institute for Information Industry, Taiwan

Yan-Ju Chen
Dept. of Computer Science, National Chiao Tung University, Taiwan.

Gwo-Hao Juang
Dept. of Computer Science, National Chiao Tung University, Taiwan.

Figure 1: The synthesized frames with our approach

Abstract

In this paper, we present an example-based motion synthesis technique. Users can interactively control the virtual character to perform desired actions in any order. The desired action can be not only recorded or precomputed motion, but also parametric synthesized one to attain the precise control of avatars. Moreover, a user can change their commands any time to switch to another action according to the instant response of opponents in fighting. The quality transition motions between consecutive actions are rapidly synthesized through traversing a simple graph structure which represents the transition relationships between different poses. The graph is constructed according to clustering on frames in a corpus of motion capture data. With the precomputation of path finding, our approach can also be applied to real-time applications. Besides, this precomputed graph structure can be used to transit those motions not included in the database. Furthermore, our approach is automatic without any human intervention. The final results demonstrate the potential of our algorithm.

Keywords: motion synthesis, motion capture, transition.

1 Introduction

In recent years, character animations widely appeared in entertainment industry. Among the production methods of computer animation, motion capture is a popular and efficient way to acquire the desired movements. The subtleties and human styles of performances can also be well preserved. Hence, the movements of virtual avatars with motion capture data look more realistic than keyframed ones by experienced artists. However, reusing those captured data take some efforts to rearrange or modify the motion data for different conditions. It is usually difficult when the configuration is obvious different from the original one. For this reason, the related topics with data-driven methods became the main stream of motion synthesis techniques in the last decade.

In this paper, we propose an adaptable framework for example-based motion synthesis. The proposed approach exploits a structure called "flexible transition graph" which represents the transition relationships between different poses to synthesize the reliable transition motions. Our goal is to interactively control the avatars in real-time applications. Therefore, we not only pre-compute the candidate paths for each node-pair in the graph but also combine with a parametric motion synthesis technique to attain the controllable avatars. The results demonstrate the potential of our method. Unlike other motion synthesis techniques, our approach animates the avatars through specifying a sequence of desired actions directly. Even with distinct dissimilarity between consecutive actions, our algorithm could produce smooth transitions. Besides, the pre-computed graph can also be applied for any newly synthesized motion not included in the database. In the condition of data lack, we can still use the node information of our graph to improve the interpolation through an integrated novel Bezier scheme.

The remainder of this paper is organized as follows. Section 2 reviews related literatures about the motion synthesis techniques with data-driven methods. Section 3 gives an overview of our approach, while Section 4 and Section 5 describe the preprocess works and run-time works respectively. Our results are shown in Section 6. Finally, Section 7 makes the detailed discussions and conclusions.

2 Related Works

Data-driven motion synthesis techniques can be classified into several categories. Researchers mainly focused on the techniques of blending and transition in the last decade. Blending is a common technique in early researches to synthesize a novel motion by weighted combination of a set of example motions [1][2][3][4]. Besides, blending is also useful for synthesizing continuous locomotions in real-time [5][6]. L. Kovar and M. Gleicher further used the parameterization of motions to obtain the precise control of avatars [7].

Motion transition is used to assemble two different clips into a single continuous motion sequence. The strategy of exploiting a graph structure was frequently adopted in recent studies for creating transitions and interactive controls of avatars.

In graph-based transition techniques, A. Sch”odl et al. [8] first introduced “Video textures” which searches the coherence in frames of image sequence. By reassembling images, they could produce endless videos with a small database. L. Kovar et al. [9] applied this concept to character animation and built a motion graph which concatenates small motion clips to generate a continuous and controllable stream of frames. Furthermore, J. Lee et al. [10] also adopted a two-layer graph structure to control the avatars based on the same idea. We utilize a similar graph structure. However, while they use their graphs to find possible actions in the following frames, we exploit the proposed graph to find appropriate transitions after users specify particular actions. O. Arikan and D. Forsyth [11][12] also presented a similar graph, but they collapsed the frames belonging to the same motion sequence together. Each node represents a motion in order to simplify the original graph based on frames. Although it can also create the transitions between different actions, they can’t directly apply their method to motions not included in the database due to the unavoidable precomputation of graph. Later, T. H. Kim et al. [13] constructed a movement transition graph by finding the transitions among the prototype movements representing the groups. Each node in their graph indicates a group of similar movements. Gelicher et al. [14] created the transition motions around the common poses. The common poses can be found automatically by their system or manually selected by human interventions. A user can control the avatars through guiding the traverse order in their graph. Notwithstanding, they can create the transition between different poses and specify a path to synthesis a stream of frames. The motion selection through their interface is not intuitive; by contrast, our system controls the avatars through motion specification. P. S. A. Reitsma and N. S. Pollard [15] applied the motion graph for the character navigation. H. J. Shin and H. S. Oh [16] proposed the Fat Graphs inspired by Snap-Together Motion [14]. They blended the similar edges between a node pair to interactively control the avatars in continuous parameter spaces. Their example motions for parameterization must be both similar in the beginning and in the end. Furthermore, they also inherited the limitation of Snap-Together Motion. R. Heck and M. Gleicher [17] proposed a parametric motion graph to synthesize the transition between two parametric synthesized motions. Therefore, they can interactively control the avatars with some specified motions. For each new type of actions, they must insert a node to extend their graph even there is only one example. For this reason, users’ choices will be bounded while we can reach the same goal without this limitation.

Ikemoto et al. [] blended multiple motion clips to create compact transitions. We share the same goal of synthesizing a frame-to-frame transition and both applied the clustering on motion data. But the significant difference is that their clustering method is clip-based while ours is frame-based. Besides, they cached the high-score intermediary clip for each representative pair while we precompute the transition paths.

Besides, there are some related approaches that can be used to improve the visual quality of motions synthesized by graph-based transition techniques. J. Wang and B. Bodenheimer not only tuned the cost evaluation for selecting transitions [18] but also computed the optimal duration for blending [19]. Moreover, the footskate cleaning methods [20][21] can be used as a post-process to avoid the artifacts caused by foot sliding. In order to improve the physical correctness, A. Safonova and J. K. Hodgins interpolated the position of center of mass instead of the position of root joint [22]. Furthermore, the naturalness of synthesized moiton can be verified by the classification method [23].

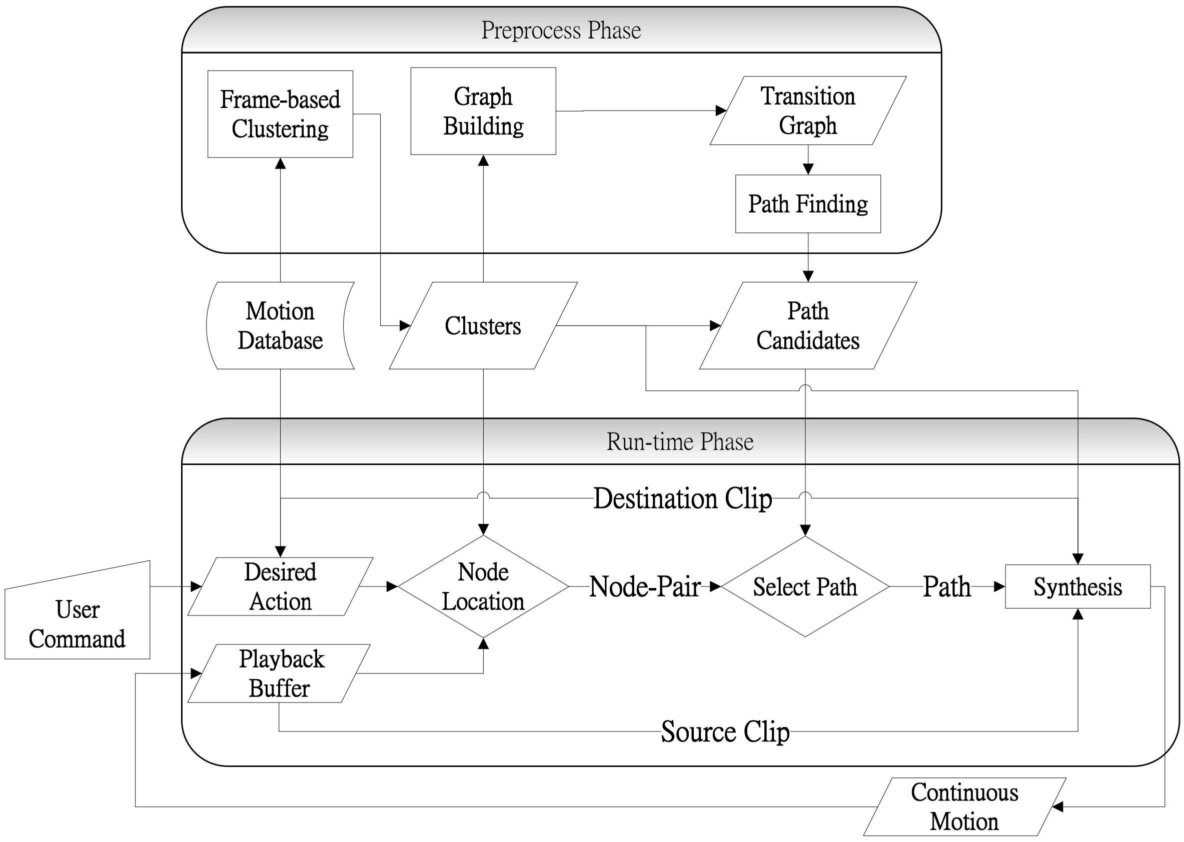


Figure 2 : The framework of our approach

3 Overview

Our approach can be divided into two phases: the preprocess phase and the run-time phase. Figure 2 shows the framework of our approach. In the preprocess phase, given a corpus of motion data, we apply a frame-based clustering first and extract information hidden among those clusters such as the consecution of frames to construct a transition graph. Finally, we precompute candidate transition paths for each node-pair in the graph. With this precomputation, we can apply our approach efficiently in real-time applications.

In the run-time phase, given a user command, the system turns it into specified desired actions. We first locate the last frame of the playback buffer and the first frame of the desired action in the graph. Then, we query the pre-computed data with this located node-pair and select one transition path from possible candidates. Finally, the transition motion is synthesized according to the order of visiting nodes in that path. Users can get a continuous motion by assigning a sequence of actions. The desired motion can be the parameterized motion to attain the control of avatar.

4 Preprocess Phase

4.1 Frame-based two-pass Clustering

In the first step, we apply clustering for all frames in our motion database. A two-pass algorithm is proposed to classify the frames into different clusters. This algorithm is efficient due to its low memory requirement and less computation time. In the beginning of clustering, we just create one typical cluster and take the first frame to be its center frame. In practice, the center frame is not only a frame in the center of a cluster but also represents the cluster's location in the pose space. For each incoming frame later, we calculate the distance between the new one and each center frame of existing clusters. Our distance calculations for two poses in text are all referred to Lee's cost metric. [10] In the first pass, if the minimum distance is less than a user-specified threshold, we will label the new frame with the ID of that cluster. If the minimum distance is larger than two times of the threshold, we will create a new cluster and assign that frame to be its center frame. Otherwise, this frame will be labeled as "undefined". Those undefined frames will be assigned once again in the second pass, but we only label them with the ID of the nearest cluster in this time without creating any new cluster. After the second pass, all frames in the database are assigned with a cluster ID.

As shown in Figure 3, in the first pass, the first frame is assigned as the center of 1st cluster. Frame 5 to frame 7 are all out of the radius of the first cluster but not far enough to form a new cluster. Therefore, we label them with "undefined". Until to 8th frame, the second cluster is created. After that, frame 9 and 10 are also located in same cluster while the frame 11 and 12 leave the cluster again. In the second pass, we check the undefined frames and find their nearest clusters. Frame 5 is closer to first cluster than second cluster while frame 6, 7, 11 and 12 are all closer to second cluster.

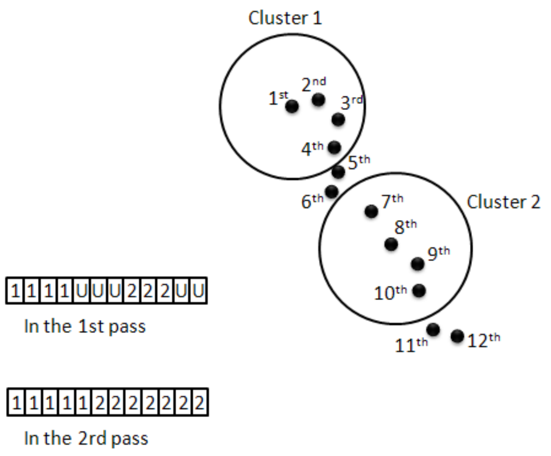


Figure 3 : Frame-based two-pass clustering

Although this clustering method is simple, it is useful enough to accomplish the task of dividing the whole pose space into several smaller regions. After the clustering, we observe that each cluster covers a range of pose space and all of its member frames are distributed over this region. Furthermore, the contiguous frames in the same motion are either in the same cluster or be separated into neighboring clusters. With these properties, we can construct a sparse transition graph in the next step.

4.2 Build the Transition Graph

In this step, we will introduce how to build a transition graph that represents the transition relationships between different poses.

In the initial configuration, we represent each cluster as one node in the graph. We then concatenate the contiguous frames in the same cluster to form a clip. Hence, each node will attach a list of short clips instead of individual frames now. After that, the edge connected any pair of nodes which we called "node-pair" later will be represented by the concatenation of contiguous clips in the corresponding clusters. In general, there always exist multiple edges between the same node-pair.

As illustrated in Figure 3, the motion is divided into two shorter clips after the clustering. The first clip belong to cluster 1 contains frame 1 to frame 5 while the second clip belong to cluster 2 contains frame 6 to frame 12. Therefore, there is one edge between the nodes of cluster 1 and cluster 2 and it contains frame 1 to frame 12.

Instead of checking all possible edges, we propose a more efficient way here to rapidly detect edges in one pass. For each motion in our database, we had labeled each frame with a cluster ID in Section 3.2. With the above mentioned work, we create an edge if there are two consecutive frames in a motion belong to different clusters. The created edge will connect these two corresponding nodes.

As shown in figure 4, there are three different motions. Each of them has been divided into small clips belonged to different clusters. We can construct the directed edges according to consecutive frames belong to different clusters. The right graph was built according to the left information.

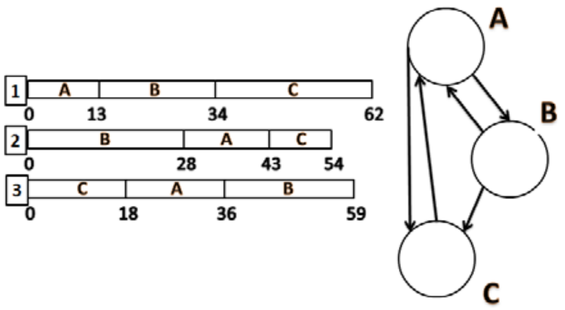


Figure 4: The graph building

Our transition graph can be either directed or undirected. If we take the former assumption, the node of the preceding clip will direct toward that of the succeeding clip when creating an edge. Therefore, the direction of an edge represents smooth transition from one to another. On the other hand, if we use a clip to create an undirected edge, the node-pair can transit to each other only when we permit the backward playing of that clip. For example, if we want to create a transition from node A to node B, we find those clips representing this edge. Some of them may be the transition motions from node B to node A, we can still use them in a backward playback order.

4.3 Pre-computed Paths

After constructing the graph, we compute several candidate paths for each node-pair in the graph with the dynamic programming. The cost of each edge is the number of frames of its shortest clip multiplied by the distance between the two center frames of the corresponding clusters. We summed up the total cost of all passed edges in that path and termed it as "transition cost" in text.

$$Cost = \sum_{k=1}^{m} Min(E_k) * Dis\tan ce(E_k) \qquad (1)$$

where m is the number of visiting node in that path, E_k is the k-th edge in the path. The function Min return the minimum frame numbers of candidate clips in E_k while the Distance answer the pose difference between the two end nodes of E_k.

In order to balance the storage space and the flexibility, for each node-pair, we only keep a maximum number of five paths whose transition costs are less than a threshold. These preserved paths called "candidate paths" and paths with high costs called "invalid path". Since these invalid paths usually cause noticeable artifacts in our experiences, we discard them even when there is no option.

Even though, all motions in the database are used for edge connection, the graph is usually not strongly connected and there are still node-pairs without path connection in practice. We will discuss how to perform motion transition under such condition in the run-time phase.

We also recorded the transition cost for each candidate path in order to provide some references for later choices.

5 Run-time Phase

In our system, we provide users with various character actions. They can be input by keying the action ID or associated with any button of keyboard or joystick. The user requests can be responded interactively either by choreographing the performances in any order or change the current action on the half way. The middle transitions will be automatically synthesized by our approach.

5.1 Locate the Node-pair

In the run-time applications, if we want to create the transition from a source clip to a destination clip which may be the action specified by users, we pick the last frame of source clip and the first frame of destination clip. In the following, we find their nearest clusters respectively by comparing those distances between them and the center frame of each cluster. Therefore, these frames may not belong to any cluster but can still find a closest one. Although there will appear more and more artifacts with the increasing distance from center frame, the strategy of trying to find the nearest cluster instead of the exact one can help us to handle some conditions we never concerned about in the database.

For example in figure 5, the last frame of source clip is located in the cluster N_s and the first frame of destination is located in the cluster N_t.

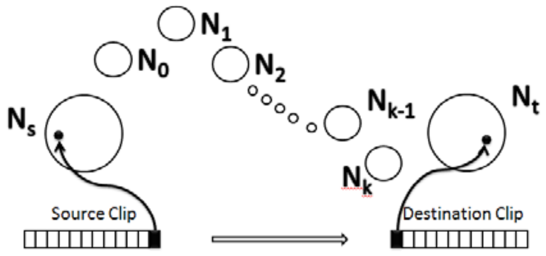


Figure 5: The location of node-pair

Now, we have one cluster ID for the last frame of source clip and the other for the first frame of destination clip. We will use this information in the next step.

5.2 Select One Path

Given a node-pair, we can query the pre-computed data to get some candidate paths instead of finding the shortest path in the run time. Between these candidate paths, we randomly select one in order to provide the flexibility. However, the selection is accorded to a probability which is estimated in an inverse proportion to the transition cost. In the example of figure 5, we select one path that passes through from N_0 to N_k in an increasing order.

5.3 Synthesize the Transition

Once the optimal transition path is selected, we can synthesize the transition motion according to the order of visiting nodes in that path. In our approach, we adopt a greedy scheme to synthesize the transition.

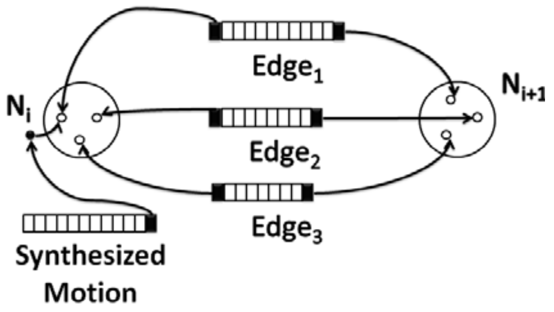


Figure 6 : The best match selection

For each node-pair in the path, such as N_s-N_0, N_0-N_1 or N_1-N_2 in the figure 5, we find a clip that can best match the synthesized motion while its length is also less than a threshold. Here, the best match means that the distance between the first frame of that clip and the last frame of the synthesize motion is the minimum. As shown in Figure 6, we will select Edge$_1$ from the three candidate edges between the node-pair N_i-N_{i+1}.

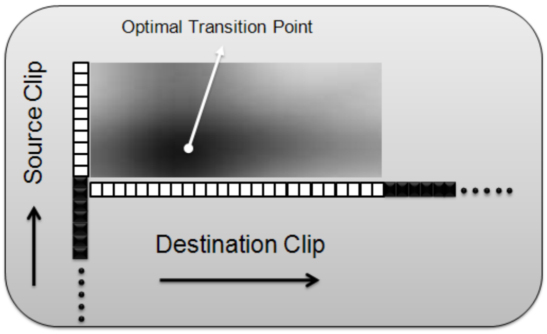


Figure 7: The optimal transition point selection

This solution is not optimal but efficient for real-time applications. We then compute the distance grid of the back end of synthesized motion and the front end of that clip to find an optimal transition point. The optimal transition point [17] is a point with the minimum distance in this grid. In our experiment, we take 10 frames for the synthesized motion and 25 frames for the incoming clip. As illustrated in Figure 7, the distance grid is normalized and represented by the color of gray level.

We repeat this process until all selected clips of node-pairs have been merged. Now, the synthesize transition motion is a continuous motion.

5.4 Bezier Interpolation

For those node-pairs which can't find any path in the graph, which indicates their transitions doesn't exist in the database, we propose a novel scheme of Bezier interpolation to overcome this difficult challenge. This scheme not only efficiently exploits the cluster information in the pose space but also creates more smooth transitions. However, the traditional linear interpolation is superior only when the difference is subtle. In the condition of no path, we assume that the pose difference between the node-pair is obvious. Therefore, we prefer using the Bezier scheme instead of applying the linear one.

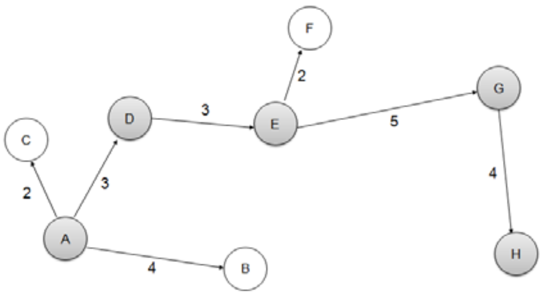


Figure 8: Path Selection

In order to get the control points of Bezier function, we execute a simple traverse process with all cluster medoids. Starting from the source node such as node A in Figure 8, we first filter out the nodes whose distance from the target node is larger than the distance between the current node and target node. For example, we never consider node C even it is the closest one. Then, we select the nearest node in the remaining nodes to be the next visiting node. Instead of choosing node B nearer to target node, we select node D nearer to the current node. This traverse process will be repeated until reaching the target node. In Figure 8, the visited order is A, D, E, G and H. After that, we divide these nodes into two half and compute their center nodes which are virtual nodes computed by averaging the visited nodes. The center nodes such as the dotted nodes C1 and C2 in Figure 9 are used to be the control points in a quadratic Bezier interpolation. Therefore, the interpolated results are not only smooth but contain the clues of posture distributions in the same time.

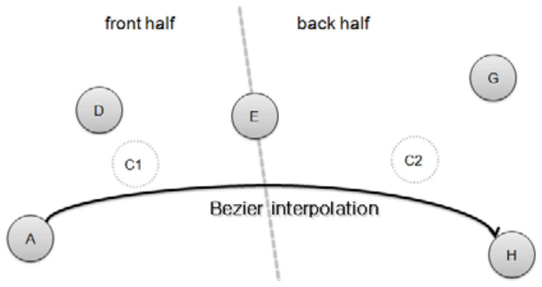


Figure 9: The quadratic Bezier interpolation

The number of frames produced by an interpolation is decided by the distance between the source pose and the target pose. This criterion can avoid the jerky effects and its detailed equation is listed below.

$$T = D(Pose_{source}, Pose_{target}) / \sigma \qquad (2)$$

where T means the suitable duration of the interpolation, D is the cost function referred to [10] and δ is a user specified parameter that controls the average length of interpolated motion.

6 Experimental Results

6.1 Platform

Our experiments perform on a laptop with a 1.73GHz Pentium-M processor, 1 GB main memory with Windows XP operating system. Besides, the graphics card we used is ATI Mobility Radeon X700.

6.2 Database

Our database includes 73 motions with a 30 fps sampling rate. In order to control the avatar by specifying actions, each motion only contains a simple action of martial art or locomotion. These short motion sequences were separately captured or manually segmented while it can also use the automatic schemes to divide the long sequences into shorter sequences. For parametric motion synthesis, we captured nine motions of different directions for any type of actions. The skeleton we adopted contains 19 joints and 60 degrees of freedom (DOFs) and the total number of frames in our database is only 3648. Even with such a small database, we can still produce the quality results.

6.3 Examples

All the examples in this section can be found in our accompanying video. In the first example, we will demonstrate a sequence of actions of martial art. The synthesized frames are shown in Figure 1. We arrange the six arbitrary actions in martial art with a specific order and perform them once at a time to show the audience their original movement first. Finally, our approach automatically synthesizes the continuous, high-fidelity streams of motion.

In addition to make the transition between different clips, we also combine the parametric motion synthesis technique to attain the precise control of avatars. In this example, the avatar successfully hit the target point after performing various actions, as shown in Figure 10. Here, our parametric motion synthesis technique is referred to [7]. In order to save the huge storage space for the parametric parameters, we divided the parameterized space into discrete voxels. This scheme is similar to that of Lee's [24] but they used the space segmentation for the captured motions.

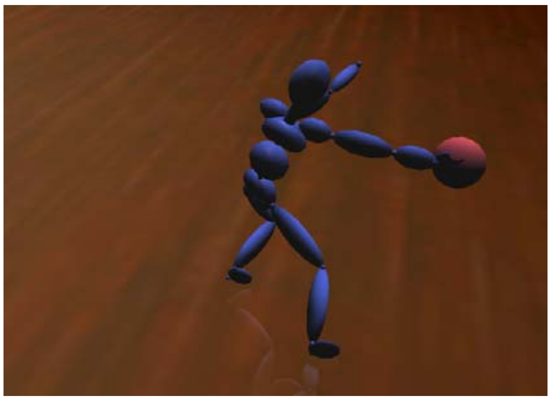
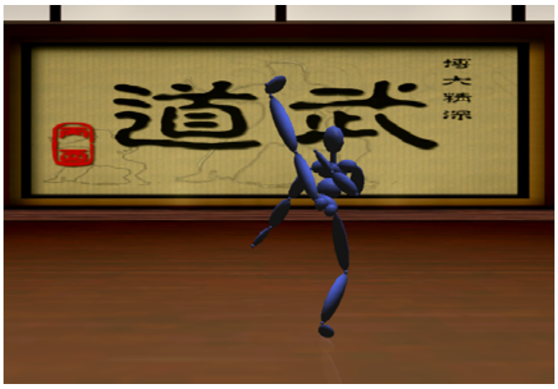
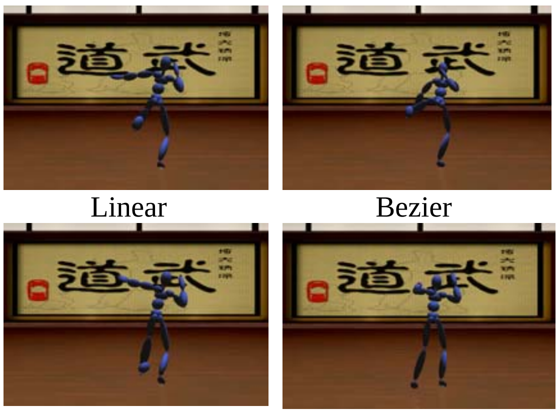


Figure 10: Parametric motion synthesis

The third example shows the comparison between the traditional linear interpolation scheme and our scheme. We create an interpolation between a kick posture and a punch posture. The traditional scheme moves the right hand and right leg in the same time causing the unbalanced artifact while our scheme knows to shrink the leg first before punching. The results are illustrated in Figure 11. Besides, we can preserve the C1 continuity through the Bezier interpolation while linear interpolation only provides C0 continuity





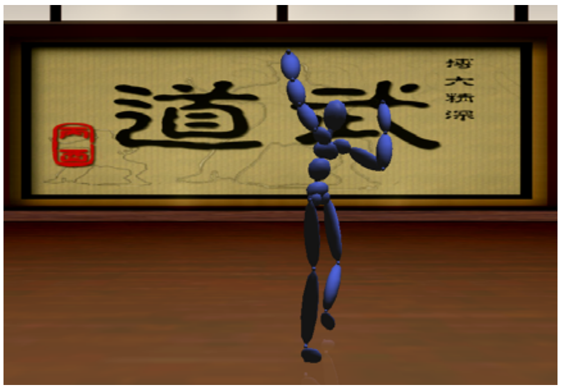


Figure 11: Different interpolation scheme

Finally, we demonstrate the flexibility of our approach. The user commands one character to walk forward, but there is another sneaker which wants to attack him. The player was aware of the approaching punch and squat immediately to dodge this attack. We just cleaned up the end of playback buffer and locate the new end of playback buffer and the first frame of desired action.

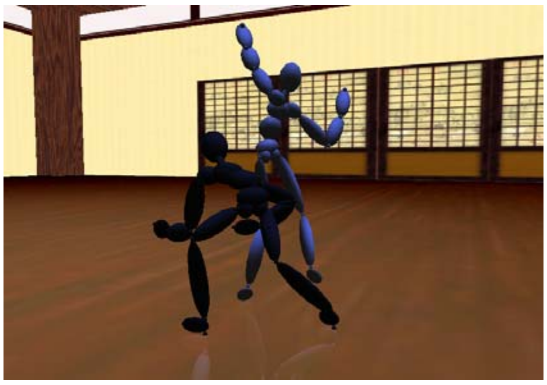


Figure 12: Dodge example

6.4 Statistics

The clustering divided the frames into 164 clusters with 785 short clips attaching to these nodes. The transition graph includes 660 edges.

Example	Time(sec)	Frames
Consecutive Attacks	0.157	150
Parametric Synthesis	0.561	233
Bezier Interpolation	0.029	20
Dodge	0.155	121

Table 1 : The synthesis time for examples

The computation time for synthesizing each example is shown in Table 1. All examples use the same transition graph where the clustering needs about 0.76 seconds and the precomputation of path candidates spends about 463.46 seconds. We observed that the maximum average time for each frame is less than 0.0024 second. Therefore, we claim that the proposed approach is adequate for run-time applications on regular PCs or laptops with multi-threaded programming.

7 Discussion

In this paper, we present a novel motion synthesis approach for producing continuous, controllable sequences of motion with motion capture data.

Our two pass clustering method is simple but reliable. Other more complicated clustering techniques such as k-means or mean-shift may provide more delicate clusters but will cost much more pre-computation time. For example, the k-means clustering spends 8211.88 second for 100 iterations when k is 150 while ours is less than 1 second.

The user specified threshold for our clustering indicates the covering radius of each cluster. Therefore, the bigger the value, the less clusters. Although the size of the preprocess data can be reduced substantially with a small radius, the artificial errors obviously increase on the contrary. The necessary space for storing preprocess data are huge with the storage complexity of $O(n^2)$. Therefore, how to balance the accuracy and necessary space is critical in our approach.

As shown in the final examples of our video, our approach can permit the avatar to change his action extemporarily. This technique is especially useful in fighting games. In traditional video games, they usually use the technique of move tree where the avatar must return to the ready pose first before performing the next action. However, we can overcome this limitation with the proposed approach. The player can interactively control their character according to the reaction of opponents.

The transition motions from poses to poses are usually short because a long transition motion will substantially destroy the timing of motion. Due to the short duration property of transition, we assume that the transition can still keep the naturalness even in backward playing.

The future works includes the reduction of the preprocess data and speed up the computation of synthesis. For example, we will adopt a hierarchical clustering scheme to reduce the location time. The time complexity will decrease to $O(\log_n)$ from $O(n)$ where n is the number of existent clusters.

Acknowledgements

This research was supported by the III Innovative and Prospective Technologies Project of Institute for Information Industry and sponsored by MOEA, Taiwan. This research is also partially supported by National Science Council, Taiwan, with grant no. NSC-95-2221-E-009-164-MY3.

Reference

[1] M. Unuma, K. Anjyo and R. Takeuchi. Fourier principles for emotion-based human figure animation. In Proceedings of ACM SIGGRAPH, pages 91-96, 1995.

[2] D. Wiley and J. Hahn. Interpolation synthesis of articulated figure motion. IEEE Computer Graphics & Applications 17(6):39-45, 1997.

[3] C. Rose, M. F. Cohen and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. IEEE Computer Graphics & Applications 18(5):32-40, 1998.

[4] M. Brand and A. Hertzmann. Style machines. In Proceedings of ACM SIGGRAPH, pages 183-192, 2000.

[5] S. I. Park, H. J. Shin, and S. Y. Shin. On-line motion blending for real-time locomotion generation. Computer Animation and Virtual Worlds, 15(3):125-138, 2004.

[6] P. Glardon, R. Boulic and D. Thalmann. A Coherent Locomotion Engine Extrapolating Beyond Experimental Data. In Computer Animation and Social Agents (CASA), pages 73-84, 2004.

[7] L. Kovar and M. Gleicher. Automated Extraction and Parameterization of Motions in Large Data Sets. ACM Transactions on Graphics, 23(3):559-568 , 2004.

[8] A. Sch”odl , R. Szeliski, D. Salesin and I. Essa. Video textures. In Proceedings of ACM SIGGRAPH 2000, Annual Conference Series, ACM SIGGRAPH, pages 489-498, 2000.

[9] L. Kovar, M. Gleicher and F. Pighin. Motion graphs. ACM Transactions on Graphics, 21(3):473-482, July 2002.

[10] 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 Transactions on Graphics, 21(3):491-500, July 2002.

[11] O. Arikan and D. Forsyth. Interactive motion generation from examples. ACM Transactions on Graphics, 21(3):483-490, 2002.

[12] O. Arikan, D. Forsyth and J. O'Brien. Motion Synthesis from Annotations. ACM Transactions on Graphics, 22(3): 402-408, 2003.

[13] T.H. Kim, S. I. Park, and S. Y. Shin. Rhythmic-Motion Synthesis Based on Motion-Beat Analysis. ACM Transactions on Graphics, 22(3):392-401, 2003.

[14] M. Gleicher , H. J. Shin , L. Kovar and A. Jepsen. Snap-together Motion: Assembling Run-time Animation. In ACM SIGGRAPH Symposium on Interactive 3D Graphics, pages 181-188, 2003.

[15] P. S. A. Reitsma and N. S. Pollard. Evaluating Motion Graphs for Character Navigation. In Proceedings of Symposium on Computer Animation, pages 89-98, 2004.

[16] H. J. Shin and H. S. Oh. Fat Graphs: Constructing an interactive character with continuous controls. In Proceedings of Symposium on Computer Animation, pages 291-298, 2006.

[17] R. Heck and M. Gleicher. Parametric Motion Graphs. In Proceedings of Symposium on Interactive 3D Graphics and Games 2007, April 2007.

[18] J. Wang and B. Bodenheimer. An Evaluation of a Cost Metric for Selecting Transi-tions between Motion Segments. In Proceedings of Symposium on Computer Animation, pages 234-238, 2003.

[19] J. Wang and B. Bodenheimer. Computing the Duration of Motion Transitions: An Empirical Approach. In Proceedings of Symposium on Computer Animation, pages 337-346, 2004.

[20] L. Kovar, J. Schreiner and M. Gleicher. Footskate Cleanup for Motion Capture Editing, In Proceedings of the Symposium on Computer Animation, pages 97–104, 2002.

[21] L. Ikemoto, O. Arikan and D. Forsyth. Knowing when to put your foot down. In ACM SIGGRAPH Symposium on Interactive 3D Graphics, pages 49-53, 2006.

[22] A. Safonova and J. K. Hodgins. Analyzing the Physical Correctness of Interpolated Human Motion. In Proceedings of Symposium on Computer Animation, pages 171-180, 2005.

[23] L. Ren, A. Patrick, A. Efros, J. Hodgins, and J. M. Rehg. A Data-Driven Approach to Quantifying Natural Human Motion. ACM Transactions on Graphics, 24(3): pages 1090-1097, 2005.

[24] J. Lee and K. H. Lee. Precomputing avatar behavior from human motion data. In Proceedings of Symposium on Computer Animation, pages 79-87, 2004.