Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2006)
M.-P. Cani, J. O'Brien (Editors)

Fat Graphs: Constructing an interactive character with continuous controls

Hyun Joon Shin and Hyun Seok Oh

Division of Digital Media, Ajou University, Suwon, Korea

Abstract

This paper proposes a methodology that allows users to control character's motion interactively but continuously. Inspired by the work of Gleicher et al. [GSKJ03], we propose a semi-automatic method to build fat graphs where a node corresponds to a pose and its incoming and outgoing edges represent the motion segments starting from and ending at similar poses. A group of edges is built into a fat edge that parameterizes similar motion segments into a blendable form. Employing the existing motion transition and blending methods, our run-time system allows users to control a character interactively in continuous parameter spaces with conventional input devices such as joysticks and the mice. The capability of the proposed methodology is demonstrated through several applications. Although our method has some limitations on motion repertories and qualities, it can be adapted to a number of real-world applications including video games and virtual reality applications.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Encouraged by powerful 3D graphics hardware devices, interactively controlled characters are widely used in many applications including video games, broadcasting, and virtual reality applications. In order to make an interactive character believable, its motion should be realistic. Simultaneously, the animation synthesis must be efficient enough to make the character respond to the user's control immediately.

There has been a good amount of research results on interactively controlled characters. One of the successful approaches is the graph-based approach. In order to rearrange an existing animation temporally, this approach analyzes the given animation data and builds a motion graph while rearranging pieces in a connected way. Sometimes, graphs with contrived structures are built to enhance responsiveness of character controls while providing the user with discrete choice. Another widely used approach is motion blending. Combining the existing animation in the space-domain, this approach enables the user to control the animated character continuously and precisely in parameter spaces.

This paper suggests a methodology that allows users to interactively and continuously control animation of a character with a little run-time overhead. To do this, we propose *fat graphs*. A node in a fat graph denotes a group of poses and the incoming and outgoing edges represent the motion segments starting from and ending with the poses. A group of edges builds a *fat edge* that parameterizes similar motion segments into a blendable form. At each node, the user can choose one of the outgoing edges which correspond to available actions starting from the pose at the node. While the motion segment corresponding to a selected edge is being realized, motion can be controlled precisely in a continuous parameter space.

Character motion control methods in interactive applications involving graphs and motion blending have been active for decades. For instance structures similar to motion graphs, called *move trees* have been utilized in the game industry while applications requiring continuous control over character motion have employed motion blending techniques. However, in most cases, graphs are constructed and motion segments are parameterized manually. Similarly to the work of Gleicher et al. [GSKJ03], we develop a efficient near-automatic method to build an interactive character controlled in continuous parameter spaces.

Our work involves three main steps:

- We propose a novel data structure that allows users to produce a variety of character motion based on two existing methods: graph based approach and motion blending.
- Our scheme entails building parameterization on a set of motion segments for run-time control with conventional input devices.
- We also provide a run-time methodology to control the character motion with conventional input devices such as joysticks and mice coupled with keyboards.

This paper consists of five sections. In Section 2, we review all related work. In Sections 3 and 4, we exhibit our method for authoring an interactive character. In Section 5, a scheme that allows the user to synthesize character motion interactively while controlling motion continuously is presented. Finally, we demonstrate our experimental results in Section 6 and discuss our contributions and the limitations of the proposed method in Section 7.

2. Related Work

A graph-based approach [AF02, KGP02, LCR*02, CLS03, AFO03, GSKJ03, KPS03, LL04] is one of the most successful approaches to produce a novel animation based on an existing motion. This approach builds sophisticated data structures which store the connectivity information between animation frames of the given motion. This information is then used to rearrange the animation frames into a novel sequence. Using this information, one can resort to search a quality animation sequence that satisfies a user specified goal can be synthesized efficiently by means of searching.

The work of Gleicher et al. [GSKJ03], which has inspired us involves design of a special data structure to efficiently produce animation for an interactively controlled character together with an authoring method for such a data structure. We use a similar data structure and also build graphs with contrived structures to construct the initial form of fat graphs. However, our data structure is further improved to allow users to control animation of a character in a continuous manner. This is done via grouping the edges that contain similar motion segments automatically with little user intervention.

Given a set of similar motion segments, motion blending is the most widely used technique to produce a motion satisfying the user specification represented by continuous values. Wiley and Hahn proposed synthesis of a motion that approximately passes through the user specified point [WH97]. They parameterized the reachable space of a body part geometrically using a dense grid and produced a motion segment that reached the user specified point on the space. Rose et al. placed given motion segments on a parameter space based on the characteristics of the motion segments and adopted radial basis functions (RBF) to compute weights of the motion segments corresponding to the user-specified parameters [RCB98]. Park et al. proposed a motion blending technique for on-line real-time locomotion generation [PSKS04]. They parameterized a variety of transportation motion segments based on their speeds, turning angles, and styles and adopted RBF to compute the weights of the motion segments. Kovar and Gleicher proposed a generalized framework to blend two motion segments [KG03]. They extended their work to search blendable pieces from a large motion database and to parameterize the pieces automatically. Mukai and Kuriyama exhibited that the geostatistical blending method produces better quality motion than the conventional ones [MK05].

Our approach to parameterizing motion segments is motivated by those of Wiley and Hahn [WH97], and Kovar and Gleicher [KG03], where motion segments are paremeterized based on their spatial properties. Here, we merge their ideas to establish the mapping between a parameter and corresponding motion spaces. Kovar and Gleicher densely sampled the parameter space to establish the piecewise linear mapping between motion space and parameter space. Wiley and Hahn accelerated run-time composition by carefully supervising the motion capture session to build regular grid on the parameter spaces. Our approach consists of two steps: 1. randomly sampling the parameter space, and 2. building a regular grid on it. By doing this, we can apply our parameterization method to the motion capture data from unsupervised sessions and accelerate motion composition at run-time. Simultaneously we approximate the parameter space with a 2D plane such that the user can choose a parameter with conventional 2D input devices.

3. Fat Graph Authoring

A fat graph consists of nodes corresponding to poses in the given corpus of motion data and edges representing the groups of similar motion segments connecting the poses. Traversing the graph connects the motion segments to produce a sequence of animation. At a node, a user can select an outgoing edge to produce the corresponding action that follows the previous animation. An edge in a fat graph may include more than one similar motion segments, so these segments in the selected edge are blended with one another to produce the precise user-specified motion (See Figure 1).

At a node, the number of outgoing edges corresponds to the actions that start from the pose at the node. Therefore, a graph having a small number of nodes but many outgoing edges offers many choices of action at a time. On the other hand, one can produce a wide variety of quality motions with a large set of blendable motion segments. Therefore, versatility in motion synthesis depends on the connectivity of the graph and the cardinalities of the motion segments in its edges.

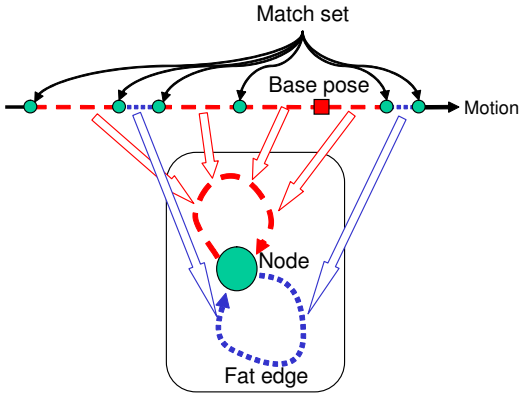


Figure 1: *Fat graph construction: Given a base pose (red box) in the motion, the similar poses (green dots) are chosen to be built into a hub-node (green circle). The motion segments connecting the node (horizontal line segments at top) are grouped (red dashed line and blue dotted line) to construct fat edges (red dashed arc and blue dotted arc.)*

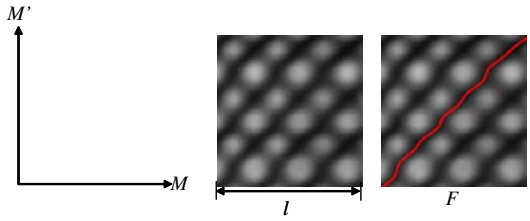


Figure 2: *The distance between two motion segments: Given two motion segments M and M' (left), the distance between every pair of poses are computed (middle). We then cumulate the distance along the correspondence curve (red curve) and normalize the distance by the motion length l (right).*

3.1. Graph Structure Construction

A graph with a contrived structure can be built employing the notion of hub-nodes as in *Snap-together motion* suggested by Gleicher et al. [GSKJ03]. This is a graph node with many incoming and outgoing edges. Therefore, the pose at a hub-node should enable many motions to start from and end at it. For example, a hub-node for a set of martial arts motions would correspond to a ready pose. For a set of walking motion any pose could be built into a hub-node since one can start many walking cycles from at every animation frame. Such graphs were constructed using the method proposed by Gleicher et al. [GSKJ03].

Their approach maintains, a hub-node can be built either automatically or semi-automatically. High connectivity of the graph is enabled by a hub-node being built automatically such that the corresponding pose, called *base pose*, has the maximum number of similar poses, called *match set*. To do this, given the corpus of motion capture data, the distances between every pair of poses are computed adopting the distance metric proposed by Kovar et al. [KGP02]. Every frame, whose distance from the base pose is locally minimal and smaller than a threshold, is collected into the match set of the base pose. We construct a hub-node automatically using the base pose with the largest match set. Here, the threshold can be set to control the trade-off between the quality of output motion and the connectivity of the graph.

Using the base pose and the match set suggested by the system, users can either accept or choose an alternative base pose based on their knowledge on the motion repertories. Post the hub-node selection, the system then provides the user with the corresponding match set. In our implementation, we enable the user to add, delete, and tweak an element of the match set for better motion segmentation. In most of the experiment, the system suggests reasonable base poses and match sets for the given motion. Hub-nodes can be added repeatedly to obtain a desirable structure is obtained.

3.2. Fat Edges

Once the hub-nodes are built, we make fat edges by grouping edges with similar actions. Every edge connecting the same nodes containing similar action can be blended to produce a novel motion. The distance between two motion segments is the squared sum of pairwise distances between every matching pose normalized by the length of a motion segment. In detail, we first establish the dense temporal correspondence between the motion segments. Since the motion segments in the edges are relatively short and we are interested in a pair of motion segments similar to each other, we discovered that a standard dynamic time warping technique provided reasonable results. Given the correspondence map F from a motion M to M', the distance between M and M' is

$$\sum_{t}^{l} \frac{d\left(M(t), M'\left(F(t)\right)\right)^2}{l}, \qquad (1)$$

where $M(t)$ is the pose at t of motion M, $d(\cdot)$ is the distance between two poses as stated above, and l is the length of M (See Figure 2).

Given a set of edges connecting the same nodes, each edge in the set is initially built into an individual group. Every pair of groups is merged into a group when the distance between any edge in one group and any edge in the other group is smaller than a threshold. This threshold can also be set to improve the motion quality sacrificing the versatility of motion control. Repeating the merging process till any pair of groups which cannot be merged yield the final groups which form the fat edges of the graph. In our implementation, the user can manually add or delete edges of a group and alter their ownership.

4. 2D Motion Parameterization

Each of the fat edges contains one or more motion segments to be blended with one another at run-time. Since it is almost impossible for users to specify the blending weight of each motion segment, we parameterize the set geometrically in the authoring step, and then provide the users with geometric control over the blending weights.

Our idea is based on the observation that many users want to control the motion using the position of a joint at a particular time instance. For example, users often want to specify the translation of the body represented by the root translation in order to control walking motion. For kicking motions, they probably want to be precise with the hitting point, which corresponds to that on a foot at impact. Therefore, users first select the joint of interest and specify the time instance. Based on this information, the system generates a parameter space covering the location of the joint at the time instance.

In order to find the relation between the blending weights and the positions of the joint obtained by blending the motion segments, we adopt the idea proposed by Kovar et al. [KG04], which densely samples parameter space randomly and approximates the mapping between them with a piecewise linear function. With randomly generated weight combinations, we sample the joint position acquired by blending the motion segments with the method that we will describe in Section 5.1. In our experiment, we found that a few hundred samples are enough to form a reachable space of the joint achieved by blending the given motion segments.

To incorporate conventional input devices with two control axes such as joysticks and the mouse for motion control at run-time, we need to parameterize the motion with two parameters. This can be done in four steps: plane determination and projection, parametric axis determination, space segmentation, and regular sampling (See Figure 3). In the first step, the 3D sample positions are projected onto the 2D plane that approximates the samples best. For intuitive control at run-time, the parameter space is parameterized once again with novel parametric axes corresponding to control axes of devices. The plane is then segmented into triangles such that a given point on the plane locates the surrounding three samples to form the barycentric coordinate frame. The space is then regularly sampled to reduce the run-time overhead.

Plane determination and projection: Conventional input devices, especially gaming devices usually have two control axes. Therefore, we reduce the dimensionality of the parameter space to two, so the parameter value can be selected with such devices intuitively and immediately. To do this, we project the randomly sampled joint positions onto a plane. The best approximating plane is spanned by two principal axes of the samples. The samples are then projected onto the plane and the projections construct a two-dimensional parameter space. Since we sample the joint position of only similar motion segments at the matching time instance, our experiments showed that the samples form a near planar surface in 3D space and the projections do not lead to much error (See the second and third columns of Figure 3.)

Parametric axis determination: Now, we need to determine two orthogonal axes on the plane that correspond to the control axes of input devices. Note that the principal axes do not necessarily match the parametric axes with which to control the motion intuitively. We observed that it is a good choice to assign one axis (usually y-axis of input device) to 3D y-axis if possible. Where the plane is near horizontal, the axis of an input device is mapped to the sagittal axis which corresponds to the front direction of the character. Based on those observations, we first examine the plane to determine if it is upright enough to be parameterized with 3D y-axis by measuring the angle between y-axis and the normal vector of the plane. If the angle is smaller than $\frac{\pi}{4}$, the projection of y-axis and its orthogonal axis on the plane are selected as the control axes. If the angle is larger than $\frac{\pi}{4}$, the forward direction of the character, which is usually z-axis aligned to the orientation of the root, is projected onto the plane to form a parametric axis, and its orthogonal axis on the plane is selected as the other parametric axis.

Space segmentation: Given a point on the plane, our goal is to compute a weight combination with which the blended motion passes through the point. Since we have the weight combinations for many samples on the plane, we can calculate this at a point by interpolating those of the neighboring samples. When we have a small number of motion segments at a fat edge, RBF interpolation produces a nice mapping from a point on the parameter space to a weight combination. However, when the number of the motion segments at an edge becomes larger, it needs more computing time to compute the weight combination. Therefore, we triangulate the space taking the projected samples as the vertices of the triangles. Delaunay triangulation method is adopted to do this, by which, the given point can be represented with the barycentric coordinates uniquely in the triangle constructed with three samples.

Regular sampling: In the triangulated space, we need to frequently visit every triangle at run-time to find one that includes the given point. It is computationally quite expensive especially in interactive applications. To make this process more efficient at run-time, we sample the space regularly with a two-dimensional grid. For each grid point, we find the included triangle and compute its weight combination by linearly interpolating those at the triangle vertices. At run time, the weight combination for a given point can be efficiently computed by bilinear interpolation of those at the grid points nearby.

5. Run-time Motion Synthesis

At run-time, traversing the graph produces a sequence of animation, connecting the motion segments on the edges tra-

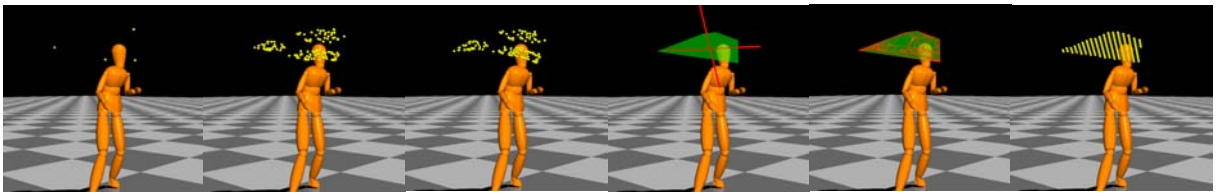


Figure 3: *Parameterization of kicking motion at impact: (Left to Right) the positions of the foot, randomly sampled points, their projections, parameterization axis, triangulation, and grid.*

versed. As the conventional graph-based motion synthesis methods, the motion segments are blended around the node to connect them seamlessly. Moreover, the motion segments in the fat edge being traversed are blended based on the user-specified weight combination. In this section, we discuss blending the motion segments in the fat edge with the given weights. We then exhibit our method that lets the user control the graph traversal and the weights.

5.1. Motion Composition

At a particular time instance, we interpolate three types of information separately: timing information, horizontal movement, and joint angles. Given the lengths of motion segments are different from one another in a fat edge. Moreover, the temporal correspondence between a pair of motion segments is not generally linear, so timing information need to be handled carefully to produce quality animation. Provided with a set of temporally matching poses, we blend the horizontal positions and orientations of character while ignoring the absolute positions and orientations in the original motion data. The angles of an individual joint are blended to be copied into the character pose at the animation frame.

Timing information: At each animation frame, we collect the corresponding poses from the source motion segments. Similarly to the previous approaches [RCB98, PSKS04, KG03], we first establish the correspondence between a reference time and the animation frames of the source motion segments. This can be done simply by taking the timing information of a motion segment (the longest segments in our implementation) as the reference time, and reusing its dense correspondence maps with the other segments, which we have already computed to compare the motion segments. Given a reference time instance T, the corresponding pose from $i-$th motion is $M_i(F_i(T))$, where $M_i(t)$ is the pose of $i-$th motion at t and $F_i(t)$ is the time correspondence map from the reference motion segment to $i-$th motion segment. The increment ΔT of the reference time for the next animation frame is computed by blending the changing ratio of T to the timing of the motion segments:

$$\Delta T = \sum_i w_i(T) \frac{dT}{dF_i} \Delta t, \qquad (2)$$

where $w_i(T)$'s are the current blending weights and Δt is the time increment for the desirable animation frame rate, which is usually $1/30$.

Position and Orientation: The position and orientation of the character are computed incrementally so that we can blend a motion segment with ones whose actions are similar but starting directions and locations are different. Instead of the absolute position and orientation of the root segment, we store its horizontal displacement and its rotation about y-axis from the previous animation frame. Computing the planar displacement is trivial. The orientation change about y-axis is computed as follows. Given the orientations of the root segment at two consecutive animation frame $\mathbf{q}(t)$ and $\mathbf{q}(t+\Delta t)$, we first rotate z-axis (or any axis that lays on the horizontal plane) with $\mathbf{q}(t)$ and $\mathbf{q}(t+\Delta t)$ to compute $\mathbf{v}(t)$ and $\mathbf{v}(t+\Delta t)$, respectively. Let θ be the signed angle between the normalized projections of $\mathbf{v}(t)$ and $\mathbf{v}(t+\Delta t)$ onto the horizontal plane. The rotation vector representing the rotation from $\mathbf{q}(t)$ to $\mathbf{q}(t+\Delta t)$ about y-axis is $\hat{y}\theta/2$, where $\hat{y}$ is y-axis. In motion synthesis step, the displacements and rotations represented with rotation vector are blended with the weights and cumulated into the previously synthesized animation.

Joint angles: In order to blend the angles of a particular joint, we adopt the approach proposed by Park et al. [PSS02] since it implicitly eliminates any possible problem due to the antipodal equivalence property of the unit quaternion space. This approach finds a unit quaternion from which the weighted squared sum of the cosine angles to the given joint angles is minimized. Using this method, the blended joint angles can be computed efficiently and applied atop the orientation and position obtained above.

5.2. Run-time Control

At run-time, a user directs the graph traversal by choosing actions at a time. Since there are a few fat edges starting from a hub-node, the user can select the action to follow starting from the pose at the node. A run-time application may adopt the devices that provide discrete choices, such as keyboards and buttons of joysticks. Each fat edge from a hub-node is assigned to a key or a button, and an action can be chosen by pressing the corresponding key or button.

Once a fat edge is selected, a weight combination of the

motion segments in the fat edge is specified to produce an output motion. Since we have already parameterized the motion segments in the fat edge with the two parametric axes, the user can specify parameters using input devices with two control axes, such as joysticks and mice. The first parametric axis, which corresponds to the projection of y-axis onto the parametric plane (or sagittal axis) would be map to y-axis of the control device. The other parametric axis is to mapped to x-axis of the device. Given the x and y value from the device, the surrounding four grid points are found and their weight combinations are blended by means of bilinear interpolation to produce the final motion on the fly.

6. Result

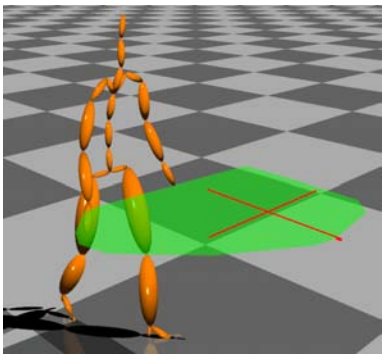
We tested our authoring system for four sets of input motion data: random walking, random sneaking, baseball batting, and kicking. All the input motion data was captured with a Vicon optical motion capture system at the rate of 120 frames per second and time-scaled to 30 frames per second for real-time animation.

Our system consists of two subsystems: graph authoring and parameterization system. Our graph authoring system reads the motion data, and finds the most frequent pose in it. The user can alter the threshold to determine similar poses. We often set the threshold as 1 in our experiment where the height of character is about 10. The user can also pick a base pose manually and the system automatically finds the match set with the given threshold. While a hub-node is being chosen, the system displays the fat edges by marking the motion segments in a fat edge with a color. The user can tweak the threshold that is used determine the similarity between motion segments. The value is highly dependent on motion repertories, but we found that with the character, 1 prevents variations of an action from being combined together, and 4 sometimes allows different actions to be grouped.

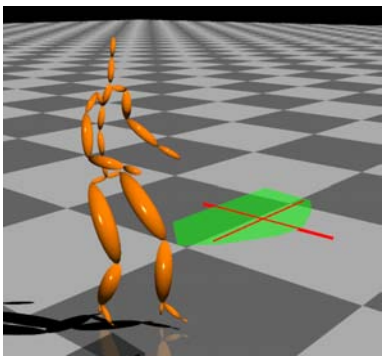
Once our parameterization system reads the graph, it allows the user to choose a fat edge to work on. After the user selects a time instance on the time-line and picks a joint of interest, our system visualizes the parameter space. The user can generate and review the blended motion either by specifying the weights or by picking a point in the parameter space.

The most time consuming part in the work flow is calculation of the distance between every pair of animation frames. Ignoring it, the graph authoring process generally takes few minutes including the time to examine the motion segments in edges. In our experiment, parameterizing an edge took less than a minute. Moreover, since the number of the fat edges is much smaller than that of the original edges, assigning the edges to a controller can be done much quickly compared to the method proposed by Gleicher et al. [GSKJ03].

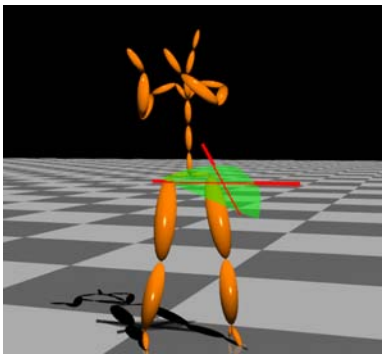
Random walking: For our random walking example, we



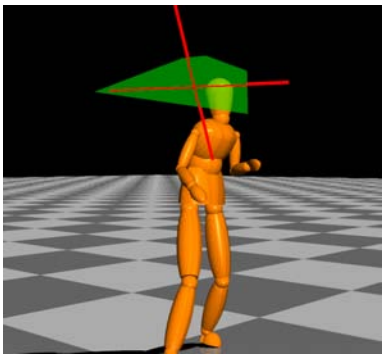
(a) Random walking



(b) Random sneaking



(c) Baseball batting



(d) Martial arts

Figure 4: *Parameter spaces*

captured the motion of an actor walking around. The motion capture data contain 22 complete walking cycles. Our authoring system found a pose as the base pose and one fat edge corresponding to walking cycles. We built a graph with one hub-node and one fat edge. The fat edge contained 21 motion segments. We parameterized the edge based on the root position at the last frame of the edge. The parameter space was built with 100 random samples which form a horizontal plane (See Figure 4(a).) The parameterization axes corresponded to forward/backward and left/right directions. We implemented a test application to evaluate run-time capability of our methodology. In the system, we used a mouse to control the step width and turning angles in the parameter space.

Random sneaking: In the random sneaking example, the input motion capture data contained 32 cycles of sneaking steps. Our authoring system provided a hub-node with 22 matching frames. A few steps were excluded since they corresponded to walk cycles while standing upright. The parameterization method same to the random walking example was used for this sneaking example (See Figure 4(b).) The same test application was implemented to verify the motion quality.

Baseball batting: We used a number of baseball batting motions for the third example. In the motion capture session, the actor was asked to swing a bat aiming at nine arbitrary points. For this example, we built one hub-node graph with three fat edges. The hub-node corresponded to the ready pose of batting and the edges contained waiting, another waiting, and swing motions. The swing motions were parameterized based on the right hand position at the impact point, since we did not have the corresponding bat motions. The constructed parameter space was similar to the strike zone (See Figure 4(c).) For run-time, we implemented a simple test system that played back one of waiting motion repeatedly, and started a swing motion when the user picked a hitting point.

Martial arts: In the last example, martial arts motion data were used to build a graph. Given the many kicking and hopping motions, we built a graph with two hub-nodes corresponding to the left and right ready stance poses. Since the variation of actions was quite high, eleven fat edges were built. Every edge contained a style of kick actions starting from left and right ready stance poses except two which corresponded to short hopping motions. We could reduce the number of fat edges to build fatter edges by increasing the threshold used in edge grouping process sacrificing the quality of blended motions. The parameter space of the fattest edge is shown in Figure 4(d). By selecting a point on the space, we could produce the kicking motion T the given point.

7. Discussions

In this paper, we provide a method to synthesize an animation of a character based on the existing animation. Analyzing motion data with little user intervention, a graph with a contrived structure is built and the similar motion segments on the edge are parameterized for continuous control. At run-time the animation is controlled interactively based on the user selection of action and the blending weights.

In the version of the interactive character system proposed by Gleicher et al. [GSKJ03], the motion segments are modified such that the motion can be connected without any processing at run-time. Contrary to this, the proposed method stitches the motion segments to connect them seamlessly on the fly. This reduces quality degradation due to the high connectivity at hub-node, which is often found in the original version. Although this approach clearly requires more computation at run-time causing overheads, it is still quite small compared to the overhead due to motion blending for parametric control. In fact, we found the proposed method is efficient enough to drive a few ten of characters simultaneously in real-time.

Our approach suffers from the limitations like the types of motion that can be built into a graph with the contrived structure more than the original version. Our approach is effective only when many actions start and end with similar poses so they are built into a graph with a small number of hub-nodes. Moreover, the given motion needs to have many similar actions to be parameterized. We believe, however, that these restrictions do not necessarily prevent the proposed method from being adopted in applications, such as video games where characters perform a limited repertory of motion and most of them start and end with very similar poses. For example, in a martial arts game, many actions start from the ready pose of the martial arts and stop at the same pose. Moreover, most of actions can be classified into a few sets with many variations. Such motion repertories fit well into the proposed method.

Another limitation of the proposed method is that the motion cannot be extrapolated. This is because we triangulate the parameter space and compute the weight combination at a point based on its barycentric coordinate. We believe that selecting a few motion segments for an edge and adopting RBF interpolation technique would be one of the most effective solutions.

Acknowledgement

This work was supported (in part) by the Ministry of Information & Communications, Korea, under the Information Technology Research Center(ITRC) Support Program.

References

[AF02] ARIKAN O., FORSYTH D. A.: Interactive motion generation from examples. *ACM Transactions on Graph-*

ics 21, 3 (2002), 483–490.

[AFO03] ARIKAN O., FORSYTH D. A., O'BRIEN J. F.: Motion synthesis from annotations. *ACM Transactions on Graphics 22*, 3 (2003), 402–408.

[CLS03] CHOI M. G., LEE J., SHIN S. Y.: Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Transactions on Graphics 22*, 2 (2003), 182–203.

[GSKJ03] GLEICHER M., SHIN H. J., KOVAR L., JEPSEN A.: Snap-together motion: assembling run-time animations. In *Proceedings of 2003 ACM Symposium on Interactive 3D Graphics* (Apr. 2003), pp. 181–188.

[KG03] KOVAR L., GLEICHER M.: Flexible automatic motion blending with registration curves. In *Proceedings of the 2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (Aug. 2003), pp. 214–224.

[KG04] KOVAR L., GLEICHER M.: Automated extraction and parameterization of motions in large data sets. *ACM Transactions on Graphics 23*, 3 (2004), 559–568.

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Transactions on Graphics 21*, 3 (2002), 473–482.

[KPS03] KIM T., PARK S. I., SHIN S. Y.: Rhythmic-motion synthesis based on motion-beat analysis. *ACM Transactions on Graphics 22*, 3 (2003), 392–401.

[LCR*02] LEE J., CHAI J., REITSMA P. S. A., HODGINS J. K., POLLARD N. S.: Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics 21*, 3 (July 2002), 491–500.

[LL04] LEE J., LEE K. H.: Precomputing avatar behavior from human motion data. In *Proceedings of the 2004 ACM SIGGRAPH / Eurographics symposium on Computer animation* (2004), pp. 79–87.

[MK05] MUKAI T., KURIYAMA S.: Geostatistical motion interpolation. *ACM Transactions on Graphics 24*, 3 (Aug. 2005), 1062–1070.

[PSKS04] PARK S. I., SHIN H. J., KIM T., SHIN S. Y.: On-line motion blending for real-time locomotion generation. *Computer Animation and Virtual Worlds 15*, 3 (2004), 125–138.

[PSS02] PARK S. I., SHIN H. J., SHIN S. Y.: On-line locomotion generation based on motion blending. In *Proceedings of ACM SIGGRAPH Symposium on Computer Animation* (July 2002), pp. 105–112.

[RCB98] ROSE C., COHEN M. F., BODENHEIMER B.: Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics & Applications 18*, 5 (September - October 1998), 32–40.

[WH97] WILEY D., HAHN J. K.: Interpolation synthesis of articulated figure motion. *IEEE Computer Graphics and Applications 17*, 6 (November 1997), 39 –45.