State-Annotated Motion Graphs

Bill Chiu, Victor Zordan, Chun-Chih Wu*
University of California, Riverside

Abstract

Motion graphs have gained popularity in recent years as a means for re-using motion capture data by connecting previously unrelated segments of a recorded library. Current techniques for controlling movement of a character via motion graphs have largely focused on path planning which is difficult due to the density of connections found on the graph. We introduce "state-annotated motion graphs," a novel technique which allows high-level control of character behavior by using a dual representation consisting of both a motion graph and a behavior state machine. This special motion graph is generated from labeled data and then bound to a finite state machine with similar labels. At run-time, character behavior is simply controlled by switching states. We show that it is possible to generate rich, controllable motion without the need for deep planning. We demonstrate that, when applied to an interactive fighting testbed, simple state-switching controllers may be coded intuitively to create various effects.

Keywords: motion capture, human animation, behavior control

1 Introduction

Motion graphs provide a means for re-using motion capture data by building transitions between motion segments and forming a tightly connected network of movements. This network, or graph, allows endless, seamless motion to be generated from a finite collection of data. However, the control methods by which the motion graph is traversed remain limited because the structure of graph is often too dense to perform deep planning. A large branching factor for a motion graph, which is desirable for building variety into the final motion, is at odds with the need to find paths quickly through the graph to accomplish a given goal. Previous techniques generally provide the means for control by performing local searches to discover longer paths through the graph. We take a different approach by re-framing the control problem as a series of high-level decisions, specifically focusing on situations where planning is not as important as intelligent, responsive behavior. This behavior is required of characters that are engaged in highly interactive activities, such as fighting.

We introduce the "state-annotated motion graph," a novel, dual representation where the motion graph is automatically embedded into a finite state machine (FSM) that encapsulates high-level behaviors. The usage of such a representation allows for action to be considered at two levels: as the individual node active in the motion graph; and as the behavior that exists within the state machine. Under this dual representation, we can give direction at the behavior level while the system generates smooth and varied motion at the level of the motion capture data. Unlike previous efforts that combine FSM and motion capture [Lau and Kuffner 2005], our technique has far fewer restrictions on the motion graph traversal. For example, our system may generate a motion capture path that connects several nodes to accomplish a single FSM transition. For control, we present a method for combining user-coded heuristics to yield various effects. Because no planning is necessary at run-time, our system is capable of interactive rates.

*e-mail: billlvbz|ccwu@cs.ucr.edu

Figure 1: Scene generated with state-annotated motion graph

We show our results using the testbed of martial arts fighting (as seen in Figure 1) which is valuable in settings such as electronic games and crowd generation for special effects. Our results include basic locomotion in the form of *shadowing*, where an opponent maintains a desired distance and facing direction relative to another fighter, as well as fighting actions for attacking and defending in a fighting match.

2 Background

Schoedl and colleagues introduced video textures as a mean of reusing prerecorded motion sequences by rearranging and connecting video clips based on visual similarity [Schoedl et al. 2000]. Shortly afterward, several research groups simultaneously proposed similar graph-based approaches aimed at motion capture reuse [Arikan and Forsyth 2002; Kovar et al. 2002a; Lee et al.] These techniques use a similarity metric to find transitions within a motion library based on visual quality and physically based constraints. Some semi-automatic approaches for building more structured motion graphs have since been proposed [Gleicher et al. 2003; Lau and Kuffner 2005] in which motions are grouped into behavior states, largely by hand. Arikan and colleagues also propose annotating a motion database and using labels to describe a desired action, which is then built using dynamic programming [Arikan et al. 2003].

Our work in state-annotated motion graphs is related to the above efforts, but differs in that we integrate annotation within the motion graph data structure. One contribution is thus in *providing context in the form of behavior labels for character motion so that it may be used to guide the construction and decrease the complexity of the motion graph* (e.g. by pruning unwanted transitions and unused nodes.) To accomplish this goal, our system uses control information provided by the user in the form of a finite state machine, and annotated state labels from the source motion to choose appropriate transitions for the motion graph. We contrast this with others' efforts in building a motion graph from unlabeled motions.

Searching a motion graph of any substantial depth is problematic

due to the high branch factor of the graph and the short duration of many nodes found in the graph. This problem has been addressed by previous efforts using adhoc routines to extend the planning horizon. For example, Schoedl and Essa [2002] precompute motion paths for video sprites using subsequence replacement. Arikan and Forsyth [2002] arrange motions into a hierarchy of clusters offline and perform planning over them at run time. Choi and colleagues [2003] use road map construction, a technique that stitches together shorter motion segments to fulfill the constraints for each leg of a path. Lau and Kuffner [2005] suggest limiting the branching with a highly structured motion graph built manually from hand-segmented clips. Kuffner's results for long-range path planning using A* or beam search are orders of magnitude faster than search compared with a general motion graph. In contrast, our approach uses an automatically generated motion graph and focuses on controlling high-level behavior transitions using precomputed paths between states. At run-time, *we perform no path planning* and instead control the character with shallow, depth-limited local search.

We demonstrate our results using interactive virtual fighters and fighting data. Close to this domain are two efforts focused specifically on fighting. In [Lee and Lee 2004], reinforcement learning is used to pre-compute motion paths for fighting behaviors. This work is complementary to our approach presented in this paper. Rather than looking for optimal paths for individual behaviors, we focus on high-level control for our character and aim to create a flexible control system (driven by a finite state machine) that may be modified at runtime. Graepel and colleagues [2005] propose a mechanism for learning a policy for fighting within a fixed video game setting where a character is pit against a heuristic-coded fighter. They focus on the problem of optimizing the transition selection from a small set of choices within the game's existing hand-crafted motion graph (aka *move tree*). Our work is similar to theirs, though we generalize control to a richer and more diverse set of motions, which are connected automatically in the state-annotated motion graph.

3 State-Annotated Motion Graphs

Behavior state machines, such as the example in Figure 2a, are a simple and flexible framework for character control and are the basis of many commercial animation systems. Most often these state machines are built by hand with the following layers: (1) the structure is chosen based on the goal of the application; (2) the behavior states are populated with appropriate animation segments; and (3) transitions are crafted to move between animation sequences. The power of the approach comes from the fact that the state description (Layer 1) is usually defined in a straightforward manner with semantic meaning such as "move forward" or "turn left" leading to intuitive state definitions. And Layer 2 can often be populated easily with animations taken from motion capture. But Layer 3 is more difficult to generate, often done in a naive manner with clean-up left to the discretion of a human animator. Even when high quality motion capture is employed for behaviors, transitions can reduce the overall quality if they include unwanted artifacts due to poor construction. Thus, the number of animations included in the state machine is often limited in order to keep the number of transitions required to a minimum. This can lead to repetitive, uninteresting motion. In contrast, motion graphs automate the process of *transitioning* from one motion clip to another and may combine large and rich databases of motion with seamless transitions. However, animation produced from a motion graph, say by a random walk, can be nonsensical because high-level context is missing. And as pointed out in the background section, general control and planning for motion graphs remains an open problem.

We combine these two techniques into a single representation, the state-annotated motion graph. It offers high-level task specification, and with it, the constraints needed to produce meaningful motions while simultaneously upholding visual continuity. This continuity is made possible by automatically generated motion graph-like transitions. In comparison, [Lau and Kuffner 2005] construct their

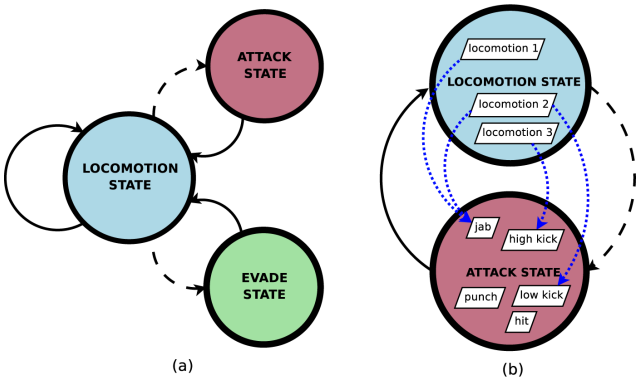

Figure 2: (a) Fighter state-machine. (b) State-transition with representative edges from motion graph that go from locomotion state to attack state shown (blue, dotted arrows)

well-structured motion graph by hand equivalent to building a state machine and choosing the behavior motions as well as the state transitions. An important distinction in their work is that *they treat the transition as a narrow gateway* from state to state. And even though they include multiple motion capture examples within a single behavior state, they still aim to have all of these clips start and end on similar frames. State-annotated motion graphs include no such restrictions on the transitions between states as long as the change in state labels matches the desired pattern. Thus, our transitions are much more broadly defined and can be traversed with any of a number of edges within a given graph (see Figure 2b). Also, unlike Lau and Kuffner's method, our approach allows motion-graph transitions freely within the context of a single behavior state as long as the state label does not change. The effect of these differences is the automatic construction of a controllable character with a large repetoire of motion capture-driven actions.

4 Graph Construction

State-annotated motion graphs contain dual information regarding a behavior state diagram and a standard motion graph. We describe one method for deriving this duality though this method is not unique. To begin, we must start with the FSM (as in Figure 2a) and an annotated motion database containing labeled behaviors that correspond to the desired states in the FSM. In our implementation we perform the labeling step simply by assigning a single label to complete files, such as "locomotion actions" for a long recording of an idling fighter. Note, no individual clip segmentation is necessary. Next, we follow an existing method [Kovar et al. 2002a] to construct a basic motion graph, saving the annotation label information during the process. However, we modify the existing algorithm to throw out all motion graph edges which are not allowed in the FSM (e.g. transitions from attack to evade in our fighting example.) As Kovar and his colleagues describe, we also compute the strongly connected components (SCCs) of the motion graph. In an SCC as defined by Tarjan [1972], any node can reach any other node within the same component. We take advantage of this property of the SCC in subsequent steps of our construction. Finally, the system chooses the largest SCC and discards any unused motion nodes.

To strengthen the utility and responsiveness of the state-annotated motion graph, we define two additional pre-processing steps. The first is to guarantee self transitions at the behavior (FSM) level. The goal of performing this step is to allow a character to remain within a single behavior state without the need for exiting the state. In FSMs, self-transitioning edges are useful for common behaviors such as the locomotion behavior in our fighting example. To compute a behavior-level, self transition from any given state, s, we find SCCs for all nodes with state label s. Note, each SCC will guarantee self transitioning by definition. We save the largest of these

SCCs and throw out the remaining nodes. To ensure consistency, we rerun the SCC subroutine on the overall motion graph to remove any dead ends potentially introduced during the production of the self transition for state *s*.

Next, we increase the responsiveness of the character by performing some offline search. To ensure timely access of certain behaviors, we introduce the notion of *fully connected* transitions which guarantee paths from every source node to every target node. In the construction of our fighting characters, we found such fully connected nodes to be useful for increasing availability of "attack" and "evade" behaviors. In our state machine in Figure 2a, fully connected transition edges are denoted by dashed lines; standard transition edges are solid. Pre-processing for fully connected transitions requires the search for paths from each source node to all target nodes. SCC again provides a guarantee that some path exists, but given that our motion graph likely includes multiple paths, we search for the temporally shortest path between the nodes. To accomplish this goal, we employ dynamic-programming (though any method would be sufficient) and store the found paths with the motion graph node. The result of this preprocessing step is that, *at runtime, the quickest path to any node in the desired state is immediately available, without search.*

5 Control

Once the state-annotated motion graph is constructed, the user can immediately control the character's behavior manually by selecting the desired state behavior. This will result in a character continuously remaining in that state if there is a self-transitioning edge or transitioning to that state until the desired state is switched to a new behavior.

To automate control, say for a non-player character, we propose a straightforward hierarchical control routine which is composed of a set of *activity* controllers, and a "supervisor" which prioritizes the activities to accomplish a high-level goal. In this simple scheme, the supervisor polls the individual controllers which determine the best path in the motion graph that will satisfy their unique subtask. They report to the supervisor an assessment of their ability to achieve their subtask and the supervisor selects the activity based on a priority scheme and the system adds the motion corresponding to the selected path to the character's animation queue.

The activity controllers correspond to the behaviors of the FSM and can be designed from simple rules. For example, to walk forward, the controller nearly be given a desired speed and the activity controller would compare the desired speed with the possible speeds for the nodes and choose the best node. A slightly more sophisticated controller that manages both speed and direction would need to reconcile between the two subgoals but the process of building the controller is as straightforward as the first step in the construction of any FSM (as outlined in Section 3.) In our experiments, the activity controllers are built from such basic rules. For example in fighting, for locomotion, we develop a behavior to *shadow* an opponent. Shadowing is a common fighting activity where the goals are to keep the opponent in front of the fighter and at a desired distance. Our rules for shadowing match these goals. During the shadowing activity, the best step (node) to take is the greedy one which minimizes the errors associated with the distance and facing direction. To leverage the two subgoals, a weighted sum of the errors is computed. If we employ this simple controller alone, we generate a pair of characters which shadow and circle each other, as if anticipating a fight. To create animation of fighting, we developed only three activity controllers, one each for shadowing, attacking, and evading. More details about the specific controllers appear in the next section.

Activity controllers are combined by a supervisory control system. We found that a simple supervisor was sufficient for fighting - we merely prioritize the activities based on the following ordering: evade; attack; then shadow. The supervisor thus polls each activity controller in turn and selects the first activity that is reasonable based on the given ordering, where the default behavior appears last. The assessment for what is reasonable is application-specific, e.g. the evade activity is only employed if the character is too close to the opponent. *The power of the state-annotated motion graph is that the supervisor controller can be thought of (and coded) as if it is executing solely on the FSM's state diagram, even though it is traversing precomputed paths at the motion-graph level.* The individual activity controllers work between the two representations but even these controllers do not perform any deep search. This is because clearly defined rules help to find the optimal path, which gives the best action for a given set of conditions.

	Attack	Evade	Locom.
Number of Nodes	76	54	702
Ave. Length (sec)	2.57	1.30	0.19
Ave. Time from Locom. (sec)	0.04	0.05	–

Table 1: Fighting state-annotated motion graph statistics: Total number of nodes in each state, average length of nodes, and average time from any locomotion node to each attack or evade node.

6 Implementation and Results for Fighting

To show our results, we implemented fighting controllers that use state-annotated motion graphs populated with several minutes of (solo) martial arts movement data. The high-level state diagram for fighting, shown in Figure 2a, specifies character behavior. Our implementation for fighting includes three primary sections: (1) the state-annotated motion graph, (2) the activity controllers, and (3) the run-time system. Constructing the graph is expected to be done once off-line, and takes several hours of computation on our database. Development of control requires some design and parameter-tuning (e.g. weighting competing subgoals properly) which was the most labor intensive stage of the implementation. The on-line system runs at 15 fps for two characters using an AMD Athlon64 CPU with 2 Gigabytes of memory.

For fighting, there are many possible factors for successful attack and evasion, but speed plays a uniformly important role and we used this as a unifying principle in the design of our activity controllers. As Figure 2a shows, the locomotion state, which is used for the shadowing activity, has a self-transitioning edge and two fully-connected edges to attack and evade states. Thus, we compute a local SCC for the locomotion state and appropriate (shortest) paths from all locomotion nodes to all attack and evade nodes to uphold our definition of "fully connected". Our computation of the shortest path is defined by the cumulative delay of all nodes in the path and is intended to give the controller the most responsive control space. Some statistics in regards to responsiveness appear in Table 1.

To assess the value of selecting a specific path for any activity, we used the following metrics employed at the stated future predicted time:

- **Attack.** Advance time to the first contact of the attack and measure the distance between the attack body (fist or foot) and designated targets (head, chest, or abdomen.)
- **Evade.** Advance to the end of the evade action and compute the distance between the fighter and the opponent.
- **Shadow (default activity).** Advance to the end of the locomotion node and assess quality based on the accuracy of two sub-goals: distance and relative facing direction computed at the end time of the nodes being tested.

In each of these metrics, a lookahead is performed. To make this prediction, we advance both characters on their current path. Although this does not guarantee a perfect prediction since the other character's controller might change its path before that future time, we found this as a reasonable predictor of future state.

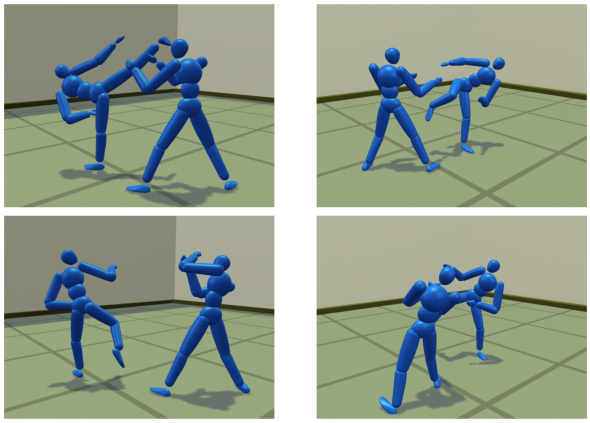

Figure 3: Left and right columns: Two fighting interactions generated using our technique

The priority between activities is a single, easily-tunable parameter which may be modulated to create unique effects. For example, when the character is assigned with a heavy weighting for the evade activity, the result is a fighter that acts more defensively. Conversely, when attack is given priority over evade, the result is a more aggressive fighter. We show a result pitting both fighting styles against one another in the accompanying animations. In addition, we show an animation where the attack behavior label is (trivially) split into two labels, one for punching and one for kicking attacks. In this case, we vary the priority of the two attack activities to both extremes. The result, as one might expect, is a fighter which prefers to punch versus one which only kicks. The granularity of the state labels is up to the discretion of the animator. That is, if warranted by the application, single attack motions (e.g. left hooks) could be labeled as unique behavior states. More details and results can be found in the primary author's thesis along with preliminary results on using reinforcement learning to find controllers automatically [Chiu. 2007].

Interactive fighting is difficult because several factors are involved simultaneously. We focus on behavior selection using a motion graph, but other factors - such as precise collisions or modifying the behavior motion on the fly - are somewhat orthogonal to the emphasis of our work. To produce the animations that are included with the paper (as shown in Figures 1 and 3), we perform post-processing in the form of blending and foot-skate clean-up to improve the quality of the final motion. Foot skate is introduced during motion graph transitions and we remove some foot skate using a naive algorithm. We also add dynamic response, as described by [Zordan et al. 2005], to select impacts to increase the perceived interaction of the fighters. In addition, more refined post processing would certainly improve the motion further: e.g. for better foot-skate clean up [Kovar et al. 2002b; Ikemoto et al. 2005], and for more precise motion control [Lee and Lee 2004]. But these modifications do not affect the behavior control of the fighters, and remain outside the scope of this paper.

7 Discussion and Conclusion

As we perform experimentation for fighter character motion, our approach will be powerful in a number of other domains, especially in domains where large numbers of interacting characters are to be controlled. The construction of state-annotated motion graphs is only limited by the need to assign state labels and to define a coherent FSM from those labels. Further assumptions are that ample data is available for the behaviors and that a densely connected motion graph may be generated from the motion data. Given these factors, character motions where interaction is prevalent should be readily controlled with our approach.

We present a novel technique for controlling interactive fighting characters using state-annotate motion graphs. The benefit of our approach stems from the ability to generate controllable, interactive characters that move based on motion capture data. The novelty of our approach lies in the dual nature of our representation because it affords high-level control via a FSM and motion-capture animation due to the motion graph. We introduce a simple hierarchy of heuristics to control for fighting. This method of control is particularly applicable to settings such as fighting where deep planning is replaced by responsiveness.

Acknowledgments: We acknowledge the help of Nick Novak for his assistance with video editing and Karen Liu for her help in capturing motion capture data. The primary author was partially supported by an NSF GAAN grant during a portion of this research.

References

ARIKAN, O., AND FORSYTH, D. A. 2002. Interactive motion generation from examples. In *ACM Transactions on Graphics*, ACM Press vol. 21, 483–490.

ARIKAN, O., FORSYTH, D., AND O'BRIEN, J. 2003. Motion synthesis from annotations. In *ACM Transactions on Graphics*, ACM Press, vol. 22, 402–408.

CHIU, B. 2007. State-annotated motion graphs. *Dissertation, University of California, Riverside*.

CHOI, M. G., LEE, J., AND SHIN, S. Y. 2003. Planning biped locomotion using motion capture data and probabilistic roadmaps. In *Proceedings of ACM SIGGRAPH '03*, ACM Press, vol. 22(2), 182–203.

GLEICHER, M., SHIN, H. J., KOVAR, L., AND JEPSEN, A. 2003. Snap-together motion: assembling run-time animations. *ACM Trans. Graph. 22*, 3, 702–702.

GRAEPEL, T., HERBRICH, R., AND GOLD, J. 2005. Learning to fight. In *Proceedings of the International Conference on Computer Games*, Artificial Intelligence, Design and Education.

IKEMOTO, L., ARIKAN, O., AND FORSYTH, D. A. 2005. Knowing when to put your foot down. In *Symposium on Interactive 3D Graphics and Games (I3D)*.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. In *ACM Transactions on Graphics*, vol. 21, 473–482.

KOVAR, L., SCHREINER, J., AND GLEICHER, M. 2002. Footskate cleanup for motion capture editing. In *ACM SIGGRAPH Symposium on Computer Animation*, 97–104.

LAU, M., AND KUFFNER, J. J. 2005. Behavior planning for character animation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

LEE, J., AND LEE, K. H. 2004. Precomputing avatar behavior from human motion data. *Eurographics/ACM SIGGRAPH Symposium on Computer Animation*.

LEE, J., CHAI, J., REITSMA, P., HODGINS, J., AND POLLARD, N. Interactive control of avatars animated with human motion data. In *Proceedings of ACM SIGGRAPH '02*, ACM Press, vol. 21(3) of *ACM Transactions on Graphics*, 491–500.

SCHOEDL, A., AND ESSA, I. A. 2002. Controlled animation of video sprites. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*.

SCHOEDL, A., SZELISKI, R., SALESIN, D. H., AND ESSA, I. A. 2000. Video textures. *SIGGRAPH 2000, Proceedings*, 489–498.

TARJAN, R. 1972. Depth first search and linear graph algorithms. In *SIAM Journal of Computing*, vol. 1, 146–160.

ZORDAN, V. B., MAJKOWSKA, A., CHIU, B., AND FAST, M. 2005. Dynamic response for motion capture animation. In *ACM Transactions on Graphics*, vol. 24, 697–701.