Eurographics/ACM SIGGRAPH Symposium on Computer Animation (2004)
R. Boulic, D. K. Pai (Editors)

Precomputing Avatar Behavior From Human Motion Data

Jehee Lee and Kang Hoon Lee

Seoul National University †

Abstract

Creating controllable, responsive avatars is an important problem in computer games and virtual environments. Recently, large collections of motion capture data have been exploited for increased realism in avatar animation and control. Large motion sets have the advantage of accommodating a broad variety of natural human motion. However, when a motion set is large, the time required to identify an appropriate sequence of motions is the bottleneck for achieving interactive avatar control. In this paper, we present a novel method of precomputing avatar behavior from unlabelled motion data in order to animate and control avatars at minimal runtime cost. Based on dynamic programming, our method finds a control policy that indicates how the avatar should act in any given situation. We demonstrate the effectiveness of our approach through examples that include avatars interacting with each other and with the user.

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

1. Introduction

Realtime animation and control of three-dimensional avatars is an important problem in the context of computer games and virtual environments. Recently, large sets of motion capture data have been exploited to provide increased realism in interactive applications as well as off-line animation production. Creating lifelike motion of avatars requires many components: acquiring a rich set of avatar behaviors that appear natural, representing the behaviors in a connected way for seamless animation, and giving the user control over those behaviors. We are interested in the last of these: selecting appropriate behaviors so that the avatar responds interactively to user commands.

A set of avatar motions has often been represented as a directed graph that encodes connectivity between behaviors (e.g., squatting can be followed by jumping). Combined with statistical models, such as Markov processes and hidden Markov models, this graph-based representation combines flexibility in an avatar's behavior with the ability to control the avatar's actions. However, applying this approach to interactive avatar control is challenging because the graph must be very large in order to accommodate a rich variety of natural human motions, and this large graph must be searched at runtime to select appropriate motions interactively.

In this paper, we present a pre-computation method that allows avatars to be animated and controlled interactively from a large collection of human motion data at minimal runtime cost. Our method tabulates the utility of taking different actions in any given state so that an appropriate sequence of actions can be found efficiently using table lookup. Our approach is based on dynamic programming which allows us to produce a control policy (or behavior) for a given state-action model. We compute a small collection of control policies that are used to control the motion of our avatars.

There exist a number of path planning and state-space search algorithms that find a path when start and goal states are given. Our approach is significantly different from those algorithms. Instead of finding a point-to-point path, we find a control policy that indicates how the avatar should act in any given situation. Once the policy is computed at a preprocessing phase, our avatar can find a sequence of actions to the goal very efficiently at runtime. Our approach is resolution-complete in the sense that the optimal (with an infinite-horizon delayed reward cost function) control pol-

† email:{jehee,zoi}@mrl.snu.ac.kr

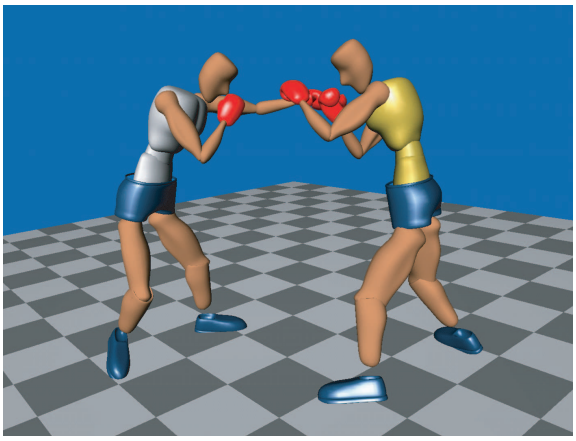

Figure 1: *The realtime match simulation is created using motion data captured from a single performer who shadowboxed. Shadowboxing is sparring with an imaginary opponent as a form of training. From motion data, our controllable animated boxers learned several behaviors to interact with each other and interact with the user.*

icy can be found at a given resolution and range of the state space.

We demonstrate the realtime capability of our approach through examples that include animated avatars interacting with each other and with the user in a dynamic environment. The user can have direct and immediate control over the motion of the avatar. Our approach is relatively easy to implement and can be applied to a wide variety of human activities. For our experiments, we selected a domain of human motion which involves significant physical interactions (see Figure 1).

2. Background

Animation and control of three-dimensional synthetic characters has been an active research topic in computer graphics for decades [BG95, Blu98, BC89, NZB00, PG96]. More recently, attention has been given to data-driven approaches using motion capture data. The use of a motion capture system provides an easy solution to interactive avatar control simply by transferring the movements of a performer to an animated avatar in realtime. A number of researchers explored puppetry techniques that map a motion from the performer to the avatar, which is usually of a different size and proportion than the performer [BHG93, DYP03, MBT96, SHS98, SLSG01]. This paper focuses on controlling avatars without any special equipment for user interface.

Statistical models have been frequently used to capture the underlying structure of a large collection of motion data. Several researchers have explored methods for introducing statistical variations into motion [BS97, PB00, SBS02]. A more popular approach is to exploit PCA dimensionality reduction for simplifying the data, clustering for grouping similar motions, and a Markov process model for allowing transitions between clusters. The transitions between clusters yield a directed graph that is often represented as a transition table. Brand and Hertzmann [BH00] use a variation of hidden Markov models to generalize motion data and introduce stylistic variations into motion. Galata and her colleagues [GJH01] use variable length hidden Markov models to allow the length of temporal dependencies to vary. Bowden [Bow00] uses piecewise non-linear principle component analysis with a Markov chain to compactly represent motion data. Molina-Tanco and Hilton [MH00] developed a PCA- and clustering-based system that identifies a sequence of motion segments interpolating user-specified keyframes. Li et al. [LWS02] adopted linear dynamic systems to provide better approximation to motions within clusters and introduce small variations into motion dynamics by perturbing the parameters of linear dynamic systems. The statistical models of Kim et al. [KPS03] also make use of k-mean clustering and a transition graph to synchronize dance motions with music.

Several research groups have explored techniques for synthesizing new motions by cutting pieces from existing motion data and reassembling them in a novel order. This process is facilitated by a graph representation that allows transitions between individual motion frames rather than clusters of motions. Frame-level transition methods avoid the risk of smoothing out fine details of motion during PCA dimensionality reduction and clustering, but create a bigger transition graph that makes avatar control even more challenging. Pullen and Bregler [PB02] segmented motion data into small pieces and rearranged them to meet user-specified keyframes. Kovar et al. [KGP02] generated a graph structure from motion data and employed the branch and bound algorithm to control an avatar to follow a sketched path. Arikan and Forsyth [AF02] created a hierarchy of graphs and employed a randomized search algorithm for synthesizing a new motion subject to temporal and position constraints. Both algorithms are intended to find a global solution and may not be suited for interactive control. Lee et al. [LCR*02] use a local on-line search algorithm which is demonstrated to generate 5 to 8 frames of avatar motion per second when the length of time the avatar looks ahead is moderately limited. However, this performance is still far from realtime in an environment with multiple characters. They also noted that graph search can be quite efficient if the graph is embedded into a specific environment and thus the flexibility of the avatar's motion is limited. Choi et al. [CLS03] generated a graph representation of motion embedded into a cluttered environment from motion data that is captured in a different (usually empty) environment. This environment-specific graph facilitates creating biped locomotion through a complex environment with obstacles. Arikan et al. [AFO03]

showed that motion synthesis from a graph structure can be cast as dynamic programming. Our work is built upon this previous work and aims to pre-compute the graph search in every possible situation and tabulate the results to find an appropriate sequence of motions at minimal runtime cost. Our system is several orders of magnitude faster at runtime than the aforementioned graph search and dynamic programming algorithms.

Machine learning techniques have been extensively exercised in computer animation. Ngo and Marks [NM93] create dynamic controllers of simple creatures using genetic programming. Sims [Sim94] also exploits genetic programming for simulating a virtual creature that can evolve its morphology. Grzeszczuk et al. [GT95, GTH98] explored automatic learning techniques for animating dynamic models such as fish and lunar landers. These techniques were based on control abstraction and neural networks. Faloutsos et al. [FvT01] employ Support Vector Machine (SVM) classifiers for identifying the "pre-conditions" under which dynamic controllers are expected to function properly. Arikan et al. [AFO03] use SVM classifiers to annotate motion databases with intuitive keywords and allow users to describe a desired scenario with those annotations.

Reinforcement learning refers to a set of problems that involve agents having to learn how to act in any given situation through trial-and-error interactions with a dynamic environment. Excellent surveys can be found in [KLM96, SB98]. A number of variants of reinforcement learning have been used in robotic control and path planning. Atkeson et al. [AMS97] constructed a two-armed robot that learns to juggle a tapered stick by hitting it alternately with each of two hand sticks. Mataric [Mat94] presented a reinforcement learning method to train a group of robots with a steerable wheeled base and a gripper to travel, collect small disks, and transport them to a destination region. Reinforcement learning is used relatively little in computer graphics. Blumberg et al. [BDI*02] created an autonomous animated dog that is trained to recognize acoustic patterns as cues for actions. They focused on training proximate causality because that is suitable for animal training from an ecological point of view.

Schödl and his colleagues investigated a variety of search algorithms for creating video textures [SSSE00] and animating video sprites [SE01, SE02]. For scripted animations, they suggested beam search and repeated subsequence replacement that search a path through state spaces given start and goal configurations. They suggested a reinforcement learning technique known as Q-learning for interactively controlling video sprites to animate following a specified path. Q-learning is the most popular reinforcement learning method, but often suffering from excessive memory usage. We are interested in creating more complicated human-like characters that hit and react, and employ a simpler dynamic programming method that requires less memory.

Our work is related to data-driven pre-computation methods in global illumination and dynamic deformation. A number of researchers have explored techniques to pre-compute global radiance transfer effects over surfaces in a scene and tabulate them to render the scene quickly at any viewpoint in a variety of lighting conditions [NRH03, SHHS03, SLSS03]. James and Fatahalian [JF03] pre-computed the dynamics of deformable models to facilitate realtime simulation. They allowed a limited form of user interaction in order to simplify the dimensionality of the state-action space.

3. State-Action Model

Our state-action model assumes an avatar that has a discrete set of states S and actions A, and a target that also has a discrete set of states E. The entire system is modelled as a state-action pair $\{(S,E),A\}$. At each step of a simulation, an avatar in the current state chooses an action. The action changes the states of the avatar and the target. This process is *Markovian* because the avatar's decision as to how to act depends only on the current state. To learn behaviors effectively, it is very important to have a low-dimensional state-action space with good discretization. Both human motion and target states are high-dimensional and continuous in general. A simplified model of the system must be provided to make precomputation practical. In this section, we describe how to construct a discrete state-action model in a preprocessing phase.

Our representation of human motion largely follows that of Lee et al. [LCR*02], which forms a directed graph with its nodes corresponding to motion frames and its edges corresponding to connecting transitions between frames. This representation can be automatically constructed from extended, unlabelled sequences of motion data by identifying similar frames and creating transitions between them. The construction method of Lee et al. creates a relatively small number of frames with multiple out-going transitions and leaves many frames with a single out-going transition, because additional transitions are allowed only at discrete events (e.g., the foot is about to touch or leave the ground). The set of avatar states S includes all frames that have multiple out-going transitions. Consecutive frames with a single out-going transition are collapsed into an action in A. With this definition, a state in S represents a static pose of the avatar and taking an action results in a transition from one state to another (see Figure 2(c)).

We are particularly interested in controlling avatars in a dynamic environment with moving tagets such as a moving target to punch and a ball to follow and hit. A discrete set of targets E forms a grid of locations. It is convenient to represent the grid with respect to a polar coordinate system that is centered at the location of the avatar. The dimension and range of the grid depends not only on the geometry of the environment but also on the type of behavior to be learned. In our boxing example, we consider two behaviors: "approach-

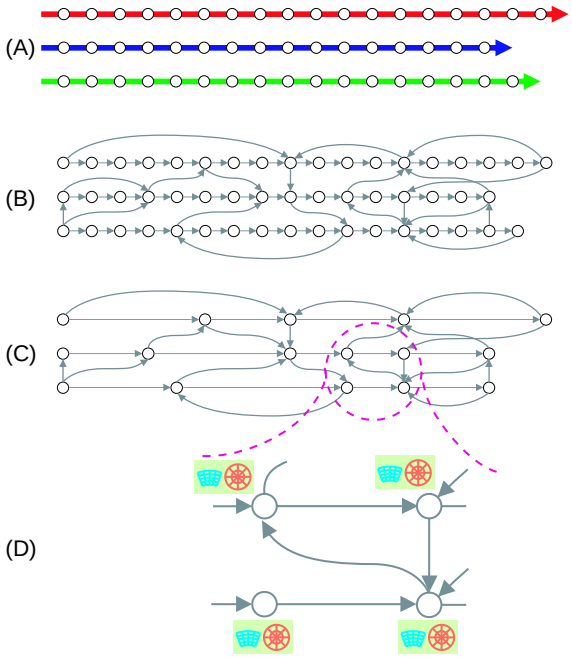

Figure 2: *The state-action model of avatar motion. (A) The motion data initially consists of a number of motion clips containing many frames. Each frame represents a pose of the avatar. (B) Many frame-to-frame transitions are created to form a directed graph. (C) Frames with a single out-going transition are collapsed. The simplified graph represents a discrete state-action model (S,A) of avatar motion, where the nodes of the graph correspond to states S and the edges correspond to actions A. (D) Target states are discretized into grids and associated with nodes.*

the-target" and "throw-punches-at-the-target" (see Figure 3). Computing the latter requires consideration of a relatively small region of the entire state space within arm's length, and the region must be three-dimensional to accommodate the x, y, and z-coordinates of the target. In contrast, computing the approach behavior requires a two-dimensional state space because the height of the target is not necessary, and the region of interest is larger to allow tracking of a target at an arbitrary location. In theory, the entire (perhaps infinitely large) state space should be considered to find the optimal path if the target can be located arbitrarily far away. In practice, we do not need to search the entire space exhaustively because search in a partial region around the avatar gives a good approximate path. The resolution of the grid depends on the expected precision of control. For example, the punch behavior should be computed with a fine grid to control the direction of the punch precisely, while the approach behavior needs a relatively coarse grid to provide a rough direction toward the target.

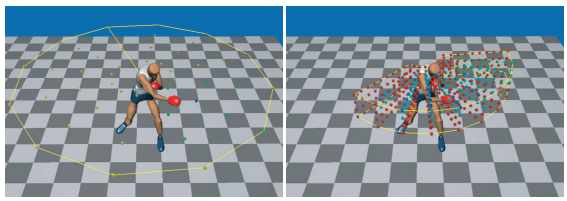

Figure 3: *The state space of a moving target is discretized differently for two behaviors: (Left) A grid for "approach-the-target" is two-dimensional, omni-directional, and relatively coarse. (Right) A grid for "throw-punches-at-the-target" is three-dimensional and dense. It covers a relatively narrow and small region around the avatar where the avatar's punches reach.*

4. Precomputing Control Policies

The graph representation of motion data provides the avatar with a broad variety of action choices, and the avatar must select one action $a \in A$ at each step of the simulation based on the current state $(s,e) \in S \times E$. Finding an optimal action involves a search through the exponentially-growing tree of actions that can move the avatar through the state-action space. This search is the primary bottleneck limiting interactive avatar control. Our goal is to pre-compute which action to take at any given situation in order to find an appropriate action very efficiently at runtime. A naive approach is to expand the search tree for every state-action pair $\{(s,e),a\}$ to evaluate the utility of taking action a at state (s,e). This approach can be prohibitively slow (even as a preprocessing step) for any environment and motion data of practical use. The major reason for this sluggish performance is that large portions of the state-action space are redundantly traversed by a number of search trees rooted at different states. Dynamic programming provides an efficient solution for this problem.

4.1. Formulation

Our approach was inspired by reinforcement learning that is designed to allow autonomous agents to learn a particular behavior through trial-and-error experiments. On each trial, the learner may receive a reward for taking an action. Through repeated trials, the learner must discover which actions tend to increase the long-run sum of rewards in future trials. In our boxing example, given target e, the boxer at its state s has to choose an action from a set of actions immediately available to the boxer at that state. Taking one of those available actions may not result in any immediate reward, but rewards may be received later when the target is hit by the punch. Taking those *delayed rewards* into account, the boxer learns to choose a sequence of actions that maximizes

the sum of future rewards

$$G = \sum_{t=0}^{\infty} \gamma^t r_t, \tag{1}$$

where discount factor $0 < \gamma < 1$ gives a penalty for long-delayed rewards in order to model the uncertainty of a dynamic environment and to make the infinite sum converge.

The basic idea of reinforcement learning is to create a lookup table that indicates which action to take given a target. Each entry of the table, indexed by (s,e), represents the expected discounted sum of reward that the avatar will gain if it starts in that state and executes the optimal policy. It has been shown that the optimal values for the entries can be learned by randomly sampling states and applying a local update rule to each state. This rule is intended to reflect an immediate reward for taking an action from that state and propagate rewards gradually to preceding states.

We describe here how to define reward functions for training our avatars. Reward $R(s,e,a)$ is a scalar-valued function of state (s,e) and action a. The reward function is specific to a behavior to be learned. Typically, an impulsive reward signal is received at a specific time instance if some conditions are satisfied. The reward function takes the maximum of discounted signals over the duration of action a.

$$R(s,e,a) = \max_t \left(\gamma^t I(t) w(t) \right), \tag{2}$$

where $I(t) = 1$ if a desired condition is satisfied at time t (e.g., the punch lies on the target). Otherwise, $I(t)$ is zero. $w(t)$ is a weight term. Some behaviors can be modelled to provide a continuous form of reward signals, which leads to

$$R(s,e,a) = \max_t \left(\gamma^t w(t) \exp(-\frac{\|e(t) - e_d\|}{\sigma}) \right), \tag{3}$$

where e_d is the destination state that receives the maximal reward. The learning process can converge more rapidly with this form of the reward function.

4.2. Dynamic Programming

Computing a control policy is a simple iterative process of sampling states and applying a local update rule to incrementally refine values in the table. On each iteration, we randomly choose a pose s of the avatar and a target e among grid points. The avatar needs to decide which action to take among a set of actions immediately available to the avatar at state s. A greedy policy is to select the one that gains the highest reward in one step. Taking action a brings the avatar to state s' and the target to a new location e' (since the location is represented with respect to a local moving coordinate system). According to the greedy policy, the value at (s,e) should be updated to reflect the immediate reward and the value of the next state (s',e'), using the best available action. This process is called *value iteration* in reinforcement learning community and can be shown to converge to the optimal values. We repeat the process until all states have been visited dozens of times.

Our updating rule is

$$V(s,e) := \max_a \left(R(s,e,a) + \gamma^t V(s',e') \right), \tag{4}$$

which asserts that taking action a at state (s,e) results in the transition to state (s',e') and produces immediate reward $R(s,e,a)$. Reward $R(s,e,a)$ is added to the discounted value of the next state and the result is used to update $V(s,e)$. t is the duration of action a. Since e' may not coincide with a grid point, we approximate $V(s',e')$ by linearly interpolating values at adjacent grid points.

Our implementation of table $V(s,e)$ produces a number of small tables. Each of the small tables $V_s(e)$ is associated with avatar state s and indexed only by target e (see Figure 2(d)).

5. Runtime Synthesis

Once the table has been filled with appropriate values, finding a sequence of actions at runtime is straightforward. Whenever the avatar is provided with more than one available action, the avatar selects the one that makes transition to the state with the highest value. This simple greedy strategy causes the avatar to act optimally at a given resolution and range of the state space. If multiple behaviors are active, the avatar must reconcile behaviors that produce different reward values. We simply select an action that maximizes the weighted sum of values from multiple behaviors.

To create lively avatars, randomness in choosing actions is as important as optimality with respect to getting a reward. In fact, real people often do not act optimally. An athlete may deliberately choose different actions in similar situations, for example, in order to trick his opponents. To incorporate randomness into our system, we identify a small number of preferable actions and then select one randomly instead of choosing the best action.

6. Experiments

All of the motion data used in our experiments was captured from a Vicon optical system at the rate of 120 frames/second and then down-sampled to 15 frames/second for realtime display. Motion capture data contains trajectories for the position and orientation of the root node (pelvis) as well as relative joint angles for each body part. Timing data was measured on an Intel Pentium IV 2.4GHz computer with 1Gbyte main memory. Our controllable animated boxers are created through following steps.

Data Acquisition. We recorded motions of about 8 minutes duration from a professional boxer. In the recorded data, our subject shadowboxed alone in an empty environment. Shadowboxing is a popular self-training method of sparring with

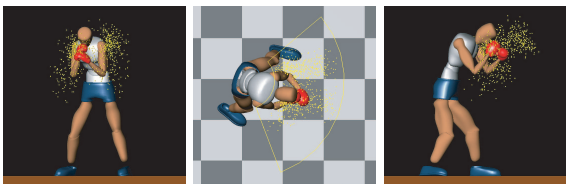

Figure 4: *The distribution of effective hitting points. Front, top, and side views are depicted.*

an imaginary opponent to practice various combinations of punches and footwork. Our subject performed about 20 different combinations of punches that included a left jab followed by a right straight punch, a right uppercut followed by a left hook, and so on. Most of combinations consisted of two to four punches. He also performed assorted defensive actions such as ducking, dodging, and blocking punches. To avoid the introduction of unnatural transitions, we did not provide our subject with any scenario or specific sequence of punches, except to constrain the motion within the capture region (5 by 5 meters). We recorded in long clips (about 90 seconds each) to allow our subject to perform natural transitions between actions.

Data Annotation. Identifying contact with the environment is important for generating good transitions between motion segments and computing reward functions. Motion capture data does not explicitly indicate when the feet and the ground are in contact and when punches are effectively hitting an imaginary target. Our system automatically annotates motion data with this information. As in Lee et al. [LCR*02], feet are considered to be on the ground if one of their adjacent joints (either the ankle or the toe) is sufficiently close to the ground and its velocity is below some threshold. Punches can deliver power effectively when the direction of motion of the fist and the forearm axis are parallel at the moment of hitting. In our system, punches are considered to be effective if the magnitude of the velocity vector of the fist projected onto the forearm axis is above some threshold and the directions of these vectors are not opposite. The tip of the fist at the moment of hitting is called an *effective hitting point*. We found 788 effective hitting points in our data set (see Figure 4).

Graph Construction. We construct a directed graph from motion data by adding connecting transitions between frames. A transition from frame i to frame j is added if poses at frame i and $j-1$ are similar and the left foot is about to leave the ground in both frames. The similarity between poses is measured by considering differences in joint angles and velocities [LCR*02]. To avoid dead ends in the graph, we find a strongly connected component in which every frame can be reached from every other frame through transitions [KGP02, LCR*02]. The strongly connected component contains 6453 frames, which corresponds to about 7 minutes and 17 seconds of motion. Collapsing frames with a single out-going transition leaves 437 frames and 27072 transitions. These frames and transitions form the discrete set of states and actions of our animated boxers.

Reward Functions. Our boxers learned two behaviors: "approach-the-target" and "throw-punches-at-the-target". The reward function for the former behavior is

$$r = \max_t \left(\gamma^t \exp(-\frac{\|p(t) - p_d\|}{10}) \right), \tag{5}$$

where p_d is the 2-dimensional location of the target. Because the character is approaching the target and not yet striking it, $p(t)$ is the 2-dimensional trajectory of the centroid of effective hitting points. The discount factor γ is 0.97. All coordinates are represented with respect to a (body local) polar coordinate system. The circular region around the avatar of 2 meter radius is discretized into a grid of size 5 (distance) by 13 (angle) (see Figure 3 (left)).

The reward function for the latter behavior is

$$r = \max_t \left(\gamma^t I(t) w(t) \right), \tag{6}$$

where $I(t) = 1$ if the location of a fist is sufficiently close to the target and the annotation at that frame indicates that the punch is effective. Otherwise, $I(t)$ is zero. Parameter $w(t) \leq 1$ is the normalized velocity of the blow at the moment of hitting and is set equal to one for the fastest blow. This function is associated with a $10 \times 10 \times 4$ grid that is overlayed on the bounding volume of effective hitting points (see Figure 3 (right) and Figure 4). The bounding volume has an angle range of -1.114 to 1.317 radians, a distance range of 29.7 to 129.3 mm, and a height range of 126.6 to 162.4mm. The policy tables require 0.9 MB of storage space.

Computing control policies. We randomly sampled a million of states (s,e) and applied the update rule to learn the "approach-the-target" behavior. Running time to compute this behavior was approximately one hour. The "throw-punches-at-the-target" behavior required 5 million iterations and 7 hours of computation time.

Animation and Control. Our first example in Figure 5(middle) shows an animated boxer that tracks and punches a standing target. The user can drag the target interactively. The approach behavior is always active and the punch behavior is activated only when the target is in the bounding box of effective hitting points. Collision between the hands and the target is checked at frames that have effective punches. When a collision is detected, the target reacts by deflecting in the direction of the punch by an amount proportional to the velocity of the punch and then returning slowly to its original position. The second example in Figure 5(bottom) shows two animated boxers that spar with each other. They are controlled by the same motion data and control policies and consider the head of the opponent to be their target to hit. Collision checking is done between the

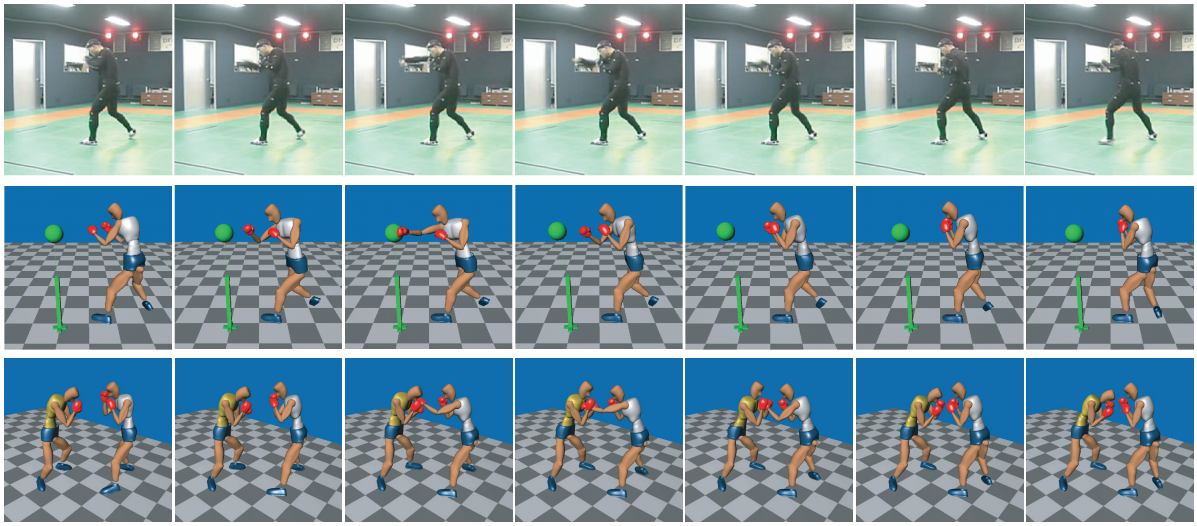

Figure 5: *Controllable animated boxers that hit and react. (Top) Motion data was recorded in an empty environment. (Middle) Our animated boxer learned to track and hit the target. (Bottom) Two animated boxers spar with each other.*

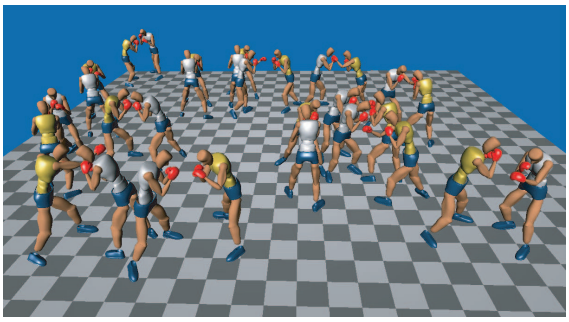

Figure 6: *Thirty animated boxers sparring.*

hands and the upper body of the opponents. Reaction to the collision is simulated by displacing the body segment contacted by a punch in the direction of the blow using an inverse kinematics solver.

Performance. To evaluate the performance of our system, we created 30 animated boxers sparring (see Figure 6). Our system required about 251 seconds to create 1000 frames of video images. Actually, rendering dominated the computation time. Our system required only 9 seconds to create the same animation with video and sound disabled. In other words, the motion of 30 avatars is computed and controlled at a rate of more than 100 frames per second.

7. Discussion

We have presented a precomputation method that allows our avatars to be animated and controlled interactively by constructing control policies from a collection of motion capture data. Our method finds an optimal (with an infinite-horizon delayed reward cost function) control policy at a given resolution and range of the state space. With a sufficiently dense discretization of the state space, the computed policy is able to find an optimal sequence of actions, which can also be found by global search algorithms used in [AF02, KGP02]. The local on-line search algorithm used by Lee et al. [LCR*02] trades off the optimality of solutions for interactive performance. We achieved a similar trade-off by finding a sequence within the resolution and range of a grid. This results in restricting the search space.

For realtime interactive systems, using precomputed control policies has a significant advantage in performance over local on-line search algorithms. Once the policy is computed, motion synthesis at runtime can be extremely fast. However, there is a downside to using precomputed policy. The precomputed policy does not allow us to change optimization objectives and parameters at runtime, which makes it difficult to coordinate multiple goals.

The presented method is memory intensive. Computing each behavior requires $O(NM)$ storage space, where N is the number of avatar states and M is the number of grid points. Recent results in machine learning show that function approximators and adaptive resolution models can be used to store large state spaces compactly [KLM96, SB98]. Moore and Atkeson [MA95] reported that their adaptive-resolution algorithm could learn policies in spaces of up to nine dimensions. Learning in high-dimensional state spaces will allow more convincing training scenarios such as learning policies from sparring with an opponent instead of a static target.

We believe that our method can be generalized to a broad variety of human motion other than boxing. A challenge of this generalization is to understand the context of interactions between avatars and an environment from unlabelled motion data. The current motion capture systems can hardly capture physical interactions and contacts precisely. In our boxing example, we recorded motion data without any physical target and identified effective hitting points later by automatically processing the data. It should be possible to identify other types of physical or indirect interactions such as gaze directions.

Collision detection and response provide an important visual cue for multiple character animation. We simply discarded the physics of collision and chose to displace body segments in contact somewhat arbitrarily for the purpose of visualizing collision events. Zordan and Hodgins [ZH02] used a physical collision model to compute the reaction of simulated humans to impact. Incorporating this physics based approach into realtime interactive systems is an interesting area for future work.

References

[AF02] ARIKAN O., FORSYTH D. A.: Interactive motion generation from examples. In *Proceedings of SIGGRAPH 2002* (2002), pp. 483–490.

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

[AMS97] ATKESON C., MOORE A., SCHAAL S.: Locally weighted learning for control. *AI Review 11* (1997), 75–113.

[BC89] BRUDERLIN A., CALVERT T. W.: Goal-directed, dynamic animation of human walking. In *Computer Graphics (Proceedings of SIGGRAPH 89)* (Boston, Massachusetts, July 1989), vol. 23, pp. 233–242.

[BDI*02] BLUMBERG B., DOWNIE M., IVANOV Y., BERLIN M., JOHNSON M. P., TOMLINSON B.: Integrated learning for interactive synthetic characters. In *Proceedings of SIGGRAPH 2002* (2002), pp. 417–426.

[BG95] BLUMBERG B. M., GALYEAN T. A.: Multi-level direction of autonomous creatures for real-time virtual environments. In *Proceedings of SIGGRAPH 95* (August 1995), pp. 47–54.

[BH00] BRAND M., HERTZMANN A.: Style machines. In *Proceedings of SIGGRAPH 2000* (July 2000), pp. 183–192.

[BHG93] BADLER N. I., HOLLICK M., GRANIERI J.: Real-time control of a virtual human using minimal sensors. *Presence 2* (1993), 82–86.

[Blu98] BLUMBERG B.: Swamped! Using plush toys to direct autonomous animated characters. In *SIGGRAPH 98 Conference Abstracts and Applications* (1998), p. 109.

[Bow00] BOWDEN R.: Learning statistical models of human motion. In *IEEE Workshop on Human Modelling, Analysis and Synthesis, CVPR2000* (July 2000).

[BS97] BRADLEY E., STUART J.: Using chaos to generate choreographic variations. In *Proceedings of the Experimental Chaos Conference* (august 1997).

[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.

[DYP03] DONTCHEVA M., YNGVE G., POPOVIC Z.: Layered acting for character animation. *ACM Transactions on Graphics (SIGGRAPH 2003) 22*, 3 (2003), 409–416.

[FvT01] FALOUTSOS P., VAN DE PANNE M., TERZOPOULOS D.: Composable controllers for physics-based character animation. In *Proceedings of SIGGRAPH 2001* (2001), pp. 251–260.

[GJH01] GALATA A., JOHNSON N., HOGG D.: Learning variable length markov models of behaviour. *Computer Vision and Image Understanding (CVIU) Journal 81*, 3 (March 2001), 398–413.

[GT95] GRZESZCZUK R., TERZOPOULOS D.: Automated learning of muscle-actuated locomotion through control abstraction. In *Proceedings of SIGGRAPH 95* (1995), pp. 63–70.

[GTH98] GRZESZCZUK R., TERZOPOULOS D., HINTON G.: Neuroanimator: fast neural network emulation and control of physics-based models. In *Proceedings of SIGGRAPH 98* (1998), pp. 9–20.

[JF03] JAMES D. L., FATAHALIAN K.: Precomputing interactive dynamic deformable scenes. *ACM Transactions on Graphics (SIGGRAPH 2003) 22*, 3 (2003), 879–887.

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. In *Proceedings of SIGGRAPH 2002* (2002), pp. 473–482.

[KLM96] KAELBLING L. P., LITTMAN M. L., MOORE A. W.: Reinforcement learning: A survey. *Journal of Artificial Intelligence Research 4* (1996), 237–285.

[KPS03] KIM T., PARK S. I., SHIN S. Y.: Rhythmic-motion synthesis based on motion-beat analysis. *ACM Transactions on Graphics (SIGGRAPH 2003) 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. In *Proceedings of SIGGRAPH 2002* (2002), pp. 491–500),.

[LWS02] LI Y., WANG T., SHUM H.-Y.: Motion texture: a two-level statistical model for character motion synthesis. In *Proceedings of SIGGRAPH 2002* (2002), pp. 465–472.

[MA95] MOORE A., ATKESON C.: The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning 21* (1995).

[Mat94] MATARIC M. J.: Reward functions for accelerated learning. In *Proceedings of the Eleventh International Conference on Machine Learning* (1994).

[MBT96] MOLET T., BOULIC R., THALMANN D.: A real-time anatomical converter for human motion capture. In *EGCAS '96: Seventh International Workshop on Computer Animation and Simulation* (1996), Eurographics.

[MH00] MOLINA TANCO L., HILTON A.: Realistic synthesis of novel human movements from a database of motion capture examples. In *Proceedings of the Workshop on Human Motion* (2000), pp. 137–142.

[NM93] NGO J. T., MARKS J.: Spacetime constraints revisited. In *Proceedings of SIGGRAPH 93* (1993), pp. 343–350.

[NRH03] NG R., RAMAMOORTHI R., HANRAHAN P.: All-frequency shadows using non-linear wavelet lighting approximation. *ACM Transactions on Graphics (SIGGRAPH 2003) 22*, 3 (2003), 376–381.

[NZB00] NOMA T., ZHAO L., BADLER N. I.: Design of a virtual human presenter. *IEEE Computer Graphics & Applications 20*, 4 (July/August 2000).

[PB00] PULLEN K., BREGLER C.: Animating by multilevel sampling. In *Computer Animation 2000* (May 2000), IEEE CS Press, pp. 36–42.

[PB02] PULLEN K., BREGLER C.: Motion capture assisted animation: Texturing and synthesis. In *Proceedings of SIGGRAPH 2002* (2002), pp. 501–508.

[PG96] PERLIN K., GOLDBERG A.: Improv: A system for scripting interactive actors in virtual worlds. In *Proceedings of SIGGRAPH 96* (August 1996), pp. 205–216.

[SB98] SUTTON R. S., BARTO A. G.: *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[SBS02] SIDENBLADH H., BLACK M. J., SIGAL L.: Implicit probabilistic models of human motion for synthesis and tracking. In *European Conference on Computer Vision (ECCV)* (2002), pp. 784–800.

[SE01] SCHÖDL A., ESSA I.: Machine learning for video-based rendering. In *Advances in Neural Information Processing Systems (NIPS)* (2001), vol. 13, pp. 1002–1008.

[SE02] SCHÖDL A., ESSA I.: Controlled animation of video sprites. In *Proceedings of the First ACM Symposium on Computer Animation* (2002), pp. 121–127.

[SHHS03] SLOAN P.-P., HALL J., HART J., SNYDER J.: Clustered principal components for precomputed radiance transfer. *ACM Transactions on Graphics (SIGGRAPH 2003) 22*, 3 (2003), 382–391.

[SHS98] SEMWAL S., HIGHTOWER R., STANSFIELD S.: Mapping algorithms for real-time control of an avatar using eight sensors. *Presence 7*, 1 (1998), 1–21.

[Sim94] SIMS K.: Evolving virtual creatures. In *Proceedings of SIGGRAPH 94* (1994), pp. 15–22.

[SLSG01] SHIN H. J., LEE J., SHIN S. Y., GLEICHER M.: Computer puppetry: An importance-based approach. *ACM Transactions on Graphics 20*, 2 (2001), 67–94.

[SLSS03] SLOAN P.-P., LIU X., SHUM H.-Y., SNYDER J.: Bi-scale radiance transfer. *ACM Transactions on Graphics (SIGGRAPH 2003) 22*, 3 (2003), 370–375.

[SSSE00] SCHÖDL A., SZELISKI R., SALESIN D. H., ESSA I.: Video textures. In *Proceedings of SIGGRAPH 2000* (July 2000), pp. 489–498.

[ZH02] ZORDAN V. B., HODGINS J. K.: Motion capture-driven simulations that hit and react. In *Proceedings of ACM SIGGRAPH Symposium on Computer Animation* (2002), pp. 89–96.