Responsive Characters from Motion Fragments

James McCann[*]
Carnegie Mellon University

Nancy Pollard[†]
Carnegie Mellon University

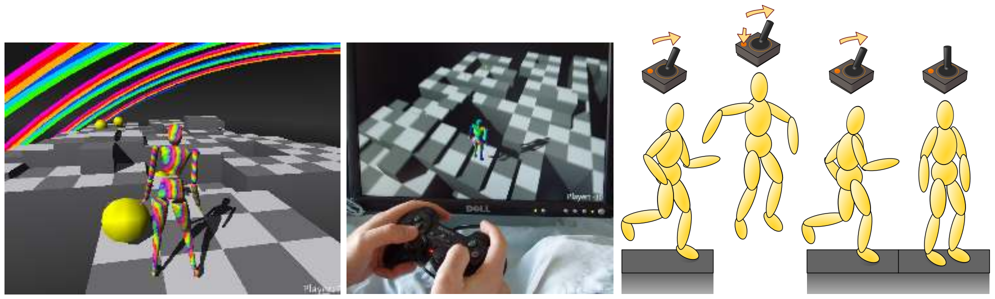

Figure 1: By modeling user behavior and not thresholding transitions, we create a high overall quality on-line motion generator suitable for directly controlled characters. **Left**, a screen capture. **Middle**, users control the character with a gamepad. **Right**, characters must respond immediately to user input, lest they run afoul of environmental hazards.

Abstract

In game environments, animated character motion must rapidly adapt to changes in player input – for example, if a directional signal from the player's gamepad is not incorporated into the character's trajectory immediately, the character may blithely run off a ledge. Traditional schemes for data-driven character animation lack the split-second reactivity required for this direct control; while they can be made to work, motion artifacts will result. We describe an on-line character animation controller that assembles a motion stream from short motion fragments, choosing each fragment based on current player input and the previous fragment. By adding a simple model of player behavior we are able to improve an existing reinforcement learning method for precalculating good fragment choices. We demonstrate the efficacy of our model by comparing the animation selected by our new controller to that selected by existing methods and to the optimal selection, given knowledge of the entire path. This comparison is performed over real-world data collected from a game prototype. Finally, we provide results indicating that occasional low-quality transitions between motion segments are crucial to high-quality on-line motion generation; this is an important result for others crafting animation systems for directly-controlled characters, as it argues against the common practice of transition thresholding.

CR Categories: I.3.6 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; E.1 [Data]: Data Structures—Graphs and Networks

Keywords: motion generation, motion graphs, character control

1 Introduction

In a game scenario with direct character control, such as our prototype (Figure 1), the character must react rapidly to changing player inputs. Quick reactions enhance the player's feeling of immersion and aid in avoiding environmental hazards. Game engines are able to enforce this reactivity at a whole-body level by translating and rotating the root of the character to obey the control signals; however, this does not excuse the pose-level character animation controller from rapid reaction, since discrepancies between the displayed animation and the character root motion manifest as visual artifacts like foot-skate.

One effective data-driven approach to this character animation problem works by stringing together short animation clips from a motion library; we dub this method *fragment-based character animation*. This approach is analogous to a motion graph where no long 'set-piece' motions are allowed. In this paper, we improve on an existing fragment-based animation algorithm ([Schödl and Essa 2000]) by developing a tabular-policy-based controller (Figure 2) which uses a conditional probability distribution to model user behavior. We evaluate our controller by comparing the fragment selections it makes to those of several other policy-based on-line controllers as well as the off-line, optimal selection. Additionally, we demonstrate that pruning the set of possible next fragments based on transition continuity, as is done when building motion graphs, can substantially decrease the overall output quality of on-line controllers such as ours.

2 Related Work

In industry, hand-constructed blend trees are often used to create on-line character animation [RenderWare 2001]; these trees con-

[*]e-mail:jmccann@cs.cmu.edu
[†]e-mail:nsp@cs.cmu.edu

tain carefully synchronized leaves (e.g. walk left, walk right, walk forward, walk back) which are interpolated by a hierarchy of blend nodes whose factors are given by player control signals. Some engines use simpler nearest-neighbor methods [Torque 2000]. Again, a set of synchronized motion loops is required. Instead of blending, however, each is annotated with an apparent control direction and only the closest to player input is displayed. The controllers in this paper produce higher-quality results than nearest-neighbor methods with comparable runtime CPU usage, while requiring fewer animator-hours to set up than blend trees.

Constructing fragment-based motions is similar to using a motion graph [Kovar et al. 2002; Arikan and Forsyth 2002; Lee et al. 2002] where no long motion segments are allowed and all segments may transition to all other segments. These modifications were motivated by the observations of Reitsma and Pollard [2004; (in press)], who explored the responsiveness of motion graphs by unrolling them into a game environment. By repeatedly sampling states and performing searches they calculated a mean lag time of over three seconds between user input and resulting action.

An alternative motion graph formulation emphasizing connectivity is given in [Gleicher et al. 2003]. This method does enable faster transitions, but still is unable to cope with immediate responsiveness in situations where a visual artifact would be introduced (such as transitioning from mid-stride to a jump). In contrast, our method is able to weigh this lack of visual quality against potential benefits in control-following.

Recently, methods to augment motion graphs using parameterized motions have been proposed [Shin and Oh 2006; Heck and Gleicher 2006]. These methods have been exploited to produce high quality controllable motion in restricted scenarios, but no full character controller with high responsiveness has been demonstrated. In particular, switching rapidly between actions – say, from a run to a crouch mid-step – is still difficult for these methods. Safonova [2006; 2007] introduces blending to motion graphs in a different way by making a graph in the space of $(\text{motion}_1, \text{motion}_2, \text{blend})$ tuples. Her method produces high-quality motion, but is not currently suited to on-line control because of computational (and memory) overhead.

Ikemoto el al. [2006] consider the problem of blending between motions and propose a method of caching the best fixed-length transitions between any two motions. In their present work, the transition window is too long (one second) to be considered for direct control situations; however, if no artifacts are introduced by shrinking this window, then their work may prove the basis of an interesting alternative to our controller.

Our method uses a reinforcement learning algorithm known as value iteration [Bellman 1957; Kaelbling et al. 1996; Sutton and Barto 1998]. Reinforcement learning is not new to motion control in graphics. Lee and Lee [2006; 2004] use reinforcement learning to create an avatar control model. They, however, prune their motion graph before planning, which we demonstrate can negatively impact quality. Ikemoto et al. [2005] use reinforcement learning on a more complex state space in order to aid a global planner. Their final controller is thus destination instead of control signal based. They also prune their motion graph. Schödl and Essa [2000] present work very close to our steady-state controller in a different control domain (video sprites). None of these previous methods employ a user model.

Using learning with a user model is common in other fields (e.g. [Levin et al. 2000], where a user model aids a dialog system). To our knowledge, however, such an approach has not been explored for reactive character animation.

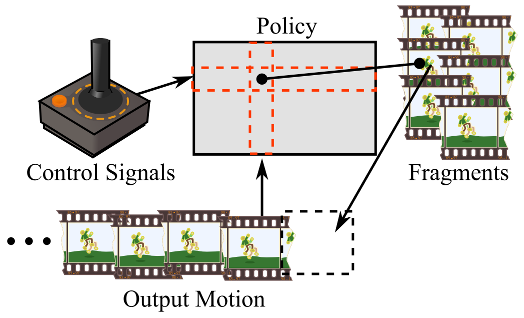

Figure 2: Our controller operates by using the current control signals and previously played fragment to look up the next fragment in a control policy table.

More complicated reinforcement learning methods exist which would allow us to learn on the continuous control signals instead of a discrete approximation [Moore and Atkeson 1995]. Treuille et al. [2007] explore this possibility, making use of basis functions to represent value functions over continuous control signals.

Another topic in graphics that deals extensively with user control is performance animation (e.g. [Shin et al. 2001; Hsu et al. 2004; Chai and Hodgins 2005; Dontcheva et al. 2003]). These systems provide fine control over character pose, and often deal with more exotic input data (e.g. full or partial motion capture), while we focus on providing control over gross motion with more conventional input devices.

An attractive alternative to data-driven animation for on-line character motion generation is to use physics-based controllers. A framework for composing such controllers is given by Faloutsos et al. [2001], while Grzeszczuk and Terzopoulos [1995] use learning to create a physical controller. Van de Panne and his colleagues explore interactive control of simulations [Laszlo et al. 2000; van de Panne and Lee 2003; Zhao and van de Panne 2005]. Abe et al. [2004] use momentum to parameterize motions for interactive control. James et al. [2003; 2006] combine physics-based data with precalculation techniques to create interactive animations of deformable objects.

3 The Controller

Our character animation controller works by generating a stream of motion fragments. As each fragment completes, the next fragment to play is selected based on the current player input and previous fragment (see Figure 2); these decisions are stored in a tabular *control policy*. In this section we discuss how this table is generated. An overview of the data flow during synthesis is given in Figure 3. We will motivate and describe each portion of the process below.

The key observation in our generation approach is that if we knew exactly how the player would change their input in the future, then we could always make the best possible choice of next fragment. Failing that perfect foreknowledge, however, we can get some idea of how players will act by setting up our game environment with a stand-in for the animated character and collecting example input streams, henceforth called *traces*. With these traces we can build a rough model of player behavior.

One caveat, however, is that control signals generally exist in a continuous, high-dimensional space. We deal with this by selecting a set of points in this control space and calling the region of the space closer to a given point than any other its *control bin*. (We note here that this requires a notion of distance in the control space, some-

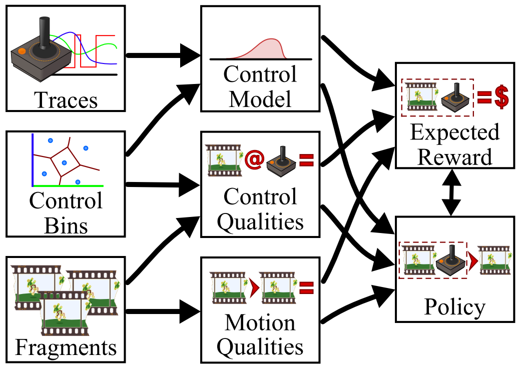

Figure 3: The data flow involved in synthesizing our controller. Example user inputs (traces), a discretization of control, and a database of motion fragments are used to calculate a user control model and matrices of qualities. These intermediate structures are used by a value iteration process to generate a final policy.

thing we will touch on in Section 4.1.) In our implementation, we chose these bin centers by hand (Figure 4), though one could conceive of choosing them automatically by using a tactic like vector quantization on the example traces.

The final input to our synthesis process is, of course, the motion fragments themselves. The only requirement placed on these fragments is that they be both short and annotated with evident control – an indication of what control signal they represent.

Now that we have described the inputs to our system, we move on to the intermediate data (column two of Figure 3). This is the data we construct so that we can evaluate different policy choices.

The control model, which we mentioned earlier, is the key piece of intermediate data, since it tells us how player input is expected to change. We use a table of conditional probabilities T, where $T_{cc'}$ gives the probability that the next control signal will be in bin c', given the previous control signal was in bin c. This table is computed by performing frequency counts on the example traces. Using such an elementary model has two advantages. First, it keeps our policy table small ($|\text{control bins}| \times |\text{motion fragments}|$), because each decision only depends on the current input and the previous fragment. Second, since the model has relatively low representative power it generalizes well to players for whom we do not have example traces.

With a control model we can reason about the future impact of decisions. To capture the immediate impact of decisions, we need to have a notion of the quality of selecting a given next fragment. This quality will tell us how artifact-free a motion is. We find it expedient to factor quality:

$$\mathsf{Quality}(f \to f'|c) = \mathsf{MotionQuality}(f \to f') \cdot \mathsf{ControlQuality}(f'|c) \tag{1}$$

Here, $\mathsf{MotionQuality}(f \to f') \in [0,1]$ indicates smoothness and realism of the transition between two fragments f and f', with zero being jerky or unrealistic and one being pleasing to the eye. $\mathsf{ControlQuality}(f'|c) \in [0,1]$ indicates how closely fragment f''s evident control matches the control signal c, with zero being performing the wrong action [e.g., jumping at the wrong time] and one corresponding to performing the correct action [e.g. walking in the correct direction]. Our implementations of MotionQuality and ControlQuality are discussed in Section 4.1. This factorization allows for a natural trade-off between artifacts introduced through bad transitions and those introduced through inaccurately following control-induced character root motion.

We now have all that we need to create our policy: the control bins and fragments give us a discretized space in which to represent our decisions, the control model lets us reason about future player inputs, and the ControlQuality and MotionQuality functions provide a means to compute the immediate benefit of decisions.

The only thing that remains is to compute the expected benefit of each decision. Value iteration is a computational framework that does just this, computing both optimal behavior and the expected discounted sum of future rewards given that behavior. By 'discounted' we mean that the reward n time steps in the future is multiplied by λ^n– our choice of λ is given in Section 4.1.

Value iteration may be succinctly expressed as an expected reward update rule

$$R_{fc} \leftarrow \max_{f'} \left(\mathsf{Quality}(f \to f'|c) + \lambda \cdot \sum_{c'} T_{cc'} \cdot R_{f'c'} \right) \tag{2}$$

where R_{fc} is initialized to 0 and stores the expected reward at fragment f with current control signal c. Once this update rule has been iterated until R converges, the policy may be constructed as

$$P_{fc} \leftarrow \operatorname{argmax}_{f'} \left(\mathsf{Quality}(f \to f'|c) + \lambda \cdot \sum_{c'} T_{cc'} \cdot R_{f'c'} \right) \tag{3}$$

where P_{fc} stores the index of the next fragment to play given previous fragment f and current control signal c. It is a property of value iteration that this policy is optimal (given the modeling assumptions). For more information on value iteration we recommend Sutton and Barto's book [1998].

3.1 Alternate Controllers

Having described our controller, we now briefly describe those controllers which we compare against. With the exception of globally optimal, all the below controllers result from modification of the reward update, with the policy still computed as per Equation 3.

Greedy control does not care about future reward, and thus its reward update is simply:

$$R_{fc} \leftarrow \max_{f'} \mathsf{Quality}(f \to f'|c) \tag{4}$$

Steady State control (as per [Schödl and Essa 2000]) assumes that control signals never change, which eliminates matrix T from the reward update:

$$R_{fc} \leftarrow \max_{f'} \left(\mathsf{Quality}(f \to f'|c) + \lambda \cdot R_{f'c} \right) \tag{5}$$

Spread control eliminates T by assuming that control signals change uniformly:

$$R_{fc} \leftarrow \max_{f'} \left(\mathsf{Quality}(f \to f'|c) + \lambda \cdot \frac{1}{|C|} \cdot \sum_{c'} R_{f'c'} \right) \tag{6}$$

where $|C|$ is the number of control bins.

Optimal control is possible, given knowledge of the future path. This restriction makes it unsuitable for on-line control, but we use it as a point of comparison. We use dynamic programming to build table V where entry V_i^f gives the value of choosing fragment f at step i along the path:

$$V_i^f \leftarrow \max_{f'} \left(\mathsf{Quality}(f \to f'|c_i) + V_{i+1}^{f'} \right) \tag{7}$$

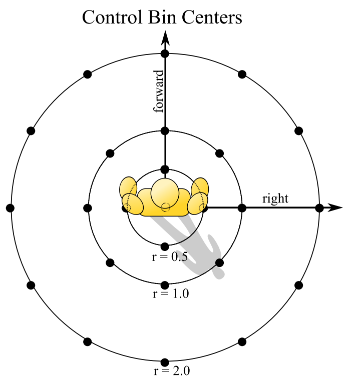

Figure 4: Ground-plane velocities (black dots) used for control signal binning in the example application. Velocities are in body-local coordinates. Each black dot corresponds to six bin centers, with different turning rates in $\{-1,0,1\}$ and airborne state in $\{0,1\}$.

where c_i gives the control signal at step i along the path and value V_n at the last step is set to 0. The optimal path $f_0,\dots,f_n$ may then be read back by selecting the maximum V_0 and stepping forward through time:

$$\begin{aligned} f_0 &\leftarrow \operatorname{argmax}_f V_0^f, \\ f_{i+1} &\leftarrow \operatorname{argmax}_f \left(\mathsf{Quality}(f_i \to f|c_i) + V_{i+1}^f\right) \end{aligned} \tag{8}$$

4 Results

4.1 The Scenario

We put our animation controller to work in the context of a simple sphere-collection game (Figure 1). In this game, players use one joystick on their gamepad to set ground-plane velocity of the character, while the other joystick controls the character's turning rate. A button is used to jump. Our prototype game fixes the character root motion to that defined by the control signals in order to provide perfect responsiveness. Additionally, the engine "cleans up" the character animation by blending between fragments and adjusting the fragment playback rate to best match the current control signals; these post-processes are not considered in our evaluation, since they could be equally well applied to the output of any character animation controller.

We begin our discussion of the implementation details by reviewing the inputs to our algorithm.

Traces. Seven minutes of two-player gameplay was recorded using a dummy character animation controller: the game engine constrained the character's root to follow the user's control signals exactly, but the character's pose was not animated. An additional two minutes of gameplay was recorded in a different session from one of the players to be used in evaluating the controllers.

Control bins were centered on 150 points selected in the game's control space – all possible combinations of ground plane velocities in three concentric circles (Figure 4), turning speeds in $\{-1,0,1\}$ radians per second, and airborne flag in $\{0,1\}$. The velocities are expressed in character-local coordinates.

Fragments were drawn from approximately five minutes of motion in the CMU motion capture database [CMU 2001]. This motion was selected to include walking, running, jumping, and sidestepping. Fragments 0.1 seconds in length were started every 0.05 second in each motion. This overlap provides some flexibility in starting positions without generating too many extra fragments. The fragment pool generated in this manner contained 5934 fragments.

	Regular		Thresholded	
Method	% opt	std.dev.	% opt	std.dev.
model	90	1.9	45	4.1
steady	84	2.6	41	5.1
spread	81	2.4	48	7.1
greedy	46	2.8		
opt			91	2.3

Table 1: Summary of quality as a percentage of optimal. Our controller, model, is better than the other controllers in normal operation. Thresholding causes all of the policy controllers to falter, even though the optimal path value has only decreased by 9%.

Having described the inputs used in our test scenario, we move on to describing the intermediate data.

Control Model. The control model is always computed via frequency counting on the traces and so requires no adaption to this scenario.

Motion Quality was based on the intuition that reflecting a joint angle at frame i about the joint angle at frame $i+j$ provides a prediction of the future angle at frame $i+2j$. (That is, if the elbow is bent 90° at frame 2 and bent 95° at frame 4, one might reasonably expect it to be bent 100° at frame 6). Bearing this in mind, let the quality of transitioning between fragments f and f' consisting of poses $f_1,\dots,f_m$ and $f'_1,\dots,f'_n$ respectively be:

$$\mathsf{MotionQuality}(f \to f') = \left(1 + \sum_{i=1}^{3} \omega_i \cdot \mathsf{Dis}(\mathsf{Refl}(f_{m-\delta_i}, f_m), f'_{\delta_i})\right)^{-1} \tag{9}$$

where the weights $\omega = [0.8, 0.5, 0.3]$ and the offsets $\delta = [1,2,3]$ where chosen by hand to produce good results, $\mathsf{Dis}(f,f')$ takes the distance between poses by summing euclidean distance between bone tips, and $\mathsf{Refl}(f,f')$ reflects pose f over pose f' in angle space. As a special case, we force $\mathsf{MotionQuality}(f \to f') = 0$ whenever f' appears within a short window before f in the source motion. This helps to prevent "stuttering" artifacts.

Control Quality was computed using a distance measure between the evident control $\mathsf{Ev}(f)$ in a fragment (estimated from center-of-mass motion) and the given control c:

$$\mathsf{ControlQuality}(f|c) = (1 + \mathsf{ControlDis}(\mathsf{Ev}(f), c))^{-1} \tag{10}$$

where control distance is a weighted euclidean distance on control vectors $c = (c_v, c_t, c_a)$:

$$\mathsf{ControlDis}(c,c') = w_v(\mathsf{P}(c_v) - \mathsf{P}(c'_v))^2 + w_t(c_t - c'_t)^2 + w_a(c_a - c'_a)^2 \tag{11}$$

and $\mathsf{P}(c)$ projects non-zero velocities into a log-magnitude space:

$$\mathsf{P}(c_v) = \frac{\log(|c_v|+1)\cdot c_v}{|c_v|} \tag{12}$$

(This projection seems to match human perception of changes in velocity.) We choose $w_v = 10$, $w_t = 6$, and $w_a = 20$ because these weights give good results in practice.

We have now described all of the intermediate data required for our algorithm. The only remaining implementation detail is the discount factor λ, which we set to 0.999.

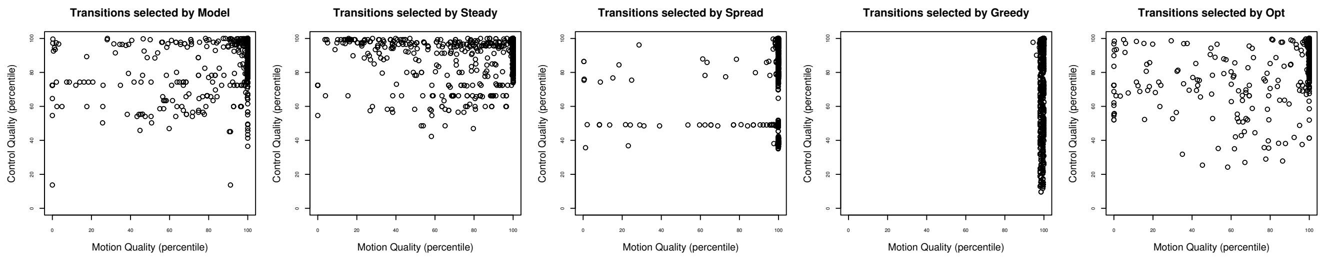

Figure 6: Distribution of transitions in motion quality percentile versus control quality percentile space; greedy and spread seem very concerned with motion quality, while steady favors high control quality. Our controller, model, strikes a similar balance to opt.

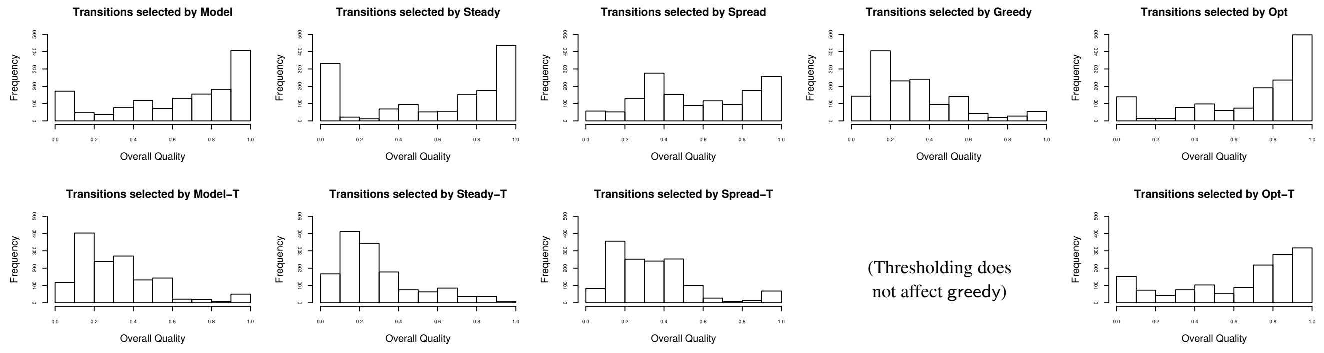

Figure 7: Distribution of transitions in overall quality; greedy is clearly the lowest quality, while opt does a very good job of avoiding low-quality transitions. Both model and steady have similar profiles, but steady is forced to take more low quality transitions because it is unprepared for changes in user input. Thresholding (bottom row) forces the policy-based controllers to take far more bad transitions.

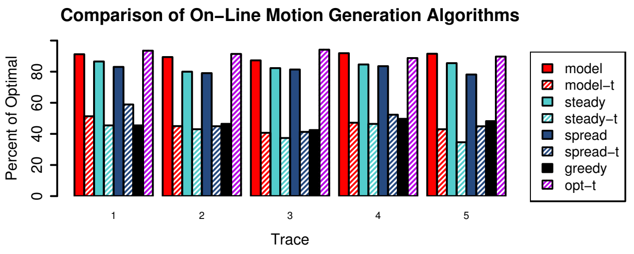

Figure 5: Comparison of trace quality as a percentage of optimal. The model-based controller performs better than a controller without a user model. Thresholding transitions based on motion quality has a substantial negative impact.

4.2 Evaluation

For the purposes of evaluation, we define nine controllers. We designate our controller from Section 3 model; the controllers from Section 3.1 will be designated greedy, steady, spread, and opt. We also introduce the model-t, steady-t, spread-t, and opt-t variants, which are the same controllers restricted to only use transitions in the top 8% by motion quality. (This is akin to the sort of transition quality thresholding commonly run on motion graphs.) We omit greedy-t, since it is identical to greedy.

Synthesis of steady, spread, and model controllers took several hours, while greedy required two minutes to compute. Synthesis times for '-t' variants were faster (since there were fewer transitions to consider). Each controller was run over two minutes of example control signal recordings (traces). These traces were distinct from those used in building the user model. Runtimes for model, steady, spread, and greedy were trivial (since policy evaluation is simply a matter of table lookup), while opt took approximately thirty minutes to calculate. Once the controllers finished in their fragment selection, qualities of the generated paths were calculated using the Quality function defined earlier.

Results of running the controllers on individual traces are given in Figure 5, with a summary in Table 1. From this overview, two conclusions may be drawn: first, the user model is useful – model performs at 90% of optimal, while steady (the previous state-of-the-art) achieves 84% and spread achieves 81%. Second, thresholding negatively – and dramatically – impacts the overall quality of generated motion (model, steady, and spread all lose more than 30% of optimal).

For insight into this second point, consider the individual fragment-fragment transitions used by the non-thresholded controllers (Figure 6). With the exception of greedy, all the methods rely, at least to some extent, on low-motion-quality transitions. An interesting side-note is that opt is not hampered as severely by the lack of low motion quality transitions (only an 9% loss); from this we may conclude that, while reasonably high quality paths still exist in the thresholded data set, they are much harder to find.

This conclusion is also supported by comparing the overall quality of chosen transitions before and after thresholding (Figure 7). Before thresholding, the transition quality histograms of model and steady appear close to opt (though steady is forced to take some very low-quality transitions when control signals change); after thresholding, the transition histograms for model, steady, and spread seem closer to those for greedy. It is also interesting to note that while spread does not have as pronounced a high-quality spike as does model, steady, and opt, it still does quite well before thresholding.

In order to determine how our user model generalized, we synthesized model using example traces from just the user who did not perform the evaluation traces, as well as just the user who did. In both cases it still operated near mean 90% of optimal, with a slightly higher standard deviation (2.0% vs. 1.7%) when sythesizing with the user who did not perform the evaluation traces. This seems to indicate that our user model is able to capture broad features of the control domain without fitting the training data too closely.

Investigating the scaling behavior of our controller, we upped the number of input fragments from 5934 to 7577 (adding examples of a motion which the previous controller had performed badly on), and re-ran the value iteration. This increased the value iteration step time and memory usage from 22 seconds and 200 megabytes to 35 seconds and 330 megabytes. (We will discuss theoretical scaling behavior later.) In the presence of these additional fragments, opt increased by about 1%, while model performed at the same percentage of this new opt. The new motion was used by the controller. The lack of a larger percentage increase may be due to the relative infrequency in the example traces of the additional motion (right-sidestepping).

5 Discussion

In the previous sections we formulated the problem of on-line character animation control as that of assembling a high-quality sequence of motion fragments; we then presented and evaluated an animation controller for this domain. This controller improved on previous efforts by modeling user behavior and by considering all possible transitions, instead of just ones with high motion quality. Below, we discuss some of the advantages and disadvantages of our chosen problem formulation and controller.

Fragment-based motion generators as we presented in this paper have the great advantage of existing in a domain where optimal paths can be calculated. This allows methods to easily be compared to optimal, as we did in the results section, and for the completeness of metrics and source data to be vetted by watching optimal paths. (For if the optimal paths do not look right, what chance does a mere on-line algorithm stand?) Furthermore, in the case of generators built around a precomputed control policy, all transitions that will ever be taken and the circumstances for those transitions are known before the controller is deployed, so it possible to examine the worst possible motion – either to determine that no bad surprises await or that adjustments are required. Additionally, the computed expected future reward at each (fragment, control bin) pair tells us what situations are bad to be in not just from an immediate quality perspective, but in the long term. Finally, if modification is required, value iteration will converge quickly given a good guess – for instance, the existing (pre-adjustments) policy. And even when not fully converged, policies tend to be reasonably close to their final states. These factors conspire to allow rapid iteration when users are searching for good parameters and source motions.

A limitation of all fragment-based motion generators is that they will never be able to respond in less than one fragment's time; thus fragment length selection is a trade-off between increased memory usage and precomputation time and run-time responsiveness, for which an application-specific balance must be struck. By selecting only the best motion in a situation, our policy will lose some of the variation present in the source database. However, one could attempt to amend the policy by keeping a family of good next paths instead of simply one, then choosing at random to provide variety. Transitions with high motion quality are sometimes simply not available, even for the globally optimal motion. We believe this problem can never be completely solved. For example, a truly "natural" transition into a jump from the middle of a walking stride at the speed a gamer would demand may be impossible. However, blending techniques in [Shin and Oh 2006; Heck and Gleicher 2006; Safonova 2006; Ikemoto et al. 2006] could be combined with our approach to improve the overall quality of the output motion.

Our algorithm, given $|F|$ fragments and $|C|$ control bins, requires $O(|C|)$ time per fragment selection (since we perform a linear scan when finding the proper control bin; temporal locality should allow this to be amortized to $O(1)$) and $O(|F| \cdot |C|)$ memory (to store the policy table). In our example controller, this corresponds to well under 1ms for fragment selection and approximately two megabytes of memory. Synthesis is more expensive, requiring $O(|F|^2|C|^2)$ time per value iteration step (because Equation 2 is executed for all fragments and control bins and scans all fragments and control bins), and $O(|F|^2 + |C|^2)$ space (in order to store precomputed quality matrices). Convergence generally requires $O(n)$ steps, where n is the size of the largest cycle in the optimal controller. For our example controller this corresponds to roughly 22 seconds per iteration step and 200 megabytes of process memory. While large controllers are feasible at runtime, actually synthesizing them using our straightforward value iteration becomes infeasible at around 15000 fragments (we run out of memory). With a more advanced policy selection scheme, one could envision using much larger data sets.

Finally, there are several interesting directions to explore in future work. First, value iteration is an inherently parallel process and it should be possible to exploit this to accelerate the synthesis process, thus providing faster feedback to parameter changes. Second, control policies are well-suited to evaluation on a GPU and could enable large numbers of animated characters to be computed without CPU load. Third, fragments do not need to be drawn directly from a motion database, but could instead be synthesized; for instance, interpolations and simple transformations – reflection, reversing – of existing fragments could be introduced. This procedure could be automated to fill deficiencies in the fragment set (missing control directions, for instance).

6 Conclusion

In this paper we presented and evaluated several policy-based online character animation controllers over real-world example data. In this way, we demonstrated that a controller generated using value iteration with a user model was an improvement over previous controllers. We also determined that even relatively conservative trimming of transitions (by motion graph standards, at least) has a disastrous impact on the overall quality of the on-line controllers. This result demonstrates the need, in data-driven motion generation, to preserve low motion quality transitions in order to avoid becoming stuck in low control quality situations.

Acknowledgments

Thanks to Jimmy Andrews for suggestions on metrics, Chris Atkeson for pointers on value iteration, Christopher Twigg for matrix palette skinning examples, and Irfan Essa for valuable discussion. Motion data used in this project was obtained from mocap.cs.cmu.edu; this database was created with funding from NSF EIA-0196217. This research was supported with NSF grants IIS-0326322 and ECS-0325383.

References

ABE, Y., LIU, C. K., AND POPOVIĆ, Z. 2004. Momentum-based parameterization of dynamic character motion. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 173–182.

ARIKAN, O., AND FORSYTH, D. A. 2002. Interactive motion generation from examples. ACM Press, New York, NY, USA, vol. 21, 483–490.

BELLMAN, R. 1957. *Dynamic Programming*. Princeton University Press, Princeton, NJ.

CHAI, J., AND HODGINS, J. K. 2005. Performance animation from low-dimensional control signals. *ACM Transactions on Graphics 24*, 3, 686–696.

CMU, 2001. http://mocap.cs.cmu.edu.

DONTCHEVA, M., YNGVE, G., AND POPOVIĆ, Z. 2003. Layered acting for character animation. *ACM Transactions on Graphics 22*, 3, 409–416.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composable controllers for physics-based character animation. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press, New York, NY, USA, 251–260.

GLEICHER, M., SHIN, H. J., KOVAR, L., AND JEPSEN, A. 2003. Snap-together motion: Assembling run-time animations. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA.

GRZESZCZUK, R., AND TERZOPOULOS, D. 1995. Automated learning of muscle-actuated locomotion through control abstraction. In *Proceedings of ACM SIGGRAPH 1995*, ACM Press, New York, NY, USA, 63–70.

HECK, R., AND GLEICHER, M., 2006. Parametric motion graphs. ACM SIGGRAPH / Eurographics Symposium on Computer Animation (Poster), August.

HSU, E., GENTRY, S., AND POPOVIĆ, J. 2004. Example-based control of human motion. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 69–77.

IKEMOTO, L. K. M., ARIKAN, O., AND FORSYTH, D. A. 2005. Learning to move autonomously in a hostile world. Tech. Rep. UCB/CSD-05-1395, EECS Department, University of California, Berkeley.

IKEMOTO, L. K. M., ARIKAN, O., AND FORSYTH, D. 2006. Quick motion transitions with cached multi-way blends. Tech. Rep. UCB/EECS-2006-14, EECS Department, University of California, Berkeley, February 13.

JAMES, D. L., AND FATAHALIAN, K. 2003. Precomputing interactive dynamic deformable scenes. *ACM Transactions on Graphics 22*, 3 (July), 879–887.

JAMES, D. L., TWIGG, C. D., COVE, A., AND WANG, R. Y. 2006. Mesh ensemble motion graphs. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, ACM Press, New York, NY, USA, 69.

KAELBLING, L. P., LITTMAN, M. L., AND MOORE, A. P. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research 4*, 237–285.

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. ACM Press, New York, NY, USA, vol. 21, 473–482.

LASZLO, J., VAN DE PANNE, M., AND FIUME, E. 2000. Interactive control for physically-based animation. In *Proceedings of ACM SIGGRAPH 2001*, ACM Press, New York, NY, USA.

LEE, J., AND LEE, K. H. 2004. Precomputing avatar behavior from human motion data. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 79–87.

LEE, J., AND LEE, K. H. 2006. Precomputing avatar behavior from human motion data. *Graph. Models 68*, 2, 158–174.

LEE, J., CHAI, J., REITSMA, P. S. A., HODGINS, J. K., AND POLLARD, N. S. 2002. Interactive control of avatars animated with human motion data. ACM Press, New York, NY, USA, vol. 21, 491–500.

LEVIN, E., PIERACCINI, R., AND ECKERT, W. 2000. A stochastic model of human-machine interaction for learning dialog strategies. *IEEE Transactions on Speech and Audio Processing 8*, 1, 11–23.

MOORE, A., AND ATKESON, C. 1995. The parti-game algorithm for variable resolution reinforcement learning in multidimensional state-spaces. *Machine Learning 21*.

REITSMA, P. S. A., AND POLLARD, N. S. 2004. Evaluating motion graphs for character navigation. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 89–98.

REITSMA, P. S. A., AND POLLARD, N. S. (in press). Evaluating motion graphs for character animation. *ACM Transactions on Graphics*. in press.

RENDERWARE, 2001. http://www.renderware.com.

SAFONOVA, A., AND HODGINS, J. 2007. Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics 26*, 3 (Aug.).

SAFONOVA, A. 2006. *Reducing the search space for physically realistic human motion synthesis*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA.

SCHÖDL, A., AND ESSA, I. A. 2000. Machine learning for video-based rendering. Tech. Rep. GIT-GVU-00-11, Georgia Institute of Technology.

SHIN, H. J., AND OH, H. S. 2006. Fat graphs: Constructing an interactive character with continuous controls. In *ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 291–298.

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

SUTTON, R. S., AND BARTO, A. G. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, ch. 4.4.

TORQUE, 2000. http://www.garagegames.com/products/torque/tge/.

TREUILLE, A., LEE, Y., AND POPOVIĆ, Z. 2007. Character animation with continuous user control. *ACM Transactions on Graphics 26*, 3 (Aug.).

VAN DE PANNE, M., AND LEE, C. 2003. Ski stunt simulator: Experiments with interactive dynamics. In *Proceedings of the 14th Western Computer Graphics Symposium*.

ZHAO, P., AND VAN DE PANNE, M. 2005. User interfaces for interactive control of physics-based 3d characters. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 87–94.