Near-optimal Character Animation with Continuous Control

Adrien Treuille Yongjoon Lee Zoran Popović

University of Washington

Figure 1: **Spin and Avoid.** A character spins while following a straight line. When fixed obstacles suddenly appear in the path, the character automatically switches to obstacle avoidance mode and navigates around the obstacles.

Abstract

We present a new approach to realtime character animation with interactive control. Given a corpus of motion capture data and a desired task, we automatically compute near-optimal controllers using a low-dimensional basis representation. We show that these controllers produce motion that fluidly responds to several dimensions of user control and environmental constraints in realtime. Our results indicate that very few basis functions are required to create high-fidelity character controllers which permit complex user navigation and obstacle-avoidance tasks.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

Keywords: Optimal Control, Motion with Constraints, Human Animation

1 Introduction

Despite decades of advances, interactive character animation still lacks the fluidity and variability of real human motion. Capturing the fine nuances of human motion requires a high-dimensional motion repertoire and a multivariate space of input parameters which can change continuously, often unpredictably. For example, a human might fluidly move to avoid unexpected moving obstacles while approaching an exit and walking next to another person. Unfortunately, algorithms for solving such problems remain highly sensitive to the size of the state space and number of concurrent goals.

In this paper, we look at these issues through the lens of a difficult and important subproblem in character animation: synthesis of a kinematic controller that blends precaptured motion clips to achieve complex multi-objectives in realtime. In particular, we assume continuously changing environmental conditions and several dimensions of user control over the character's motion. For example, a virtual sports application might require that the character's speed, direction, and torso orientation all be independently controllable. Tasks like obstacle avoidance introduce additional control variables. For simple navigational control, interactive applications have typically used precomputed motion cycles and carefully constructed graphs that blend between these cycles. However building appropriate graphs and transitions requires a great amount of skill and manual adjustment. For complex objectives, researchers have also studied learning algorithms that automatically determine optimal sequences of actions. The primary challenges for both manual and learning approaches are high-dimensionality and the need for both rapid and natural changes in motion.

This paper argues that *low-dimensional reinforcement learning* presents an attractive unifying approach to these problems. This approach outperforms greedy methods for navigational problems while retaining the low memory overhead. At the same time, despite the compact representation, we can still solve difficult problems such as obstacle avoidance.

Our technique has two components. First, an underlying motion model stitches together captured motion clips in realtime. Second, a controller selects sequences of clips to achieve some goal. The controller is based on a compact, parametric *value function* over all possible states and goals. This approach has the following advantages: given a corpus of motion capture data and a set of controller objectives, we automatically produce a highly compact realtime controller that blends motion near-optimally to achieve user goals. There is no need to explicitly construct walk cycles or otherwise edit the underlying data. We show that our technique is also expandable, in that we can construct a set of controllers separately for different tasks, then transition between these controllers near-optimally, even if they were learned on different underlying sets of motion. This greatly reduces the cost of constructing complex controllers, as each individual task can be optimized separately before connecting the controllers together. A similar idea allows us to parallelize precomputation along dimensions that do not change over time. We demonstrate our technique for a number tasks including navigational control of a character and moving obstacle avoidance. The latter allows us to create crowd simulations in which each character fits a local linear model to the movement of near characters, then avoids those characters in realtime without an explicit collision avoidance algorithm.

ACM Reference Format
Treuille, A., Lee, Y., Popović, Z. 2007. Near-optimal Character Animation With Continuous Control. *ACM Trans. Graph. 26*, 3, Article 7 (July 2007), 7 pages. DOI = 10.1145/1239451.1239458 http://doi.acm.org/10.1145/1239451.1239458.

Contributions. Our main contribution is to show that the inherent smoothness of the value function for many tasks in character animation enables the construction of low-dimensional, near-optimal controllers for these tasks. We also focus on controllers that can be optimized in parallel for a subset of input parameters, allowing us further to increase the dimensionality of the controller. We hope these ideas will extend the applicability of optimal control theory to more problems in motion synthesis.

2 Related Work

Human motion capture data provides an effective basis for creating new animations. By interpolating and concatenating motions, realistic new animation can be generated. A graph representation often describes allowable transitions between poses. Pullen and Bregler [2002] segmented motion data into small pieces and rearranged them to meet user-specified keyframes. Kovar et al. [2002] generated a graph structure from motion data allowing characters to follow sketched paths using a branch and bound algorithm. Arikan and Forsyth [2002] created a hierarchy of graphs and employed a randomized search algorithm for synthesizing a new motion subject to temporal and position constraints. These algorithms employ complex graph search and optimization techniques to produce animations and are not well suited for interactive control. Lee et al. [2002] use a local on-line search algorithm which is demonstrated to generate 5 to 8 frames of motion per second when the search horizon is limited. The graph search can be made more efficient by using a simple cyclic graph structure [Gleicher et al. 2003], by composing the world from smaller environment-specific graphs [Lee et al. 2006], by precomputing search trees [Lau and Kuffner 2006], and by grouping similar motion segments into parametrized graph nodes and building sparse hierarchical motion graphs [Kwon and Shin 2005]. Choi et al. [2003] generated environment-specific graph representations that enable biped locomotion through a complex environment with obstacles. Arikan et al. [2003] showed that motion synthesis from a graph structure can be cast as dynamic programming. Our motion model uses a graph structure, but the blending algorithm admits blends between *any* two clips of motion and automatically prevents foot-skate without inverse kinematics for a wide class of motion. Researchers have also generalized transition graphs to form richer, more complete motion spaces [Shin and Oh 2006; Safonova and Hodgins 2007]. It would be highly interesting to extend our idea to these more general spaces.

Our controller uses *reinforcement learning* methods, which are well studied in the control community. Kaelbling et al. [1996] provide a good overview of the discrete reinforcement learning problem. Bertsekas [2001] presents in-depth coverage of the continuous case, as well as of the delayed rewards model which we use. LaValle [2006] provides a good overview with many applications to robotics and motion planning. Atkeson et al. [1997] frame reinforcement learning in the broader context of function approximation. However, these techniques suffer acutely from dimensionality, making control difficult or impossible in high-dimensional spaces. Moore and Atkeson [1995] address this problem specifically with a multi-resolution reinforcement learning algorithm; however this method relies heavily on storing samples which can become intractable for the large-scale problems we wish to solve. Researchers have explored linear programming to efficiently solve reinforcement learning problems [Singh and Yee 1994; Trick and Zin 1997]. In this vein, Pucci de Farias presented a linear programming approach that uses basis functions to compactly approximate the value function without storing samples [2002]. This algorithm forms the basis for our approach to character animation because it greatly mitigates the cost of dimensionality by avoiding runtime sampling. In this paper, we also reduce the dimensionality burden by learning optimal controllers only within certain subspaces, while parallelizing precomputation along other dimensions.

Reinforcement learning is increasingly appearing in the graphics literature, for example to create video textures [Schödl and Essa 2001] or for local character navigation [Ikemoto et al. 2005]. The closest work to our own is that of Lee and Lee [2004] which used value iteration to construct a sample-based value function for boxing. Relative to this method, our precomputations are 7× faster on comparably sized problems, but require more memory. At runtime, however, our basis approximation requires 30-90× *less* memory than Lee and Lee's sample representation. To be similarly compact, sample-based methods such as Lee and Lee's would require very coarse discretizations, incurring significant risk of missing minima and other important features of the value function. Concurrently with our work, McCann and Pollard [2007] have integrated a model of user behavior into reinforcement learning, enabling highly responsive realtime character animation.

3 Motion Model

Our approach has two main components: a motion engine blends through captured motion clips to produce realtime human animation, while a control policy determines the best sequence of clips to achieve some multivariate control objective. We begin by describing the motion engine.

Our model generates poses in realtime by blending sequences of prerecorded motion clips. Unlike standard motion graphs [Kovar et al. 2002; Arikan and Forsyth 2002; Lee et al. 2002], we allow transitions between *any* two clips, and our method automatically prevents foot-skate for a large class of motion. Our model assumes that we have captured a set $\mathcal{C}$ of motion clips, where each clip $C \in \mathcal{C}$ consists of a sequence of poses: $C = (\mathbf{p}_1, \ldots, \mathbf{p}_m)$. Each pose $\mathbf{p} \in \mathbb{R}^n$ is a vector specifying all joint positions in a kinematic skeleton. We further assume that each clip C covers a single walk cycle and is divided into two subsequences C_{in} and C_{out}. The subsequences start and end during flight or mid-stance, and each covers one foot plant. We specify one constraint frame in each subsequence that occurs during the middle of the ground contact phase (Figure 2).

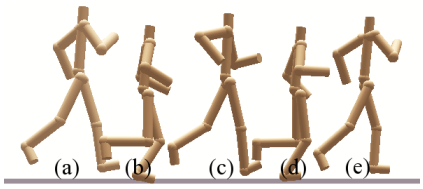

Figure 2: **A clip.** The first subsequence C_{in} runs from (a) to (c), while the second C_{out} comprises (c) through (e). The constraint frames are (b) and (d).

To create longer motions, we partially overlap successive clips and blend between them. The constraint frames allow us to prevent foot-skate during the foot plant without inverse kinematics. Two realizations make this possible. First, when blending from C to C', we may mirror or arbitrarily reorient the root of C' while preserving continuity. Second, kinematic blending is linear in the root of the skeleton (although nonlinear in all other skeletal nodes). Therefore, if we properly orient the foot and "re-root" the skeleton at its foot, we can satisfy a foot position constraint. Blending between clips C and C' is a four step process. First, clip C' is mirrored if necessary to blend along the same foot. Second, we overlap the constraint frames from C_{out} and C'_{in} (Figure 3). Third, clip C' is reoriented so that its ground-contact foot coincides with that of C at the constraint frame. Finally, we blend from C_{out} to C'_{in}, *with the ground contact foot treated as the root of the kinematic skeleton.* By the linearity of blending at the root, if the foot is fixed during interval $[t_a, t_b]$ of C_{out} and during $[t'_a, t'_b]$ of C'_{in}, then there cannot be foot-skate on the interval $[t_a, t_b] \cap [t'_a, t'_b]$ of the blended animation.

In summary, in our model, any subsequence of clips produces a valid animation. Because our control algorithm avoids nonsmooth blends, we do not need an explicit graph structure to ensure good transitions, yet our model's high branching factor allows for quick motion changes. Another advantage of this model is that maximum blending occurs during ground contact when ground forces can account for changes in direction; minimal blending occurs during the flight phase, better preserving the linear and angular momentum of correct motion. The major drawback is that we can enforce only one constraint per blend, thus preventing foot-skate for only one foot at a time. When two feet are on the ground, one foot might slide unless all double-stance clips have the feet spaced at a fixed distance. Nonetheless, we believe our model is highly useful for realtime foot-skate prevention for the wide class of motions which do not have double stance, such as walking and running.

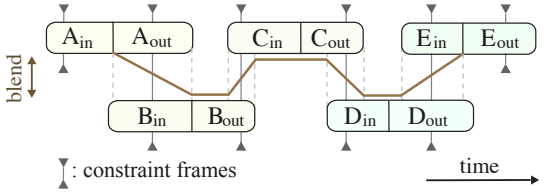

Figure 3: **Clip blending.** We blend the sequence $A,B,C,D,E \subseteq \mathcal{C}$. Constraint frames (shown as solid vertical bars) are aligned across successive clips. Blending occurs where two clips overlap.

4 Control

The motion model described above creates a valid motion for any sequence of clips. The controller must decide *which* sequence best achieves it goals quickly and naturally. Before explaining the control algorithm itself, we describe the state space $\mathcal{S}$ on which the controller is defined.

Since our model blends through sequences of clips, it would seem natural to define the state $X \in \mathcal{S}$ as the current clip. Unfortunately, this representation is insufficient for many tasks. For example, if the character is to walk along a line, the state must keep track of the character's orientation relative to this line. Other tasks may require other state variables. Therefore, although our control formulation is general, the specific state representation depends on our controller's objectives. In our results, we learned four tasks:

- **Navigation.** We divide the clips into several gaits, including standing, walking, and running. The user controls the gait, linear motion path, and torso orientation.
- **Spinning Navigation.** The user controls the motion direction as the character spins in circles (Figure 1, left).
- **Fixed Obstacle Avoidance.** The character follows a line, avoiding a fixed planar obstacle (Figure 1, right).
- **Moving Obstacle Avoidance.** The character follows a direction, avoiding a planar obstacle with linear motion.

Although we learn these objectives independently, we consider them jointly in the state:

$$X = \left(C, x, z, \theta, u, v, \dot{u}, \dot{v}, \hat{G}, \hat{\tau}, \hat{\omega}\right). \tag{1}$$

Here, $C \in \mathcal{C}$ is the current clip, (x,z,θ) is the character's position and orientation on the x-z plane, (u,v) is the relative position of the obstacle, and $(\dot{u},\dot{v})$ is the obstacle's speed (Figure 4). Finally $\hat{G}$, $\hat{\tau}$, and $\hat{\omega}$ are the character's *desired* gait, torso orientation, and spin, respectively. The latter three variables represent intentions, and only change if the user directly edits them (for example, by pushing the button to change the desired gait speed). Nonetheless, it will be useful to consider these part of the state.

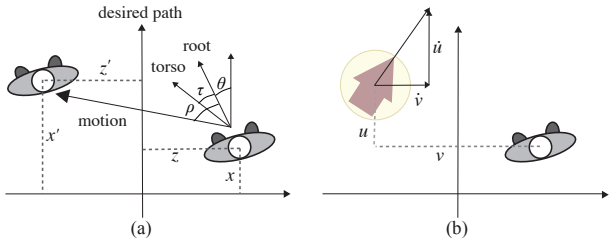

Figure 4: **State Variables.** Root orientation θ is measured relative to the desired path (along the positive x-axis). The torso orientation τ and actual motion direction ρ are measured relative to the root.

For every pair of clips, the blending algorithm (Section 3) determines the length of the blend Δt, as well as the linear and angular change $(\Delta x, \Delta y, \Delta\theta)$ in the character's position. Suppose we transition from clip C to C'. Then the state changes from $X = (C, \cdots)$ to $X' = (C', \cdots)$ according to the *transition function* $f : \mathcal{S} \times \mathcal{C} \to \mathcal{S}$ as follows:

$$f(X,C') = X' = \begin{pmatrix} C' \\ x' \\ z' \\ \theta' \\ u' \\ v' \\ \dot{u}' \\ \dot{v}' \\ \hat{G}' \\ \hat{\tau}' \\ \hat{\omega}' \end{pmatrix} = \begin{pmatrix} C' \\ x + \cos(\theta)\Delta x - \sin(\theta)\Delta z \\ z + \sin(\theta)\Delta x + \cos(\theta)\Delta z \\ \theta + \Delta\theta \\ u + \dot{u}\Delta t - \cos(\theta)\Delta x + \sin(\theta)\Delta z \\ v + \dot{v}\Delta t - \sin(\theta)\Delta x - \cos(\theta)\Delta z \\ \dot{u} \\ \dot{v} \\ \hat{G} \\ \hat{\tau} \\ \hat{\omega} \end{pmatrix}. \tag{2}$$

4.1 Cost

We express goals by assigning a cost to each state and transition

$$\begin{array}{lcl} c_s : \mathcal{S} \to \mathbb{R} & : & \text{State cost} \\ c_t : \mathcal{S} \times \mathcal{S} \to \mathbb{R} & : & \text{Transition cost} \end{array} \tag{3}$$

which are inversely proportional to how well these fulfill the goal. This framework is very broad and can handle a great number of objectives. We now describe the costs used specifically for our four tasks. Again, although we learn these tasks independently (Table 1), we consider them jointly for now.

The gaits partition the clip set $\mathcal{C}$ into different kinds of motion. For the navigation task, we treat the desired gait $\hat{G}$ as a hard constraint so that if $X = (C, \cdots)$ but $C \notin \hat{G}$, then $c_s(X) = \infty$. Otherwise, we ignore the gait and orient the coordinate system so that the desired path coincides with x-axis. We penalize deviation from this path and proximity to the obstacle:

$$c_s(X) = \underbrace{\gamma_x |x|}_{\textit{deviation}} + \underbrace{\gamma_O \exp\left(-(u^2+v^2)\sigma_O^{-2}\right)}_{\textit{obstacle}}. \tag{4}$$

Here, γ_x and γ_O are scaling parameters, and σ_O controls the width of the obstacle. The transition cost c_t ensures smooth character motion in the right direction and with the correct torso orientation and spin:

$$c_t(X,X') = \underbrace{\gamma_\Psi \Psi(C,C')}_{\text{Physics.}} + \underbrace{\gamma_\rho |\theta + \rho|}_{\text{Direction.}} + \underbrace{\gamma_\tau |\tau - \hat{\tau}|}_{\text{Torso.}} + \underbrace{\gamma_\omega |\omega - \hat{\omega}|}_{\text{Spinning.}}. \tag{5}$$

Again, γ_Ψ, γ_ρ, γ_τ, and γ_ω are scaling constants. The physics cost $\Psi : \mathcal{C} \times \mathcal{C} \to \mathbf{R}$ measures the "physical error" in blending from C to C' as

a weighted sum of squared differences of the position and velocity across all joints. The motion direction ρ is measured relative to the the root (Figure 4). Finally, the average of torso direction τ and torso spin ω are compared to the *desired* torso angle and spin, $\hat{\tau}$ and $\hat{\omega}$, respectively.

5 Policies

Having cast our goals in terms of numerical costs, we now look at strategies for achieving these goals. We formalize this idea as a *policy* $\Pi : \mathcal{S} \to \mathcal{C}$, a function that picks the next clip based on the current state. The most obvious policy is greedy:

$$\Pi_{\text{greedy}}(X) = \operatorname*{argmin}_{C' \in \mathcal{C}} \left\{ c_t(X, X') + c_s(X') \right\}, \tag{6}$$

where X' is given by the transition function $X' = f(X, C')$ as in Equation (2). This policy simply chooses the clip that directly minimizes our state and transition costs. In most cases this policy does not produce good results because motion requires *planning*: sacrificing short-term objectives to achieve more efficient results in the long run. For example, when told to turn around, a digital character with planning will step so as to produce a sharper, more realistic turn than under greedy control (Figure 5).

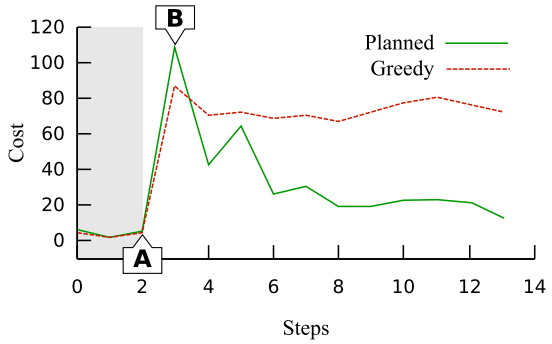

Figure 5: **Planning vs. greedy controllers.** This graph compares the cost $c_s + c_t$ of a greedy navigational controller (Equation (6)) with our planning controller. The characters act almost identically until the user reverses the intended motion direction (**A**). The planning controller then chooses a high cost step (**B**) to enable a sharp and realistic turn one step later.

5.1 The Optimal Policy

Instead of minimizing the immediate cost, we will minimize the entire policy cost, taking into account the future. Suppose policy Π produces the sequence $(X_0, X_1, X_2, \ldots)$ with $X_t = f(X_{t-1}, \Pi(X_{t-1}))$. We measure the cost $c_\Pi(X_0)$ of this sequence as:

$$c_\Pi(X_0) = \sum_{t=0}^{\infty} \alpha^t \left[c_s(X_t) + c_t(X_t, X_{t+1}) \right]. \tag{7}$$

The discount factor $\alpha \in [0, 1)$ accounts for future uncertainty and conveniently also ensures that the series converges. However, the main reason for this functional form is that it can be rewritten recursively:

$$c_\Pi(X_t) = c_s(X_t) + c_t(X_t, X_{t+1}) + \alpha c_\Pi(X_{t+1}). \tag{8}$$

Under general conditions, minimizing Equation (7) for each initial state results in the optimal policy $\Pi_\star$ [Bertsekas 2001]. This allows us to define a *value function* $V : \mathcal{S} \to \mathbb{R}$ that measures the long-term state cost under the optimal policy: $V(X) = c_{\Pi_\star}(X)$. Thus for all X we can rewrite (8) as:

$$V(X) = c_s(X) + c_t(X, X') + \alpha V(X'). \tag{9}$$

where $X' = f(X, C')$ and $C' = \Pi_\star(X)$ is the optimal next clip. An ingenious and well known reversal of logic turns this equation into a formula for the optimal policy. If Equation (9) truly reflects the optimal long-term cost, then the best policy must be to minimize the right hand side of this expression:

$$\Pi_\star(X) = \operatorname*{argmin}_{C \in \mathcal{C}} \left\{ c_t(X, X') + \alpha V(X') \right\}. \tag{10}$$

Thus, the value $V(X)$ completely specifies the optimal controller.

5.2 Near-optimal Policy

We generally cannot compute the exact value for continuous state spaces, and must content ourselves with near-optimal policies based on some approximation of the value function. Sample-based approaches work well, but are efficient neither in learning time nor in controller memory costs. We instead adapt a linear programming approach used primarily in operations research [de Farias 2002]. The idea is to define *basis* $\Phi = [\phi_1, \cdots, \phi_n]$ for the value function where each basis function $\phi_i : \mathcal{S} \to \mathbb{R}$ can be evaluated in closed form, such as polynomials or Gaussians. We then approximate the value function by a linear combination of these basis functions:

$$V \approx r_1 \phi_1 + \cdots r_n \phi_n = \Phi \mathbf{r}. \tag{11}$$

Thus, we have reduced the problem of solving for the complete value function to the much simpler (and lower dimensional) problem of solving for the n-dimensional vector $\mathbf{r}$ to approximate it.

To derive this approximation, we draw draw a set of state samples $\bar{\mathcal{S}} \subset \mathcal{S}$ at which we will evaluate the value function. We also consider a set $\mathcal{L} \subseteq \bar{\mathcal{S}} \times \mathcal{S}$ of state transition pairs, each starting at a sample point. The algorithm starts with $\mathcal{L} \leftarrow \{\}$ and $\mathbf{r} \leftarrow \mathbf{0}$, and iterates towards a better value of $\mathbf{r}$ by alternating between the following two steps until convergence.

1. Compile a set of optimal actions according to the current V from every sample in $\bar{\mathcal{S}}$ and add them as constraints:

   $$\mathcal{L} \leftarrow \mathcal{L} \cup \{(X, X') \mid X \in \bar{\mathcal{S}}\}$$

   where X' is the optimal next state from X given our value function approximation $V \approx \Phi \mathbf{r}$.

2. Find V by solving the linear program

   $$\max_{\mathbf{r}} \sum_{X \in \bar{\mathcal{S}}} V(X)$$
   $$\text{s.t.} \quad V(X) \le c_s(X) + c_t(X, X') + \alpha V(X') \quad \forall (X, X') \in \mathcal{L}.$$

Step 2 essentially inflates the value function as much as possible subject to the bounds. These inequality constraints generalize (9) to the case where the transition $(X, X') \in \mathcal{L}$ may not be optimal. This technique has proved very successful for our purposes. The convex optimization step of the algorithm yields *globally* optimal approximations of the value function, up to the representative power of the basis and constraint sampling. Moreover, we can discard all samples after optimization, retaining only the highly compact basis coefficient vector $\mathbf{r}$.

5.3 Runtime Control

The runtime control algorithm is extremely fast and simple: every time a clip finishes, the controller scans through all clips, picking the minimum value transition as in Equation (10). User inputs such as changing the desired gait, torso orientation, or motion path alter the underlying state variables in Equation (1), and thus affect the next state transition.

5.4 Dimensionality

Despite the advantages of approximate dynamic programming, high dimensional control remains challenging. Basis approximations use far fewer variables than sample-based approaches, but the number of bases must generally still grow exponentially with the size of the state. We mitigate this problem by employing two related but subtly different concepts: *switchability* and *separability*.

Both techniques exploit the property that some quantities remain constant through clip transitions. For example, Equation (2) tells us that the desired gait $\hat{G}$ does not change. Somewhat more subtly, the expression $x+u$ (the distance between the object and the desired path along the x-axis) also remains unchanged, but *only* in the case of a stationary obstacle. Since the controller cannot affect these quantities, we do not need to couple them during optimization. Instead, we tabulate the value function along these dimensions and learn them separately. The difference between switchability and separability is in how we recombine the value function afterwards.

• **Switchability** simply means switching between the tabulated value functions. This is well suited to discrete dimensions such as the desired gait $\hat{G}$. We can even produce near-optimal transitions across value functions that were optimized with different parameters or used different clips. This is because Equation (10) holds even if state X is drawn from a different value function than X'. For example, when transitioning from a walking gait to a running gait, the optimal policy is still to pick the running clip with lowest value. This feature enables us to optimize the parameters of several controllers independently, and automatically combine them with near-optimal transitions.

• **Separability** means blending the value function, producing a continuous value along the separable dimension. For example, we can learn separate value functions for various fixed obstacle locations and blend between these for intermediate obstacle positions. Separability can be interpreted as learning a single value function with linear interpolation (or "hat") bases along the separable dimension. The space of value functions is particularly well suited for blending for the same reason that the value functions can be represented with a small number of bases: the value functions for character animation tend to be smooth.

Switchability and separability do not fundamentally solve the dimensionality problem inherent in reinforcement learning. Nonetheless, they allow for some degree of sub-exponential memory consumption (in serial implementations) or sub-exponential time usage (in parallel implementations). Intrinsically dependent dimensions still require exponential resources, however.

6 Results

To produce our results, we learned four controllers (Section 4) to which we give the following symbolic names: navigation (N), spinning navigation (SN), fixed obstacle avoidance (FOA), and moving obstacle avoidance (MOA).

Table 1 summarizes the controllers. Columns 2-4 indicate the active variables for each task. Variables that are neither switchable nor separable are *dependent* and must be coupled during value function optimization. Note that for FOA, the distance between the the obstacle and the desired motion line $x+u$ is fixed. Therefore, we treat this expression as separable. For moving obstacles, this expression is not fixed and cannot be treated as separable. In fact, to save memory in MOA, we ignore deviation from the desired path and measure only the character's angular direction.

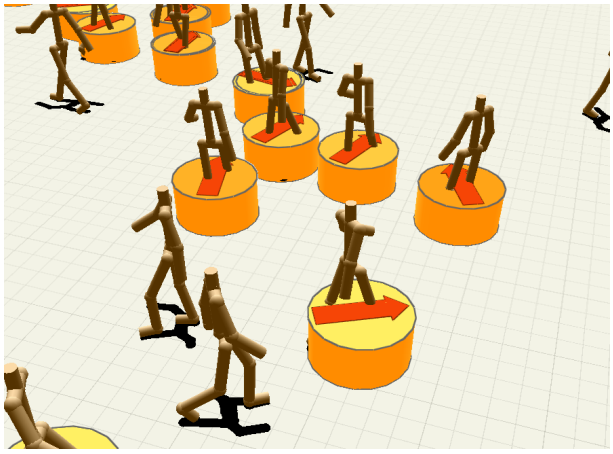

Figure 6: **Moving obstacle avoidance.** Moving obstacle avoidance allows us to model short-horizon crowd dynamics. Each character who is viewed as an obstacle by another is marked by a cylinder. Thus, characters automatically avoid one another.

The current clip C is both discrete and dependent. Therefore, we tabulate the value function along this dimension, but optimize for all clips at once. For our results we used 384 clips, manually partitioned into gaits by speed: 18 standing clips, 87 slow walks, 124 normal walks, 65 fast walks, and 90 runs. In practice, we allowed adjacent gaits to share some clips, smoothing gait transitions. The navigation (N) task used all these clips, SN used a subset of only 60 walking clips, while FOA and MOA used a set of 18 walking clips.

The rest of the dependent dimensions are continuous and we learn continuous functions along these dimensions drawn from one of two families. The first is the polynomials:

$$\mathcal{P}_N(x) \equiv \left\{ f(x) = x^k \;\middle|\; k = 0,\ldots,N \right\}.$$

The second is a family of N Gaussian bumps evenly spaced along the interval $[a,b]$:

$$\mathcal{G}_{a,b,N}(x) \equiv \left\{ f(x) = e^{\frac{-(x-\bar{x})^2}{\sigma^2}} \;\middle|\; \bar{x} = a + i\frac{b-a}{N-1}, i = 0,\ldots,N-1 \right\},$$

where $\sigma^2 = 2$ throughout this paper. Multivariate families of functions are defined through outer products. For example, we denote:

$$\mathcal{Q}(x)\mathcal{R}(y) \equiv \left\{ f(x,y) = q(x)r(y) \;\middle|\; q \in \mathcal{Q},\ r \in \mathcal{R} \right\}.$$

The bases for each task are shown in the last column of Table 1. In FOA for example, we used the basis $\mathcal{P}_4(u)\mathcal{P}_2(v)$ of polynomials (Figure 8, (1)-(15)) for general control, and added a set of Gaussian bases $\mathcal{G}_{0,-\frac{3}{2},4}(u)\mathcal{G}_{x+u,x+u,1}(v)$ (Figure 8, (16)-(18)) to improve the resolution of the value function in the crucial region around the obstacle.

All optimizations were performed with samples uniformly covering each dimension, with between 5 and 8 samples per dimension. For cyclic dimensions, samples divide up the interval [-π, π), while along linear dimensions samples cover a 4 meter interval about the origin. Our sampling is therefore much higher resolution than the value function bases. In fact, we found we could delete up to 50% of the samples at random without appreciable detriment to the controller. Because of our low-dimensional value function representation, learning the controllers was relatively quick: our 2D navigational controllers with several hundred clips never took longer than

Task	Active State Variables			Cost Terms						Continuous Basis Functions
	Switchable	Separable	Dependent	γ_x	γ_O	γ_Ψ	γ_ρ	γ_τ	γ_ω	
N	$\hat{G}$	$\hat{\tau}$	C, x, θ	•	-	•	•	•	-	$\mathcal{P}_2(x)\mathcal{P}_2(\theta)$
SN	-	-	C, θ	-	-	•	•	-	•	$\mathcal{P}_2(\theta)$
FOA	-	$x+u$	C, u, v, θ	•	•	•	•	-	-	$\big(\mathcal{P}_4(u)\mathcal{P}_2(v) \cup \mathcal{G}_{0,\text{-}\frac{3}{2},4}(u)\mathcal{G}_{x+u,x+u,1}(v)\big)\mathcal{P}_2(\theta)$
MOA	-	$\dot{u}, \dot{v}$	C, u, v, θ	-	•	•	•	-	-	$\mathcal{P}_2(\theta) \cup \mathcal{G}_{\text{-}2,2,5}(u)\mathcal{G}_{\text{-}2,2,5}(v)\mathcal{P}_2(\theta)$

Table 1: **Tasks.** Columns 2-4: Active state variables. Columns 5-10: Nonzero cost terms are indicated by bullets. Column 11: Continuous basis functions. (See Section 6.)

45 minutes nor more than 2GB of memory to learn. Our more complicated obstacle avoidance controllers never took longer than one hour to learn, and also required about 2GB of memory. Precomputation time is measured as the longest single process runtime when parallelizing along separable and switchable dimensions.

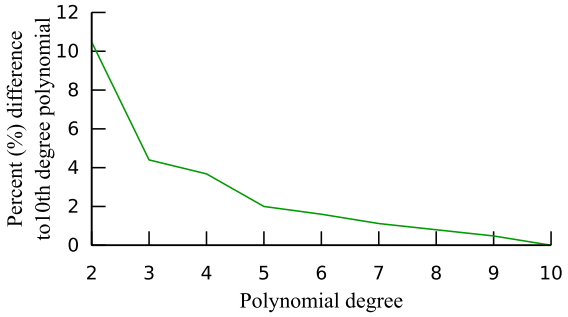

Figure 7: **Value function convergence.** This graph varies the degree d of the root orientation polynomial $\mathcal{P}_d(\theta)$ for a simple navigational controller. Errors are measured relative to the 10th-degree approximation as the sum of squared differences over a set of samples along the θ axis.

Our runtime controller requires 0.101 ms to choose state transitions and can blend poses at 0.04 ms per frame (not including rendering) on a Dual Core 3.5 GHz Intel Xeon processor. Because of our compact basis representation, the runtime memory costs of our method are low. Navigational controllers require less than 13KB of memory while our avoidance controllers require less than 18KB with double precision floating point numbers. In fact, these numbers were dwarfed by the cost of storing the motion clips, which is 23MB for our navigational controller and 1.1MB for our avoidance controller. Note that these costs are amortized across all characters. The per-character memory footprint can be measured in bytes.

Examples of our controllers can be seen in the accompanying video. Compared to greedy methods, our planning algorithm produces better motion with sharper turns, an effect shown in the video and demonstrated numerically in Figure 5. Surprisingly few bases are required to capture these results. We varied the polynomial degree along the θ axis for a simple navigational controller and compared the value at sample points to the 10th-degree approximation. While quadratic polynomials are about 11% different from the 10th-degree value, they nonetheless produce motion visually indistinguishable in quality to higher degrees. Our results also demonstrate successful avoidance of fixed and moving obstacles. The latter allows us to create simple crowd simulations. Specifically, each character fits a moving obstacle model to all neighbors within a threshold distance. The highest value neighbor is then treated as a moving obstacle. This avoidance approach is heuristic and can fail in situations with too many neighbors. However, we found it worked well at medium densities.

7 Conclusion and Future Work

This paper presents a new control model for realtime character animation subject to high dimensional, continuously changing user control. The underlying motion model enables rapid transitions while enforcing foot-skate constraints for a wide class of motion. The control mechanism chooses sequences of clips using a compact, continuous value function representation that admits fast, near optimal control, even in high dimensional spaces.

Our solution by no means solves the fundamental problem of dimensionality in control. However we show that by combining a number of different strategies, we mitigate this problem. Foremost is our use of basis functions to represent the value space. This has several advantages: the basis functions can correlate information between dimensions, reducing the pure dimensional explosion of sampling, while the wide support of each function enables significant reduction in required samples. Also, parallelizing along constant dimensions allows us to further increase the dimensionality of our controllers. This has allowed us to create complex high dimensional controllers in less than one hour of optimization.

Our framework automatically produces near-optimal controllers given a set of objectives. There is no need to explicitly construct walk cycles or otherwise edit the underlying data. Our system is able to blend through ground contact points, preventing foot-skate for single-stance motions in realtime without post-processing such as inverse kinematics. We show that our technique is also expandable, in that we can construct a set of controllers separately for different tasks, then transition among these controllers near optimally, even when they were learned on different underlying sets of motion. This greatly reduces the cost of constructing complex controllers, as each individual task can be optimized separately before connecting the results together. Finally we demonstrate our technique for a number of tasks including navigation with independent control over gait and torso orientation, as well as 2D obstacle avoidance. The latter allows us to create crowd simulations in which each character fits a local linear model to the movement of near characters and then avoids those characters in realtime, all without explicitly defining collision avoidance strategies.

We believe that the major limitation of our technique is that it is too closely tied to blending precaptured motion data, thus requiring large amounts of such data to produce highly varied controllers. For example, our navigational controllers require the motion capture subject to perform a large number of actions: locomoting and turning at a variety of speeds and torso orientations. Ideally, we would like to determine automatically the minimal set of clips that would guarantee good controllers, perhaps computing additional clips via a synthesis process. A related problem is that while our system has a very high branching factor (transitions are possible between any two clips), we are still limited to a discrete number of transitions. This makes it harder to achieve very fine control such as stepping exactly on certain points or walking on a narrow bar. In fact, we found that our system always achieved better motion when given more clips, implying that even with several hundred clips, we had not yet reached diminishing marginal returns. This is a situation where a more general notion of blending, such as “fat graphs” [Shin and Oh 2006] or multi-way blending [Safonova and Hodgins 2007] might prove useful. We note that our foot-skate constraint could be generalized to handle multi-way blends. Also, we expect that blending techniques which always produce physically correct motion [Safonova and Hodgins 2005] would enable more low-cost transitions and therefore better motion with fewer clips.

Another important advance would be to treat the clips as samples

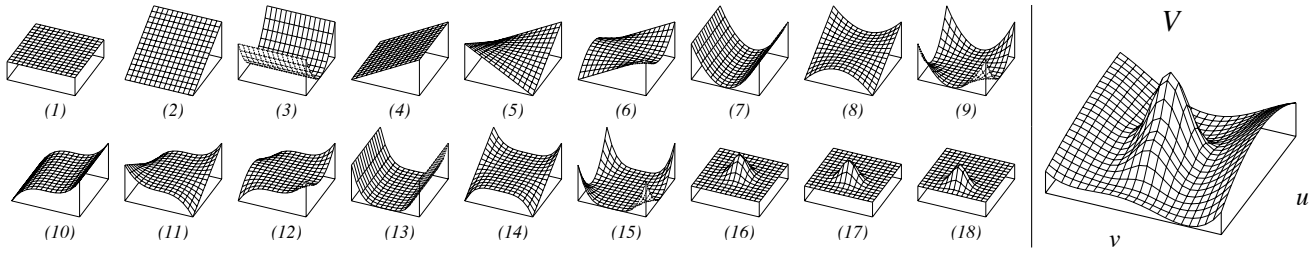

Figure 8: **Bases functions for fixed obstacle avoidance.** (1)-(15) are the polynomial bases for fixed obstacle avoidance (FOA). (16)-(18) are three of the four Gaussian bases. All bases are shown as slices through the u-v plane, holding θ, C, and $x+u$ constant. Our method solves for the linear superposition of basis functions shown on the right.

from a continuous manifold of human motion. This would allow us to define continuous bases along this manifold, further mitigating the curse of dimensionality.

Looking further, the next major advance would be to allow full kinematic motion based on the degrees of freedom in the skeleton. The principle obstacle to this research direction is the vast dimensionality of the human skeleton. In addition, human motion is based on a set of highly interconnected biomechanical systems such as bones, muscles, and tendons, has much to do with robustness and styles that are learned over years of practice. These issues add to the difficulty of the problem, and we are still far from producing motion this way. With this goal in mind, however, we think it is especially important to explore strategies that address the dimensional explosion inherent in motion controllers. We hope that the work presented here will provide a first step in that direction.

Acknowledgments. The authors would like to thank T. Scott Saponas for creating the video, Mira Dontcheva for helping with the figures, and the anonymous reviewers for their helpful comments. This work was supported by the UW Animation Research Labs, NSF grants EIA-0121326, CCR-0092970, IIS-0113007, an Alfred P. Sloan Fellowship, an Intel Fellowship, a Link Fellowship, Electronic Arts, Sony, and Microsoft Research.

References

ARIKAN, O., AND FORSYTH, D. A. 2002. Interactive motion generation from examples. *ACM Transactions on Graphics 21*, 3 (July), 483–490.

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

ATKESON, C. G., MOORE, A. W., AND SCHAAL, S. 1997. Locallyweighted learning for control. *Artificial Intelligence Review 11*, 1-5 (Feb.), 75–113.

BERTSEKAS, D. P. 2001. *Dynamic Programming and Optimal Control*, vol. 2. Athena Scientific.

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

DE FARIAS, D. P. 2002. *The Linear Programming Appraoch to Approximate Dynamic Programming: Theory and Application*. PhD thesis, Stanford.

GLEICHER, M., SHIN, H. J., KOVAR, L., AND JEPSEN, A. 2003. Snap-together motion: Assembling run-time animation. *ACM Transactions on Graphics 22*, 3 (July), 702–702.

IKEMOTO, L., ARIKAN, O., AND FORSYTH, D. 2005. Learning to move autonomously in a hostile environment. Tech. Rep. UCB/CSD-5-1395, June.

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

KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. *ACM Transactions on Graphics 21*, 3 (July), 473–482.

KWON, T., AND SHIN, S. Y. 2005. Motion modeling for on-line locomotion synthesis. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 29–38.

LAU, M., AND KUFFNER, J. J. 2006. Precomputed search trees: Planning for interactive goal-driven animation. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 299–308.

LAVALLE, S. M. 2006. *Planning Algorithms*. Cambridge University Press.

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

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 Transactions on Graphics 21*, 3 (July), 491–500.

LEE, K. H., CHOI, M. G., AND LEE, J. 2006. Motion patches: building blocks for virtual environments annotated with motion data. *ACM Transactions on Graphics 25*, 3 (July), 898–906.

MCCANN, J., AND POLLARD, N. 2007. Responsive characters from motion fragments. *ACM Transactions on Graphics (SIGGRAPH 2007) 26*, 3 (July). *To appear.*

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

PULLEN, K., AND BREGLER, C. 2002. Motion capture assisted animation: Texturing and synthesis. *ACM Transactions on Graphics 21*, 3 (July), 501–508.

SAFONOVA, A., AND HODGINS, J. K. 2005. Analyzing the physical correctness of interpolated human motion. In *In Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 171–180.

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

SCHÖDL, A., AND ESSA, I. 2001. Machine learning for video-based rendering. *Advances in Neural Information Processing Systems 13* (July), 1002–1008.

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

SINGH, S. P., AND YEE, R. C. 1994. An upper bound on the loss from approximate optimal-value functions. *Machine Learning 16*, 4 (September), 227–233.

TRICK, M. A., AND ZIN, S. E. 1997. Spline approximations to value functions: A linear programming approach. *Macroeconomic Dynamics 1*, 255277.