EUROGRAPHICS 2013 / I. Navazo, P. Poulin
(Guest Editors)

Volume 32 (2013), Number 2

Animal Locomotion Controllers From Scratch

Kevin Wampler[1,2] Jovan Popović[2] Zoran Popović[1]

[1]University of Washington
[2]Adobe Research

Abstract

There exists a large body of research devoted to creating real time interactive locomotion controllers which are targeted at some specific class of character, most often humanoid bipeds. Relatively little work, however, has been done on approaches which are applicable to creatures with a wide range of different forms – partially due to the lack of easily obtainable motion-capture data for animals and fictional creatures. We show how a locomotion controller can be created despite this dearth of data by synthesizing it from scratch. Our method only requires as input a description of the shape of the animal, the gaits which it can perform, and a model of the task or tasks for which the controller will be used. From this a sequence of motion clips are automatically synthesized and assembled into a motion graph which defines a physically realistic controller capable of performing the specified tasks. The method attempts to minimize the computational time required to generate this controller and we show its effectiveness at creating interactive controllers for a range of tasks for bipeds, tripeds, and quadrupeds.

1. Introduction

With the emergence of motion capture and the data-driven methods for interactive controller construction, humanoid motion has enjoyed a considerable body of research. Techniques have been developed which allow motion capture data to be assembled into graph-like structures that support real time interactive control. More recent advances make use of reinforcement learning to automate the logic of determining which motions to use to perform a task, and recent years have seen large strides in controllers which can be run directly inside of a physics simulator. Unfortunately, current techniques for character animation either require a database of pre-captured motion from which to construct the controller, are limited to humanoid bipeds, or require the effort of an expert designer in order to construct the controller.

Our controller generation technique takes as input a description of the shape of an animal and a model of the tasks for which the controller will be used. From this, we automatically build a graph of interconnected motions such that by traversing different paths through this graph the required tasks can be performed. For instance, a controller designed to follow a given direction would consist of various connected walks and turns. This technique works for a wide range of different animal types without any motion data or manual tuning being required.

Since we do not assume access to any pre-specified information on how the animal moves, there are several challenges which must be overcome to accomplish this. Firstly, not all combinations of motions into a graph will create controllers which perform a task equally well. Indeed, it appears that even that task of selecting a set of motions from an existing database to create a controller is difficult to achieve [LLP09]. On the other hand, we must also contend with the difficulty of actually creating the individual motions from which the graph is built. This is particularly difficult because new motions are most easily created by modifying similar existing motions, but controllers are often best constructed from a highly dissimilar set of motions. Thus the choices needed to quickly build highly effective controllers must be balanced against possible difficulties in actually creating these motions.

We approach this problem by synthesizing both the individual motions used in a motion graph as well as their connectivity with a technique combining aspects of spacetime constraints with ideas from optimal control. This approach considers not only how effective the resulting graph will be at performing a set of user-specified tasks, but also an estimate of the difficulty in solving for the individual motions from which it is constructed. Furthermore, as new motions are added they are used to improve the ability to solve for

© 2013 The Author(s)
Computer Graphics Forum © 2013 The Eurographics Association and Blackwell Publishing Ltd. Published by Blackwell Publishing, 9600 Garsington Road, Oxford OX4 2DQ, UK and 350 Main Street, Malden, MA 02148, USA.

future motions, allowing an effective controller to be incrementally constructed.

2. Related Work

Locomotion behaviors are challenging to create because they must both nimbly respond to varying tasks and faithfully synthesize motions that accomplish them. One way to proceed is to rely on motion data [BW95, WP95], motion graphs [LCR*02, AF02, KGP02], or other parametric structures derived from motion data [SO06, HG07, KG04]. A recent research trend is use reinforcement learning to automate the runtime task of selecting which data clips to play in what order [LL04, LK05, TLP07, MP07, LZ08], or how to navigate in a continuous abstract motion space [LWB*10, LWH*12] in order to comply with user-specified tasks. These methods explain how to compose responses by flexibly following precaptured motion sequences, but they do not establish methods for constructing such motions for arbitrary characters.

Because of their predictability and generally speedy runtime performance, these kinematic behaviors are routinely employed in computer games. Although most of these approaches are limited to directly playing back pre-recorded data, one notable exception to this is given in [HRE*08]. In this approach an artist designs motions using an interface that explicitly aids in making these motions retargetable to a wide range of different creatures. Although both this and our approach can construct controllers for many different creature types, they differ in two primary ways. Firstly, although the controllers generated with our approach are kinematic, any motions generated by these controllers are physically accurate. Secondly, we do not rely on a database of artist-generated motions but instead create the motions directly from the shape of the animal.

Other approaches [FvdPT01, SKL07, CBvdP09, WP10, MdLH10, WHDK12] leverage rapid advances in the design of control strategies for walking, running, and other motion primitives [RH91, HWBO95, HP97, WH00, CBYvdP08, CBvdP10, dLMH10]. Recent advances have also enabled dynamic quadruped controllers [CKJ*11] with a wide range of simulated gaits and skills. Although the variety and quality of the motions generate by this technique are impressive, it is not a general-purpose controller synthesizer that can be applied to an arbitrary animal since, aside from being limited to quadrupeds (demonstrated on a dog), constructing the controllers requires either motion capture data or a careful tuning of parameters. Our approach differs from existing ones in that it requires no data or manual tuning, yet can be applied to a wide range of different animals.

Trajectory synthesis with spacetime constraints offers another promising direction. For our task, methods that construct motions from constraints alone are more relevant [WK88, FP03, MTP12] than methods that transform motion data [PW99, LHP05, RGBC96, SHP04, LYvdP*10]. In particular, the methods for synthesis of animal gaits [WP09, NCNV*12] and motion transitions [CRS10] hint at the possibilities for automatic computation of locomotion behaviors. In this paper, we show how to go beyond the computation of steady-state animal gaits by synthesizing motions with turning and varying gait transitions. We then combine these optimizations with an outer loop that builds and chooses which motions to generate in order to build a locomotion behavior.

3. Building Locomotion Controllers

Basic Definitions The input to our method is given as an animal, specified as a kinematic tree of *limbs* connected by *joints*. Each joint defines a parameterized transformation from its parent limb to its child limb, and there is an additional 'joint' specifying the global position and orientation of the character's root. A concrete pose for the animal is specified by a vector of *degrees of freedom* (DOFs) giving the parameters of each joint. A sequence of such vectors, along with information about when each foot is in contact with the ground forms a *motion clip*, denoted $\mathbf{M}$. We refer to an individual frame f_i in a motion by $\mathbf{M}(f_i)$. When we wish to index into a motion by a continuous time parameter rather than at a discrete frame, we instead use the notation $\mathbf{M}(t)$, which is linearly interpolated from the pair of frames surrounding t. We denote a *motion graph* with $\mathbf{G}$. The nodes in this graph are equivalent to frames in a motion, and we let $\mathbf{G}(f_i)$ refer to such a node. The edges in a graph represent frames which can be transitioned between, and also store the relative 2D transformation which the root of the character undergoes during the transition. Throughout our controller construction process we will also find it useful to create approximations of various quantities. Such approximations are signified by a tilde.

3.1. Controller Optimality

Our approach attempts to create a controller which is optimal with respect to a combination of energetic efficiency and the ability to adeptly adeptly satisfy the user's commands. This controller is constructed from a set of *motions*, $\mathbf{M}_1, \ldots, \mathbf{M}_n$ interconnected into a single graph $\mathbf{G}$. Each branch in this graph corresponds to a point where the character has a choice in its future motion, the goal being to pick the motions which best comply with the user's commands.

Because this approach solves for both the individual motions in the controller and the global structure of the motion graph build from these motions, it is necessary to have a definition of what an 'optimal motion graph' is which encompasses both of these aspects. To this end, we define an objective function which combines features from both spacetime constraints and optimal-control approaches to animation. First we will describe an optimization formulation which allows for the synthesis of individual motions, after which we will

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.

show how this formulation can be extended to the optimization of an entire motion graph.

Motion Optimality It is useful to begin by considering how to generate single motions offline. One technique for achieving this which has been successful in generating motions for a wide range of animals is the method of spacetime constraints [WK88, WP09]. This approach specifies a motion as the minimum of a large constrained optimization problem. We restrict our attention to the most common formulation, which represents the objective function as a sum of per-frame costs and specifies the constraint functions independently for each frame. Assuming that the desired motion consists of n different frames, this has the form:

$$\begin{aligned} \mathbf{M} = \operatorname{argmin} & \frac{1}{n}\sum_i cost(\mathbf{M}(f_i)) \\ \text{s.t. } & g_j(\mathbf{M}(f_i)) = 0 \qquad \forall j,i \\ & h_k(\mathbf{M}(f_i)) \leq 0 \qquad \forall k,i \end{aligned} \tag{1}$$

Typically $cost(\mathbf{M}(f_i))$ is taken to be the sum of the joint torques or muscle forces exerted by the character at frame f_i and the constraint functions g_j and h_k enforce that the resulting motion satisfy the laws of physics.

Although this formulation can work well for synthesizing single motions, it does not provide any means by which user commands can be incorporated. To this end, we first define a vector of *task parameters* $\theta(f_i)$ for each frame $\mathbf{M}(f_i)$. The definition of the task parameters is task specific, and provides the means by which the user can control the character. For example, in a controller which allows the character to be steered to walk in a desired direction, the task parameters would simply include the character's desired heading. We then alter the objective function used by spacetime constraints to penalize deviations from performing the desired task at each frame:

$$cost(\mathbf{M}(f_i),\theta(f_i)) = cost_b(\mathbf{M}(f_i)) + cost_t(\mathbf{M}(f_i),\theta(f_i)) \tag{2}$$

Here $cost_b(\mathbf{M}(f_i))$ is the *biomechanical cost* inherent in the character's motion at frame $\mathbf{M}(f_i)$, and $cost_t(\mathbf{M}(f_i),\theta(f_i))$ is the *task cost* penalizing frames in which the character's state $\mathbf{M}(f_i)$ is not in accordance with the task parameters $\theta(f_i)$. Details of the specific task parameters we use are given in section 4.

Motion Graph Optimality In order to measure how effective a given motion graph will be as a locomotion controller, the preceding definition of the optimality of single motions must be generalized in two ways. Firstly it must allow for a complex connectivity of multiple motions into a larger graph, and secondly it must account for the fact that the task parameters are not known in advance, but will rather be dictated at runtime by a user. These requirements naturally lead to a definition of optimality based on the formulation of a controller as a Markov decision process (MDP). This MDP consists of a *state space* defined by the set of tuples $s = (f, r, \theta)$ where f is a frame in $\mathbf{G}$, r is a matrix giving the global 2D position of the character, and θ is some setting of the problem's task parameters.

Since a user controls a character by changing the task parameters as the character moves, we no longer restrict ourselves to a fixed sequence of task parameters, but instead specify a dynamical model by which they are expected to change over time. This model specifies the expected behavior of the user as they control the character. Although in principle this task model could take any form, in our implementation we only use Markovian models which only specify a probability $\mathrm{P}[\theta(f_{t+1})|\mathbf{M}(f_t),\theta(f_t)]$ of the user selecting task parameters $\theta(f_{t+1})$ for the next frame if the task parameters in the current frame are $\theta(f_t)$ and the character's current state is $\mathbf{M}(f_t)$. Although this is a restricted model of a user's expected behavior, we have found it to be sufficient for controller synthesis, and it can also be easily learned from examples of real user behavior [MP07].

The more complex connectivity between the motions in a graph is accounted for with a *transition function*. This function defines how the character can move from one point in state space $s = (f, r, \theta)$ to another $s' = (f', r', \theta')$, done by changing f to f', altering r by the transformation associated with the edge in $\mathbf{G}$ between f and f', and stochastically updating θ according to $\mathrm{P}\left[\theta'|\mathbf{M}(f),\theta\right]$. Finally, we define a *cost function* giving the cost incurred by transitioning from s to s' by $cost(s,s') = cost(\mathbf{M}(f),\theta)$. The goal of such a MDP controller is to pick which edges in $\mathbf{G}$ to follow so as to minimize the cost of the corresponding state transitions, given the current values of the task parameters.

Because we do not know the sequence of task parameters a user will choose in advance, we cannot use the objective function in equation 1 directly to evaluate the cost of using a controller built from a particular graph. We therefore alter it so that instead of being defined by the average per-frame cost of a motion, it represents the expected value of the average per-frame cost incurred while using the controller. This expected cost depends on the *policy* π which dictates which edge the $\mathbf{G}$ should be followed from each possible combination of f and θ. Evaluating a policy at a point in state space yields a new point in state space $s' = \pi(s)$. We make the standard choice that this policy be required to be an *optimal policy* π^* which minimizes the associated expected cost over the space of all possible policies.

In order to numerically calculate the expected average performance of a controller we use reinforcement learning [SB98]. This is done with a two-step process: First we calculate an optimal policy π^* which determines which edge should be taken in the graph for each possible combination of node and task parameters. This can be efficiently computed by calculating a *value function* over the combined space of graph nodes and task parameters. Since we formulate our controller as an MDP, there are a num-

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.

ber of methods which can be automatically applied in order to compute such a value function. Our implementation uses *fitted value iteration* [Gor95], but we also refer the reader to any of a number of character animation papers [LL04, LK05, TLP07, MP07, LZ08] which employ alternative approaches to learning the value function.

Once the optimal policy π^* has been determined, we need to compute an *influence function* which gives the probability with which a character moving according to π^* will be found in any particular state, taken in the limit as the controller is run for an infinite amount of time. Intuitively, this influence function captures the notion that some motions may be used more often than others, and thus costs for these motions should likewise matter more. This approach has proven successful in creating compact controllers from a database of pre-captured motions [LLP09].

Rather than explicitly constructing an approximation to the influence function, we generate a number of samples $s_1, \ldots, s_n$ of according to its distribution. We initially distribute $s_1, \ldots, s_n$ evenly over the state space, and then repeatedly advect these points according to π^*. This is done by, at each point and iteration, using π^* to determine which edge in $\mathbf{G}$ to follow and then following the MDPs transition function for this edge to arrive at a new point in state space. After a sufficient number of iterations the state points accurately sample from the influence function. We can then calculate the expected value of the average per-frame cost of using the controller with the given task model as:

$$cost(\mathbf{G}) = \frac{1}{n} \sum_{s_i} cost(s_i, \pi^*(s_i)) \tag{3}$$

We note that when solving a spacetime constraints problem to generate a motion these samples only alter the objective function in equation 1, leaving the constraint functions unaltered. This means that this sampling procedure does not impact the physical validity of the generated motions.

So far we have only accounted for how the objective function in equation 1 generalizes from motion synthesis to controller synthesis. It is still necessary to address the constraint functions. Fortunately the answer here is simple. Each motion $\mathbf{M}$ of which $\mathbf{G}$ is comprised is a minimum solution to equation 1, and thus satisfies the constraint functions. It merely remains to ensure that the constraint functions are also satisfied at the junctions between motions. It is sufficient to simply require that when adding a new motion $\mathbf{M}$ to $\mathbf{G}$, the first and last pairs of frames in $\mathbf{M}$ each correspond to a pair of consecutive frames in $\mathbf{G}$. This ensures that the motions seamlessly stitch together and that all animations generated by a controller using $\mathbf{G}$ will be physically valid.

3.2. Controller Generation

Equation 3 allows us to judge the merits of one motion graph versus another, so it remains to actually construct a graph which minimizes this cost. We construct such a graph incrementally starting from an initial graph $\mathbf{G}_0$ consisting of a single walk cycle as generated by [WP09]. Each $\mathbf{G}_{i+1}$ is then created by generating a new motion $\mathbf{M}_{i+1}$ and adding it to $\mathbf{G}_i$. At a high-level, this process uses the following steps:

1. Search over all candidate optimizations to generate $\mathbf{M}_{i+1}$. For each candidate:
   a. Create $\widetilde{\mathbf{M}}$ as an efficient-to-compute approximation of $\mathbf{M}_{i+1}$. This is used in the next two steps.
   b. Determine c, an estimate of $cost(\mathbf{G}_{i+1})$ assuming the optimization used to generate $\mathbf{M}_{i+1}$ succeeds.
   c. Determine p, an estimate of the probability that the optimization used to generate $\mathbf{M}_{i+1}$ will succeed.
   d. Compute the expected cost of $\mathbf{G}_{i+1}$ after attempting to solve for $\mathbf{M}_{i+1}$ as $(1-p) \cdot cost(\mathbf{G}_i) + p \cdot c$.
2. Perform a spacetime constraints optimization to solve for the $\mathbf{M}_{i+1}$ for which this expected cost is minimized. If the optimization succeeds, add $\mathbf{M}_{i+1}$ to $\mathbf{G}_i$.
3. Repeat until termination

A more detailed pseudocode overview our algorithm is given in figure 1.

There are several sub-problems which must be addressed in this method.

- A method allowing a wide range of candidate $\mathbf{M}_{i+1}$ motions suitable for augmenting $\mathbf{G}_i$ to be generated.
- A technique to generate $\widetilde{\mathbf{M}}$, which serves as an efficient-to-compute guess at $\mathbf{M}_{i+1}$.
- A way of quickly estimating c and p in the above outline.
- A an algorithm for searching over the space of possible new motions to determine the single next motion to solve in full and add to $\mathbf{G}_i$.

These sub-problems will be addressed in turn.

Motion Optimization We generate each motion $\mathbf{M}_i$ by starting with a 'guess' initialization motion $\widetilde{\mathbf{M}}_i$ and use this as the initialization for a spacetime constraints optimization. Taking $\widetilde{\mathbf{M}}_i$ for granted for the moment, we solve for $\mathbf{M}_i$ with a spacetime constraints formulation is based on the method of [WP09]. Since the approach of [WP09] can only generate cyclic gaits, we extend it to support the creation of a wider range of motions by exposing parameters of the optimization and then searching for the values of these parameters which are expected to generate the most useful new motion.

Each new motion $\mathbf{M}_{i+1}$ is used to augment the graph $\mathbf{G}_i$. Although our algorithm's structure supports any number of ways of augmenting a graph, we use two methods for achieving this: Add a new motion providing a transition between

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.

```
1: G_0 ← single walk cycle generated using [WP09]
2: while not done do
3:     minCost ← ∞
4:     S*, σ* ← none, none
5:     for all S_j do                          ▷ find next optimization to perform
6:         search σ_k
7:             M̃ ← initialization motion for S_j, σ_k
8:             G̃ ← G_i ∪ M̃
9:             p ← P̃(G̃, σ_k)
10:            c ← (1 − p) · cost(G_i) + p · cost̃(G̃, σ_k)
11:            if c < minCost then
12:                minCost ← c
13:                S*, σ* ← S_j, σ_k
14:            end if
15:        end search
16:    end for
17:    M̃_{i+1} ← initialization motion for S*, σ*
18:    M_{i+1} ← S*(M̃_{i+1}, σ*)               ▷ perform optimization
19:    G_{i+1} ← G_i ∪ M_{i+1}
20: end while
```

Figure 1: *Pseudocode for our controller generation algorithm. S and σ represent an optimization template and its associated parameters respectively. The variable p stores the probability with which an optimization is expected to converge to a physically realistic result, and c stores the expected value of the cost of the new graph created by performing an optimization to create a motion augmenting $\mathbf{G}_i$ (taking into account that this optimization will only succeed with probability p).*

two existing states, and create a new cyclic motion and connect it to $\mathbf{G}_i$. To represent each of these two possible modifications we define a pair of *optimization templates*, denoted S_1, S_2 (figure 2). Each optimization template further exposes a space of parameters such that any setting of these parameters for S_j, denoted σ_j, yields a concrete optimization problem which may be solved to generate a new motion $\mathbf{M}' = S_j(\widetilde{\mathbf{M}}, \sigma_j)$.

Which parameters are exposed to be varied and which are fixed depends on the nature of the controller being generated, but we have found the following to be useful for creating our locomotion controllers:

parameter	symbol	connecting	cyclic
start node	f_a	✓	✓
end node	f_b	✓	✓
turn angle	θ	✓	
strafe angle	ϕ		✓
speed	s		✓
gait type	g		✓

The parameters f_a, f_b determine where $\mathbf{M}_{i+1}$ connects from and to $\mathbf{G}_i$, θ gives the angle by which the character should turn during a transition motion, and ϕ, s, and g give the strafing angle, speed, and gait type of the cycle portion of $\mathbf{M}_{i+1}$. Each such possible gait g is specified by a set C of *contact intervals* $(foot_1, start_1, end_1), \cdots, (foot_n, start_n, end_n)$, specifying that $foot_j$ is in contact with the ground from time $start_j$ to end_i. We have found that specifying these contact intervals is relatively easy and generally takes only a few minutes. Further details on the formulation of the spacetime constraints optimization are given in the appendix.

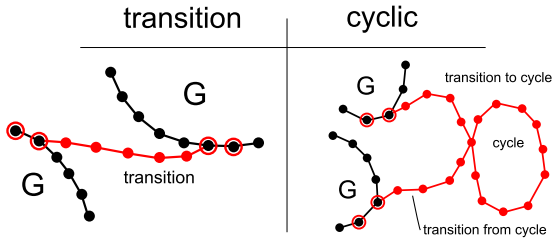

Figure 2: *The two optimization templates we employ. The transition template adds a transition between two existing frames in $\mathbf{G}$. The cyclic template creates a new cyclic motion $\mathbf{M}_c$, then creates a transition motion from $\mathbf{G}$ to $\mathbf{M}_c$ and from $\mathbf{M}_c$ to $\mathbf{G}$. Note that the first and last pair of frames in each transition motion must match existing frames in $\mathbf{G}$.*

Motion Initialization Unfortunately, spacetime constraints optimizations such as the ones we employ are notoriously sensitive to their initialization, $\widetilde{\mathbf{M}}_{i+1}$. Depending on the quality of its initialization an optimization may converge to a good result, converge to a poor local minimum (which is physically valid but energetically inefficient), or fail to converge to a physically valid motion at all. Since we do not assume any example data from which a good initialization might be formed this presents a difficulty. The key is noting that $\mathbf{G}_i$ is constructed entirely of physically valid motions seamlessly stitched together, and thus $\mathbf{G}_i$ in some sense represents a database of 'good' motions which we can use to create $\widetilde{\mathbf{M}}_{i+1}$.

Since there are in general many possible initialization motions that can be generated from $\mathbf{G}_i$, we select the single motion which minimizes a a *quality function* $q(\widetilde{\mathbf{M}})$. The function associates each candidate motion with a real number representing how good an initialization it provides. A quality of 0 means that the optimization is virtually guaranteed to converge to a good result, while motions with large quality measures are likely to be more problematic. We have found that a simple choice for q works well – the sum of the distance between each frame in $\widetilde{\mathbf{M}}$ and the nearest frame in $\mathbf{G}_i$:

$$q(\widetilde{\mathbf{M}}) = \sum_j \min_k d(\widetilde{\mathbf{M}}(f_j), \mathbf{G}_i(f_k)) \tag{4}$$

Here $d(\mathbf{M}(f_i), \mathbf{G}(f_j))$ is defined according to the metric used in [KGP02].

For each potential optimization, defined by an optimization template S_j and setting of its parameters σ_j, we search

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.

through $\mathbf{G}_i$ to find a motion which best serves as an initialization for an optimization with parameters in σ_j. The end result will be a motion which correctly satisfies the f_a, f_b, θ, and g parameters in σ_j, but which may not satisfy ϕ or s nor be physically realistic until the final spacetime constraints optimization is solved. This search for the best initialization motion in $\mathbf{G}$ is performed slightly differently depending on if we are creating a cyclic or a transition motion.

For cyclic motions we search over all cycles in $\mathbf{G}_i$ up to 1.5 seconds in length. Each such cycle defines a motion $\mathbf{M}_c$, but the timing of this motion may not match the that of the motion to be generated. This is the case when S_i and σ_i specify a set of foot contact intervals C which the resulting gait should follow. We address this by first discarding any motions in which a foot has a different number of contact intervals than is specified by S_i and σ_i. For each remaining motion, we warp $\mathbf{M}_c$ so that its foot contact timings match those in C using the approach described in appendix B. Finally, in the case that S_i has a parameter corresponding to the character's velocity, we alter the motion of the root of the character in $\widetilde{\mathbf{M}}_i$ so that it achieves the desired velocity.

For transition motions we perform a similar operation. For a transition motion starting at graph node f_a and ending at f_b, we begin by searching over all pairs of paths in $\mathbf{G}_i$, one starting from f_a (and ending anywhere) and the other ending at f_b (but starting anywhere). For each pair of paths we compute a linear blend from the path starting at f_a into the path ending at f_b, and choose the pair for which the result scores the best according to equation 4. In the case that S_i parameterizes over a turn angle θ, we circularly warp the resulting clip so that it achieves the desired turn. We then calculate $\widetilde{\mathbf{M}}_{i+1}$ by performing a more expensive but higher quality blend of these two paths using a registration curve [KG03].

Graph Construction We create a controller incrementally by augmenting a motion graph $\mathbf{G}_i$ with new motions generated from a spacetime constraints optimization. This requires determining which motion to add at each iteration. Since the goal is to arrive at a controller which minimizes equation 3, in an ideal world we could search directly over the parameter spaces for each optimization template for the motion when when added to $\mathbf{G}_i$ gives the lowest value of $cost(\mathbf{G}_{i+1})$. Unfortunately correctly evaluating $cost(\mathbf{G}_{i+1})$ requires $\mathbf{M}_{i+1}$ which involves solving a spacetime constraints optimization – an extremely expensive computation and one which may often fail to converge to physically valid solution at all. This makes it infeasible to use $cost(\mathbf{G}_{i+1})$ as the objective function for determining how $\mathbf{G}_i$ should be augmented.

Since actually calculating $cost(\mathbf{G}_{i+1})$ is expensive, we seek to efficiently approximate it with a pair of *surrogate functions*: $\widetilde{cost}$ and $\widetilde{P}$. The function $\widetilde{cost}(\mathbf{G}_i, \sigma_{i+1})$ estimates $cost(\mathbf{G}_{i+1})$ under the assumption that the optimization $S_{i+1}(\widetilde{\mathbf{M}}_{i+1}, \sigma_{i+1})$ is used to create $\mathbf{M}_{i+1}$ and that this optimization succeeds. $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1})$ gives an estimate of the probability with which this optimization can be expected to succeed. We then search for the optimization parameters σ^*_{i+1} which maximize the expected value of the decrease in the cost of $\mathbf{G}_{i+1}$:

$$(1-p)\cdot cost(\mathbf{G}_i) + p\cdot \widetilde{cost}(\mathbf{G}_i, \sigma^*_{i+1}) \tag{5}$$

where $p = \widetilde{P}(\mathbf{G}_i, \sigma^*_{i+1})$. Then only solve a single spacetime constraints problem $S_{i+1}(\widetilde{\mathbf{M}}_{i+1}, \sigma^*_{i+1})$ is solved to create $\mathbf{M}_{i+1}$ and thereby $\mathbf{G}_{i+1}$.

We define $\widetilde{cost}(\mathbf{G}_i, \sigma_{i+1})$ by simply using $\widetilde{\mathbf{M}}_{i+1}$ as a stand-in for $\mathbf{M}_{i+1}$; creating $\widetilde{\mathbf{G}}_{i+1}$ by adding $\widetilde{\mathbf{M}}_{i+1}$ to $\mathbf{G}_i$ and defining $\widetilde{cost}(\mathbf{G}_i, \sigma_{i+1}) = cost(\widetilde{\mathbf{G}}_{i+1})$. This avoids performing a spacetime constraints optimization, but has the caveat that it still requires estimates for the biomechanical costs in equation 2 associated with each frame in $\widetilde{\mathbf{M}}_{i+1}$. We do this by interpolating the costs associated with each frame in $\mathbf{G}_i$ using:

$$cost_b(\widetilde{\mathbf{G}}_{i+1}(f_k)) \frac{\sum_j cost_b(\mathbf{G}_i(f_j)) d(\widetilde{\mathbf{G}}_{i+1}(f_k), \mathbf{G}_i(f_j))^{-2}}{\sum_j d(\widetilde{\mathbf{G}}_{i+1}(f_k), \mathbf{G}_i(f_j))^{-2}}$$

Where $d(\mathbf{M}(f_i)', \mathbf{M}(f_j))$ is measured according to the metric in [KGP02].

The definition of $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1})$ is necessarily more *ad hoc* as the probability that a given optimization will converge depends in complicated ways on the specifics of the optimization and the particular nonlinear optimizer used. Nevertheless, we expect that optimizations for which $q(\widetilde{\mathbf{M}}_{i+1}) \approx 0$ will be expected to succeed, as well optimizations for which σ_{i+1} is similar to an optimization which has previously succeeded. Similarly, optimizations for which σ_{i+1} is similar to those for a previously failed optimization will be likewise expected to fail, although this probability can be reduced by a new higher quality initialization. We believe that there are many possible functions satisfying these properties which would serve as a suitable definition for $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1})$, but the particular formulation use is given in appendix C.

Parameter Search Having defined $\widetilde{cost}(\mathbf{G}_i, \sigma_{i+1})$ and $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1})$, it only remains to solve for the optimization parameters σ^*_{i+1} which minimize equation 5. Although in principle any optimization technique would suffice for this task, we note that although $\widetilde{cost}(\mathbf{G}_i, \sigma_{i+1})$ and $\widetilde{P}(\mathbf{G}_i, \sigma_{i+1})$ are significantly cheaper to compute than actually performing a spacetime constraints optimization, they are still relatively expensive (on the order of 0.2 to a few seconds). We therefore choose an approach which attempts reduce how many evaluations of these surrogate functions are required.

Because the the parameter spaces for each optimization template are low-dimensional, we employ a method which utilizes all previous evaluations of equation 5 for the current $\mathbf{G}_i$ iteration. Specifically, given the previous evaluations of equation 5 $(\sigma_1, cost_1), \ldots, (\sigma_n, cost_n)$ such that $cost_j$ is the

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.

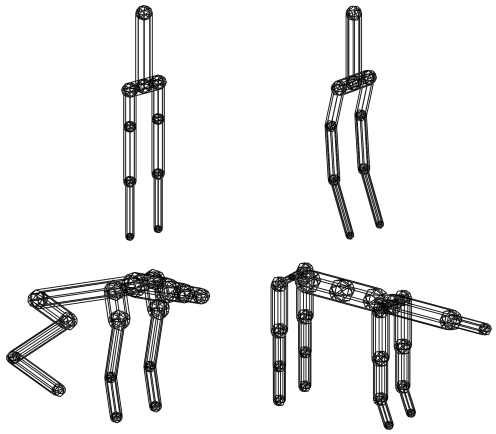

Figure 3: *The skeletal structured for two different bipeds, a triped, and a quadruped for which we synthesize controllers.*

evaluation of equation 5 for the parameters σ_j, we choose the next parameters σ_{n+1} to sample as those which minimize $\sum_j \frac{w_j\, cost_j}{w_j+1}$ where $w_j = d(\sigma_{n+1}, \sigma_j)^{-2}$. This equation has the property that it is exactly equal to $cost_j$ when $\sigma_{n+1} = \sigma_j$, and generally decreases elsewhere, leading to a global exploration of the optimization parameter space.

Once this search process converges, we have a set of optimization parameters σ^*_{i+1} which minimize equation 5, so we can construct a graph by using σ^*_{i+1} to create $\mathbf{M}_{i+1}$ and thereby $\mathbf{G}_{i+1}$, repeating the process until either a given size of $\mathbf{G}$ is attained or when the decrease of $cost(\mathbf{G}_{i+1})$ from $cost(\mathbf{G}_i)$ falls below some threshold. Pseudocode for this process appears in lines 4-15 of figure 1. The final motion graph resulting from this optimization defines a controller which can animate the input animal efficiently performing the given tasks.

4. Results

We demonstrate the effectiveness of our automatic controller generation on several different animals with widely varying morphologies: Two different simplified bipedal creatures both with 16 DOFs and weighing 10.6kg, a 65kg triped with a large back leg and two smaller front legs with 22 DOFs, and a 500kg horse-like quadruped with 29 DOFs. The skeletal structures for these animals are illustrated in figure 3.

The task models used to define the input to these controllers consist of both single- and multi-objective tasks. Each of these task models is created by combining one or multiple of four different sub-tasks: a direction following task, a torso angle matching task, a speed matching task, and a gait matching task, each parameterized by a single variable and with an associated cost function. The total task cost function for equation 2 is simply the sum of the individual cost functions:

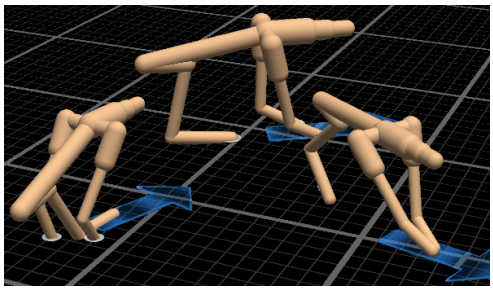

Figure 4: *Frames of a triped direction and gait type controller in use.*

task	**t**ask parameter	**c**ost
direction following	θ_v	$\lvert\theta_v - \text{atan}(\frac{v_y}{v_x})\rvert$
torso orientation	θ_r	$\lvert\theta_r - \phi\rvert$
speed	s_t	$\lvert s_t - \lVert v\rVert\rvert$
gait type	g_t	$10 \cdot (1 - \delta_{g_t,g})$

Figure 4 shows some frames from a triped controller constructed for a task model parameterizing over both direction and gait type.

We demonstrate our controller on four different task combinations. In all cases the synthesized controller successfully performs the desired tasks without need for any manual per-task or animal tweaking:

demo	**t**ask parameters	**o**ptimization parameters
direction	θ_v	θ
direction + torso	θ_v, θ_r	θ, ϕ
direction + gait	θ_v, g_t	θ, g, f_a, f_b
speed	s_t	s, f_a, f_b

The learning times observed for our controllers range from 5-10 minutes for a biped direction controller to 8-10 hours for a horse torso angle plus direction controller. Most of this difference in running time is due the the speed of the space-time constraints solver when applied to these different animals, with the rest due to the fact that multi-task controllers generally require more motion clips. Successful controllers tend to require between 5-10 constituent motions for the single-parameter controllers to 15-30 clips for more complicated controllers.

We evaluate the effectiveness of our process for iteratively constructing $\mathbf{G}$ by comparing it against a technique which attempts to sample the parameter space of each S_i as uniformly as possible. A graph of $cost(\mathbf{G}_i)$ at each iteration for our approach versus uniform sampling for three different task descriptions is shown in figure 5. For the two parameter task models, our approach achieves a better cost after its first few iterations than was achieved by uniform sampling even after over 20 iterations. The improvement for single parameter tasks is less dramatic since the state space is small enough to be sampled simply. Nevertheless the relative improvement of

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.

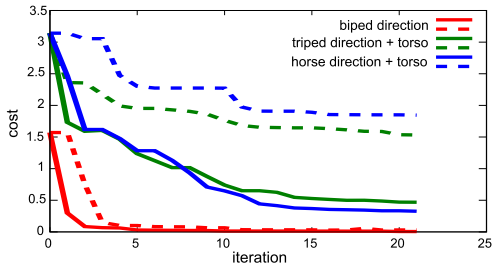

Figure 5: *Solid lines plot cost*$(\boldsymbol{G}_i)$ *over 20 iterations of our algorithm for three different task models. For comparison the costs achieved by a uniform sampling approach for the same tasks are shown as dashed lines. Each iteration represents the selection and application of a single optimization template (see figure 2).*

our approach is still substantial – after 20 iterations achieving a cost over five times lower than by uniform sampling.

For each controller we also define the dynamics of how the task parameters change over time. Our implementation does this independently for each task parameter by randomly choosing between keeping the parameter unaltered, switching it randomly to a new value, or changing it to a value randomly selected from a Gaussian distribution centered at the parameter's current value. Changing the relative probability with which these options occur impacts the type of controller created. For instance, controllers generated under large and frequent changes to the task parameters focus on motions which allow the character to quickly change its state, while controllers which assume that the task parameters stay primarily constant excel at making sure that the character can eventually satisfy the task accurately.

Additionally, setting the weights given to biomechanical versus task costs (see equation 6) provides a simple and useful way to alter the nature of the generated controllers. Controllers with a low weight on biomechanical costs are more agile, but may appear realistic, while controllers emphasizing the biomechanical costs are more smooth and natural looking, but adapt more slowly to changes in the task parameters. We have found a that a wide range of weights give reasonable results, and typically weight the biomechanical costs with a factor between 1 and 0.1, where a factor of 1 gives a weight equal to that of the task costs.

5. Conclusion and Future Work

We have described a completely automatic method for building animal locomotion controllers. The only input needed is a skeletal description of the animal, a set of timings for the gaits it can perform, and a statistical model of how the user is expected to control it. The tasks may be single- or multi-valued and the technique works for a wide range of different animal forms.

As is the case with almost all methods which control a character using a Markov decision process, our method scales poorly with the number of task parameters used simultaneously. This is both because solving for $\widetilde{cost}$ is more expensive, and because larger motion graphs are usually required to perform well in such cases. This limits the applicability of our approach to problems with two or perhaps three task parameters. Another limitation on our approach is its fundamental reliance on a powerful trajectory optimizer. Although our spacetime constraints implementation performs reasonably well, it has trouble synthesizing motions for animals with more than 30-40 DOFs, particularly where said animal is executing a higher speed run or jog. The spacetime constraints solver we employ also cannot synthesize motions for non-legged animals (such as snakes) or for animals with a large number of degrees of freedom (such as centipedes), nor for animals with distinct heel and toe contacts such as humans, although these present interesting challenges for future work.

Although there are several parameters that can be adjusted to tweak the style of the motions in the controllers we generate, our approach does not currently support artist interaction. One way to address this would be to incorporate a degree of artist control into the spacetime constraints solver. Although there has been some recent promising work in this direction [NCNV*12], it remains an open question how employ artist interaction within the context of our motion graph synthesis without requiring artist intervention at every time another spacetime constraints problem needs to be solved.

Appendix A: Spacetime Constraints Solver

The spacetime constraints solver used to generate the individual motions in our controllers is based on the formulation described in [WP09]. When generating cyclic motions, the constraint functions and the biomechanical component of the cost in equation 2 may be used directly from their description. For motions intended to connect states within an existing motion graph, we must ensure that the generated motion transitions seamlessly from its start and end states in the graph. To achieve this it is sufficient to ensure that the first and last pairs of frames each match a pair of consecutive states in the graph. We therefore fix the optimization variables corresponding to the first two and the last two frames and only optimize over the variables associated with the interior frames.

It remains to combine the biomechanical and task costs into the objective function as in equation 2. To do this first attach the initialization motion $\widetilde{\mathbf{M}}$ to the current graph $\mathbf{G}$. We generate a set of samples in the state space of the controller's MDP distributed according to the controllers influence function using the method described at the end of section 3.1, and retain the m samples which lie at frames in $\widetilde{\mathbf{M}}$. We can

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.

then calculate the objective function as:

$$\frac{1}{n}\sum_{f_i}\left[\alpha\,\log(cost_b(\mathbf{M}(f_i))) + \frac{1}{m}\sum_{s_j=(f_i,r_j,\theta_j)} cost_t(s_j,\pi^*(s_j))\right] \tag{6}$$

This is equivalent to computing equation 3 just along the motion being optimized. Here $cost_b(\mathbf{M}(f_i)$ is the objective function as described by [WP09] and α is a weighting term setting the tradeoff between minimizing biomechanical versus task costs. We typically employ values of α from 0.1 to 1. The logarithmic scaling of the biomechanical cost is used to make it easier to define the task costs. For instance a gallop may require significantly more energy than a slow walk, but that should not outright prevent the optimization from ever using such motions.

Appendix B: Cyclic Motion Temporal Alignemt

Given a motion $\mathbf{M}_c$ with foot contact times C_c, and target foot contact times C, we calculate a temporal alignment between C_c and C independently for each foot by pairing off the contact intervals for that foot in order. We then define a per-foot *time warp* function $t_c = T_{foot_j}(t)$ such that $T_{foot_j}(start_j) = start_{c,j}$ and $T_{foot_j}(end_j) = end_{c,j}$ for each $foot_j, start_j, end_j$ in C and $foot_j, start_{c,j}, end_{c,j}$ in C_c which match the given foot. Times in between the interval endpoints are linearly interpolated. Since T_{foot_j} perfectly aligns the contact timings of $foot_j$, we create an initialization motion $\widetilde{\mathbf{M}}_i$ by blending between these different warps according to

$$\widetilde{\mathbf{M}}_i(t)_d = \frac{\sum_{foot_j} w(foot_j,d)\mathbf{M}_c(T_{foot_j})_d}{\sum_{foot_j} w(foot_j,d)}$$

where d ranges over the DOFs for the character and $w(foot_j,d)$ is 1 if DOF d is on a path from $foot_j$ to the root and 0.01 otherwise. This ensures that the contact timings of each foot in $\widetilde{\mathbf{M}}_i$ match those in C_i and that the other degrees of freedom are reasonably interpolated.

Appendix C: Estimating Optimization Success Probability

Our definition of the function $\widetilde{P}(\mathbf{G}_i,\sigma_{i+1})$ estimates the probability with which the optimization defined by σ_{i+1} and initialization quality $q = q(\widetilde{\mathbf{M}}_i)$ will successfully converge to a physically valid motion. The motions constituting $\mathbf{G}_i$ are created in a series of previous optimizations, some of which have succeeded and others which have failed. Let the parameters and initialization quality be denoted by $\sigma_{s,j}$ and $q_{s,j}$ for the successful optimizations, and $\sigma_{f,j}$ and $q_{f,j}$ for the failed optimizations. Then we define:

$$\widetilde{P}(\mathbf{G}_i,\sigma_{i+1}) = \frac{s_1+s_2}{s_1+s_2+f_1+f_2} \tag{7}$$

where

$$s_1 = \sum_{q_{s,j}<q} \frac{q_{s,j}}{q} d(\sigma_{s,j},\sigma_{i+1})^{-2} \tag{8}$$

$$s_2 = \sum_{q_{s,j}>q} \frac{1}{d(\sigma_{s,j},\sigma_{i+1})^2 + \log(\frac{q}{q_{s,j}})} \tag{9}$$

$$f_1 = \sum_{q_{f,j}>q} \frac{q}{q_{f,j}} d(\sigma_{f,j},\sigma_{i+1})^{-2} \tag{10}$$

$$f_2 = \sum_{q_{f,j}<q} \frac{1}{d(\sigma_{f,j},\sigma_{i+1})^2 + \log(\frac{q_{f,j}}{q})} \tag{11}$$

and the function $d(\sigma_a,\sigma_b)$ computes the distance between the optimization parameters σ_a and σ_b as the square root of a sum per-parameter similarity measures as given below:

parameter	**d**istance
f_a	$d(\mathbf{G}_i(f_1),\mathbf{G}_i(f_2))^2$
f_b	$d(\mathbf{G}_i(f_1),\mathbf{G}_i(f_2))^2$
θ	$2-2\cos(\theta_2-\theta_1)$
ϕ	$2-2\cos(\phi_2-\phi_1)$
s	$(s_2-s_1)^2$
g	$1-\delta_{g_1,g_2}$

where $d(\mathbf{G}_i(f_1),\mathbf{G}_i(f_2))^2$ is measured according to the metric in [KGP02], and δ_{g_1,g_2} is 1 when g_1 and g_2 represent identical foot contact timings and 0 otherwise. An unweighted sum of these terms has sufficed in our experiments.

References

[AF02] ARIKAN O., FORSYTH D. A.: Interactive motion generation from examples. *ACM Transactions on Graphics (ACM SIGGRAPH 2002) 21*, 3 (2002), 483–490. 2

[BW95] BRUDERLIN A., WILLIAMS L.: Motion signal processing. In *Proceedings of SIGGRAPH 95* (Aug. 1995), Computer Graphics Proceedings, Annual Conference Series, pp. 97–104. 2

[CBvdP09] COROS S., BEAUDOIN P., VAN DE PANNE M.: Robust task-based control policies for physics-based characters. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 28*, 5 (2009), Article 170. 2

[CBvdP10] COROS S., BEAUDOIN P., VAN DE PANNE M.: Generalized biped walking control. *ACM Transctions on Graphics 29*, 4 (2010), Article 130. 2

[CBYvdP08] COROS S., BEAUDOIN P., YIN K., VAN DE PANNE M.: Synthesis of constrained walking skills. *ACM Transactions on Graphics 27*, 5 (2008), 113:1–113:9. 2

[CKJ*11] COROS S., KARPATHY A., JONES B., REVERET L., VAN DE PANNE M.: Locomotion skills for simulated quadrupeds. *ACM Transactions on Graphics 30*, 4 (2011), Article TBD. 2

[CRS10] CHENG REN L. Z., SAFONOVA A.: Human motion synthesis with optimization-based graphs. 2

[dLMH10] DE LASA M., MORDATCH I., HERTZMANN A.: Feature-based locomotion controllers. *ACM Transactions on Graphics 29*, 4 (July 2010), 131:1–131:10. 2

[FP03] FANG A. C., POLLARD N. S.: Efficient synthesis of physically valid human motion. *ACM Trans. Graph. 22*, 3 (2003), 417–426. 2

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.

[FvdPT01] FALOUTSOS P., VAN DE PANNE M., TERZOPOULOS D.: Composable controllers for physics-based character animation. In *Proceedings of ACM SIGGRAPH 2001* (2001), Annual Conference Series, pp. 251–260. 2

[Gor95] GORDON G. J.: Stable function approximation in dynamic programming. In *IN MACHINE LEARNING: PROCEEDINGS OF THE TWELFTH INTERNATIONAL CONFERENCE* (1995), Morgan Kaufmann. 4

[HG07] HECK R., GLEICHER M.: Parametric motion graphs. *Proceedings of Symposium on Interactive 3D Graphics and Games (I3D) 2007* (Apr. 2007). 2

[HP97] HODGINS J. K., POLLARD N. S.: Adapting simulated behaviors for new characters. In *Proceedings of SIGGRAPH 97* (1997), Annual Conference Series, pp. 153–162. 2

[HRE*08] HECKER C., RAABE B., ENSLOW R. W., DEWEESE J., MAYNARD J., VAN PROOIJEN K.: Real-time motion retargeting to highly varied user-created morphologies. *ACM Trans. Graph. 27*, 3 (2008), 1–11. 2

[HWBO95] HODGINS J. K., WOOTEN W. L., BROGAN D. C., O'BRIEN J. F.: Animating human athletics. In *Proceedings of ACM SIGGRAPH 95* (1995), Annual Conference Series, pp. 71–78. 2

[KG03] KOVAR L., GLEICHER M.: Flexible automatic motion blending with registration curves. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2003), SCA '03, Eurographics Association, pp. 214–224. 6

[KG04] KOVAR L., GLEICHER M.: Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph. 23* (August 2004), 559–568. 2

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Transactions on Graphics 21*, 3 (July 2002), 473–482. 2, 5, 6, 9

[LCR*02] LEE J., CHAI J., REITSMA P. S. A., HODGINS J. K., POLLARD N. S.: Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics 21*, 3 (July 2002), 491–500. 2

[LHP05] LIU C. K., HERTZMANN A., POPOVIĆ Z.: Learning physics-based motion style with nonlinear inverse optimization. *ACM Trans. Graph. 24*, 3 (2005), 1071–1081. 2

[LK05] LAU M., KUFFNER J. J.: Behavior planning for character animation. In *2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (July 2005), pp. 271–280. 2, 4

[LL04] LEE J., LEE K. H.: Precomputing avatar behavior from human motion data. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (July 2004), pp. 79–87. 2, 4

[LLP09] LEE Y., LEE S. J., POPOVIĆ Z.: Compact character controllers. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 papers* (New York, NY, USA, 2009), ACM, pp. 1–8. 1, 4

[LWB*10] LEE Y., WAMPLER K., BERNSTEIN G., POPOVIĆ J., POPOVIĆ Z.: Motion fields for interactive character locomotion. *ACM Trans. Graph. 29*, 6 (Dec. 2010), 138:1–138:8. 2

[LWH*12] LEVINE S., WANG J. M., HARAUX A., POPOVIĆ Z., KOLTUN V.: Continuous character control with low-dimensional embeddings. *ACM Transactions on Graphics 31*, 4 (2012), 28. 2

[LYvdP*10] LIU L., YIN K., VAN DE PANNE M., SHAO T., XU W.: Sampling-based contact-rich motion control. *ACM Transactions on Graphics 29*, 4 (July 2010), 128:1–128:10. 2

[LZ08] LO W.-Y., ZWICKER M.: Real-time planning for parameterized human motion. In *2008 ACM SIGGRAPH / Eurographics Symposium on Computer Animation* (July 2008), pp. 29–38. 2, 4

[MdLH10] MORDATCH I., DE LASA M., HERTZMANN A.: Robust physics-based locomotion using low-dimensional planning. *ACM Transactions on Graphics 29*, 4 (July 2010), 71:1–71:8. 2

[MP07] MCCANN J., POLLARD N.: Responsive characters from motion fragments. *ACM Transactions on Graphics (SIGGRAPH 2007) 26*, 3 (July 2007). 2, 3, 4

[MTP12] MORDATCH I., TODOROV E., POPOVIC Z.: Discovery of complex behaviors through contact-invariant optimization. *ACM Trans. Graph. 31*, 4 (2012), 43. 2

[NCNV*12] NUNES R. F., CAVALCANTE-NETO J. B., VIDAL C. A., KRY P. G., ZORDAN V. B.: Using natural vibrations to guide control for locomotion. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2012), I3D '12, ACM, pp. 87–94. 2, 8

[PW99] POPOVIĆ Z., WITKIN A.: Physically based motion transformation. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1999), ACM Press/Addison-Wesley Publishing Co., pp. 11–20. 2

[RGBC96] ROSE C., GUENTER B., BODENHEIMER B., COHEN M. F.: Efficient generation of motion transitions using spacetime constraints. pp. 147–154. 2

[RH91] RAIBERT M. H., HODGINS J. K.: Animation of dynamic legged locomotion. In *Computer Graphics (Proceedings of SIGGRAPH 91)* (1991), Annual Conference Series, ACM SIGGRAPH, pp. 349–358. 2

[SB98] SUTTON R., BARTO A.: *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts, 1998. 3

[SHP04] SAFONOVA A., HODGINS J. K., POLLARD N. S.: Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Trans. Graph. 23*, 3 (2004), 514–521. 2

[SKL07] SOK K. W., KIM M., LEE J.: Simulating biped behaviors from human motion data. *ACM Transactions on Graphics 26*, 3 (2007), 107:1–107:9. 2

[SO06] SHIN H. J., OH H. S.: Fat graphs: constructing an interactive character with continuous controls. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland, Switzerland, 2006), Eurographics Association, pp. 291–298. 2

[TLP07] TREUILLE A., LEE Y., POPOVIĆ Z.: Near-optimal character animation with continuous control. *ACM Trans. Graph. 26*, 3 (2007), 7. 2, 4

[WH00] WOOTEN W. L., HODGINS J. K.: Simulating leaping, tumbling, landing and balancing humans. *International Conference on Robotics and Automation (ICRA)* (2000), 656–662. 2

[WHDK12] WANG J. M., HAMNER S. R., DELP S. L., KOLTUN V.: Optimizing locomotion controllers using biologically-based actuators and objectives. *ACM Trans. Graph. 31*, 4 (2012), 25. 2

[WK88] WITKIN A., KASS M.: Spacetime constraints. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1988), ACM, pp. 159–168. 2, 3

[WP95] WITKIN A. P., POPOVIĆ Z.: Motion warping. In *Proceedings of SIGGRAPH 95* (Aug. 1995), Computer Graphics Proceedings, Annual Conference Series, pp. 105–108. 2

[WP09] WAMPLER K., POPOVIĆ Z.: Optimal gait and form for animal locomotion. *ACM Trans. Graph. 28*, 3 (2009), 1–8. 2, 3, 4, 5, 8, 9

[WP10] WU J.-C., POPOVIĆ Z.: Terrain-adaptive bipedal locomotion control. *ACM Transactions on Graphics 29*, 4 (July 2010), 72:1–72:10. 2

© 2013 The Author(s)
© 2013 The Eurographics Association and Blackwell Publishing Ltd.