Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2012)
P. Kry and J. Lee (Editors)

Learning Motion Controllers with Adaptive Depth Perception

Wan-Yen Lo[†1,2], Claude Knaus[2], and Matthias Zwicker[1,2]

[1]University of California, San Diego [2]Universität Bern

Abstract

We present a novel approach to real-time character animation that allows a character to move autonomously based on vision input. By allowing the character to "see" the environment directly using depth perception, we can skip the manual design phase of parameterizing the state space in a reinforcement learning framework. In previous work, this is done manually since finding a minimal set of parameters for describing a character's environment is crucial for efficient learning. Learning from raw vision input, however, suffers from the "curse of dimensionality", which we avoid by introducing a hierarchical state model and a novel regression algorithm. We demonstrate that our controllers allow a character to navigate and survive in environments containing arbitrarily shaped obstacles, which is hard to achieve with existing reinforcement learning frameworks.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation;

1. Introduction

Reinforcement Learning (RL) has recently attracted considerable attention in character animation. RL algorithms allow the character to learn optimal behaviors in a preprocessing step by interacting with the environment. At run-time, the character can make (near) optimal decisions in real-time.

Unfortunately, defining suitable state spaces for RL algorithms is difficult. On one hand, using general but high-dimensional state spaces is impractical, since the learning time and memory requirements increase superlinearly with the number of state variables. On the other hand, too specific state representations have disadvantages: a controller learned to navigate through round obstacles that are parameterized by radii, can be confused with sharp obstacles, as they are not well described by radii. Hence, controllers have to be manually parametrized for each task and environment. This process is tedious, and a small change in the state representation requires the learning process to be repeated.

We propose a general approach that avoids the need of any ad-hoc parametrization of the environment. With our approach, low-dimensional state representations can be built automatically for learning intuitive behaviors efficiently. Instead of letting the character read a manually crafted description of the environment, we allow the character to perceive depth directly. Since the large amount of visual information is intractable for learning, we use a hierarchical representation of the visual percepts. We build a hierarchy by subsampling the input percept, based on the observation that high resolution vision is not always required. When objects are far, blurry vision is enough to make a good decision; when objects are close, clear vision is needed. Also, in general, objects in front of the character require clearer vision than those in other directions. We do not, however, explicitly specify a level of detail for any given situation. Instead, we allow the level of detail to be selected automatically during learning, and found that the learned controllers confirm this intuition. The main technical contribution of our approach is to integrate the hierarchical state model with an efficient RL algorithm, and to allow the depth percepts to be interpreted adaptively and automatically in the learning process.

First of all we show that the performance of previous approaches based on parameterizing each object explicitly drops quickly as the environment gets more cluttered. In contrast, learning with depth perception is relatively more robust. We further show that complicated situations require high resolution vision, but using raw vision input suffers from the curse of dimensionality. Our hierarchical state rep-

† walo@ucsd.edu

DOI: 10.2312/SCA/SCA12/145-154

resentation, however, allows a controller to be learned efficiently by adjusting the resolution adaptively. We also show how a learned controller is flexible enough to handle objects of novel shapes. Finally, we use a game-like scenario to highlight the advantage of RL approaches over path searching techniques in real-time applications.

In summary, we make the following contributions:

- A hierarchical state model for reinforcement learning to replace a single state definition of fixed dimensionality.
- The use of a hierarchical state model in character animation to allow characters to perceive depth. Our approach avoids the need to carefully design ad-hoc parameterizations of environments based on their specific properties.
- An efficient reinforcement learning algorithm that integrates our hierarchical state model. To this end, we present a regression algorithm with automatic resolution adaptation based on scene complexity.

2. Related Work

Motion graphs and related approaches are commonly used to represent plausible transitions between motion segments [KGP02, SO06, HG07, BCvdPP08]. By traversing the graph, natural-looking and complex motions can be synthesized. Most systems allow users to specify the desired motion, and formulate the input constraints as a cost function. The desired motion can then be solved by searching a path through the graph that minimizes the total cost [KGP02, AF02, LK05, LK06, SH07, LZ10]. However, the search complexity for an optimal or near-optimal solution is exponential to the connectivity of the graph and the length of motion. Applying these techniques for real-time applications is therefore challenging. Other approaches use probabilistic roadmaps, rapidly-exploring random tree, or limited-horizon search, trading optimality for efficiency [LK00, KL00, LCR*02, CLS03, CKHL11].

Reinforcement learning, on the other hand, allows precomputing optimal actions for every situation [KLM96, SB98]. The objectives and constraints are represented as a reward function, and RL algorithms aim at finding a mapping from states to actions that maximizes the rewards accumulated over time. Several works have demonstrated effective use of RL in character animation [LL04, IAF05, TLP07, MP07, LZ08]. To increase the applicability of RL in character animation, Lee et al. [LLP09] reduce the size of the action space by presenting a motion selection algorithm; Lee and Popović [LP10] allows the reward function to be inferred from examples provided by the user; Lee et al. [LWB*10] define a continuous action space by introducing motion fields in order to make more responsive controllers. Wampler et al. [WAH*10] extend the RL framework with game theory to generate controllers in two-player adversarial games. Levine et al. [LLKP11] combines RL with heuristic search to enable space-time planning in highly dynamic environments.

In robotics and artificial intelligence, vision-based sensing has been combined with local path planning [MCKK05, CTS*09] and reinforcement learning [MSN05, JP05, JP07], allowing the agent to navigate toward the goal while avoiding obstacles. Michels et al. [MSN05] define a state by dividing the image into a set of directions. Each direction is a vertical stripe encoding the log distance to the nearest obstacle. However, the number of stripes is pre-defined and does not adapt to scene complexity. On the other hand, Jodogne and Piater [JP07] argue that most approaches in RL rely on preprocessing the visual input to extract task-relevant information. To avoid task specific coding, they use an automatic image classifier to partition the visual space adaptively. However, they use a fixed map to train the classifier and any change in the map requires the training to be repeated.

In computer graphics, synthetic vision are used for steering virtual humans [NRTMT95, KL99, PO03, OPOD10]. However, without pre-planning, computing the solution at run-time leads to a trade-off between optimality and efficiency. In applied perception, Sprague et al. [SBR07] introduce a more elaborate visual model by allowing the character to control the gaze, and apply RL algorithms to build mappings from visual percepts to body movements. However, their controller can only take into account one nearest obstacle at a time, while our perception model does not limit the number of objects in the scene. Our state definition is similar to the one used by Michels et al. [MSN05], but inspired by Jodogne and Piater's work [JP05], we subdivide the depth percepts adaptively to adjust the level of discretization dynamically and automatically according to the complexity of the scene. Applying our state model to RL, we can achieve near-optimal character control in real-time.

3. Learning Motion Controllers

In this section, we review relevant RL background and explain our choice of learning algorithms. The motion control problem is formulated as a *Markov decision process* (MDP) $(\mathcal{S}, \mathcal{A}, T, R)$ with discrete-time dynamics:

- $\mathcal{A}$ is the action space, composed of all motion fragments.
- $\mathcal{S}$ is the state space, and generally a state $s_t \in \mathcal{S}$ is defined as a vector (a_{t-1}, θ_t), where t is a discrete time step, $a_{t-1} \in \mathcal{A}$ denotes the currently played motion, and $\theta_t \in \mathbb{R}^n$ is a vector of *task parameters* describing the character's current situation in the environment.
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the transition model, where $T(s, a, s')$ denotes the probability of reaching state s' when action a is taken in state s.
- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, defined as $R(s_t, a_t) = R_s(s_t) + R_t(a_{t-1}, a_t)$. The state reward R_s measures how well the character respects user objectives and environmental constraints. The transition cost R_t ensures smooth transitions between motions.

A solution to the MDP is a *policy* $\pi : \mathcal{S} \to \mathcal{A}$, and the quality of a policy is measured by the value function V^π :

$\mathcal{S} \to \mathbb{R}$,

$$V^{\pi}(s) = E^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s\right] \quad (1)$$

$$= R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V^{\pi}(s'), \quad (2)$$

where $a_t = \pi(s_t)$ and γ accounts for future uncertainty. There exists at least one optimal policy π^* that yields the optimal value for every state [SB98]. Given the optimal value function V^* , an optimal policy can be defined as,

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') V^*(s') \right). \quad (3)$$

If the state and action spaces are finite, and the transition model and reward function are known, the optimal solution to the MDP can be found with dynamic programming algorithms, such as value iteration or policy iteration. In our case, however, the state space is large and continuous while the transition model is unknown.

Therefore, we can only approximate the optimal value function to obtain a near-optimal policy. To learn motion controllers, Lo and Zwicker [LZ08] and Lee et al. [LLP09] propose to use regression trees, where each leaf node determines a constant approximation. A regression tree starts with the whole state space as a root node, and is recursively refined until it provides enough representation power. Since this adaptive refining process has proven effective to solve high dimensional problems, we also use regression trees as our approximation architecture. More specifically, we adopt the *Extra-trees* algorithm [GEW06] to build an ensemble of randomized trees. In order to integrate our hierarchical state representation into the existing learning framework, we further propose a modification to the original Extra-trees algorithm, as detailed in Section 4.3.

Since the transition model is unknown in our framework, instead of the optimal value function, we approximate the optimal *action-value function*,

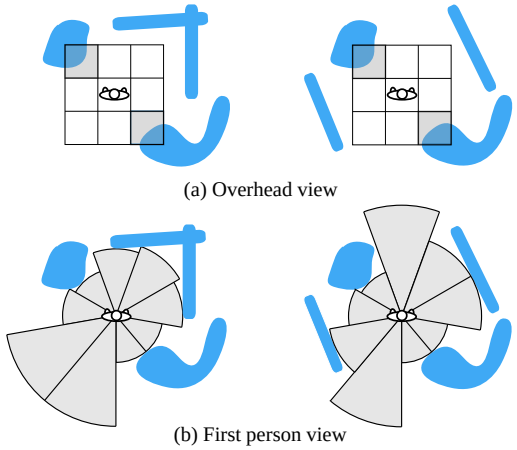
$$Q^{\pi}(s,a) = E^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s, a_0 = a\right] \quad (4)$$

$$= R(s,a) + \gamma \sum_{s'} T(s,a,s') V^{\pi}(s'), \quad (5)$$

Given the optimal action-value function Q^*, an optimal policy can be derived as

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s,a). \quad (6)$$

Equation 6 is favorable for decision making, especially when performing one-step look-ahead is expensive or the transition model is unknown. In our work, we approximate Q^* using the *fitted Q iteration algorithm* [EGW05], which has been explored in previous work [LZ08, LWB*10]. The algorithm takes as input an approximation architecture (Extra-trees in our case), and a sequence of the character's interactions with the environment. The algorithm iteratively updates the action-value function and uses the updated policy to generate new sequences of interactions. We present the algorithm in Appendix A for completeness.



(a) Overhead view

(b) First person view

Figure 1: *Comparison of perception models using overhead view (a) and first person view (b). The blue regions denote obstacles while the gray regions are their representations in the perception model. In this example, the overhead view classifies the two different environments as the same, while the first person view successfully discriminates them.*

4. Adaptive Depth Perception

In Section 4.1, we first explain how environments are parameterized in previous RL frameworks, and then propose our approach based on visual percepts that does not require ad-hoc parameterizations of different environments. Learning directly from the visual percepts, however, suffers from the curse of dimensionality. To overcome the problem, we introduce a hierarchical state model in Section 4.2, and present a novel regression algorithm to allow learning motion controllers with adaptive depth perception in Section 4.3.

4.1. State Representation

Previous works show that finding a state representation for efficient learning remains challenging. Most state representations are tailored for each specific task. Lee and Popović [LP10] demonstrate motion controllers on environments with varying number and shapes of obstacles, but they assume the environments are fixed. For every change in the environment, a new controller must be learned. To support dynamic environments, many previous works parameterize each object explicitly [TLP07, LZ08, LLP09, LLKP11], and typically the task parameters (Section 3) are defined as

$$\theta = (x_1, y_1, \mathbf{s}_1, \ldots, x_m, y_m, \mathbf{s}_m), \quad (7)$$

where m refers to the number of objects, and (x_i, y_i) and $\mathbf{s}_i$ denote the relative position and shape description of the ith

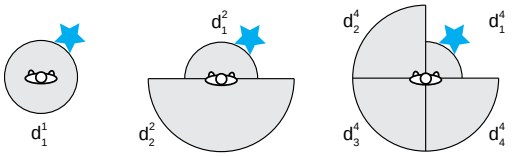


Figure 2: *Visualization of our hierarchical state model. (a) When $n = 1$, a state indicates the distance to the closest object in the scene. (b) When $n = 2$, a state indicates the distances to the closest objects in the front and back. (c) When $n = 4$, a state describes the shortest distances to the objects in four directions.*

object. The value of m is predefined, yet allowing more objects makes the learning problem harder.

A common solution is to consider only a few closest objects in the environment, but there may be more than m objects that are equally close to the character. Unable to perceive its complete situation, the character may easily make a fatal decision. In addition, the descriptive power of shape parameter $\mathbf{s}_i$ is usually limited to avoid high dimensionality, which explains the popular use of cylinders in previous work. Finally, a controller learned with one shape description cannot be easily generalized to novel shapes.

A natural way to resolve this complexity is to let the character "see" the environment directly, by defining a state with the character's vision. Some previous work use an overhead camera to model the character's perception [IAF05, MCKK05]. However, we found that by rendering the scene from the first person view [KL99, MSN05], we achieve better learning results, because a more descriptive state can be defined with fewer parameters, as shown in Figure 1. More specifically, we capture n depth values from the character's panoramic field of view, and store for each vertical stripe i the log distance d_i to the closest object. Therefore, we define the task parameters in a state as

$$\theta = (d_1, \ldots, d_n). \tag{8}$$

However, it is difficult to use this representation directly in existing RL frameworks, since the resolution n needs to be predefined. If n is too small, the state representation is not descriptive enough for the learning algorithms to converge. If n is too large, the problem is cursed by dimensionality. Therefore, we propose a hierarchical state model and a novel regression algorithm to allow the use of depth perception in RL frameworks.

4.2. Hierarchical State Model

We propose a novel approach to adjust the dimensionality of the perception automatically and adaptively in the learning process. Our approach is based on the observation that high resolution vision is not always required, but sometimes

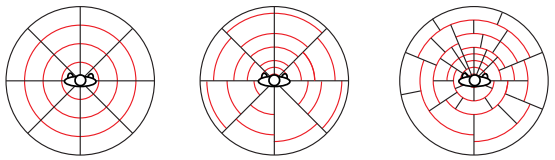


Figure 3: *The depth perception space discretized with (a) uniform grid, (b) adaptive regression, and (c) hierarchical adaptive regression. The resolution is fixed for (a) and (b), shown by the black angular dividers. But with adaptive regression (b), each dimension is discretized adaptively, shown as red radial dividers. With our hierarchical regression algorithm, the resolution can be adaptively adjusted and each dimension can also be adaptively discretized.*

low resolution vision is enough for making good decisions. Hence, we build a hierarchy by sub-sampling the input percept, and reformulate the task parameters using a hierarchical representation

$$\theta = (\mathbf{d}^n, \mathbf{d}^{\frac{n}{2}}, \mathbf{d}^{\frac{n}{4}}, \ldots, \mathbf{d}^1), \tag{9}$$

where n denotes the finest resolution of the character's perception, and we let n to be a power of 2 to simplify the definition. The vector $\mathbf{d}^i$ is defined as

$$\mathbf{d}^i = (d_1^i, d_2^i, d_3^i, \ldots, d_i^i), i = 1, 2, 4, \ldots, n, \tag{10}$$

where i represents the level of sub-sampled vision, and d_j^i corresponds to a non-negative depth value. The perception with the highest resolution is directly obtained from the camera mounted on the character, and for other resolutions the depth values are computed as

$$d_j^i = \min\left(d_{2j}^{2i}, d_{2j-1}^{2i}\right). \tag{11}$$

These definitions are visualized in Figure 2.

4.3. Adaptive Learning

In this section, we introduce our hierarchical regression algorithm, which is based on the Extra-trees algorithm but allows the state hierarchy to be automatically adapted to the scene complexity. Compared to the use of a uniform grid (Figure 3a), the original Extra-trees algorithm adaptively discretizes the state space in each dimension (angular bin in Figure 3b). This allows some parts of the state space to be discretized more finely than others, providing the character with clearer vision when the object is up front. However, the original Extra-trees algorithm only works with a state representation of fixed dimensionality, such as Equation 8. It does not allow the number of dimensions n to be adjusted adaptively.

In order to adaptively discretize the perception space in both radial and angular directions, as shown in Figure 3c, we

Algorithm 1 Hierarchical Extra-Trees Algorithm

Input: $\mathcal{T}' = \{(x_t = (s_t, a_t), y_t)\}$ and $\mathcal{D} = \{(i, j)\}$

1: **procedure** SPLITNODE($\mathcal{T}', \mathcal{D}$)
2: **for all** (i, j) in $\mathcal{D}$ **do**
3: Find a random split value $c_{i,j}$
4: Compute the relative variance reduction $r_{i,j}$
5: **end for**
6: $(a, b) \leftarrow \text{argmin}_{i,j}\, r_{i,j}$
7: **if** $r_{a,b} < \varepsilon_s$ **then**
8: $\mathcal{D}' \leftarrow \{(2i, 2j), (2i, 2j-1), \forall (i, j) \in \mathcal{D}\}$
9: Repeat 1-6 using $\mathcal{D}'$ to get (a', b')
10: **if** $r_{a',b'} > r_{a,b}$ **then**
11: $(p, q) \leftarrow (\frac{a'}{2}, \lfloor \frac{b'}{2} \rfloor)$
12: $\mathcal{D} \leftarrow \mathcal{D} \setminus \{(p, q)\}$
13: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(2p, 2q), (2p, 2q+1)\}$
14: $(a, b) \leftarrow (a', b')$
15: **end if**
16: **end if**
17: $\mathcal{T}'_L \leftarrow \{(x_t, y_t) | s_t.d^a_b < c_{a,b}\}$
18: $\mathcal{T}'_R \leftarrow \{(x_t, y_t) | s_t.d^a_b \geq c_{a,b}\}$
19: SplitNode($\mathcal{T}'_L, \mathcal{D}$)
20: SplitNode($\mathcal{T}'_R, \mathcal{D}$)
21: **end procedure**

present a regression algorithm that works with the hierarchical state model introduced in Section 4.2. Our Hierarchical Extra-trees algorithm maintains the original framework of iteratively selecting a node split that leads to highest relative variance reduction, while allowing the set of dimensions be adjusted in a hierarchical way. In Algorithm 1, Line 7–16 is our modification, while the rest is from the original Extra-trees algorithm.

We start this recursive regression algorithm using the lowest resolution of depth perception, so the input $\mathcal{D}$ equals to $\{(1, 1)\}$, that is, only the value of d^1_1 is considered in θ. Initially, the entire state space is treated as a root node, so the input $\mathcal{T}'$ is obtained from all sampled transitions (using Line 6–7 in Algorithm 2). To split a node, a potential split position is generated randomly for each dimension in $\mathcal{D}$ and each potential split is evaluated by computing the relative variance reduction [GEW06]. The split with the highest variance reduction is selected (Line 6) to divide $\mathcal{T}'$ into two subsets (Line 17–18), which are recursively split to produce a tree (Line 19–20) .

In the first few recursions, blurry or low-dimensional vision is sufficient for discriminating between favorable and unfavorable states. However, after the character has learned enough using the blurry vision, further splitting along any particular input dimension yields no satisfying variance reduction (Line 7, where ε_s is a pre-defined threshold). Since the current set of dimensions cannot discriminate more complex situations, our adaptive strategy is to increase the resolution of the perception, by *refining* the input set of dimensions $\mathcal{D}$. For each dimension in use, we check if we can get better variance reduction by replacing it with its two finer-level dimensions (Line 8–9), and we replace the one with the best improvement (Line 10–15). The subtrees from this node will use this new set of dimensions (Line 19–20) until it is not descriptive enough and refined again. Hence, each node in the regression tree might use a different set of dimensions for splitting the state space, as illustrated in Figure 4. The adaptive refinement stops when the finest resolution level is reached.

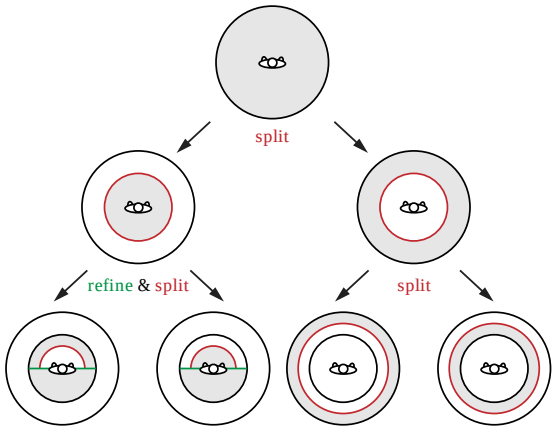


Figure 4: *Illustration of building a regression tree with our hierarchical regression algorithm. "Split" (shown in red) is what the original Extra-trees algorithm does, "refine" (shown in green) means refinement of our hierarchical state model. The initial state consists of one-dimensional perception with lowest resolution. First we find a cut along the dimension and split the root node into a left and right subtree. The right subtree represents those states where even the closest object is distant from the character. When all the objects are far, blurred vision is enough, so no state refinement is required. If the character is in a state represented by the left subtree, there is at least one close object, and the character requires clearer vision to judge the situation. Hence, only after the resolution of the perception is increased can we find a cut to sufficiently reduce the variance.*

With the original Extra-trees algorithm, the use of high resolution input is prohibitive, as running Line 2–5 in Algorithm 1 is computationally intensive. With our hierarchical regression algorithm, however, the dimensionality is adaptively increased, so we can gain considerable speedup.

5. Results

We collect a few minutes of motion capture data of a person walking around, and organize the data by building a well-connected motion graph [ZS08], which contains 2173 nodes and 2276 links. We further compress the graph by collapsing the links with only one successor and one predecessor. The

final graph contains 25 nodes and 128 links. Each graph link corresponds to an action in the MDP.

To simulate the character's vision, we place four cameras on the body, each with 90 degrees field of view, covering the panoramic vision. We use object false-coloring for rendering different types of objects, e.g., goals and obstacles. Instead of introducing extra dimensions in the state definition to store the types, we use a set of dimensions for each type of object separately.

The remaining section is organized as follows. We first explain the experiment setup in Section 5.1. We demonstrate the effectiveness of using the depth perception model in 5.2. We further show in Section 5.3 that by adjusting the perception adaptively, the learning efficiency is significantly improved. We analyze the optimality and run-time performance of the result controllers in Section 5.4 and Section 5.5 respectively. In Section 5.6, we show how a learned controller is flexible enough to handle objects of novel shapes. Finally, we use a game-like scenario in Section 5.7 to highlight the advantage of applying reinforcement learning in real-time applications.

5.1. Experiment Setup

In Section 5.2–5.6, we make comparisons by learning *navigation controllers*. The objective of a navigation controller is to guide the character to a goal without colliding with any obstacle in the scene. Our state reward function is defined as: $+100$ for stopping in a goal region, -200 for collision, and -1 for each second elapsed. When acquiring the character's perception, we render the goal objects in a separate pass so the character can always see the goal, but the obstacles will occlude each other. All the comparisons are made on a PC with a Xeon 2.50GHz dual 4-core CPU and 8GB memory, using the same learning framework:

- In the fitted Q iteration algorithm (Algorithm 2), we rebuild regression trees using the (hierarchical) Extra-trees algorithm in each of the first 50 iterations; after 50 iterations, we freeze the tree structure, stop generating new training samples, and let the algorithm run until convergence.
- In each iteration, we generate N trajectories of training samples. Each trajectory starts at a random position and finishes at a goal or after exceeding 100 time steps.
- We build 10 regression trees to approximate an action-value function, that is, having 128 actions in our MDP, we produce 1280 regression trees for a motion controller.

To assess the quality of a motion controller, we run simulations with the controller to compute the average expected return. We start each simulation by placing the character randomly in a random environment, and run the simulation by using the controller to navigate the character. The expected return of a simulation is computed as $\sum_{t=0}^{1000} \gamma^t R(s_t, a_t)$. In the end, we average the expected returns from all simulations to obtain the *score* of a controller. The higher the score is, the better the controller performs. To make fair comparisons, we fix a set of 10,000 random environments and initial states, so the average scores of different controllers are directly comparable. Finally, since the learning algorithm is randomized, we repeat the learning process three times for each experiment and average the scores and running time.

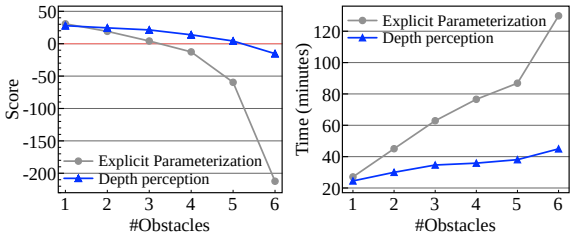


Figure 5: *Performance comparison between a controller learned with explicit parameterization of the environment (gray) and that learned with depth perception (blue).*

5.2. Depth Perception

We first compare the conventional state representation using explicit parameterization with our novel approach based on depth perception. The environment contains a goal object and m obstacles. All objects are cylinders with variable radii. In this comparison, the task parameters θ^e (explicit parameterization) and θ^d (depth perception) are defined respectively as

$$\theta^e = (x_1, y_1, a_1, \ldots, x_{m+1}, y_{m+1}, a_{m+1})$$
$$\theta^d = (d_1^g, d_2^g, d_3^g, d_4^g, d_1^o, d_2^o, d_3^o, d_4^o),$$

where (x_i, y_i) and $a_i = \tan^{-1}(r_i / \sqrt{x_i^2 + y_i^2})$ denote the relative position and viewing angle of the ith object whose radius is r_i; $(d_1^g, d_2^g, d_3^g, d_4^g)$ denote the character's depth perception of the goal in four directions; $(d_1^o, d_2^o, d_3^o, d_4^o)$ are defined similarly for the obstacles. In this comparison, $N = 200$ trajectories are generated in each iteration of the learning algorithm to expand the training set.

We compare the two representations with increasing number of obstacles m, and the results are presented in Figure 5. When there is only one obstacle in the scene, the results are comparable both in score and learning time. However, as the number of obstacles increases, so dose the dimensionality of θ^e, and the performance of the explicit parameterization declines significantly. Moreover, with explicit parameterization, after m increases to four, the score drops below zero, meaning that the character often takes unnecessary detours or collides with obstacles. On the contrary, the depth perception model is more robust with respect to the scene complexity. The results show that even a perception model with low resolution ($n = 4$ in Equation 8) can lead to a better controller in a shorter amount of time.

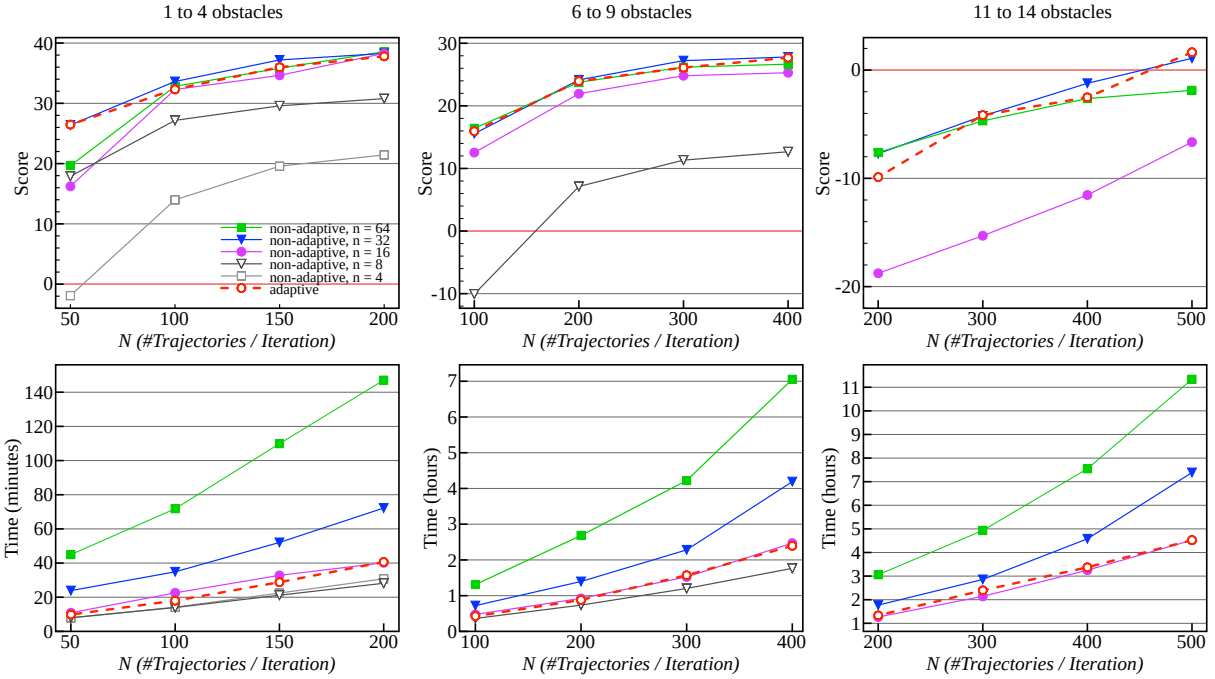


Figure 6: *Performance comparisons among non-adaptive depth perception, with n fixed to 4, 8, 16, 32, and 64 plotted in solid lines, and adaptive depth perception, plotted in dotted lines. We make comparisons using three different scene densities, and we consistently obtain improved learning performance with our approach.*

5.3. Adaptive Depth Perception

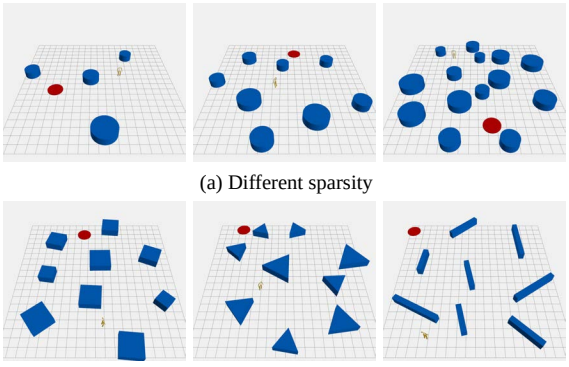
In this section, we analyze how the learning performance scales with the resolution of perception. First of all, we experiment with sparse environments containing one to four cylindrical obstacles whose positions and radii are generated randomly. We compare resolutions of $n = 4, 8, 16, 32$, and 64 in Equation 8 (when $n = 4$, the state representation is the same as θ^d in Section 5.2), and plot the scores and learning time with respect to N, the number of trajectories sampled in each iteration. The plots in Figure 6 show that the first two increases of the resolution greatly improves the learning results, but the increase from $n = 16$ to $n = 32$ improves the scores only moderately with the expense of a noticeable increase in learning time. The resolution of $n = 64$, while consuming a significant amount of time for learning, does not improve the scores at all. Consequently, with limited samples, the scores cannot be infinitely improved by simply increasing the resolution, because as the dimensionality increases, the volume of the state space grows so fast that the available samples become sparse. It is thus very difficult to predefine an ideal value of n. In this example, when the environment is sparse, $n = 16$ is ideal among all the tested resolutions for learning good controllers efficiently.

However, our adaptive depth perception model outperforms any of the fixed resolutions: compared to $n = 16$, higher scores can be obtained in a shorter amount of time. The result is plotted with dotted red lines in Figure 6. More specifically, the multi-scale state representation used in the experiments is defined as

$$\theta^a = (\mathbf{d}_g^n, \mathbf{d}_g^{\frac{n}{2}}, \ldots, \mathbf{d}_g^4, \mathbf{d}_o^n, \mathbf{d}_o^{\frac{n}{2}}, \ldots, \mathbf{d}_o^4), \tag{12}$$

where $\mathbf{d}_g^i$ and $\mathbf{d}_o^i$ are defined as in Equation 10 with the subscript g and o denote the perception of the goal and obstacles respectively. The threshold ε_s, which controls the degree of adaptivity in Algorithm 1, is set to 0.5.

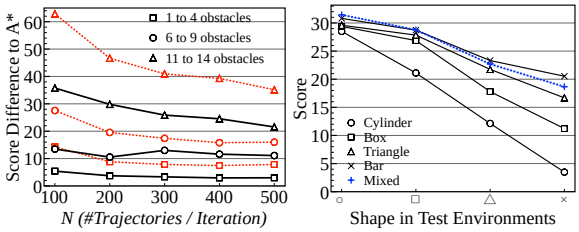
We further extend the experiments by increasing the density of the environment. The next two experiments are performed on environments containing 6 to 9 and 11 to 14 obstacles respectively, and the densities are visualized in Figure 7a. When there are 6 to 9 obstacles, a similar conclusion can be drawn from the plots in Figure 6: 1. Both $n = 16$ and $n = 32$ lead to good scores, but the former is more efficient. 2. Learning with $n = 64$ is much slower, while the score is not improved. 3. Our adaptive model has the best performance: it obtains the good scores from high resolutions and short learning time from low resolutions. However, as the environment gets even more cluttered, Figure 6 (11 to 14 obstacles) shows that $n = 16$ is no longer comparable with $n = 32$, and the latter is now the ideal fixed resolution. Our adaptive model, however, still gains the better scores from high resolutions and faster learning from low resolutions.



(a) Different sparsity

(b) Different shapes

Figure 7: *Environments of varied density (a) and containing obstacles of varied shapes (b).*



(a) Optimality (b) Generalizability

Figure 8: *(a) We analyze the optimality of our controllers by comparing the results to those generated with the A* search algorithm, and plot the differences. The red dotted lines and the black solid lines denote the controllers without and with one-step look-ahead respectively. (b) We quantify the generalizability of the controllers trained for different shapes by evaluating them in environments containing other shapes.*

5.4. Optimality

In order to evaluate the optimality of our controllers, we apply the A* search algorithm to compute the optimal score for the same sets of random environments. We define the heuristic function as the product of the straight line distance from the current position to the goal and the minimal cost required to travel one unit distance. In Figure 8, we plot in red dotted lines the difference between the scores of our controllers (learned with adaptive depth perception) and the optimal scores from A* search. The differences decreases as the number of sampled trajectories N increases. In general, more samples are required in more cluttered environments to achieve the same optimality.

5.5. Run-time Performance

Since it takes less than 2 milliseconds (500 fps) for a controller to make a decision in run-time, we can also allow the controller to look one step ahead into the future for making better decisions. Expanding the policy in Equation 3, we can obtain the one-step look-ahead policy,

$$\pi(s) = \underset{a}{\operatorname{argmax}} \left(R(s,a) + \max_{a'} \hat{Q}^*(s',a') \right), \qquad (13)$$

where s' is the consequent state of taking action a from the currents state s, and the depth values in s' are rendered by simulating the action a while assuming the environment is temporarily static. The one-step look-ahead evaluation is commonly used in animation literature [TLP07, LZ08, LLP09, LP10]. Making a decision with the look-ahead policy takes about 15 milliseconds (66 fps) on average, and the scores are shown in solid black lines in Figure 8a. When the environments are sparse, by looking ahead one step, the scores become very close to the optimal ones; when the environments are cluttered, the scores can be greatly improved but more samples are still required for the controllers to converge toward the optimal behaviors. However, given a start state, A* requires 15 seconds on average to compute the optimal path, while our controller can make decisions sequentially in real-time.

5.6. Generalizability

An advantage of using depth perception is that the learned controllers can be directly applied to environments containing arbitrarily-shaped objects, and no preprocessing, such as re-parameterization of the environment, is required. To quantify this property, we generate several controllers, each trained for obstacles of a specific shape, and evaluate the controllers in environments containing obstacles of other shapes. In this experiment, we start with cylinders that are used throughout the previous sections, and then change the shapes into boxes, triangles and bars, as shown in Figure 7b. To maintain the density of the environment, the volume of each obstacle is preserved when changed into other shapes. The evaluation results are shown in Figure 8b. The plot shows that the controller trained for cylinders can respond to other shapes, but the scores depend on the roundness of the obstacles in the environments used for evaluations. In general, bars and triangles produce more difficult scenes, as they tend to create wider blocks and narrower passages, while cylinders produce the simplest environments. So the controllers trained for cylinders are exposed to only few samples to learn difficult situations. This also explains why the controller trained for bars has the best scores in all shapes.

Finally, we mix all four shapes to learn a controller, whose scores are plotted in dotted blue line in Figure 8b. It has good scores in all tests, even though when compared with the controller trained for bars, it has worse performance in the environments containing only bars. This is due to the fact that the use of all shapes make difficult situations appear less often in the training samples, while the test set of bars involves only difficult situations. We conclude that using depth perception, the learned controllers can respond to novel shapes,

but with limited capability, as the controllers can only infer the situations from experience

5.7. Survival Game

In the end, we use a game-like scenario to highlight the advantage of reinforcement learning over path search techniques in real-time applications. We build a closed environment where all the obstacles move toward the character in a constant speed. The only goal of the character is to survive by not colliding with any obstacles or walls. Since the goal is not a concrete state or position, it is difficult to define a heuristic for path search algorithms like A*. In addition, applying search algorithms in real-time requires collision detection to be performed for each expansion of the search trees, which introduces considerable computation overhead. On the contrary, the scenario can be easily defined in a RL framework: in the learning stage, the character is given a small reward for every surviving moment, but a deadly penalty for any collision. This is all that is needed for the character to learn a surviving strategy automatically by interacting with the environment. The learned strategy is shown in the accompanying video.

6. Conclusions

We have proposed a method that facilitates the application of reinforcement learning to character control. Traditional state models are manually made up with explicit descriptions of the environment, but our state model lets the character perceive the environment directly. When using such a generic high-dimensional sensor like depth vision, it is critical to adapt the dimensionality of the state space to the scene complexity to avoid the curse of dimensionality. To enable such adaptation, we propose a hierarchical state model and a novel regression algorithm. Our hierarchical state model replaces a single state definition of fixed dimensionality. Our regression algorithm then uses the hierarchical state model to adapt the dimensionality to the scene complexity. We demonstrate that our approach based on adaptive depth perception learns a better controller in a shorter amount of time. The learned controller can also be directly applied to environments containing objects of novel shapes, without reparameterizations of the environment.

We see two major directions to improve our work. Firstly, the character only perceives and generalizes the current situation but does not memorize its history. If the environment is complex, consisting of dead ends like in a maze, the character may get trapped repeating the same mistakes. Incorporating short-term or long-term memory into the system can resolve this problem. Maintaining global information like a scene map [NRTMT95, KL99, Bak01] or modeling the problem with partially observable MDP can also help the character to make better decisions. Secondly, in this work we only use one-dimensional depth vision for learning, but in order to let the character perform more complicated actions, two-dimensional depth vision is required. As the potential number of dimensions increases quadratically, significantly more computing resources are demanded, or the learning should be performed more adaptively. However, we would like to see our work as a first step to explore further research of using adaptive state models in reinforcement learning.

Appendix A: Fitted Q Iteration Algorithm

Fitted Q iteration algorithm reformulates the Q-function determination problem a sequence of kernel-based regression problems [EGW05]. The algorithm takes as input an approximation architecture F, and a traing set $\mathcal{T}$ that consists of sequences of the character's interactions with the environment. The algorithm starts by initializing Q to be zero (Line 2) . In each iteration, Q is used to derive the long-term reward y_t for each state-action pair x_t (Line 5–8). Then Q is updated by fitting the data points of (x_t, y_t) with the approximation architecture F (Line 9). In the end of each iteration, new sequences of transitions are generated and added to the training set $\mathcal{T}$ with the updated Q-function (Line 10).

Algorithm 2 Fitted Q Iteration Algorithm

Input: a set of transition tuples $\mathcal{T} = \{(s_t, a_t, r_t, s_{t+1})\}$ and a function approximator F
Output: an approximation of the optimal action-value function $\hat{Q}^*$

1: $n \leftarrow 0$
2: $\hat{Q}_n \leftarrow 0$
3: **repeat**
4: $\quad n \leftarrow n+1$
5: $\quad$ **for all** (s_t, a_t, r_t, s_{t+1}) in $\mathcal{T}$ **do**
6: $\quad\quad x_t = (s_t, a_t)$
7: $\quad\quad y_t = r_t + \gamma \max_{a \in A} \hat{Q}_{n-1}(s_{t+1}, a)$
8: $\quad$ **end for**
9: $\quad$ Use F to induce $\hat{Q}_n$ from $\{(x_t, y_t) : t = 1, \ldots, |\mathcal{T}|\}$
10: $\quad$ Generate new transitions for $\mathcal{T}$ using $\hat{Q}_n$.
11: **until** $\|\hat{Q}_n - \hat{Q}_{n-1}\|_\infty < \varepsilon_q$
12: **return** $\hat{Q}_n$

References

[AF02] ARIKAN O., FORSYTH D. A.: Interactive motion generation from examples. *ACM Trans. Graph. 21* (July 2002), 483–490. 2

[Bak01] BAKKER B.: Reinforcement learning with long short-term memory. In *NIPS* (2001), pp. 1475–1482. 9

[BCvdPP08] BEAUDOIN P., COROS S., VAN DE PANNE M., POULIN P.: Motion-motif graphs. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2008), SCA '08, Eurographics Association, pp. 117–126. 2

[CKHL11] CHOI M. G., KIM M., HYUN K., LEE J.: Deformable motion: Squeezing into cluttered environments. *Computer Graphics Forum* (2011). 2

[CLS03] CHOI M. G., LEE J., SHIN S. Y.: Planning biped locomotion using motion capture data and probabilistic roadmaps. *ACM Trans. Graph. 22* (April 2003), 182–203. 2

[CTS*09] CHESTNUTT J. E., TAKAOKA Y., SUGA K., NISHIWAKI K., KUFFNER J., KAGAMI S.: Biped navigation in rough environments using on-board sensing. In *IROS* (2009), pp. 3543–3548. 2

[EGW05] ERNST D., GEURTS P., WEHENKEL L.: Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res. 6* (December 2005), 503–556. 3, 9

[GEW06] GEURTS P., ERNST D., WEHENKEL L.: Extremely randomized trees. *Machine Learning 63*, 1 (2006), 3–42. 3, 5

[HG07] HECK R., GLEICHER M.: Parametric motion graphs. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2007), I3D '07, ACM, pp. 129–136. 2

[IAF05] IKEMOTO L., ARIKAN O., FORSYTH D.: Learning to move autonomously in a hostile world. In *ACM SIGGRAPH 2005 Sketches* (New York, NY, USA, 2005), SIGGRAPH '05, ACM. 2, 4

[JP05] JODOGNE S., PIATER J. H.: Interactive learning of mappings from visual percepts to actions. In *Proceedings of the 22nd international conference on Machine learning* (New York, NY, USA, 2005), ICML '05, ACM, pp. 393–400. 2

[JP07] JODOGNE S., PIATER J. H.: Closed-loop learning of visual control policies. *J. Artif. Int. Res. 28* (March 2007), 349–391. 2

[KGP02] KOVAR L., GLEICHER M., PIGHIN F.: Motion graphs. *ACM Trans. Graph. 21* (July 2002), 473–482. 2

[KL99] KUFFNER J. J., LATOMBE J.-C.: Fast synthetic vision, memory, and learning models for virtual humans. In *CA* (1999), pp. 118–127. 2, 4, 9

[KL00] KUFFNER J. J., LAVALLE S. M.: Rrt-connect: An efficient approach to single-query path planning. In *Proc. IEEE ICRA* (2000), pp. 995–1001. 2

[KLM96] KAELBLING L. P., LITTMAN M. L., MOORE A. W.: Reinforcement learning: a survey. *J. Artif. Int. Res. 4* (May 1996), 237–285. 2

[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 Trans. Graph. 21* (July 2002), 491–500. 2

[LK00] LAVALLE S. M., KUFFNER J. J.: Rapidly-exploring random trees: Progress and prospects. In *Algorithmic and Computational Robotics: New Directions* (2000), pp. 293–308. 2

[LK05] LAU M., KUFFNER J. J.: Behavior planning for character animation. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2005), SCA '05, ACM, pp. 271–280. 2

[LK06] LAU M., KUFFNER J. J.: Precomputed search trees: planning for interactive goal-driven animation. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2006), SCA '06, Eurographics Association, pp. 299–308. 2

[LL04] LEE J., LEE K. H.: Precomputing avatar behavior from human motion data. In *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2004), SCA '04, Eurographics Association, pp. 79–87. 2

[LLKP11] LEVINE S., LEE Y., KOLTUN V., POPOVIĆ Z.: Space-time planning with parameterized locomotion controllers. *ACM Trans. Graph. 30* (May 2011), 23:1–23:11. 2, 3

[LLP09] LEE Y., LEE S. J., POPOVIĆ Z.: Compact character controllers. *ACM Trans. Graph. 28* (December 2009), 169:1–169:8. 2, 3, 8

[LP10] LEE S. J., POPOVIĆ Z.: Learning behavior styles with inverse reinforcement learning. *ACM Trans. Graph. 29* (July 2010), 122:1–122:7. 2, 3, 8

[LWB*10] LEE Y., WAMPLER K., BERNSTEIN G., POPOVIĆ J., POPOVIĆ Z.: Motion fields for interactive character locomotion. *ACM Trans. Graph. 29* (December 2010), 138:1–138:8. 2, 3

[LZ08] LO W.-Y., ZWICKER M.: Real-time planning for parameterized human motion. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (2008), SCA '08, Eurographics Association, pp. 29–38. 2, 3, 8

[LZ10] LO W.-Y., ZWICKER M.: Bidirectional search for interactive motion synthesis. *Computer Graphics Forum 29*, 2 (2010), 563–573. 2

[MCKK05] MICHEL P., CHESTNUTT J., KUFFNER J., KANADE T.: Vision-guided humanoid footstep planning for dynamic environments. In *in Proc. of the IEEE-RAS/RSJ Int. Conf. on Humanoid Robots (HumanoidsŠ05)* (2005), pp. 13–18. 2, 4

[MP07] MCCANN J., POLLARD N.: Responsive characters from motion fragments. *ACM Trans. Graph. 26* (July 2007). 2

[MSN05] MICHELS J., SAXENA A., NG A. Y.: High speed obstacle avoidance using monocular vision and reinforcement learning. In *Proceedings of the 22nd international conference on Machine learning* (New York, NY, USA, 2005), ICML '05, ACM, pp. 593–600. 2, 4

[NRTMT95] NOSER H., RENAULT O., THALMANN D., MAGNENAT-THALMANN N.: Navigation for digital actors based on synthetic vision, memory, and learning. *Computers & Graphics 19*, 1 (1995), 7–19. 2, 9

[OPOD10] ONDŘEJ J., PETTRÉ J., OLIVIER A.-H., DONIKIAN S.: A synthetic-vision based steering approach for crowd simulation. *ACM Trans. Graph. 29* (July 2010), 123:1–123:9. 2

[PO03] PETERS C., O'SULLIVAN C.: Bottom-up visual attention for virtual human animation. In *CASA* (2003), pp. 111–117. 2

[SB98] SUTTON R., BARTO A.: *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998. 2, 3

[SBR07] SPRAGUE N., BALLARD D., ROBINSON A.: Modeling embodied visual behaviors. *ACM Trans. Appl. Percept. 4* (July 2007). 2

[SH07] SAFONOVA A., HODGINS J. K.: Construction and optimal search of interpolated motion graphs. *ACM Trans. Graph. 26* (July 2007). 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* (2006), SCA '06, Eurographics Association, pp. 291–298. 2

[TLP07] TREUILLE A., LEE Y., POPOVIĆ Z.: Near-optimal character animation with continuous control. *ACM Trans. Graph. 26* (July 2007). 2, 3, 8

[WAH*10] WAMPLER K., ANDERSEN E., HERBST E., LEE Y., POPOVIĆ Z.: Character animation in two-player adversarial games. *ACM Trans. Graph. 29* (July 2010), 26:1–26:13. 2

[ZS08] ZHAO L., SAFONOVA A.: Achieving good connectivity in motion graphs. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Aire-la-Ville, Switzerland, Switzerland, 2008), SCA '08, Eurographics Association, pp. 127–136. 5