Motion Synthesis for 3 Articulated Figures and Mass-Spring Models

Motion Synthesis for 3D Articulated Figures and Mass-Spring Models

Hadi Partovi, Harvard University
Jon Christensen, Harvard University
Amir Khosrowshahi, Harvard University
Joe Marks, DEC CRL
J. Thomas Ngo, Harvard University

Abstract

Motion synthesis is the process of automatically generating visually plausible motions that meet goal criteria specified by a human animator. The objects whose motions are synthesized are often animated characters that are modeled as articulated figures or mass-spring lattices. Controller synthesis is a technique for motion synthesis that involves searching in a space of possible controllers to generate appropriate motions. Recently, automatic controller-synthesis techniques for 2D articulated figures have been reported. An open question is whether these techniques can be generalized to work for 3D animated characters. In this paper we report successful automatic controller synthesis for 3D articulated figures and mass-spring models that are subject to nonholonomic constraints. These results show that the 3D motion-synthesis problem can be solved in some challenging cases, though much work on this general topic remains to be done.

CR Categories: I.2.6 [**Artificial Intelligence**]: Learning—*parameter learning*. I.2.6 [**Artificial Intelligence**]: Problem Solving, Control Methods and Search—*heuristic methods*. I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*animation*. I.6.3 [**Simulation and Modeling**]: Applications.

Additional Key Words: Spacetime constraints, controller synthesis, banked stimulus-response (BSR) controllers, stochastic optimization, evolutionary computation.

1 Introduction

The task of specifying motion for animated characters, whether the specification takes the form of key frames or parameters for a physical simulation, is still performed in an essentially manual way. A major goal in animation research is to automate this task. This is the *motion-synthesis problem* [1].

Early work on motion synthesis for 2D articulated figures led to the development of local-optimization techniques for refining coarse, user-supplied trajectories for animated characters [3, 22, 6]. This work borrowed ideas from optimal control theory [4] and numerical optimization [10].

A very different approach to motion synthesis—an approach that takes inspiration from recent similar work in AI and robotics [7, 16, 15, 2]—is predicated upon the notion of computing *motion controllers* for animated characters automatically. A motion controller governs the actuators in a character's physical model. In this context, the motion-synthesis problem becomes one of *controller synthesis*: the goal is to create a motion controller that, when executed, will cause the character to move in a manner consistent with physical law and with goal criteria supplied by a human animator.

Conventional local-optimization techniques do not appear to be appropriate for controller synthesis. One recent alternative approach is due to Van de Panne and Fiume [21]. In their approach, a motion controller is a sensor-actuator network (SAN), a nonlinear network that connects binary sensors to the actuators that move the character. Connection weights are computed using a combination of random generate and test, simulated annealing, and stochastic gradient ascent. A competing approach, due to Ngo and Marks [17, 18, 8], employs a banked stimulus-response (BSR) controller, a set of mutually independent stimulus-response rules that govern a character's behavior. The parameters for the BSR controller are found by an evolutionary search mechanism.

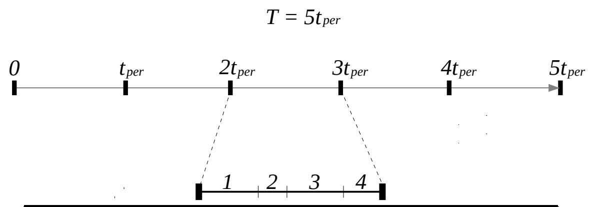
The controller-synthesis approach has produced exciting results, but only for 2D articulated figures. An open question is whether this approach will generalize to 3D, especially for characters whose physical models are subject to nonholonomic constraints [11], such as those imposed by contact with external objects like the ground.[1] There is every reason to believe that the generalization to 3D should be very difficult. The dimensionality of the controller space approximately doubles for a creature with a given number of joints, primarily because each joint can have two degrees of freedom (§2.2). More importantly, with many 3D articulated figures, a large fraction of the search space is occupied by controllers that cause the character to lose balance and fall over unrecoverably. This problem is much less severe in 2D.

Nevertheless, in this paper we report on several experiments in which effective motion controllers of the BSR variety were generated automatically for a selection of motion-synthesis problems involving 3D articulated figures subject to nonholonomic constraints imposed by the ground (§2). The problems considered involve characters that are expected to have varying degrees of stability, including relatively steady quadrupeds (§2.1) and a comically unstable biped (§2.2). We also present results for motion-synthesis problems involving a 3D mass-spring model subject to nonholonomic constraints (§3). These results constitute solid evidence that 3D motion synthesis, although harder than 2D motion synthesis, is by no means impossible for a range of interesting problems.

2 3D Articulated Figures

Our initial research in motion synthesis for 3D articulated figures began with an investigation of time-based BSR motion controllers (§2.1). Recent results [9, 8] showed that this kind of motion controller is simpler to code, easier to search, and just as able to generate useful motions for 2D articulated figures as the original BSR controller with physical sensors [17, 18]. This also proved true for stable 3D articulated figures, but not for unstable ones. For motion-synthesis problems involving an unstable biped, we had to resort to BSR controllers with physical sensors (§2.2) to get

[1] Tu et al. [20] have produced anecdotal evidence in the form of a video that the 3D motion-synthesis problem for mass-spring models that are not subject to nonholonomic constraints can be solved in a convincing fashion, at least for simple 12-spring models of fish.



Rule	*Stimulus*	*Response*
1	$[t^1_{\text{begin}}, t^1_{\text{end}})$	$(\theta^1_1, \psi^1_1, \ldots, \theta^1_{N-1}, \psi^1_{N-1}, \tau^1)$
2	$[t^2_{\text{begin}}, t^2_{\text{end}})$	$(\theta^2_1, \psi^2_1, \ldots, \theta^2_{N-1}, \psi^2_{N-1}, \tau^2)$
3	$[t^4_{\text{begin}}, t^3_{\text{end}})$	$(\theta^3_1, \psi^3_1, \ldots, \theta^3_{N-1}, \psi^3_{N-1}, \tau^3)$
4	$[t^4_{\text{begin}}, t^4_{\text{end}})$	$(\theta^4_1, \psi^4_1, \ldots, \theta^4_{N-1}, \psi^4_{N-1}, \tau^4)$

Figure 1: An illustration of a time-based BSR motion controller.

satisfactory results.

2.1 Time-based motion controllers

In a time-based BSR motion controller, the sole input "sensor" indicates the passage of time, so that the bank of stimulus-response rules is equivalent to a simple script of responses that is performed one or more times [8]. In our form of the BSR motion controller (Figure 1), a time interval with user-defined length t_{per} is broken into a small number C of mutually exclusive regimes. During each regime, the corresponding stimulus-response (SR) rule is active. The sequence of regimes is repeated to fill the length of the simulation, T, which is also chosen by the user. Thus, if $t_{\text{per}} < T$, then the sequence of actions specified by the rules is repeated periodically.[2] If, on the other hand, $t_{\text{per}} \geq T$, then the action sequence is not constrained to be periodic.

Each SR rule in a time-based controller for an N-rod creature is specified by the following parameters:

- A half-open time interval $[t_{\text{begin}}, t_{\text{end}})$ during which the rule is active. By analogy with BSR controllers whose sensors are physical, we refer to each time interval as a stimulus region, or simply a *stimulus*. The values in a valid time interval are subject to some constraints: $0 \leq t_{\text{begin}}, t_{\text{end}} < t_{\text{per}}$; the intervals in different rules must be disjoint; and the union of the intervals in all the rules must be $[0, t_{\text{per}})$.
- A set of target Euler angles θ_i, ψ_i for each joint i ($1 \leq i \leq N-1$) in the figure.
- A time constant τ.

While a given rule is active, the figure is deformed so that the actual Euler angles for each joint approach the target angles at a rate governed by the time constant. The absolute position and orientation of the figure are then determined by physical simulation of the deforming figure [12].

Thus, a time-based BSR controller contains $C-1$ independent stimulus boundaries, $C(2N-2)$ target Euler angles, and C time constants, giving a total of $2CN-1$ variables. (The value t_{per} is also essential to the specification of the controller, but because it is set by the user we do not count it as an independent parameter.) For example, Rex (depicted in Figure 14) has $N = 11$ rods and $C = 4$ SR rules, so the total number of variables is 87. It is the task of the search algorithm to find 87 floating-point values for these variables that will cause the figure to achieve the desired objective, which is expressed as a scalar *fitness function*. (A sample fitness function that is used to elicit horizontal movement is the distance traveled by the figure's center of mass.) The search algorithm used for the work reported here is a type of *stochastic population hill climbing*, described in Figure 2. Simple search algorithms of this kind have been shown empirically to be more effective for BSR-controller synthesis than other, more sophisticated kinds of evolutionary computation that involve crossover operations [9, 8].

[2]For all the time-based motion-controller results given in the paper and shown in the accompanying videotape, $t_{\text{per}} = \frac{T}{5}$ and the number of SR rules $C = 4$, just as in Figure 1.

```
Randomly generate 1,000 motion controllers
Initialize population to be the best 100 controllers
Rank order the population
for evaluation = 1 to 250,000
  Select a controller by rank-based selection
  Mutate the controller and re-evaluate it
  Insert the new controller into the population by rank
  Delete the lowest-ranked controller
end for
```

Figure 2: One form of stochastic population hill climbing.

The initial steps in the search algorithm are designed to produce a good initial population of random controllers (for details of the randomization process, see the discussion of mutation below). Once the population of controllers has been initialized, the algorithm calls for a repeated cycle of controller selection, mutation, insertion, and deletion. The selection step is done according to the rank ordering of the controllers: the higher a controller is in the ranking, the more likely it is to be selected. This causes computational effort to be focused on the areas of the search space in which the best solutions have been found, but without eliminating suboptimal areas from consideration. The mutation step modifies a copy of the selected controller in the following ways:

1. One randomly selected rule in the controller is subjected to *creep*, in which all of the independent rule parameters are changed by a small, random amount.
2. With probability $p = \frac{1}{2}$, a randomly selected rule is reinitialized. This is done by generating random values for all the parameters in the rule. The time-interval and target-angle parameters are distributed uniformly; the time-constant parameter is distributed logarithmically [17, 18].

The insertion and deletion steps are straightforward.

Figure 13 shows a trajectory that is typical of those produced by time-based BSR motion controllers. Cujo, a dog-like creature, propels himself forward by bounding repeatedly. The trajectory depicted in Figure 14 is the result of separate motion-synthesis problems that were solved seriatim: three distinct motion controllers are used to make Rex walk, then turn 90°, then walk again. The fitness function for Cujo was simply the distance traveled by his center of mass; Rex's fitness function was similar, but also included secondary terms that penalized sideways motion and falling

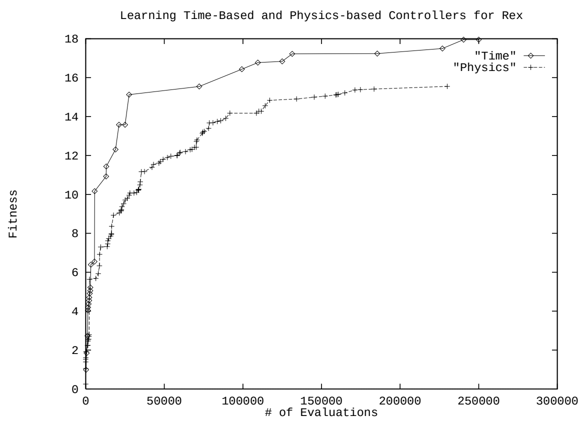


Figure 3: Learning curves for both kinds of motion controller.

down.

The progress of the search algorithm in finding one of Rex's motion controllers (for one of the walking motions) is shown in Figure 3: the top curve depicts the best time-based controller found so far, plotted against the number of controller evaluations performed. In typical fashion for the articulated figures considered here, rapid progress is made through the first 50,000 evaluations, after which progress is generally slower. One evaluation of a motion controller for Rex requires 1,000 time steps of simulation, and about 0.3 seconds of elapsed time on a Digital 3000/400 AXP workstation. An acceptable controller had therefore evolved in just under five hours.

2.2 Controllers with physical sensors

The simplicity and power of time-based BSR motion controllers make them very attractive when compared to the original BSR controllers with physical sensors [8]. Unfortunately, there is some evidence to suggest that the time-based approach may have inherent limitations. The most difficult motion-synthesis problem we have considered is that of making Bob the Biped walk. Bob, a biped with point feet (see Figure 15), is very unstable, so he falls over easily. To date, we have been unable to generate automatically a time-based motion controller for Bob that allows him to walk more than a couple of steps before keeling over.[3]

However, we have been successful in computing useful motion controllers for Bob when physical sensors are employed. These controllers [17, 18] are similar to the time-based BSR controller illustrated in Figure 1, except for the following key differences:

- Unlike a time-based stimulus, which is an interval on the time line, a physical stimulus is a multidimensional hyperrectangle in sense space (the space spanned by the physical sense variables).[4]
- Physical stimulus regions can overlap, so an arbitration procedure is required to choose between multiple applicable rules [17, 18]. Likewise, the stimuli do not necessarily cover the sense space. This situation also requires special treatment [17, 18].
- More SR rules are desirable for controllers with physical sensors (usually 10 rules suffice).

These differences in the underlying BSR representation necessitate changes in the search algorithm. The basic form of the algorithm remains the same (Figure 2), but the initialization and mutation procedures require modification. Every rule in a time-based controller is guaranteed to be executed during each time period of duration t_{per}, but no such a priori guarantee can be made for a controller with physical sensors. In fact, the addition of a randomly generated SR rule to an existing controller is exponentially unlikely to affect the generated motion, because a randomly generated stimulus region occupies an exponentially small fraction of the full volume of the sense space. Therefore, to ensure that non-creep mutations play a useful role, it is necessary to devise *relevance heuristics*—methods for generating random stimulus regions that are guaranteed to affect the generated motion. The relevance heuristic employed in the original work on BSR controllers [17, 18] prescribes that a newly generated stimulus hyperrectangle contain a point in sense space in common with the motion generated by the unmutated controller. With 3D figures, it was necessary to augment this relevance heuristic with information about the direction of the current path in sense space. Applying four rounds of this kind of mutation operation to the first set of 100 motion controllers was also found to be a useful strategy for enriching the initial population, as was the application of a low-pass filter to the values of all the physical senses to remove noise and high-frequency variation.

While we found that motion controllers with physical sensors generally attained fitness scores inferior to time-based controllers for stable articulated figures (see Figure 3), they were distinctly superior for Bob the Biped. The walk depicted in Figure 15 was one of two effective, forward-facing walking strategies discovered.[5] (The strategy not shown here was a highly periodic shuffling motion.) We also noticed a considerable variation in the quality of the BSR controllers found by different runs of the same algorithm when physical sensors were used (see Figure 4), which suggests that multiple runs of the algorithm (or a larger population of candidate solutions) might be best for 3D articulated-figure motion-synthesis problems with this level of difficulty.

3 3D Mass-Spring Models

In extending the BSR approach to motion-synthesis problems for 3D mass-spring models, we decided to stick with one character, Mr. Jello, for initial experimentation. Mr. Jello is a cube-like character

[3]Because of the particulars of the physical model and the SR-rule representation, getting Bob to walk is more like trying to learn to use stilts than trying to learn regular walking. By permitting one's ankles to bend upon contact with the ground, a human walker can maintain balance even when on only one foot. If Bob had true feet, he could not do the same because his internal deformations are determined kinematically, without information about the ground. The additional computational cost of a more general physical model and simulator able to handle these phenomena might place results comparable to those reported here beyond the reach of our current serial hardware.

[4]The physical senses used in our 3D-articulated-figure controllers are the height of the center of mass, the linear velocity, the angular velocity, the up vector (the latter three senses are expressed as 3D vectors in a figure-centric local coordinate system), contact forces and collision impulses for each possible contact point, and internal Euler angles for each flexible joint. The increased complexity of each SR rule and the desirability of having more rules in a BSR controller with physical sensors means that 880 floating-point numbers are used to specify a motion controller for Bob the Biped. The corresponding number for Rex (Figure 14) is 1,280.

[5]To our amusement (and then to our annoyance), several controllers were computed that achieved net forward movement, but in such a way that Bob either gyrated or faced backward through much of the motion. To eliminate this undesirable behavior, we modified the fitness function to penalize trajectories in which Bob experienced a net rotation about the vertical axis.

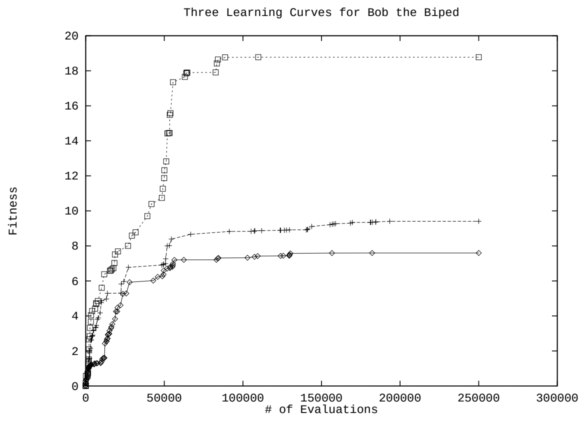


Figure 4: Variability in search-algorithm performance.

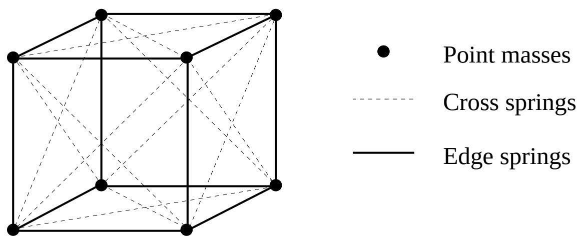


Figure 5: Mr. Jello's internal structure.

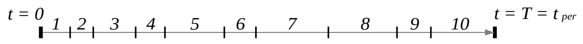
who is modeled as eight point masses connected by 24 springs, 12 of which run along the edges of the cube, and 12 of which traverse face diagonals, as shown in Figure 5. The springs serve as actuators for Mr. Jello: he moves by dynamically changing the rest lengths of his springs.[6] Our goal was to synthesize motion controllers that would make Mr. Jello perform simple tasks like walking, tumbling, and jumping.

Because of their relative simplicity, we began by considering time-based BSR controllers for these motion-synthesis problems. The time-based controllers used for Mr. Jello are similar in principle to those used for the various 3D articulated figures (§2.1), but they differ in the following details:

- Each time interval is represented by the midpoint of the interval, t_{mid}. (This seems like an inconsequential difference relative to the scheme in which interval boundaries are represented explicitly, but it appears to produce a small but noticeable improvement in the controllers generated.)
- A response consists only of a list of rest lengths, λ_i, for each spring i. (All edge and cross springs have the same stiffness and damping characteristics, respectively, which are determined by four global constants set by the user.)

For the trajectories shown in Figures 16–18 and on the accompanying videotape, we used controllers that had 10 rules, and a value

[6]The physical simulator we used for this research is straightforward and unsophisticated. It uses fixed-time-step Euler integration, the simple Coulomb model of dynamic friction, and impulse-based collision simulation for the point masses.



Rule	*Stimulus*	*Response*
1	t^1_{mid}	$(\lambda^1_1, \lambda^1_2, \ldots, \lambda^1_{24})$
2	t^2_{mid}	$(\lambda^2_1, \lambda^2_2, \ldots, \lambda^2_{24})$
$\vdots$	$\vdots$	$\vdots$
10	t^{10}_{mid}	$(\lambda^{10}_1, \lambda^{10}_2, \ldots, \lambda^{10}_{24})$

Figure 6: An illustration of a time-based BSR motion controller for Mr. Jello.

```
Initialize and evaluate a single controller
for evaluation = 1 to 50,000
  Mutate the controller and re-evaluate it
  if the new controller is better than the old one then
    Replace the old controller with the new one
  end if
end for
```

Figure 7: Stochastic hill climbing.

of $t_{\text{per}} = T$ (see Figure 6). The latter decision led to aperiodic motions, which we considered to be more interesting for Mr. Jello. Thus a complete time-based motion controller for Mr. Jello consists of $10 \times 25 = 250$ floating-point numbers. (We do not count the stiffness, damping, and t_{per} constants as independent parameters because they are user specified.)

To find good controllers automatically, we experimented with three different search algorithms, shown in Figures 7–9.[7] The first algorithm (Figure 7) is a simple form of stochastic hill climbing, the second (Figure 8) follows a standard simulated-annealing approach [14, 5], and the third (Figure 9) is a slightly simplified form of the stochastic-population-hill-climbing algorithm encountered earlier (Figure 2). The remaining unspecified details concern the initialization and mutation steps: all random initial values are uniformly distributed, and the different kinds of mutation operator and their respective probabilities of application are given in Table 1.[8]

For the problems we considered, all three search algorithms performed well. Sample trajectories for three problems are illustrated in Figures 16–18: Mr. Jello learns to jump, tumble, and shuffle. (More sample trajectories are shown on the accompanying videotape.) Learning plots for the three algorithms are shown in Figures 10–12 for the task of Mr. Jello jumping. In these learning plots, a point is plotted for each evaluation made. Thus it is possible to see not only the best solution found so far—as can be seen in learning-curve plots like those in Figures 3 and 4—but also how much time the algorithm is spending investigating good, medium, and bad solutions. Even though the differences in performance were small, stochastic population hill climbing fared best for the

[7]In choosing stochastic population hill climbing as the search algorithm for motion-synthesis problems involving 3D articulated figures, we were relying on a comparative study of search algorithms for motion-synthesis problems involving 2D articulated figures [9, 8]. Mass-spring lattices are sufficiently different that we thought it prudent to conduct a similar empirical study (though on a smaller scale) to determine an effective search algorithm for this type of physical model.

[8]The values in the table are probably not essential to proper operation of the algorithm. They are stated here merely for completeness.

```
Initialize and evaluate a single controller
Set the temperature $\mathcal{T} = \mathcal{T}_{\text{init}}$
for evaluation = 1 to 50,000
  Mutate the controller and re-evaluate it
  Compute $\Delta E$, the change in the fitness function
    caused by the random mutation
  if the new controller is worse then
    Undo the mutation with probability $P = 1.0 - e^{-\Delta E/\mathcal{T}}$
  end if
  Decrease $\mathcal{T}$ according to the annealing schedule
end for
```

Figure 8: Simulated annealing.

```
Initialize population to be 100 random controllers
Rank order the population
for evaluation = 1 to 50,000
  Select a controller randomly
  Mutate the controller and re-evaluate it
  Insert the new controller into the population by rank
  Delete the lowest-ranked controller
end for
```

Figure 9: Another form of stochastic population hill climbing.

Probability	*Mutation Operation*
0.1	Creep a time-interval midpoint t_{mid}
0.2	Creep a rest length λ
0.2	Creep every rest length λ^i for some i
0.1	Randomize a time-interval midpoint t_{mid}
0.2	Randomize a rest length λ
0.2	Randomize every rest length λ^i for some i

Table 1: Mutation operations for mass-spring models.

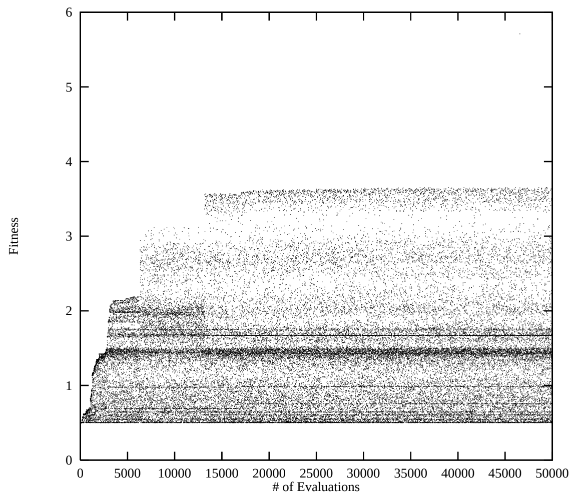


Figure 10: Learning plot for stochastic hill climbing.

majority of problems we considered.

One evaluation of a motion controller for Mr. Jello requires about 670 time steps of simulation, and about 0.6 seconds of elapsed time on a DECstation 5000. On a more current machine, *e.g.*, a Digital 3000/400 AXP workstation, this figure would decrease by about a factor of four. Given that only 15,000 evaluations were needed to get good solutions for the problems considered here, a motion controller for a simple motion-synthesis task involving a character of Mr. Jello's complexity can probably be produced in about 40 minutes on a modern mid-range workstation.

4 Conclusions and Future Work

The results described here provide the first solid evidence that motion synthesis for 3D characters can be done automatically in a variety of challenging cases.

In future work on 3D articulated figures, we would like to address the following issues:

- Better physical models and a more general way of specifying a response in the BSR framework are needed in order to get more realistic motion. For example, the current response model does not allow us to incorporate a foot that will automatically lay flat on the ground when the foot is planted—this is the reason why all the articulated figures walk on point feet.
- Fitness functions that reward graceful motion appropriately are also needed to get motion that is visually more plausible. Our current algorithms sometimes generate motion that, while physically realistic, does not match the kind of motion we expect to see for a given character. Preliminary results with 2D articulated figures [8] and with the 3D characters considered in this paper suggest that much can be gained by making fitness functions more sophisticated.
- The ability to concatenate in time previously computed motion controllers[9] for a specific character would allow an ani-

[9]We anticipate that controllers with physical senses can be concatenated with better

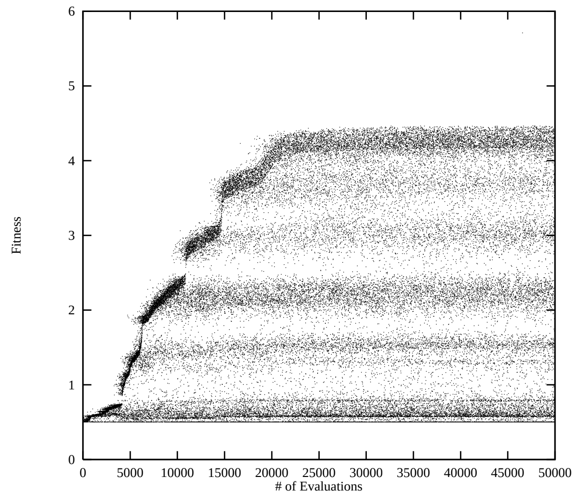


Figure 11: Learning plot for simulated annealing.

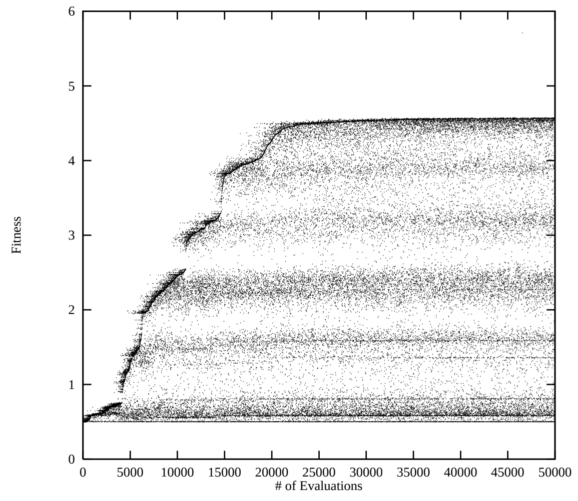


Figure 12: Learning plot for stochastic population hill climbing.

mator to assemble complex, composite motions for that character in real time, without having to wait while new motion-synthesis problems are solved. An animation editor that supports this kind of interaction for 2D articulated figures has been tested [8], but it remains to be seen whether this approach can be generalized to 3D.

For 3D mass-spring models, our focus is on the following issues:

- We need to gain experience with additional mass-spring models. We have begun to work with a worm-like character of greater complexity than Mr. Jello, but it is too early to know whether the simple methods that work well for Mr. Jello apply equally well to more complex mass-spring lattices.
- An animation editor of the kind discussed above for 3D articulated figures would also be useful for 3D mass-spring models.
- Although Mr. Jello has a simple symmetric structure, we intentionally did not take advantage of symmetry in specifying the form of the motion controller or the nature of the search algorithm, because we wanted to discover how much could be achieved with a simple, unstructured, general approach to the problem. Taking advantage of structural symmetry in 3D mass-spring models might well provide additional leverage for future motion-synthesis techniques. A related idea is to try to synthesize motion in terms of *modal dynamics* [19] for mass-spring lattices.
- Perhaps the most important outcome of this work will be renewed interest in the approximate Jell-O equation of Heckbert [13], $\mathbf{J} = 0$. Here we have been able to cast this fundamental equation in time-dependent form, and, paradoxically, it appears that

$$\frac{\partial}{\partial t}\mathbf{J} \neq 0.$$

—maybe.

5 Acknowledgments

This work was supported in part by an NSF grant to Martin Karplus, and by an equipment grant from Digital Equipment Corp. HP is grateful for a summer internship at Digital Equipment Corporation's Cambridge Research Lab.

References

[1] N. I. Badler, B. A. Barsky, and D. Zeltzer, editors. *Making Them Move: Mechanics, Control, and Animation of Articulated Figures*. Morgan Kaufmann, San Mateo, CA, 1991.

[2] R. D. Beer and J. C. Gallagher. Evolving dynamical neural networks for adaptive behavior. *Adaptive Behavior*, 1(1):91–122, Summer 1992.

[3] L. S. Brotman and A. N. Netravali. Motion interpolation by optimal control. *Computer Graphics*, 22(4):309–315, August 1988.

success than time-based controllers, which are not sensitive to initial conditions.

[4] A. Bryson and Y. Ho. *Applied Optimal Control: Optimization, Estimation, and Control*. Hemisphere Publishing Corp., 1975.

[5] V. Cerny. A thermodynamical approach to the travelling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory Applications*, 45:41–51, 1985.

[6] M. F. Cohen. Interactive spacetime control for animation. *Computer Graphics*, 26(2):293–302, July 1992.

[7] H. de Garis. Genetic programming: Building artificial nervous systems using genetically programmed neural network modules. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 132–139, Austin, Texas, June 1990.

[8] A. Fukunaga, L. Hsu, P. Reiss, A. Shuman, J. Christensen, J. Marks, and J. T. Ngo. Motion-synthesis techniques for 2D articulated figures. In review, 1994.

[9] A. Fukunaga, J. T. Ngo, and J. Marks. Automatic control of physically realistic animated figures using evolutionary programming. In *Proceedings of the Third Annual Conference on Evolutionary Programming (EP94)*, San Diego, CA, February 1994. To appear.

[10] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, San Diego, CA, 1981.

[11] H. Goldstein. *Classical Mechanics*. Addison-Wesley, 2nd edition, 1980.

[12] J. K. Hahn. Realistic animation of rigid bodies. *Computer Graphics*, 22(4):299–308, August 1988.

[13] P. S. Heckbert. Ray tracing JELL-O brand gelatin. *Computer Graphics*, 21(4):73–74, July 1987.

[14] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[15] J. R. Koza and J. P. Rice. Automatic programming of robots using genetic programming. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 194–201, Menlo Park, California, 1992. American Association for Artificial Intelligence.

[16] P. Maes and R. A. Brooks. Learning to coordinate behaviors. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 796–802, Menlo Park, California, 1990. American Association for Artificial Intelligence.

[17] J. T. Ngo and J. Marks. Physically realistic motion synthesis in animation. *Evolutionary Computation*, 1(3):235–268, 1993.

[18] J. T. Ngo and J. Marks. Spacetime constraints revisited. In *SIGGRAPH '93 Conference Proceedings*, pages 343–350. ACM SIGGRAPH, Anaheim, CA, August 1993.

[19] A. Pentland and J. Williams. Good vibrations: Modal dynamics for graphics and animation. *Computer Graphics*, 23(3):215–222, July 1989.

[20] S. Tu, D. Terzopoulos, and E. Fiume. Go Fish! ACM SIGGRAPH Video Review, Issue 91, 1993.

[21] M. van de Panne and E. Fiume. Sensor-actuator networks. In *SIGGRAPH '93 Conference Proceedings*, pages 335–342, Anaheim, CA, August 1993. ACM SIGGRAPH.

[22] A. Witkin and M. Kass. Spacetime constraints. *Computer Graphics*, 22(4):159–168, August 1988.

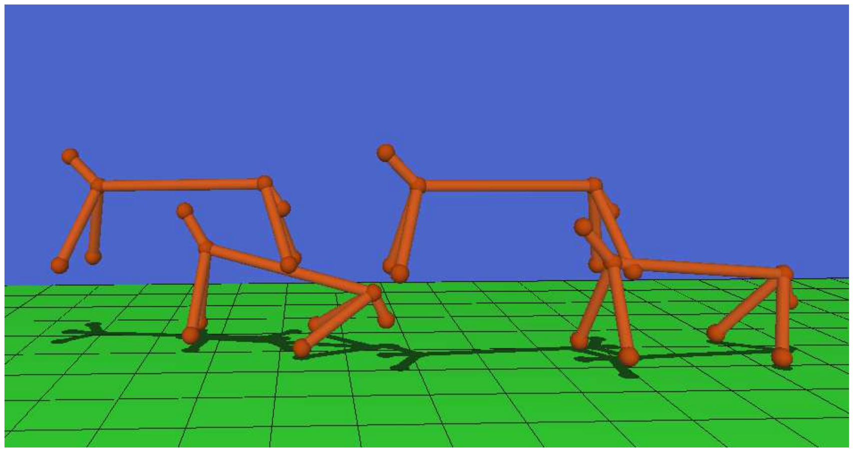


Figure 13: Cujo bounds from right to left.

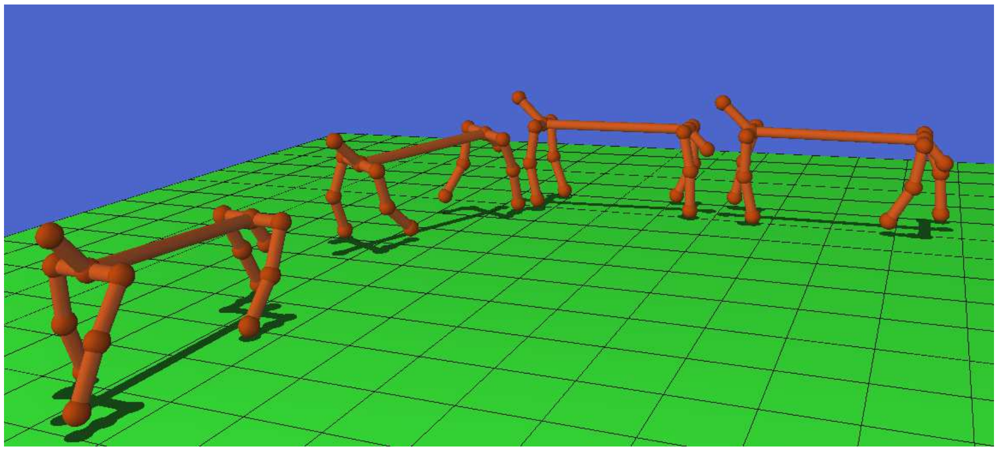


Figure 14: Rex walks, turns, and walks again.

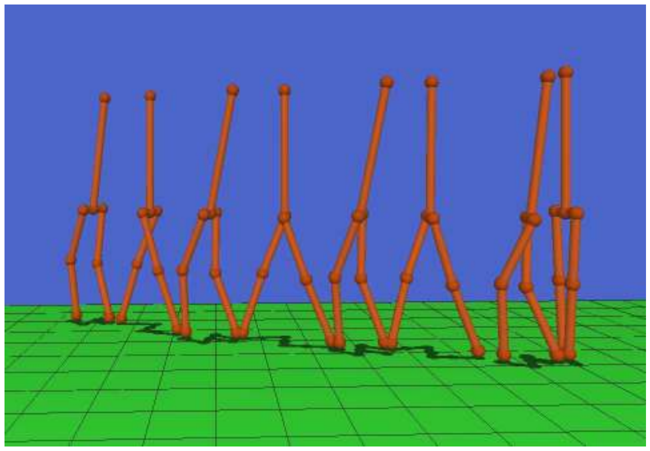


Figure 15: Bob the Biped walks.

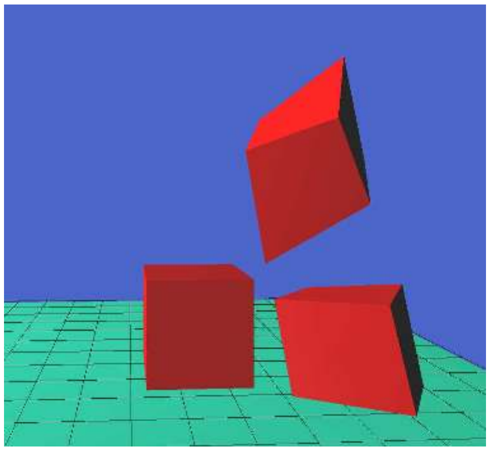


Figure 16: Mr. Jello jumps.

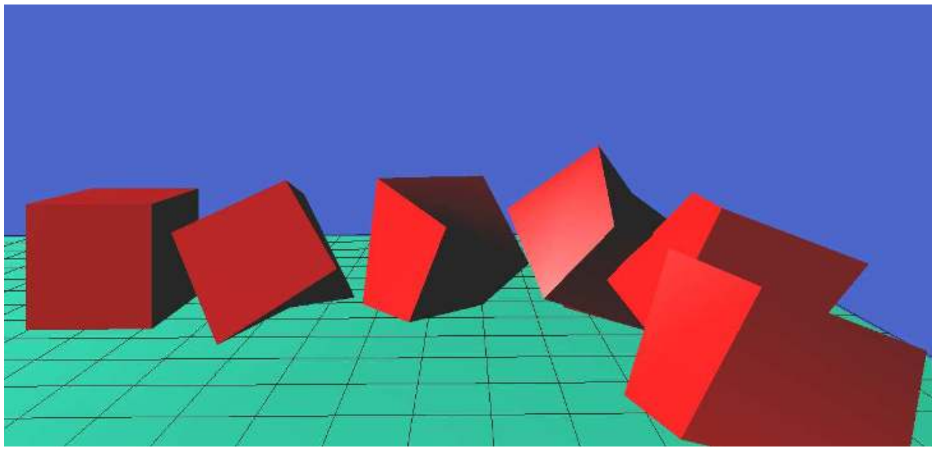


Figure 17: Mr. Jello walks by tumbling.

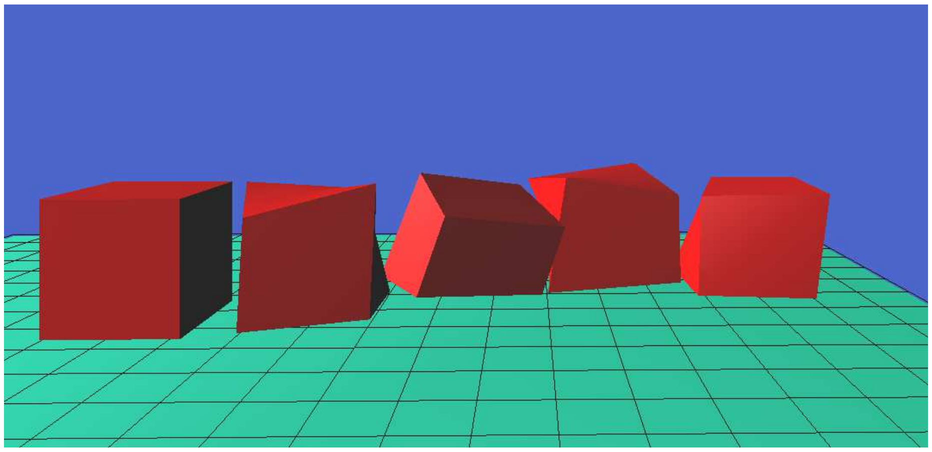


Figure 18: Mr. Jello walks by shuffling.