Reusable Motion Synthesis Using State-Space Controllers

Michiel van de Panne*, Eugene Fiume** and Zvonko Vranesic*

Department of *Electrical Engineering/**Computer Science
University of Toronto
Toronto, Canada M5S 1A4

Abstract

The use of physically-based techniques for computer animation can result in realistic object motion. The price paid for physically-based motion synthesis lies in increased computation and information requirements. We introduce a new approach to realistic motion specification based on state-space controllers. A user specifies a motion by defining a goal in terms of a set of destination states. A state-space controller is then constructed, which provides an optimal-control solution that guides the object from an arbitrary starting configuration to a goal. Motions are optimized with respect to time and control energy. Because controllers are specified in terms of destination states only, it is easy to reuse the same controller to produce different motions (from different starting states), or to create a complex sequence of motions by concatenating several controllers. An implementation of state-space controllers is presented, in which realistic motions can be produced in real time. Several examples will be considered.

CR Categories: I.3.7 [Computer Graphics]: Three Dimensional Graphics and Realism – animation; I.6.3 [Simulation and Modelling]: Applications; G.1.6 [Constrained Optimization].

1 Introduction

Computer-assisted animation embodies a wide variety of motion-synthesis techniques. Kinematic approaches still predominate and are likely to do so, but physically-based techniques are gaining in popularity. The cost of greater physical realism has been increased computational cost and information requirements. Moreover, it is not usually possible to reuse a previously-computed motion in other contexts.

The physical modelling of natural phenomena or motions requires physical *simulation*. In such cases, one typically defines some initial conditions and then invokes a physical simulation of the model. In a general animation system, some notion of motion *control* is also required. In this case, a desired goal is specified, and the system attempts to generate a suitable series of forces and torques on a moving object in order to reach the goal from some initial configuration. The control problem is difficult, and it is the focus of this paper. We propose the use of encapsulated optimal control laws in the form of *state-space controllers*. The same controller can be used in many different situations, and it can be concatenated with other controllers to produce seamless composite motions. The next section gives an overview of motion specification techniques. We then describe our approach, some examples of its use, and future work.

[0]The financial assistance of the Natural Sciences and Engineering Research Council of Canada, and of the Information Technology Research Centre of Ontario, is gratefully acknowledged.

2 Previous Work

2.1 Kinematic Motion Synthesis

Complex motion synthesis has traditionally been performed kinematically using interpolation mechanisms such as keyframing [9-11,18,21,23,27,33]. Approaches to simplifying the specification of key positions include inverse kinematic solutions [5,14,19], and procedural position specification [10,14,31,41]. Keyframes can also be obtained from real moving objects with the use of rotoscoping. Techniques for the kinematic specification of cyclic motions such as walking or hopping have also been investigated [14,41].

2.2 Physically-Based Motion Synthesis

To satisfy the physical constraints of motion, animators have turned to physical simulation [3,7,16,38]. Simulation guarantees realistic, but not necessarily desirable, motion. Achieving the desired motion is a difficult control problem. Objects such as articulated figures (AFs) are controlled by internal torques applied at the joints. The *control problem* is to find the function of the torques over time that produces the desired motion.

Several methods of generating the required torque functions have been suggested. One method requires the user to specify torques directly [3,13,37,38]. It is in general difficult to come up with the necessary torques to perform desired motions through a process of trial and error. One need only observe a backhoe operator to see that this is true. An alternative is to use inverse dynamics to solve for the torques required to produce a known acceleration [6,8,16,17]. This approach is useful when it is desired to have a portion of an object follow a particular path, or to have an initial guess of the torques needed to perform a motion. One can also obtain the

required torques by using the desired joint positions as set-points for closed-loop controllers [3,20]. This models robots controlled by position-servos, permitting kinematic control while still utilizing the equations of motion.

A solution to a specific inverse-dynamics problem can be encapsulated in a dedicated *controller* or *control procedure*. This method cleanly partitions the control from the dynamics equations. Although the concept has often been suggested in the literature, the construction of the required controller or control procedure has always been left to the animator or is constructed using *a priori* information, such as clinical data [8,16,41]. Many controllers have been carefully hand-engineered to solve specific problems. These include mechanical bipeds [26,28,35], human walking [8], six-legged robots [24], snakes [25], and one-legged hopping robots [30].

Existing controllers have thus far been carefully tuned to solve a specific problem. Consequently, they are not likely to be flexible, reusable, or optimal with respect to time and energy constraints. The state-controllers proposed in this paper seek to overcome these shortcomings.

2.3 Optimal Control Methods

Good solutions to the motion control problem have been achieved by viewing it as a problem in optimization. A motion can be formulated as a two-point boundary problem with the start and end points of the motion sequence being constraints in state space that must be met. An optimization function reflecting the control energy expended and time taken for the motion [1,7,40] is then minimized to produce the optimal solution (see Figure 1).

The method of *space-time constraints* by Witkin and Kass uses a variant of sequential quadratic programming to solve the optimization problem, and generates convincing motion [40]. The user provides expressions for the total kinetic energy of the object and must express all other constraints in a mathematical form. This is something that animators are unlikely to be adept in doing. The solution is also costly to compute. Brotman and Natravali [7] present a similar approach to solving the control problem, but make use of a different mathematical formulation. The same problems exist as for the method of space-time constraints. Neither paper suggests the possibility of saving a motion for future reuse. A generalization of these approaches would be to define a large set of optimal-control solutions in the form of a general control law.

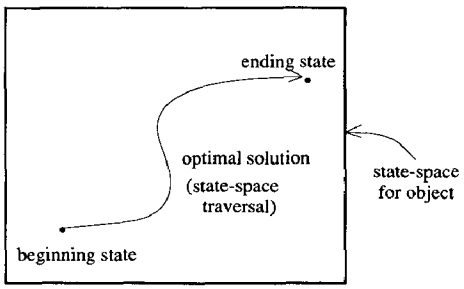

Figure 1: *Optimal-control motion synthesis.*

3 State-Space Controllers

3.1 Overview

We now introduce the main contribution of this paper. A *state-space controller* (SSC) defines a set of control torques that guides an object to a specified goal from a large domain of initial configurations, in a fashion that optimizes time taken and energy expended. A goal is characterized by a set of destination states, and depending on the nature of this set, several classes of motion are possible. Simple motions include those with a stationary destination state. A non-stationary destination state will result in periodic motions such as hopping or walking. One can also define motions with goals consisting of many destination states. This captures motions in which the terminal velocities or positions of parts of the object are irrelevant. For example, in a race, it is irrelevant which part of the body crosses the finish line first. Lastly, "conditional" motion can be defined: given more than one destination state, perform the easiest motion.

While individual SSCs may define interesting motions in themselves, the real power of the approach lies in the ability to concatenate SSCs to create a composite motion. SSCs can also be concatenated with other motion-generation techniques such as motor programs and key-framing, as we shall see later. Consider the following example. Suppose Luxo (the jumping lamp, Figure 2) is to hop forward several times, take a long forward jump to miss a ditch, and then do a backflip out of elation of surviving. Given an appropriate set of controllers, a user can build (and view) an animation sequence by writing a script consisting of the desired sequence of controllers. Figure 3 depicts the interaction of controllers with an animation system.

The concatenation of SSCs may be specified in two ways. One way is to run each SSC to its conclusion or for a specified duration. The terminal state of this SSC will then become the starting state of the next SSC, as in the Luxo example above. An alternative approach is to specify *state-space breakpoints*, in which an SSC is associated with particular regions of state space. An object may thus have a complex interleaving of SSCs attached to it.

3.2 Motion through State-Space

The *state* of an object represents all the information required to specify the position and velocity of every point on the object. The *state space* of the object is the set of all possible states that the object can assume. The state of a moving ob-

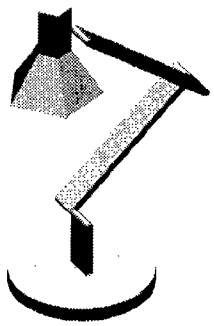

Figure 2: *The jumping lamp.*

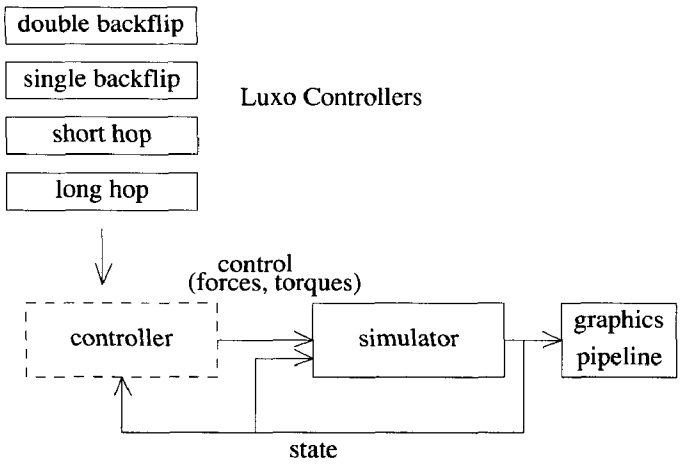

Figure 3: *Using state-space controllers.*

ject changes with respect to time. Consider a swinging pendulum, a simple articulated object (Figure 4). The state of a freely-swinging pendulum continuously changes with time under the force of gravity. Its angular velocity plotted with respect to time is a near-sinusoid, and likewise for the pendulum angle. When these two functions are combined to obtain the state, the resulting path through the state-space of the pendulum is the almost circular path shown in Figure 5. By exerting control torques, it is possible to influence the state-space trajectory taken by an articulated object. We can use this technique to guide an object toward a goal. This is the central principle underlying state-space controllers.

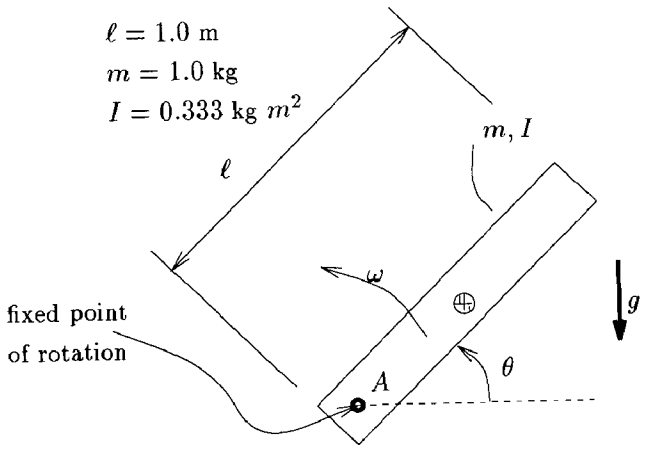

Figure 4: *The pendulum has mass m, length l, moment of inertia I, angular velocity ω; θ is the CCW angle from the positive x-axis, and g is the force of gravity.*

3.3 Specification and Concatenation

The motion to be executed by a controller is expressed in terms of a set of destination states for the object. The state transitions from an arbitrary start state to a destination state are optimized with respect to the time and energy taken to perform the motion, and can only use the internal torques that fall within the range of torques capable of being exerted by the object. The controller thus functions as a control law, which defines the optimal-control solutions to a one

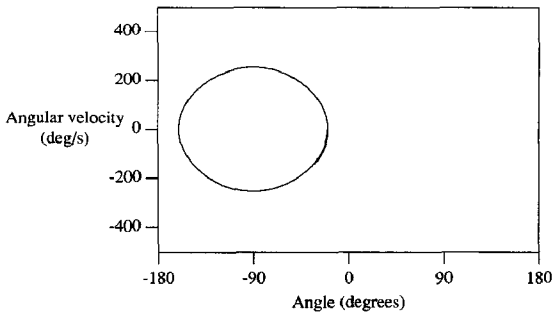

Figure 5: *Pendulum swing as represented in state space.*

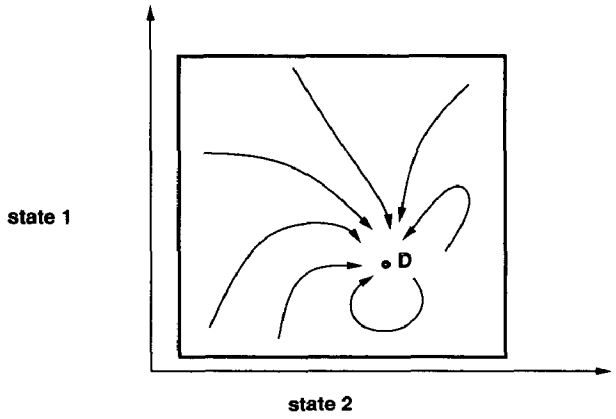

Figure 6: *A one boundary-point state-space controller.*

boundary-point problem (the destination state), as shown in Figure 6. In this case, point D represents a destination state in a two-dimensional state-space. The arrows show some possible state-space paths that will be followed by the object for various initial states.

A controller is defined over a user-specified, bounded region of state-space called its *domain*. Figure 7 shows the regions of state space over which controllers A, B, and C are defined, as well as three respective destination states, D_a, D_b, and D_c. Let S_0 represent the initial state of the object. Suppose that while controller A guides the object toward D_a, it is desired to invoke controller B. Similarly, assume that it is next desired to change to controller C. The solid line in Figure 7 indicates one such set of changes, where S_1 and S_2 are the states at which the new controllers are invoked. The only constraints on S_1 and S_2 are that $S_1 \in$ Domain(A) $\cap$ Domain(B) and $S_2 \in$ Domain(B) $\cap$ Domain(C). Clearly, concatenation of controllers need not occur only at destination states.

3.4 Structure of Controllers

A controller is defined in a local co-ordinate system, and defines relative motions. Formally, a controller denotes a vector function $f : \mathcal{S} \rightarrow \mathcal{T}$, where $\mathcal{S}$ is a state space and $\mathcal{T}$ is a set of torque tuples. It is entirely possible to define procedural controllers based on motor programs or kinematic interpolation (see below). We shall focus our discussion on the automatic generation of controllers that solve one-point optimal-control problems.

In our current scheme, a continuous state-controller f is synthesized from a discrete table of torques in state space. An n-

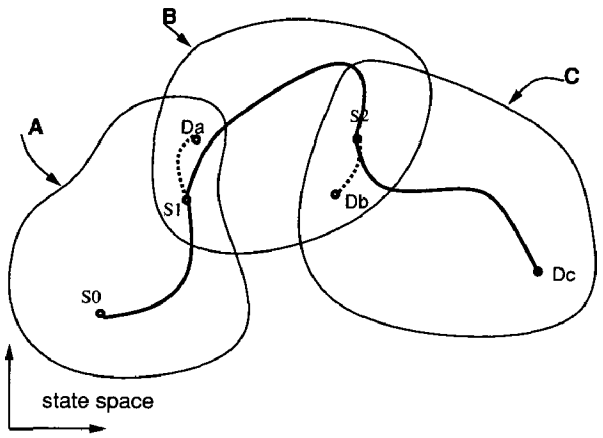

Figure 7: *Valid domains for exchanging controllers.*

dimensional volume forming the controller's domain is regularly subdivided into small n-dimensional cubes. Here n represents the number of dimensions of the object's state space, or alternatively how many numbers are required to specify the object's state. Table elements correspond to the corners of the small hypercubes. The torques provided by the tables are made continuous through n-linear interpolation in all dimensions of the object's state space (e.g., bilinear interpolation in two dimensions). Hierarchical or non-uniform sampling is advisable, and is planned. We expect to use better-quality reconstruction filters when we move to non-uniform sampling.

3.5 The Generation of Controllers

It is infeasible (and inefficient) to solve a one-point optimization problem by solving many instances of two-point problems. We instead employ a divide-and-conquer technique called *dynamic programming*. The principle of dynamic programming is illustrated in Figure 8. Suppose path AC is optimal. Then the optimal path from any state P on AC to state C is given by the subpath PC of AC. If a better alternative path had existed (as shown by the dashed line), the optimal path from A to C would contain this subpath. This property is true of any monotonically-increasing optimization function of object motion, such as time or expended energy.

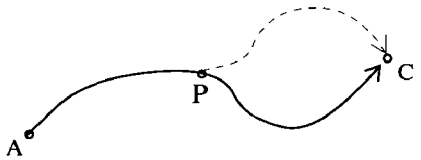

Figure 8: *The principle of dynamic programming.*

Figure 9 illustrates how dynamic programming can be applied to the generation of state-space controllers. Suppose optimal solutions are known for states located in region 1, which contains destination state D. To calculate the optimal solutions for states located in region 2, we solve a local optimization problem to get from the current state to the edge of region 1. This provides a composite optimal solution for both regions. Clearly, then, an appropriate strategy for controller generation is to work backward from D, radiating the solutions outward.

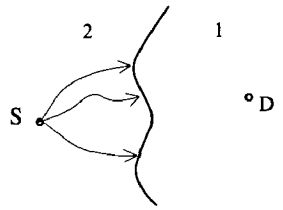

Figure 9: *Dynamic programming applied to an SSC.*

A more detailed picture of the local optimization problem to be solved is shown in Figure 10. θ_n and ω_n are the angular position and angular velocity of a link of an articulated figure. The quantization intervals of the table entries are given by q1 and q2. The figure shows the possible state-space trajectories as projected onto the $\theta_n\omega_n$ plane of state space. The trajectory taken from state S depends on the angular acceleration α_n, which in turn depends on the control value (torque vector) contained in the table element corresponding to point S. The *local optimization problem* is to find the control value at S that minimizes the optimization function.

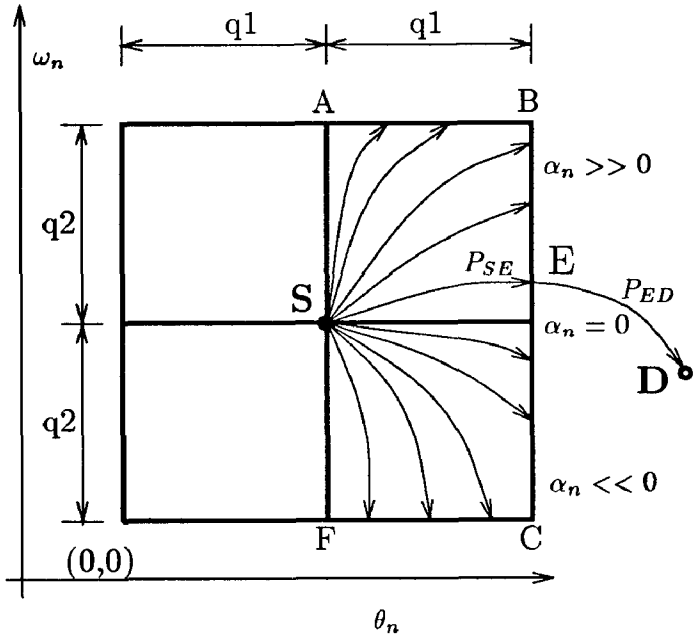

Figure 10: *The effect of various torques applied at* S.

The optimization function we use is given by two terms:

$$f_{opt}(P_{SD}) = t_{P_{SD}} + \int_0^{t_{P_{SD}}} KT(t)\,dt. \qquad (1)$$

The first term represents the time taken to perform the motion, while the second term measures the energy expended [1,40]. P_{SD} represents a path through state space from an initial state S to a destination D, and $t_{P_{SD}}$ represents the time taken to traverse this path. K is a user-defined vector of constants that specifies the time/energy tradeoff, and $T(t)$ represents the applied torques (control values). A small value of K will minimize the *time* taken to perform the motion. A large value minimizes the *energy* used to perform the motion. Typical values for K depend on the magnitude of the forces and torques capable of being exerted by the object. For our

pendulum example, a value of $K = 1$ implies we are willing to exchange one second of time taken in reaching the destination state for the "energy" expended to exert a torque of 1 Newton-metre for one second. Our "energy" term is more properly a measure of the effort required to perform a motion. Other objective functions are possible.

It is required to find the torque vector at S that produces a state-space path P_{SD} such that $f_{opt}(P_{SD})$ is minimized. The solution we currently employ consists of calculating $f_{opt}(P_{SD})$ from a set of samples of torque vectors and then choosing the vector that results in the minimal value of the optimization function. The samples are chosen by uniformly sampling the space of torque vectors (since this space is bounded). This is admittedly a slow, brute-force approach whose sole virtue is that it works. For each sample torque vector, the state-space trajectory is calculated by simulating the motion of the object. The trajectory is followed until the object enters a region of state space in which the value of the optimization function is already known. For the path P_{SD} this is indicated as point E in Figure 10.

Once the exit state is known, the value of the optimization function can be determined from the equation

$$f_{opt}(P_{SD}) = t_{P_{SE}} + \int_0^{t_{P_{SE}}} KT(t)\,dt + f_{opt}(P_{ED}). \quad (2)$$

For the local optimization problem, the values of the optimization function are assumed to be known at neighbouring points (points A, B, C, and F in Figure 10) and are linearly interpolated between the points. The first two terms in Eq. 2 are simply $f_{opt}(P_{SE})$ (Eq. 1), while the last term represents the value interpolated between the values of the optimization function at B and C. Several iterations of controller computation are required, because the values of the optimization function at neighbouring points may be unknown for the first iteration. In Figure 10, this means that the computation of a control value for state S is only accurate if the values of the optimization function are accurately known at points A, B, C, and F. It is difficult to sequence local optimizations such that each local optimization always uses accurate information. It is simpler to approximate this order and repeat this for several iterations. For each iteration, the value of the optimization function will decrease. When the maximum such change is less than a user-specified value, the solution is assumed to be complete. An upper limit can also be placed on the number of iterations to use. Technically, this approach only yields a local minimum, but the local minimum found has always been satisfactory.

3.6 The Size of State Space

While generating controllers for objects with few degrees of freedom is feasible, in our current implementation the number of state-space dimensions becomes a problem for more complex articulated figures. Both the time-complexity of the algorithm and the size of the state-space control table are exponentially dependent on the number of state-space dimensions. The size of the table is given by $n_s^{d_s}$, where d_s is the number of state-space dimensions (typically twice the number of degrees of freedom since each degree of freedom has a position and velocity), and n_s is the number of samples per state-space dimension. The time-complexity of the algorithm is given by $O(n_s^{d_s} n_t^{d_t})$, where d_t is the number of torques or forces to be applied, and n_t is the number of samples per torque or force. The problem arises because state space is defined as a large bounding hypercube that is uniformly sampled. There are two main aspects to the problem. First, the bounding hypercube is almost always far too large. A user should be allowed to eliminate irrelevant areas of state space. Second, uniform sampling is far from optimal. The sampling rate close to the destination should be high, but the rate should fall off dramatically with "distance" from the destination. Sampling artifacts do not seem to affect the stability of the controller significantly. If the interpolation of control values between table entries causes the object to drift from the optimal path to the destination state, the controller corrects itself because the control values being applied are always computed based on the current state. This allows the controller to function properly, even if the object's state changes suddenly as a result of a collision involving the object. There is a considerable amount of interesting research to be performed in studying the placement and frequency of control-table values in state space.

4 Dynamics Formulation

To embed state-space controllers effectively in our animation system (see below) it is necessary to perform physical simulations quickly. To date, we have focused our formulation on the planar forward dynamics of articulated figures (AFs). The extension to fully three-dimensional articulation is straightforward. (It involves adding a Coriolis-force term and changing moments of inertia from scalars to tensors so that the moment of inertia for a link becomes dependent on the current axis of rotation.) We assume that AFs have joints and links that can be represented as a tree, with one link serving as the root link. A link can have any number of child links, which are connected by a rotating joint.

The recursive Newtonian dynamics formulation we use is well known [4], and is based on two fundamental equations that balance the forces and moments exerted on each link. Ultimately, we solve for the linear acceleration of the root link and the angular acceleration for all the links with respect to the root link. The accelerations are then numerically integrated with an adaptive time step to determine the new velocity and position of the links. What is important about our approach is that, unlike most previous implementations of dynamics formulations, the equations of motion are formed symbolically, directly from the basic masses, forces, and inertial properties of the object. This is very useful representation. We have written a *dynamics compiler* which can compile a brief physical description of the AF into the desired equations of motion. The equations generated are of the form $Ax = b$ where A and b are dependent on the physical properties and configuration of the links, and x represents the vector of unknown accelerations. The output of the dynamics compiler gives the symbolic value of each of the elements of the matrix A and the vector b. In the implementation the values of A and b are output as lines of 'C' code so that the equations of motion can be compiled and placed in a library. This makes our system more portable, and it allows us to separate the equations of motion from the remainder of the system.

Special-purpose motions could also be compiled and placed into this library. A symbolic representation of the equations of motion also allows a programmer or a compiler to simplify the expressions to be computed.

For objects containing about 12 joints or fewer [36], the equations of motion are best solved using LU-decomposition with back-substitution [29] once the values of A and b are calculated. This solution has a complexity of $O(n^3)$ for an AF with n links. $O(n)$ methods do exist, but in practice they are only useful for AFs with many degrees of freedom [2,12].

Almost all interesting animations of objects involve collisions with the environment and have other constraints on the motion of the object, such as joint limits and friction. We use springs and dampers to implement collisions and joint limits [13,16,39].

We have developed an animation system that incorporates the above dynamics formulation, and in which controllers can be generated offline and subsequently scripted for use in realtime animation. Figure 11 depicts our animation system, called *Mosys*.

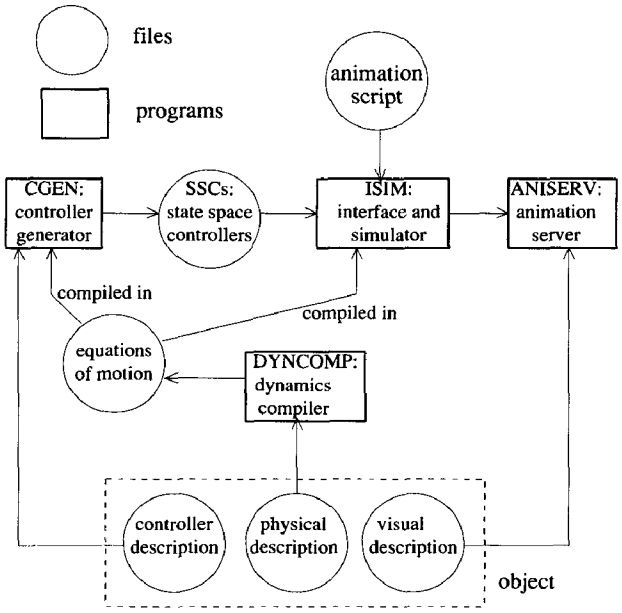

Figure 11: *Mosys implementation.*

5 Examples

We now consider several examples of the generation of state-space controllers using Mosys. The generation of controllers is entirely separate from their use, and are usually computed offline. Once computed, they allow realtime animation of the objects for which they were constructed.

5.1 Periodic Motions

A periodic motion, such as a link swinging or Luxo-lamp hopping corresponds to a cycle in the object's state-space. A controller that produces a periodic motion is created by specifying a destination state with non-zero velocities. As the controller guides the object along the state-space cycle, the object performs a periodic motion corresponding to the states passed through along the way. One of the many possible periodic, swinging motions of a frictionless, free-swinging pendulum is shown by the state-space cycle in Figure 5.

Motions such as walking or repeated hopping can be produced by making the object follow a cyclic path through state space. In this case the state representing the horizontal distance denotes the distance to complete one cycle. The value of this state increases until the object lands, at which point the state wraps back to the starting position. This allows a repeated motion to be described in terms of a state-space cycle, which can be modelled by a state-space controller. This idea was motivated by the fact that the spinal cord of animals can produce a periodic sequence of control signals resulting in periodic walking motions [15,32].

5.2 Pendulum

The pendulum has only one degree of freedom and therefore has a two-dimensional state space consisting of the angle and the angular speed (see Figure 4). The link is free to rotate in both directions (without friction) and has the force of gravity acting on it. The pendulum also has a motor or muscle located at the point of rotation that can exert a control torque on the pendulum. It is easy to represent the state-space path of a pendulum using a plot, which helps to illustrate how general SSCs guide objects to a destination.

SSCs with various destinations have been generated to control the motion of a pendulum. Figure 12 describes an SSC with a destination state of $\theta = 0$ deg, $\omega = 0$ deg/s. Applied torques are constrained to the range -10 to 10 Nm, and pendulums cannot rotate at absolute angular speeds of more than 500 degrees per second. The state-space dimensions of θ and ω are divided into 36 and 21 discrete intervals respectively, yielding SSCs with a total of 756 entries. Figure 13 summarizes the pendulum SSCs that were generated.

```
# controller for a single link
# controller description commands:
#       ssd <name> <min> <max> <steps>
#       torq <name> <tmin> <tmax> <tsteps> <Ktorq>
#       dest <state>

dyn link1              # dynamics equations

ssd omega -500.0 500.0 21
ssd angle -180 170 36
wrap angle
torq torque -10.0 10.0 11 0.1
dest angle=0 omega=0
```

Figure 12: *Specification for pendulum SSCs*

name	destination state θ (deg)	ω (deg/s)	generation time (s)
A	0	0	381
B	120	0	452
C	-120	0	341
D	0	300	460
E	-120	-300	459
F	160	250	454

Figure 13: *Six pendulum controllers. Controllers were computed on a Sun 3/60 workstation.*

Figures 14 and 16 depict some state-space plots of pendulum motion under control of the SSCs. The scales used for the state-space plots in are as in Figure 5. The curve connecting each starting state and the destination state represents the state-space path that the pendulum takes when guided by the SSC. The right and left boundaries of the state-space plots are connected, giving each plot a cylindrical topology.

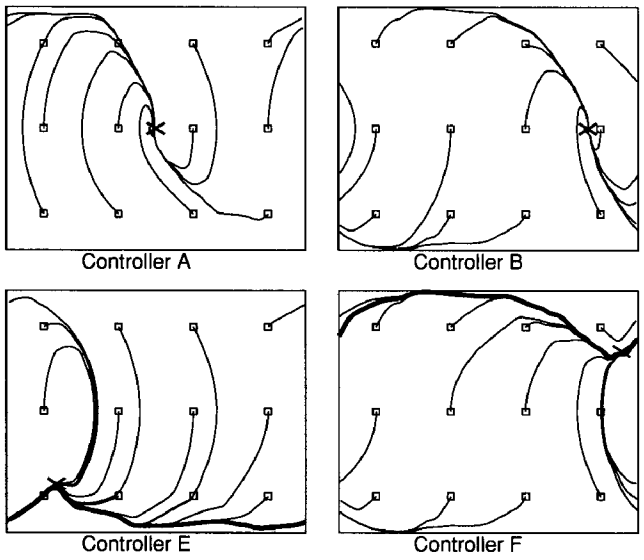

Figure 14: *State-space trajectories followed using various SSCs (C and D omitted). The destination state is given by an encircled X, while some sample start states are given by the small squares.*

Controllers D, E, and F define periodic motions because their destination states specify non-zero angular velocities (see above). As seen in Figure 14, these controllers eventually drive the pendulum toward their respective state-space cycles, from all initial states. The state-space cycles are indicated in the figure with a thick line.

The six different SSCs form a library of motion commands for the pendulum. Figure 15 shows an animation script used to 'animate' the pendulum, assuming that one would want to endow a pendulum with a 'muscle' that can exert a torque at the joint. Figure 16 depicts the resulting motion.

The animation begins with controller B being used to drive the pendulum to the destination state for controller B. The point 'sB' in Figure 16 shows the point at which controller B is invoked, and point 'dB' denotes the destination state for controller B. After 0.4 seconds of proceeding toward dB, controller A is invoked for 0.7 seconds. Before it reaches dA (the destination state for controller A), controller D is invoked for 0.6 seconds. The remainder of the animation script consists of similar exchanges of controllers.

While the state of the pendulum is continuous over time, torques (and hence accelerations) are discontinuous at the point of controller exchange. Such discontinuities might detract from the realism of the resulting motion because real actuators (muscles or motors) cannot instantaneously change their applied torque or force. A simple solution to this problem is to apply a slew limitation to the control values. This would limit the absolute rate of change of the control value,

```
state link -135,350        # starting state
swapcon link conB.ctab     # invoke controller B
sim 0.4                    # simulate 0.4 seconds
swapcon link conA.ctab     # invoke controller A
sim 0.7                    # simulate 0.7 seconds
swapcon link conD.ctab     # invoke controller D
sim 0.6                    # simulate 0.6 seconds
swapcon link conE.ctab     # invoke controller E
sim 0.4                    # simulate 0.4 seconds
swapcon link conC.ctab     # invoke controller C
sim 0.4                    # simulate 0.4 seconds
swapcon link conF.ctab     # invoke controller F
sim 0.7                    # simulate 0.7 seconds
```

Figure 15: *A pendulum animation script.*

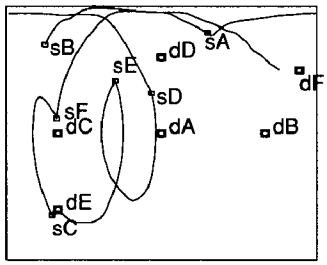

Figure 16: *State-space trajectory of pendulum.*

resulting in $C^{(2)}$ continuous motion.

6 The Self-Parking Car

We now consider the design of an SSC that is to parallel-park a car on the street as shown in Figure 17. The car has five state-space dimensions shown in Figure 18. The destination state for the car is given by the dotted line in Figure 17, with the car being at rest. The domain of the controller is marked with a dashed line. The reference point of the car, located between the front wheels, must remain within this region. The animator can place the car anywhere within the domain of the SSC and have it park in a fashion similar to people(!). The walls in Figure 17 are additional obstacles to be avoided.

The car has two control variables: ω_{st}, the rate at which the steering wheel turns, and α_w, the angular acceleration or deceleration of the front wheels of the car.

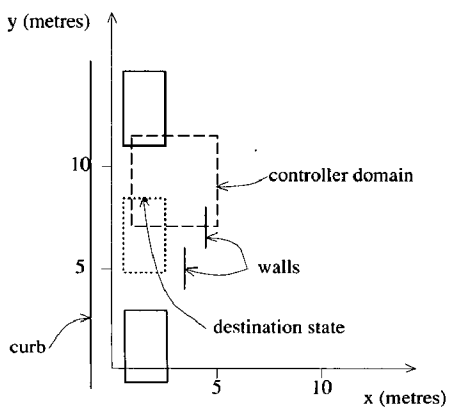

Figure 17: *The street (non-contact parking only).*

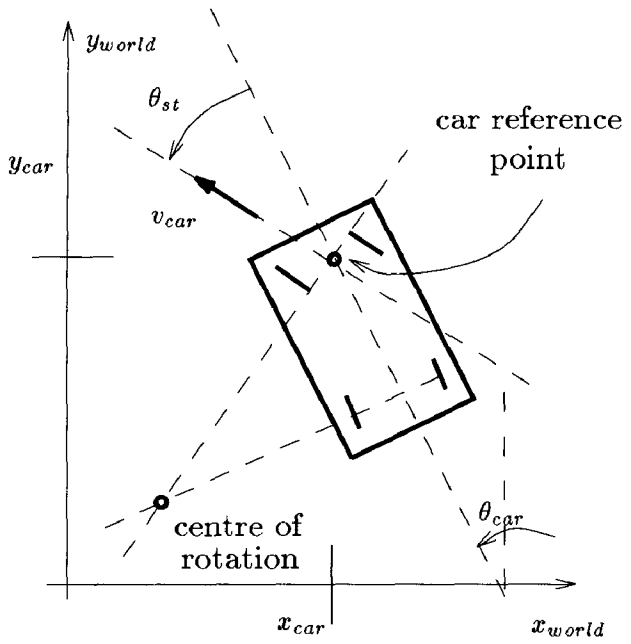

Figure 18: *State-space dimensions for a car.*

The final SSC for the car has 13250 states. All table entries that correspond to collision states are removed while the state-space is being generated; thus the SSC algorithm knows these states are out of the controller domain. Some results of the car SSC in operation are shown in a sequence of frames in Figure 19. The position of the car is shown every 0.3 seconds. The parked cars on the right are used to provide a reference showing the destination state for the car. In all six cases shown, the car is placed in the initial state at rest, but the wheels are oriented differently. The car SSC is then invoked to park the car. The bottom-left and top-right cars are especially interesting. In these cases the cars back up past the desired destination and then drive forward to straighten the wheels.

7 Luxo

In our next example, Luxo will perform a sequence of interesting motions, such as jumps and flips. Because we use uniform sampling of the state space to generate a continuous controller, two practical problems arise in trying to create a jump or flip controller for Luxo. The first is the size of the state space. As illustrated in Figure 20, Luxo has 5 degrees of freedom when in the air, and thus has a 10-dimensional state space. The second is that many of the most important states during a jump occur during takeoff and landing, when only one edge of the lamp's base is in contact with the ground. Since the state space is sampled uniformly without paying special attention to interactions with the environment (like the floor), collisions may not be properly sampled. The problem is illustrated in Figure 21, which shows the states of successive table entries. The second state has its base protruding through the floor, and would therefore be discarded from the SSC table (removing it from the controller's domain). The state that is really of interest is the third one, with the left side of the base touching the ground. Both of the above problems serve to show that our choice of uniform sampling is somewhat crude. Fortunately, the remedy for the case of Luxo is not difficult.

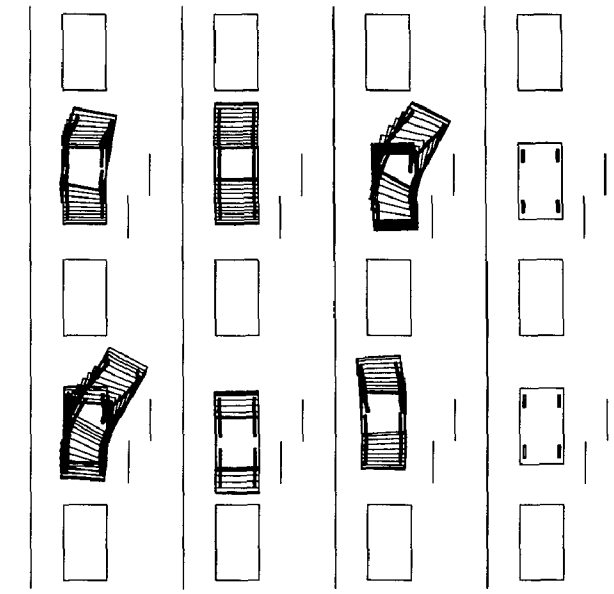

Figure 19: *Animation using the car-parking SSC.*

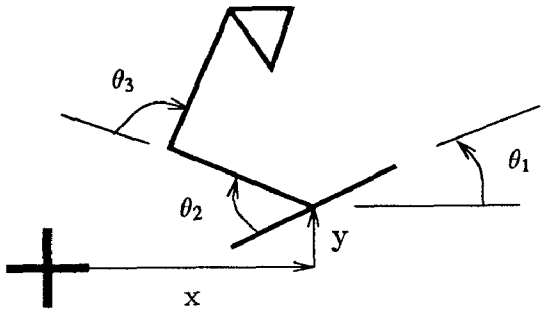

Figure 20: *Degrees of freedom for a jumping lamp.*

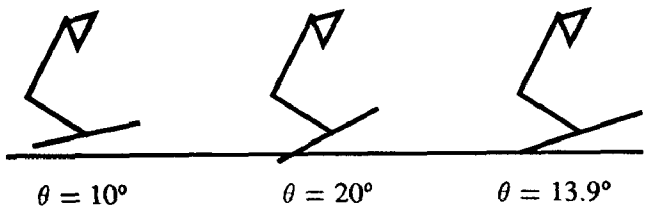

Figure 21: *The collision sampling problem.*

The problem of state-space size can be solved by breaking up canonical Luxo motions into two pieces: airborne motion, and motion on the ground (i.e., takeoff/landing motion). It is easy to write motor programs (i.e., procedural functions of torques over time) to make Luxo perform a flip or jump. It is very much more difficult to do a correct landing. We thus use motor programs for the airborne motions, and a controller to guide Luxo to a safe landing and to prepare it for the next motion. All airborne motions begin from a distinguished starting state, which coincides with the destination state of the "landing" controller. Thus motions based on motor programs can be easily concatenated with motions based on SSCs. The "landing" controller only has to deal with 2 degrees of freedom, or a four-dimensional state space, which is quite feasible.

The dynamics compiler was used to generate the equations of motion for Luxo. Some code was added to the resulting dynamics procedure in order to model collisions with the environment, consisting of the floor and a set of stairs. Four motions were created for the Luxo animation: a forward jump, a single back flip, a double backflip, and a single backflip down a step. Once controllers and motor programs for the individual motions have been generated, scripting various animations is a simple exercise. Figure 22 is a sample back-flip from the animation. See also the photographs at the end of this paper.

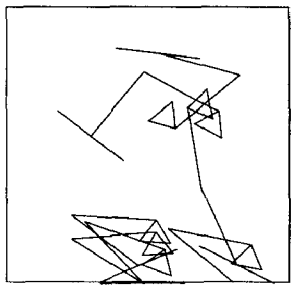

Figure 22: *A Luxo back-flip*

A brief comparison with previous animations of Luxo is informative. The *Luxo Jr.* video [22] by Pixar was produced entirely using keyframing. The motion was synthesized manually and succeeds in looking realistic because of the formidable artistic talent of the animators. The repertoire of motions that Luxo performs in the video is not large or complex; most motions are performed with the base on the ground. Witkin and Kass obtain a jumping motion for Luxo by formulating it as a two-boundary-point optimization problem [40]. The results produced are impressive, but their formulation appears to have a problem with the takeoff and landing occurring with only one edge of the base in contact with the ground. Their Luxo jump sequence has a takeoff and landing with a flat base. From our experience with the torques necessary to make Luxo perform a jump, we are convinced that a jump with a flat base on takeoff and landing is very difficult to perform and would therefore not be a natural mode of locomotion for a lamp!

8 Conclusions

We have introduced a new approach to reusable motion synthesis based on state-space controllers. The controllers produced are unique in that they are used to control the simulation of the object with no *a priori* knowledge of the object or how it should move, apart from a destination goal. The forward dynamics of articulated figures is automatically generated from a basic physical description of the object.

While the use of controllers in physically-based motion synthesis is very encouraging, there are several aspects that require more thought. First, we wish to carefully compare controller response to the actual optimal solution. This may allow us to develop error-control mechanisms for controller generation. Second, we would like to create hierarchical or distributed controller structures, in which more "abstract" controllers actually manage lower-level controllers in response to events in the system. Third, a controller is currently quite dependent on the specific physical parameters of an object. To what extent can controllers themselves be parameterized by, for example, interpolating between similar controllers? Fourth, faster controller-generation techniques are required. Fifth, better controller sampling and reconstruction techniques are needed. Sixth, we wish to develop a better user interface for scripting controllers into significant animations.

References

[1] R. Alexander. The gaits of bipedal and quadrupedal animals. *Int. Journal of Robotics Research*, Summer 1984.

[2] W. W. Armstrong. Recursive solution to the equations of motion of an n-link manipulator. *Proc. 5th World Congress Theory Mach. Mechanisms*, volume 2, 1343–1346, 1979.

[3] W.W. Armstrong, M. Green, and R. Lake. Near-real-time control of human figure models. *IEEE Computer Graphics and Applications*, 7(6):52–61, June 1987.

[4] H. Asada and J.-J.E. Slotine. *Robot Analysis and Control*. John Wiley and Sons, 1986.

[5] N.I. Badler, K.H. Manoocherhri, and G. Walters. Articulated figure positioning by multiple constraints. *IEEE Computer Graphics and Applications*, 7(6):28–38, June 1987.

[6] R. Barzel and A.H. Barr. A modeling system based on dynamic constraints. *Proc. of SIGGRAPH'88 (Aug. 1988). ACM Computer Graphics 22,4*, 179–188.

[7] L.S. Brotman and A.N. Netravali. Motion interpolation by optimal control. *Proc. of SIGGRAPH'88 (Aug. 1988). ACM Computer Graphics 22,4*, 309–315.

[8] A. Bruderlin. Goal-directed, dynamic animation of bipedal locomotion. Technical report, Simon Fraser University, 1988.

[9] N. Burtnyk and M. Wein. Computer generated keyframe animation. *Journal of the Society of Motion Picture and Television Engineers*, 80(3):149–53, March 1971.

[10] T. Calvert, J. Chapman, and A. Patla. Aspects of the kinematic simulation of human movement. *IEEE Computer Graphics and Applications*, 41–50, Nov. 1982.

[11] C. Csuri. Real time film animation. *IEEE Convention Digest*, 42–3, March 1971.

[12] R. Featherstone. The calculation of robot dynamics using articulated body inertias. *Int. Journal of Robotics Research*, 2(1):13–30, Spring 1983.

[13] D.R. Forsey and J. Wilhelms. Techniques for interactive manipulation of articulated bodies using dynamic analysis. *Proc. of Graphics Interface*, 8–15, 1988.

[14] M. Girard. Interactive design of computer-animated legged animal motion. *IEEE Computer Graphics and Applications*, 7(6):39–51, June 1987.

[15] S. Grillner. Locomotion in vertebrates: Central mechanisms and reflex interaction. *Physiological Reviews*, 55:247–304, 1975.

[16] Paul M. Isaacs and Michael F. Cohen. Controlling dynamic simulation with kinematic constraints, behaviour functions and inverse dynamics. *Proc. of SIGGRAPH '87. ACM Computer Graphics 21,4*, 215–224.

[17] P.M. Isaacs and M.F. Cohen. Mixed methods for complex kinematic constraints in dynamic figure animation. *The Visual Computer*, 4:296–305, 1988.

[18] D.H. Kochanek and R.H. Bartels. Interpolating splines for keyframe animation. *Graphics Interface*, 41–42, 1984.

[19] J. U. Korein and N. I. Badler. Techniques for generating the goal-directed motion of articulated structures. *IEEE Computer Graphics and Applications*, 71–81, Nov. 1982.

[20] B.C. Kuo. *Automatic Control Systems*. Prentice-Hall, Inc., 1987.

[21] J. Lasseter. Principles of traditional animation applied to 3-d computer animation. *Proc. of SIGGRAPH'87 (July 1987). ACM Computer Graphics 21,4.*, 35–44.

[22] J. Lasseter and W. Reeves. Luxo jr. Pixar Video, 1986.

[23] N. Magnenat-Thalmann and D. Thalmann. *Computer Animation: Theory and Practice*. Springer-Verlag, 1985.

[24] R. B. McGhee and G. I. Iswandhi. Adaptive locomotion of a multilegged robot over rough terrain. *IEEE Transactions on System, Man, and Cybernetics*, 176–182, April 1979.

[25] G.S.P. Miller. The motion dynamics of snakes and worms. *Proc. of SIGGRAPH'88 (Aug. 1988). ACM Computer Graphics 22,4*, 169–178.

[26] H. Miura and I. Shimoyama. Dynamic walk of a biped. *Int. Journal of Robotics Research*, 60–74, Summer 1984.

[27] T. J. O'Donnel and Arthur J. Olsen. Gramps - a graphical interpreter for real-time interactive three-dimensional picture editing and animation. 1981.

[28] K. Ogo, A. Ganse, and I. Kato. Quasi dynamic walking of biped walking machine aiming at completion of steady walking. *Third Symposium on Theory and Practice of Robots and Manipulators*, 340–356, Sept. 1978.

[29] W.H. Press, B.P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes*. Cambridge University Press, 1986.

[30] M.H. Raibert. *Legged robots that balance*. Artificial Intelligence series. MIT Press, Cambridge MA, 1985.

[31] C. Reynolds. Computer animation with scripts and actors. *Proc. of SIGGRAPH'81.*, 1981.

[32] M. L. Shik and G. N. Orlovskii. Neurophysiology of a locomotor automatism. *Physiological Reviews*, 56:465–501, 1976.

[33] S. Steketee and N.I. Badler. Parametric keyframe interpolation incorporating kinetic adjustment and phrasing control. *Proc. of SIGGRAPH'85 (July 1985). ACM Computer Graphics 19,3.*

[34] D. Sturman. Interactive keyframe animation of 3-d articulated models. *Graphics Interface*, 35–40, 1984.

[35] M. Townsend and A. Seirig. Effect of model complexity and gait criteria on the synthesis of bipedal locomotion. *IEEE Transactions on Biomedical Engineering*, 433–444, November 1973.

[36] M. W. Walker and D. E. Orin. Efficient dynamic computer simulation of robotic mechanisms. *Journal of Dynamic Systems, Measurement, and Control*, 205–211, Sept. 1982.

[37] J. Wilhelms. Virya: A motion control editor for kinematic and dynamic animation. *Proc. Graphics Interface 86*, 141–146. Morgan Kaufman, May 1986.

[38] J. Wilhelms, M. Moore, and R. Skinner. Dynamic animation: interaction and control. *The Visual Computer*, 4(6):283–295, 1988.

[39] J. Wilhelms. Using dynamic analysis for realistic animation of articulated bodies. *IEEE Computer Graphics and Applications*, 7(6):12–27, June 1987.

[40] A. Witkin and M. Kass. Spacetime constraints. *Proc. of SIGGRAPH'88 (Aug. 1988). ACM Computer Graphics 22,4*, 159–168, 1988.

[41] D. Zeltzer. Motor control techniques for figure animation. *IEEE Computer Graphics and Applications*, 53–60, Nov. 1982.