Spacetime Constraints

Andrew Witkin
Michael Kass

Schlumberger Palo Alto Research
3340 Hillview Avenue, Palo Alto, CA 94304

Abstract

Spacetime constraints are a new method for creating character animation. The animator specifies *what* the character has to do, for instance, "jump from here to there, clearing a hurdle in between;" *how* the motion should be performed, for instance "don't waste energy," or "come down hard enough to splatter whatever you land on;" the character's *physical structure*—the geometry, mass, connectivity, etc. of the parts; and the physical *resources* available to the character to accomplish the motion, for instance the character's muscles, a floor to push off from, etc. The requirements contained in this description, together with Newton's laws, comprise a problem of constrained optimization. The solution to this problem is a *physically valid* motion satisfying the "what" constraints and optimizing the "how" criteria. We present as examples a Luxo lamp performing a variety of coordinated motions. These realistic motions conform to such principles of traditional animation as anticipation, squash-and-stretch, follow-through, and timing.

Keywords — Animation, Constraints

1 Introduction

Computer animation has made enormous strides in the past several years. In particular, Pixar's *Luxo, Jr.* [10] marked a turning point as perhaps the first computer-generated work to compete seriously with works of traditional animation on every front. Key among the reasons for *Luxo, Jr.'s* success is that it was made by a talented animator who adapted the principles of traditional animation to the computer medium. *Luxo, Jr.*, in large measure, *is* a work of traditional animation that happens to use a computer to render and to interpolate between keyframes. John Lasseter spelled this out clearly in his presentation to Siggraph '87 [9]. Although *Luxo, Jr.* showed us that the team of animator, keyframe system, and renderer can be a powerful one, the responsibility for defining the motion remains almost entirely with the animator.

Some aspects of animation—personality and appeal, for example—will surely be left to the animator's artistry and skill for a long time to come. However, many of the principles of animation are concerned with making the character's motion look *real* at a basic mechanical level that ought to admit to formal physical treatment. Consider for example a jump exhibiting anticipation, squash-and-stretch, and follow-through. Any creature—human or lamp—can only accelerate its own center of mass by pushing on something else. In jumping, the opportunity to control acceleration only exists during contact with the floor, because while airborne there is nothing to push on. Anticipation prior to takeoff is the phase in which the needed momentum is acquired by squashing then stretching to push off against the floor. Follow-through is the phase in which the momentum on landing is absorbed.

Such physical arguments make nice *post hoc* explanations, but can physics be brought to bear in *creating* the complex active motions of characters like Luxo? If so, how much of what we regard as "nice" motion follows directly from first principles, and how much is really a matter of style and convention?

This paper presents a physically-based approach to character animation in which coordinated, active motion is created automatically by specifying:

- *What* the character has to do, for instance "jump from here to there."
- *How* the motion should be performed, for instance "don't waste energy," or "come down hard enough to splatter whatever you land on."
- What the character's *physical structure* is—what the pieces are shaped like, what they weigh, how they're connected, etc.

[0] "Luxo" is a trademark of Jac Jacobsen Industries AS.

- What physical *resources* are available to the character to accomplish the desired motion, for instance the character's muscles (or whatever an animate lamp has in place of muscles,) a floor to push off from, etc.

Our initial experiments with this approach have aimed at making a Luxo lamp execute a convincing jump just by telling it where to start and where to end. The results we present in this paper show that such properties as anticipation, follow-through, squash-and-stretch, and timing indeed emerge from a bare description of the motion's purpose and the physical context in which it occurs. Moreover, simple changes to the goals of the motion or to the physical model give rise to interesting variations on the basic motion. For example, doubling (or quadrupling) the mass of Luxo's base creates amusingly exaggerated motion in which the base *looks* heavy.

Our method entails the numerical solution of large constrained optimization problems, for which a variety of standard algorithms exist. These algorithms, while relatively expensive, spend most of their time solving sparse linear systems, and are therefore amenable to acceleration by array processors and other commonly available hardware. The greatest difficulty arises not in computing the numerical solution, but in setting up the intricate sparse matrix equations that drive the solution process. To address this problem we implemented an object-oriented symbolic algebra system that automates this difficult task almost entirely. We therefore believe the method described here can become a practical animation tool requiring no more mathematical sophistication of the end user than do current keyframing systems.

The remainder of the paper is organized as follows: the following section discusses the previous use of physical methods in animation. The spacetime method is then introduced using a moving particle as a toy example. Next, our extension of the method to complex problems is discussed. Finally, the Luxo model and the results obtained with it are described.

2 Background and Motivation

Recently, there has been considerable interest in incorporating physics into animation using simulation methods. [7, 14, 15, 1, 13, 4, 6] The appeal of physical simulation as an animation technique lies in its promise to produce realistic motion automatically by applying the same physical laws that govern real objects' behavior.

Unfortunately, the realism of simulation comes at the expense of control. Simulation methods solve *initial value problems:* the course of a simulation is completely determined by the objects' initial positions and velocities, and by the forces applied to the objects along the way. An animator, however, is usually concerned as much with where the objects end up and how they get there as where they begin. Problems cast in this form are not initial value problems. For instance, while simulating a bouncing ball is easy enough, making the ball bounce *to* a particular place requires choosing just the right starting values for position, velocity, and spin. Making these choices manually is a painful matter of trial and error. Problems such as this one, in which both initial and final conditions are partially or completely constrained, are called *two-point boundary problems*, requiring more elaborate solution methods than forward simulation.[3]

Character animation poses a still more difficult problem. Animals move by using their muscles to exert forces that vary as a function of time. Calculating the motion by simulation is straightforward once these time-dependent force functions are known, but the difficult problem is to calculate force functions that achieve the goals of the motion. Specifying these functions by hand would be hopeless, equivalent to making a robot move gracefully by manually varying its motor torques.

In an effort to reconcile the advantages of simulation with the need for control, several researchers [1, 7] have proposed methods for blending positional constraints with dynamic simulations. The idea behind these methods is to treat kinematic constraints as the consequences of unknown "constraint forces," solve for the forces, then add them into the simulation, exactly canceling that component of the applied forces that fights against the constraints.

Constraint force methods permit parts, such as a character's hands or feet, to be moved along predefined keyframed trajectories, but provide no help in defining the trajectories, which is the central problem in creating character animation. While allowing a character to be dragged around manually like a marionette, constraint forces sidestep the central issue of deciding how the character should move.

These shortcomings led us to adopt a new formulation of the constraint problem, whose central characteristic is that we solve for the character's motion and time-varying muscle forces over the entire time interval of interest, rather than progressing sequentially through time. Because we extend the model through time as well as space, we call the formulation *spacetime constraints.*

The spacetime formulation permits the imposition of constraints throughout the time course of the motion, with the effects of constraints propagating freely backward as well as forward in time. Constraints on initial, final, or intermediate positions and velocities directly encode the *goals* of the motion, while constraints limiting muscle forces or preventing interpenetration define properties of the physical situation. Additionally, Newtonian physics provides a constraint relating the force and position functions that must hold at every instant in time. Subject to these constraints we optimize functions that specify

how the motion should be performed, in terms of efficiency, smoothness, etc. Solving this constrained optimization problem yields optimal, physically valid motion that achieves the goals specified by the animator.

3 A spacetime particle

As a gentle but concrete introduction to the spacetime method, this section describes a minimal example involving a moving particle, influenced by gravity, and equipped with a "jet engine" as a means of locomotion. With no restrictions on the forces exerted by its engine, the particle can move any way it likes. The problem we formulate here is that of making the particle fly from a given starting point to a given destination in a fixed period of time, with minimal fuel consumption. This toy problem is too simple to produce any really interesting motion, but it exhibits all the key elements of the method, and will aid in understanding what follows.

3.1 Problem formulation

Let the particle's position as a function of time be $\mathbf{x}(t)$, and the time-varying jet force be $\mathbf{f}(t)$. Suppose for simplicity that the mass of the fuel is negligible compared to that of the particle, so the total mass may be treated as a constant, m, with a constant gravitational force $m\mathbf{g}$. Then the particle's equation of motion is

$$m\ddot{\mathbf{x}} - \mathbf{f} - m\mathbf{g} = 0, \tag{1}$$

where $\ddot{\mathbf{x}}$ is the second time derivative of position. Given the function $\mathbf{f}(t)$, and initial values for $\mathbf{x}$ and $\dot{\mathbf{x}}$ at some time t_0, the motion $\mathbf{x}(t)$ from t_0 could be obtained by integrating equation 1 to solve the initial value problem.

Instead we wish to make the particle fly from a known point $\mathbf{a}$ to a known point $\mathbf{b}$ in a fixed period of time. Suppose for simplicity that the rate of fuel consumption is $|\mathbf{f}|^2$. In that case, we have constraints $\mathbf{x}(t_0) = \mathbf{a}$ and $\mathbf{x}(t_1) = \mathbf{b}$ subject to which

$$R = \int_{t_0}^{t_1} |\mathbf{f}(t)|^2 \, dt$$

must be minimized. The problem then is to find a force function $\mathbf{f}(t)$, defined on the interval (t_0, t_1), such that the position function $\mathbf{x}(t)$ obtained by solving equation 1 satisfies the boundary constraints, and such that the objective function R is a constrained minimum.

There exist a variety of standard approaches to solving problems of this form. Prevalent in the optimal control literature are iterative methods that solve the initial value problem within each iteration, using the equations of motion to obtain the position function from the force function (see [12] for a good survey.) We choose instead to represent the functions $\mathbf{x}(t)$ and $\mathbf{f}(t)$ independently. The equation of motion then enters as a constraint that relates the two functions, to be satisfied along with the other constraints during the solution process. Each function is discretized, that is, represented as a sequence of values, with time derivatives approximated by finite differences. This approach leads to a classical problem in constrained optimization, for which a variety of standard solution algorithms are available.

Let the discretized functions $\mathbf{x}(t)$ and $\mathbf{f}(t)$ be represented by sequences of values $\mathbf{x}_i$ and $\mathbf{f}_i$, $0 \le i \le n$, with h the time interval between samples. To approximate the time derivatives of $\mathbf{x}(t)$ we use the finite difference formulas

$$\dot{\mathbf{x}}_i = \frac{\mathbf{x}_i - \mathbf{x}_{i-1}}{h} \tag{2}$$

$$\ddot{\mathbf{x}}_i = \frac{\mathbf{x}_{i+1} - 2\mathbf{x}_i + \mathbf{x}_{i-1}}{h^2} \tag{3}$$

Substituting these relations into equation 1 gives n "physics constraints" relating the x_i's to the f_i's,

$$\mathbf{p}_i = m\frac{\mathbf{x}_{i+1} - 2\mathbf{x}_i + \mathbf{x}_{i-1}}{h^2} - \mathbf{f}_i - m\mathbf{g} = 0, \quad 1 < i < n. \tag{4}$$

In addition we have the two boundary constraints

$$\mathbf{c}_a = \mathbf{x}_1 - \mathbf{a} = 0$$

and

$$\mathbf{c}_b = \mathbf{x}_n - \mathbf{b} = 0.$$

Assuming that $\mathbf{f}(t)$ is constant between samples, the objective function R becomes a sum

$$R = h\sum_i |\mathbf{f}_i|^2 \tag{5}$$

which is to be minimized subject to the constraints. The discretized objective and constraint functions are now expressed in terms of the $\mathbf{x}_i$'s and the $\mathbf{f}_i$'s, which are the independent variables to be solved for.

3.2 Numerical Solution

From the standpoint of the numerical solution process it is useful to suppress the structure of the particular problem, reducing it to a canonical form consisting of a collection of scalar independent variables S_j, $1 \le j \le n$, an objective function $R(S_j)$ to be minimized, and a collection of scalar constraint functions $C_i(S_j)$, $1 < i < m$, which must be driven to zero. In the current problem, the S_j's are the x, y, and z components of the $\mathbf{x}_i$'s and the $\mathbf{f}_i$'s, while the C_i's are the components of the $\mathbf{p}_i$'s, $\mathbf{c}_a$, and $\mathbf{c}_b$. Typically, setting up the linearized indices is the responsibility of a

program that keeps track of the independent variables and the constraint functions.

In these terms, the standard constrained optimization problem is "Find S_j that minimizes $R(S_j)$ subject to $C_i(S_j) = 0$. For the sake of modularity, the numerical method that solves the problem is best regarded as an object that requests answers to certain standard questions about the system, and iteratively provides updated values for the solution vector S_j. Any method must be permitted to request the *values* of R and C_i at a given S_j. In addition, most effective methods require access to *derivatives* of R and C_i with respect to S_j, in order to move toward a solution.

The solution method we use is a variant of Sequential Quadratic Programming (SQP), described in detail in [3]. Essentially, the method computes a second-order Newton-Raphson step in R, and a first-order Newton-Raphson step in the C_i's, and combines the two steps by projecting the first onto the null space of the second (that is, onto the hyperplane for which all the C_i's are constant to first order.) Because it is first-order in the constraint functions and second-order in the objective function, the method requires that we be able to compute two derivative matrices: the *Jacobian* of the constraint functions, given by

$$J_{ij} = \frac{\partial C_i}{\partial S_j},$$

and the *Hessian* of the objective function,

$$H_{ij} = \frac{\partial^2 R}{\partial S_i \partial S_j}.$$

In addition, the first derivative vector $\partial R/\partial S_j$ must be available. The SQP step is obtained by solving two linear systems in sequence. The first,

$$-\frac{\partial R}{\partial S_i} = \sum_j H_{ij} \hat{S}_j$$

yields a step $\hat{S}_j$ that minimizes a second-order approximation to R, without regard to the constraints. The second,

$$-C_i = \sum_j J_{ij} (\tilde{S}_j + \hat{S}_j)$$

yields a step $\tilde{S}_j$ that drives linear approximations to the C_i's simultaneously to zero, and at the same time projects the optimization step $\hat{S}_j$ onto the null space of the constraint Jacobian. The final update is $\Delta S_j = \tilde{S}_j + \hat{S}_j$. The algorithm reaches a fixed point when $C_i = 0$ and when any further decrease in R requires violating the constraints.

3.3 Linear system solving

The choice of a method for solving these linear systems is critically important, because the matrices can be large. Although inverting a general $n \times n$ matrix is $O(n^3)$, the matrices arising in spacetime problems are nearly always extremely sparse. Exploiting the sparsity is essential to make the problem tractable. Moreover, over- and under-constrained systems, whose matrices are non-square and/or rank-deficient, can easily arise, in which case the inverse is undefined and the system cannot be solved. The latter problem is well treated by the pseudo-inverse [8, 4], which provides least-squares solutions to overconstrained problems, and minimal solutions to underconstrained ones. To compute the pseudo-inverse while exploiting random sparsity, we adapted a sparse conjugate gradient (CG) algorithm described in [11], which is $O(n^2)$ for typical problems. The CG algorithm solves the matrix equation $\mathbf{a} = \mathbf{M}\mathbf{b}$ by iteratively minimizing $|\mathbf{a} - \mathbf{M}\mathbf{b}|^2$, giving a least-squares solution to overconstrained problems. Provided that a zero starting-point is given for $\mathbf{b}$, the solution vector is restricted to the null-space complement of $\mathbf{M}$.

3.4 Matrix evaluation.

Applying the SQP algorithm to the moving particle example requires evaluation of the sparse derivative matrices, as well as the objective and constraint functions themselves. Apart from the bookkeeping required for indexing, these evaluations are straightforward. The Jacobian of the physics constraint is given by

$$\begin{aligned}
\frac{\partial \mathbf{p}_i}{\partial \mathbf{x}_j} &= 2m/h^2, \quad i = j \\
&= -m/h^2, \quad i = j \pm 1 \\
&= 0, \quad \text{otherwise} \\
\frac{\partial \mathbf{p}_i}{\partial \mathbf{f}_j} &= 1, \quad i = j \\
&= 0, \quad \text{otherwise.}
\end{aligned}$$

The Jacobians of the boundary constraints are trivial. The gradient of R is

$$\frac{\partial R}{\partial \mathbf{f}_i} = 2\mathbf{f}_i,$$

and the Hessian is

$$\begin{aligned}
\frac{\partial^2 R}{\partial \mathbf{f}_i \partial \mathbf{f}_j} &= 2, \quad i = j \\
&= 0, \quad \text{otherwise.}
\end{aligned}$$

Although it happens that the toy problem we chose constrains initial and final positions, nothing in the solution approach depends on this configuration: initial and final conditions could be left free, and constraints at arbitrary internal points could be added. Moreover, arbitrary constraints of the form $F(S_i) = 0$, not just position constraints, may be added provided that the constraint functions and their derivatives can be evaluated.

4 Extension to complex models

In principle, the procedure described in the last section extends to complex models, constraints, and objective functions. In practice, as the model grows more complex, the problem becomes prohibitively difficult. The difficulty lies not so much in calculating the numerical solution as in creating code to evaluate the constraint and objective functions and their sparse derivatives, and in coercing the evaluations into the form of a canonical constrained optimization. In particular, the required differentiations can lead to enormous algebraic expressions that are all but impossible to derive and code by hand.

To make the method practical, we developed a lisp-based system that performs these difficult tasks automatically. The system consists of three principle elements: a specialized math compiler that performs symbolic differentiation and simplification of tensor forms, and generates optimized code to perform the evaluations; a runtime system that allows the generated functions to be composed dynamically, automatically building the vectors and sparse matrices that drive the numerical solution; and an SQP solver.

Because the mathematical operations required to define a new primitive object or constraint are highly stylized, it is possible to reduce the programmer's job to a simple cookbook procedure. Once the primitives are defined, a user with little or no knowledge of the underlying mathematics can wire them together dynamically to create animation. Although a full description is beyond the scope of this paper, this section briefly outlines the system and the operations it performs.

4.1 Function Boxes

A *function box,* the lowest level construct in the system, consists of a set of input quantities, which may be scalars, vectors, matrices, or higher-order tensors, and a collection of output quantities each defined as a mathematical function of the inputs. To define a function box, the programmer specifies the inputs, the outputs, and the functions that relate them. The function definitions are mathematical expressions that may include differentiations as well as algebraic operations. Non-scalar quantities are expressed and manipulated using index notation with the summation convention. For each output, the system performs symbolic differentiation as called for, simplifies the resulting expression, extracts common sub-expressions, and generates an optimized lisp function that evaluates the output given the inputs. In addition, the system symbolically differentiates each output with respect to each input on which it depends, creates a lisp function to evaluate the derivative, and analyzes its sparsity. These functions form the Jacobians of the outputs. The generated functions, input-output dependencies, sparsities, etc., are recorded in a data structure accessible to the runtime system.

4.2 User Interface

Once defined, function boxes are manipulated using a graphical interface in which they appear as literal boxes on the screen, with ports representing the input and output quantities.[2] The user may instantiate boxes, connecting the ports to form a graph whose arcs represent function composition. In this way, complex systems are built dynamically by composing pre-compiled primitives. By default, input ports to which nothing has been connected are treated as internal constants whose values may be inspected and modified interactively, and unconnected output ports are ignored. However, inputs may also be flagged by the user as state variables to be solved for, and outputs may be flagged either as constraints or as terms to be summed into the objective function.

4.3 Runtime System

Once the graph representing the model has been constructed, and the state-variables, constraints, and objective terms declared, a pre-runtime computation is performed to set up the constrained optimization. The user-declared state variables, constraints, and objective terms are collected and indexed to form the quantities S_j, C_i, and R required by the solver. The sparse derivatives are formed by propagation through the graph using the chain rule, with the individual Jacobian functions associated with function boxes combined by a hierarchy of sparse matrix multiplications and additions. An optimal sequence of adds and multiplies is pre-computed for each sparse matrix operation, and the sparsity patterns of the resulting global matrices are also precomputed. Evaluation of C_i, R, and their derivatives, then proceeds by recursing through the graph, calling the individual value and Jacobian functions, and performing the sparse matrix operations. The solver communicates with the model by requesting these evaluations and updating the state vector.

4.4 Defining Objects

Built on top of the basic system is a layer handling the specifics of physical object models, whose main job is to construct the object's equations of motion. In the case of the moving particle this just involved direct application of $f = ma$. However, deriving the equations of motion for more complicated objects can be difficult.

We derive the equations automatically using Lagrangian Dynamics [5], a classical cookbook procedure in which an expression for a body's kinetic energy is subjected to a

series of symbolic differentiations. Lagrange's equations of motion are given by

$$\frac{d}{dt}(\frac{\partial T}{\partial \dot{\mathbf{q}}}) - \frac{\partial T}{\partial \mathbf{q}} - \mathbf{Q} = 0, \qquad (6)$$

where T is kinetic energy, $\mathbf{q}$ is a vector of *generalized coordinates*, and $\mathbf{Q}$ is a *generalized force.* The components of the generalized coordinates are whatever variables control the positions and orientations of parts of the body (e.g. translations, rotations, joint angles, etc.) The generalized force is just the sum of ordinary forces applied to body, transformed into generalized coordinates. For point forces, this transformation is accomplished by multiplying the force vector by the Jacobian of the point at which the force is applied with respect to $\mathbf{q}$.

To define an object, the user is required to supply expressions for T, and for the coordinates of points on the body to which forces or constraints may be applied. Although T must be derived manually, this is a manageable job and need only be done once when a primitive object is defined. Given these expressions, automatic construction of a function box representing the objects is straightforward: the kinetic energy expression is subjected to the rote symbolic differentiations called for in equation 6, with an additional derivative with respect to $\mathbf{q}$ used to define the Jacobian of the physics constraint. The expressions for material points are also differentiated with respect to $\mathbf{q}$ to create "force converter" functions, small Jacobian matrices that map applied forces into generalized coordinates. The function box takes as inputs values for $\mathbf{q}$, $\dot{\mathbf{q}}$, and $\ddot{\mathbf{q}}$, for applied forces, and for constants such as masses and dimensions. It produces outputs for the "physics constraint" defined by the equations of motion, and for the positions and velocities of the material points defined by the user.

4.5 Discretized functions of time

In developing the particle example of the last section, discretized functions representing forces and positions over time were incorporated into the equations of motion by direct substitution. Given the ability to compose functions and their sparse Jacobians automatically, we adopted the alternative of constructing specialized function boxes to represent discretized functions. These boxes contain the sequence of values representing the function, and output the values and the time-derivatives obtained using finite-difference formulas. The Jacobians of these output functions are trivial constant diagonal or banded matrices. The values and derivatives are connected to the corresponding inputs on the object model, causing the discretization to be effected automatically at runtime.

Figure 1: Luxo

5 Spacetime Luxo

We are now equipped to proceed to a spacetime model of an animate Luxo Lamp. The model is composed of rigid bodies of uniform mass connected by frictionless joints. Each joint is equipped with a "muscle" modeled as an angular spring whose stiffness and rest angle are free to vary with time. The lamp is subject to the forces of its own muscles, in addition to the external force of gravity and the contact forces arising from its interaction with objects such as floors and skijumps. A picture of the model appears in Figure 1. In our initial examples, Luxo's motion is restricted to a plane. This expedient simplifies the mathematics, while still allowing the creation of complex, subtle, and interesting motion. Extension of the model to three dimensions involves no fundamental difficulties, although it leads to systems that are somewhat larger, somewhat slower, and more difficult to debug. The definition of the model consists of less than a page of tensor expressions, which expand into roughly 4000 lines of automatically generated lisp code.

5.1 Kinetic Energy

As discussed in the last section, our principle task in defining the model was to formulate an expression for the kinetic energy, T. In general, T is the volume integral over the body of the kinetic energy of each particle, $\frac{1}{2}\rho\,|\dot{\mathbf{x}}|^2$, where ρ is the mass density at point x. The kinetic energy of an articulated object is the sum of the kinetic energies of the parts. Each of Luxo's links is modeled as a rigid body rotating about an axis of fixed direction that passes through the origin in body coordinates (see Figure 2.) Because the axis is fixed, the orientation of the i-th link may be denoted by a single angle θ_i, with angular velocity $\omega_i = \dot{\theta}_i\mathbf{a}$, where $\mathbf{a}$ is a unit vector in the direction of the axis. In addition to

Figure 2: Luxo's parameters: $\mathbf{P}_0$ is a translation, and θ_i is the orientation of the i-th link. Points $\mathbf{P}_1$–$\mathbf{P}_3$ are computed from these parameters.

rotation, the body origin undergoes a translation $\mathbf{p}_i$, with translational velocity $\mathbf{v}_i = d\mathbf{p}_i/dt$. Each link has mass m_i, a constant moment of inertia I_i about the rotation axis, and a center of mass $\mathbf{c}_i$ expressed as a displacement from the body origin. In these terms, the kinetic energy of the i-th link is

$$T_i = \frac{1}{2} m_i \left|\mathbf{v}_i\right|^2 + m_i \omega_i \cdot \mathbf{v}_i \times \mathbf{c}_i + \frac{1}{2} \left|\omega_i\right|^2 I_i. \qquad (7)$$

To connect the links, each link inherits as its translation the position of the previous link's endpoint, with the base's translation, $\mathbf{P}$, serving as a translation parameter for the whole model. The translational velocity $\mathbf{v}_i$ of the i-th link is thus

$$\begin{aligned} \mathbf{v}_i &= \frac{d\mathbf{P}}{dt}, \quad i = 0 \\ &= \mathbf{v}_{i-1} + \mathbf{r}_{i-1} \times \omega_{i-1}, \quad \text{otherwise} \end{aligned}$$

where $\mathbf{r}_{i-1}$ is a vector from the $(i-1)$-th link's center of rotation to its point of attachment with the i-th link. The total kinetic energy T is obtained by recursively substituting this expression into equation 7 to obtain the T_i's, and summing over i.

5.2 Muscles

Luxo's muscles are three angular springs, one situated at each joint. The spring force on the joint connecting the i-th and $(i+1)$-th links is defined by

$$F_i = k_i(\phi_i - \rho_i),$$

where k_i is the stiffness constant, ϕ_i is the joint angle, and ρ_i is the rest angle. Our model is parameterized by link orientations rather than joint angles. The joint angle is $\phi_i = \theta_{i+1} - \theta_i$, the difference between the orientations of the surrounding links. The generalized force on θ_i, the orientation of the i-th link, due to the j-th muscle is

$$\begin{aligned} Q_i &= \sum_j F_j \frac{d\phi_i}{d\theta_j}, \\ &= k_j(\phi_j - \rho_j), \quad j = i+1 \\ &= -k_j(\phi_j - \rho_j), \quad j = i \\ &= 0, \quad \text{otherwise} \end{aligned}$$

Unlike passive springs whose stiffness and rest state are constants, k_i and ρ_i vary freely over time, allowing arbitrary time-dependent joint forces to be exerted.

6 Results

6.1 Jumping Luxo

Jumping motion was created using kinematic constraints to specify initial and final poses, with linear interpolation between the poses to create a trivial initial condition for the spacetime iteration. Another constraint was used to put Luxo on the floor during the initial and final phases of the motion. Subject to these and the physics constraint, we minimized the power due to the muscles, $F_\theta \dot{\theta}$. In one variation, we adjusted the mass of Luxo's base, leaving the situation otherwise unchanged. In another, we additionally constrained the force of contact with the floor on landing, to produce a relatively soft landing. In a final variation, we added a hurdle, together with a constraint that the jump clear the hurdle.

The pose constraints consisted of values for the three joint angles, and were applied to the first two and last two frames of motion. Because we measure velocity using a finite difference, this incorporates the additional constraint that Luxo be at rest at the beginning and end of motion. Initial values for the orientations were obtained by linear interpolation between the two poses.

The floor enters both as a kinematic constraint and as a force. In general, collision constraints appear as inequalities, but to simplify matters, we chose to specify explicitly the time intervals during which Luxo was on the floor, imposing during those times the equality constraints

$$\theta_0 - \frac{\pi}{2} = 0, \mathbf{P} - \mathbf{P}_f = 0$$

where θ_0 is the orientation of the base, $\mathbf{P}$ is the position of the center of the base, and $\mathbf{P}_f$ is a constant point on the floor. In other words, the position and orientation of the base are nailed. The limitation of this formulation,

compared to an inequality, is that the times at which contact occurs must be pre-specified, rather than allowing things to bounce freely. The floor constraint was enabled for the first and last five frames, allowing time for anticipation and follow-through. Of course, two different values were used for P_f at the start and finish, defining the start end points of the jump.

The floor constraint represents a mechanical interaction involving the transmission of force between the base and the floor. This contact force must be taken into account to satisfy the physics constraint. The simple contact model used for the jump has the base colliding with the floor inelastically with infinite friction, which means that the base comes to rest, losing its kinetic energy, at the moment of contact. The contact force is therefore whatever arbitrary force on the base—specifically, on $\mathbf{P}$ and θ_0—is required to satisfy physics in light of the floor constraint. No special provision need be made to solve for the contact forces beyond introducing additional state variables to represent them. Their values are then determined during the constraint-solving process. This method of solving for constraint forces applies to other mechanical constraints, such as joint attachments, and is closely related to the method of Lagrange multipliers.

The choice of optimization criteria is an area we have just begun to explore. In the examples shown, we sought to optimize a measure of the motion's mechanical efficiency by minimizing the *power* consumed by the muscles at each time step, which for each joint is the product of the muscle force and the joint's angular velocity. Our preliminary observation is that this criterion produces relatively fluid and natural motion, compared to kinematic smoothness criteria in terms of velocity and acceleration, which tend to come out looking somewhat arthritic.

Figure 3 shows a series of iterations leading from an initial motion in which Luxo translates, floating well above the floor, to a finished jump in which all the constraints are met and the objective function is minimized. Note that the elements of realistic motion already appear after the first iteration. The final motion shows marked anticipation, squash-and-stretch, and follow-through. From its pre-defined initial pose, Luxo assumes a crouch providing a pose from which to build momentum. The crouch is followed by a momentum-building forward-and-upward extension to a stretched launching position. While in flight, the center of mass moves ballistically along a parabolic arc determined by the launch velocity and by the force of gravity. Toward the end of the flight, Luxo once again assumes a crouched position in anticipation of landing, extending slightly while moving toward impact. This "stomp" maneuver has the effect of transferring kinetic energy into the base, where it vanishes in the inelastic collision with the floor. Following impact, luxo extends forward while compressing slightly, dissipating the remaining momentum of flight, then rises smoothly to its pre-specified final pose.

In the first variation on the basic jump, we add an additional constraint fixing the contact force on landing. The value we choose provides control over a hard-to-soft landing dimension—a large landing force leads to an exaggerated stomp, as if trying to squash a bug, while a small value leads to a soft landing, as if trying to avoid breaking something fragile. Figure 4 shows a relatively soft landing, generated under the same conditions as the basic jump except for the contact force constraint. Comparing the motion to the basic jump, we see that Luxo softened the blow of impact by squashing while moving toward impact, reducing the velocity, and hence the kinetic energy of the base. In contrast, the basic jump has a small *stretch* before impact, producing an energy-absorbing stomp.

The next variation has the same conditions as the basic jump, but the mass of the base has been doubled. The final motion is shown in Figure 5. As expected, both the anticipation and follow-through are exaggerated in compensation for the greater mass.

A final variation, shown in Figure 6, has the conditions of the soft-landing jump, but with a hurdle interposed between start and finish, and an additional constraint that Luxo clear the hurdle. As one would expect, the extra height required is gained by squashing vigorously on approaching the wall.

The jumping examples each took under 10 minutes to compute on a Symbolics 3640. While this is hardly interactive speed, it constitutes a tiny fraction of the cost of high-quality rendering.

6.2 Ski Jumping

Figure 7 shows Luxo descending a ski jump. As in the previous case, Luxo is constrained to be on the ski jump and the landing at particular time samples. The biggest difference between the ski-jump and the infinite-friction floor of the previous example is that Luxo is free to slide, with the exact positions on the ski jump and the landing left unspecified except at the top and bottom of the ski jump. In addition, there is a constraint that the orientation of the base must be tangent to the surface it is resting on.

Both the ski jump and landing exert forces on Luxo. There is a normal force which keeps him from falling through and a frictional force which is tangent to the surface and proportional to the tangential velocity. The coefficients of friction were state variables in the optimization.

At one time instant while Luxo is in the air, the height of his base is constrained. In addition, there is a term in the objective function which gives him a preference for a particular pose while in the air. This is a "style" optimization without which Luxo is content to go through the air in a bent position.

Luxo is also given pose constraints at the beginning and

end of the motion. Unlike the previous jumps, however, his initial velocity is unconstrained.

The initial condition for the optimization was a uniform translation in the air above both the ski jump and the landing. In the first iteration, Luxo puts his feet on the ski jump and landing. By iteration 4, there is significant anticipation and follow through. Figure 7 is the result after 16 iterations.

Both the ski jump and landing are built from two B-spline segments. The entire jump was computed with 28 time samples in the optimization. There were 223 constraints and 394 state variables. The Jacobian contained 3587 non-zero entries, about 4% of the total number of entries. The entire motion was computed in 45 minutes on a Symbolics 3600.

7 Discussion

Our results show that spacetime methods are capable of producing realistic, complex and coordinated motion given only minimal kinematic constraints. Such basic attributes as anticipation, squash-and-stretch, follow-through, and timing emerge on their own from the requirement that the kinematic constraints be met in a physically valid way subject to simple optimization criteria.

The principle advantage of spacetime methods over simple keyframing is that they do much of the work that the animator would otherwise be required to do, and that only a skilled animator *can* do. Motions that would require highly detailed keyframe information may be sketched out at the level of "start here" and "stop there." This is a profoundly different and more economical means of control than conventional keyframing affords, an advantage that easily outweighs the greater mathematical complexity and computational cost of the method.

Beyond sparser keyframing, spacetime methods offer really new forms of motion control. For example, we saw in the previous section that constraints on forces, such as the force of a collision, can be used in a direct and simple way to say "hit hard" or "hit softly," producing subtle but very effective changes in the motion.

Of the new opportunities for motion control, perhaps the most exciting is the selection of optimization criteria to affect the motion globally, an area we have only begun to explore. With a little thought, it is clear that a magic "right" criterion, whether based on smoothness, efficiency or some other principle, is unlikely to emerge and would in any case be undesirable. This is because the "optimal" way to perform a motion, as with any optimization, depends on what you're trying to do. Consider for example several versions of a character crossing a room: in one case, walking on hot coals; in another, walking on eggs; in another, carrying a full bowl of hot soup; and in still another, pursued by a bear. Plainly the character's goals—and attendant criteria of optimality—are very different in each case. We would hope to see these differing goals reflected in the motion. The possibility of controlling motion directly in terms of its goals, not just where it goes but how, is one we intend to explore.

References

[1] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constaints. *Computer Graphics*, 22:179–188, 1988.

[2] Kurt Fleischer and Andrew Witkin. A modeling testbed. In *Proc. Graphics Interface*, pages 127–137, 1988.

[3] Phillip Gill, Walter Murray, and Margret Wright. *Practical Optimization*. Academic Press, New York, NY, 1981.

[4] Michael Girard and Anthony A. Maciejewski. Computational Modeling for the Computer Animation of Legged Figures. *Proc. SIGGRAPH*, pages 263–270, 1985.

[5] Herbert Goldstein. *Classical Mechanics*. Addision Wesley, Reading, MA, 1950.

[6] David Haumann. *Topics in Physically Based Modeling, Course Notes*, volume 16, chapter Modeling the Physical Behavior of Flexible Objects. SIGGRAPH, 1987.

[7] Paul Issacs and Michael Cohen. Controlling dynamic simulation with kinematic constraints, behavior functions and inverse dynamics. *Computer Graphics*, 21(4):215–224, July 1987. Proc. SIGGRAPH '87.

[8] Charless Klein and Ching-Hsiang Huang. Review of Pseudoinverse Control for Use With Kinematically Redundant Manipulators. *IEEE Trans. SMC*, **13**(3), 1983.

[9] Johen Lasseter. Principles of traditionial animation applied to 3D computer animation. *Computer Graphics*, 21(4):35–44, 1987.

[10] Pixar. *Luxo, Jr.*, 1986. film.

[11] W.H. Press, B.P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C*. Cambridge University Press, Cambridge, England, 1988.

[12] Robert S. Stengel. *Stochastic Optimal Control*. John Wiley and Sons, New York, New York, 1986.

[13] Demetri Terzopoulos, John Platt, Alan Barr, and Kurt Fleischer. Elastically deformable models. *Computer Graphics*, 21(4), July 1987. Proc. SIGGRAPH '87.

[14] Jane Wilhelms and Brian Barsky. Using dynamic analysis to animate articulated bodies such as humans and robots. *Graphics Interface*, 1985.

[15] Andrew Witkin, Kurt Fleischer, and Alan Barr. Energy constraints on parameterized models. *Computer Graphics*, 21(4):225–232, July 1987.

Figure 3: From top to bottom, a series of iterations leading from an initial motion in which Luxo translates, floating above the floor, to a finished jump in which all the constraints are met and the optimization function is minimized. The final motion shows marked anticipation, squash-and-stretch, and follow-through.

Figure 4: A variation on the basic jump in which the contact force on landing is constrained to be small. The force of impact is reduced by squashing just before landing, reducing the velocity and hence the kinetic energy of the base. In contrast, the jump in Figure 3 exhibits a slight *stretch* before impact, producing an energy-absorbing stomp.

Figure 5: The mass of Luxo's base has been doubled. In other respects, the conditions are the same as those producing the basic jump.

Figure 6: Hurdle Jump

Figure 7: Ski Jump

Figure 8: Spacetime constraints: a cartoonist's view. (c) 1988 by Laura Green, used by permission.