Interactive Dynamics

Andrew Witkin
Michael Gleicher
William Welch
School Of Computer Science
Carnegie Mellon Univserity
Pittsburgh PA 15213

Abstract

Our goal is to use physical simulation as an interactive medium for building and manipulating a wide range of models. A key to achieving this goal is the ability to create complex physical models dynamically by snapping simple pieces together, integrating the process of model creation into the ongoing simulation. We present a mathematical and computational formulation for constrained dynamics that makes this possible, allowing encapsulated objects, constraints, and forces to be combined dynamically and simulated efficiently. The formulation handles arbitrary objects, including nonrigid bodies. We describe an implementation for interactive dynamics, and discuss applications to mechanism construction, geometric modeling, interactive optimization data fitting, and animation.

Keywords — Constraints, Simulation, Animation

1 Introduction

Physical simulation by computer has traditionally filled a niche as a useful, if cumbersome, tool for quantitative analysis and prediction. The skill and labor required to set up a simulation, followed by hours or days of run time, have restricted its practice to a hard core of dedicated specialists.

The increasing availability of high-performance computers with fast 3D graphics has for the first time made it feasible to perform non-trivial physical simulations—and see the results—at fully interactive speeds. This development opens the door to a host of new and exciting uses for the machinery of physics—for example, virtual worlds in which a user performs physical experiments, or builds and tinkers with simulated machines, or even visualizes and manipulates abstract mathematical objects with physics serving as the user interface. Such systems could be of value to a broad range of users, who need not necessarily understand the underlying mathematics. Because all of us are skilled at manipulating the physical world around us, it makes sense to use simulated physics as a medium for interaction even where the physical model is just an analogy to something more abstract.

Raw performance is a prerequisite to interactive physics, but creating a truly interactive physical medium entails much more than just making simulations run faster. Traditionally, the model creation phase is completely separate from the actual execution of the simulation. The former often involves manual derivation and coding of the system's equations of motion. Speeding up the simulation can provide the ability to *manipulate* a pre-defined model, but this capability is of limited use without the ability to dynamically *create* new models and modify existing ones. For example, a tinkertoy world for assembling and experimenting with virtual mechanisms would be of little interest if adding new pieces, making and breaking connections, and so forth entailed exiting the program, writing code, re-compiling and re-linking. To maintain the virtual world illusion, the pieces must snap together and apart, transparently and dynamically, while the simulation is running.

A key to dynamically creating virtual physical objects lies in the proper treatment of constraints. Constraints provide the glue that combines simple objects to form complex ones, by representing the bolts, joints, sliders, etc. that turn a bag of parts into a mechanism. Constraints also permit a user to define

[0]The work reported in this paper was supported in part by Apple Computer.

objects' form and behavior by stating what is desired, rather than by explicit specification of shape or motion.

Incorporating constraints into an interactive setting poses a difficult problem: the addition (or deletion) of a constraint on a physical system structurally changes the system's equations of motion, reflecting the exchange of forces that causes the constraint to hold. An interactive medium must respond to these changes, forming and solving the new equations of motion, automatically and without noticeable delay. The challenge becomes still more difficult if we insist on preserving the generality of the solution. In particular, we wish to avoid restricting its scope to collections of rigid bodies, as, for example, do [1, 12, 5, 11]).

In this paper we present a solution to the problem posed above, describe its implementation, and discuss several interactive applications. First, we develop a mathematical formulation for constrained dynamics, similar to that of [9], and more loosely related to [3]. A constraint force that is a linear combination of constraint gradients is imposed, projecting the system's acceleration onto the subspace of "legal" accelerations. Calculating the constraint force is a linear problem, even when the constraints are non-linear. The general constraint equations are intrinsically global, dealing with all the objects, forces, and constraints comprising the physical model. We next develop a decomposition of these equations in terms of the contributions of individual elements, without loss of generality. This allows us to reconcile their inherently monolithic nature to the requirements of dynamic construction and encapsulation. We also describe the more general system for dynamic function composition on which our implementation of this structure is based.

We conclude by describing several applications of interactive dynamics The first is a "tinkertoy world," a virtual 3-D environment in which the user is able to create and manipulate pre-defined parts, dynamically attach them using a variety of constraints, and experiment with the resulting structures and mechanisms. The second is a two-dimensional system in which parametric curves are manipulated and attached together using constraints. We then demonstrate the use of dynamics as a medium for the interactive solution of non-linear problems in constrained optimization, image interpretation and model fitting. Finally, we describe the use of interactive dynamics as a medium for creating keyframe animation of characters built by pinning together elastic pieces.

2 Constrained Dynamics

In classical mechanics, constraints play a role as a means of describing physical systems. Taking the standard example of a bead sliding freely on a rigid wire, an important aspect of the bead's behavior can be summarized just by observing that "the bead stays on the wire, no matter what."

Constraints such as the bead-on-wire have physical consequences. Treating the bead as a particle whose motion is governed by $\vec{f} = m\vec{a}$ gives a relation between its motion and the *total* force on it: the force and acceleration lie in the same direction, with their magnitudes scaled by m. But the bead-on-wire constraint implies that the bead will never accelerate in a way that moves it off the wire, whatever force is applied. In the special case of a straight wire, this just means that the bead's acceleration, and therefore the total force, must lie tangent to the wire, even if the force we apply to the bead points in some other direction. An immediate consequence is that the applied force, $\vec{f}_a$, cannot be the total force. Rather, there must also be some other force, a *constraint force* $\vec{f}_c$, such that the total force

$$\vec{f} = \vec{f}_a + \vec{f}_c = k\vec{t}, \tag{1}$$

where $\vec{t}$ is the wire's tangent and k is some scalar. In words, the force f_c is whatever force needs to be added to the applied one to make the bead accelerate in a manner consistent with the constraint.[1]

Constrained dynamics is concerned with making objects' behavior consistent with the forces of constraint. The mathematics of constrained dynamics are hardly new (see any standard classical mechanics text, such as [6],) although they have begun to appear only recently in the Computer Graphics literature [2, 10, 9, 7, 3, 14, 13]. In this section we address the problem of constrained dynamics in light of the requirements of interactivity: that we be able to freely add or delete constraints in an ongoing simulation, with minimal restrictions on the form of the constraints or the nature of the objects.

2.1 Restoring forces

Curiously, the constraint force of equation 1 depends on the applied force, as if the bead and wire were somehow sensing the applied force and responding actively to our attempts to pull them apart. Obviously, no such mechanism exists. We begin by understanding where this dependency comes from.

[1] Equation 1 obviously doesn't determine $\vec{f}_c$ uniquely. This will be dealt with later.

Naturally, the bead-on-wire constraint is an idealized approximation. A more accurate physical description would show the bead and wire deforming a tiny bit as we tried to pull them apart, inducing a restoring force that cancels the applied force. To simplify things, we can think of this restoring force as a rubber band connecting the bead to the wire, with force $-kc$, where c is the displacement of the bead off the wire, and k is the stiffness of the rubber band. In order for the constraint to hold approximately at equilibrium, i.e. $c \leq \epsilon$, the stiffness k must be sufficiently large that the restoring force $-k\epsilon$ cancels any applied force, i.e. $k = f_{max}/\epsilon$, where f_{max} is the largest force we plan to apply. To make ϵ small, we must make k large.

The difficulty with making k large is that doing so produces differential equations that are numerically intractable, appropriately called *stiff* equations. To understand the problem's origin intuitively, consider what happens when the bead is at rest on the wire and you try to pull it off with constant force f_{max}. The applied force begins to displace the bead, and the rubber band begins to exert a restoring force proportional to the displacement. The restoring force balances the applied one when $c = \epsilon$. When we solve this system using simple numerical methods, the distance traveled by the bead accelerating from rest under force f_{max} in a single timestep Δt must plainly be on the order of ϵ to avoid substantial overshoot and instability. In short, the step size must be so small that the largest permitted applied force f_{max} makes objects move only a negligible distance ϵ in a single timestep, which means you never get anywhere. So, although stiff rubber bands may be a good description of what really happens, they are not a good way to enforce constraints numerically.

2.2 Constraint forces

Ironically, the problem of stiffness is avoided by letting ϵ go to zero (and the stiffness k to infinity.) In this limiting case, the rule $c = 0$ really does govern the system exactly. Since there are no displacements, and hence nothing to restore, the restoring force is renamed a *constraint force.* In addition to depending on the state of the system and on time, as most forces do, constraint forces depend on other *forces.* In this section we develop a system of linear equations that yield constraint forces which, added in to the ordinary applied forces, lead the system to accurately satisfy the constraints.

To make this result general, we switch at this point from the specific case of a bead on a wire to the generic one of a system whose equations of motion have the form

$$M_{ij}\ddot{q}_j = C_j + Q_j \tag{2}$$

where M is a mass matrix, q is the vector of the system's independent variables, Q is the vector of (known) applied forces, and C is the vector of (unknown) constraint forces.[2] This equation is just $f = ma$ in generalized form. Ultimately, our goal is to solve for $\ddot{q}$, given q, and $\dot{q}$, allowing us to integrate the differential equation forward through time.

Rather than a single constraint, we have a vector of constraint functions $c_i(q_j, t)$, depending on the state q, and possibly directly on time. The constraints are met when $c_i(q_j, t) = 0$. The constraint equation itself provides another condition on C. For c to remain at 0 from some initial time t_0, it suffices that $c(t_0) = 0$, $\dot{c}(t_0) = 0$, and $\ddot{c} = 0$ from t_0 on. If c depends on time directly and also through the state q, we have from the chain rule

$$\dot{c}_i = \frac{\partial}{\partial t} c_i(q_j(t), t) = \frac{\partial c_i}{\partial q_j}\dot{q}_j + \frac{\partial c_i}{\partial t},$$

and differentiating again gives

$$\ddot{c}_i = \frac{\partial c_i}{\partial q_j}\ddot{q}_j + \frac{\partial \dot{c}_i}{\partial q_j}\dot{q}_j + \frac{\partial^2 c_i}{\partial t^2}, \tag{3}$$

noting that

$$\frac{\partial \dot{c}_i}{\partial q_j} = \frac{\partial^2 c_i}{\partial q_j \partial q_k}\dot{q}_k.$$

If W is the inverse of mass matrix M, then equation 2 becomes

$$\ddot{q}_j = W_{jk}(C_k + Q_k).$$

Substituting into 3 and setting $\ddot{c}$ to zero yields the condition

$$\frac{\partial c_i}{\partial q_j} W_{jk}(C_k + Q_k) + \frac{\partial \dot{c}_i}{\partial q_j}\dot{q}_j + \frac{d^2 c_i}{dt^2} = 0. \tag{4}$$

which is a system of *linear* equations with only the constraint force vector C unknown. In words, equation 4 just says that the constraint force, added into the applied force, must cause the second time derivative of the constraints to be zero. This condition is generally too weak: if the system is underconstrained, as is usually the case (otherwise nothing can move at all!) we have fewer equations than unknowns, and there exist many values for C that satisfy equation 4.

[2] In index notation, an unsubscripted quantity is a scalar, one subscript denotes a vector, and two denote a matrix. Under the *summation convention,* the appearance of any index twice in a term implies summation, so that $M_{ij}q_j$ means $\sum_j M_{ij}q_j$, which is matrix M times vector q.

2.3 Virtual work

The ambiguity of equation 4 is easy to understand. The equation states that the system's acceleration must not move the constraint functions from zero, but in an underconstrained system, a whole subspace of such "legal" accelerations exist. Given a constraint force that satisfies equation 4, nothing said so far prohibits us from adding to it any additional force we like, as long as the acceleration it induces lies in that legal subspace. To remove this ambiguity it suffices to add one reasonable restriction: that the constraint never add or remove energy from the system, which is to say that it may do no work. To guarantee this we require that the the work done by the constraint force vanish, under any small displacement of the system *consistent with the constraints.* Thus, for every legal displacement dq, C must satisfy $C_j dq_j = 0$, which simply requires the constraint force to point in a direction in which the system is forbidden to move. This requirement, known as the *principle of virtual work,* is not derived from anything else. It is a restriction, albeit a reasonable one, on the class of constraints to be considered.

In the case of a single scalar constraint c, the "legal" displacements are those lying in the tangent plane to the surface $c = 0$. Because the gradient $\partial c/\partial q$ is normal to the tangent plane, this means that every legal displacement must satisfy $(\partial c/\partial q)dq = 0$. The forbidden displacements are those that satisfy $dq = \lambda(\partial c/\partial q)$ for any scalar λ. The multidimensional generalizations of the tangent plane and the gradient direction are the *null space* and *null space complement* of the constraint Jacobian matrix. The null space contains the displacements satisfying

$$\frac{\partial c_i}{\partial q_j} dq_j = 0,$$

while the null space complement contains those that satisfy

$$dq_j = \lambda_i \frac{\partial c_i}{\partial q_j}$$

for some vector λ_i. Viewing the constraint vector as a collection of scalar constraints, the null space is the set of vectors which lie in every constraint's tangent plane, while the null space complement is the set of linear combinations of the constraint gradients.

To lie in the null space complement, the constraint force must therefore satisfy

$$C_i = \lambda_i \frac{\partial c_i}{\partial q_j}$$

for some vector λ. Enforcing the virtual work principle is simply a matter of replacing C_k by $\lambda_r \partial c_r/\partial q_k$ in equation 4, and solving for λ. The components of λ are known as *Lagrange multipliers.* This substitution, with some re-arrangement, yields

$$-\left[\frac{\partial c_i}{\partial q_j} W_{jk} \frac{\partial c_r}{\partial q_k}\right] \lambda_r = \frac{\partial c_i}{\partial q_j} W_{jk} Q_k + \frac{\partial \dot{c}_i}{\partial q_j}\dot{q}_j + \frac{\partial^2 c_i}{\partial t^2}, \tag{5}$$

in which the entire right hand side is known, and the matrix on the left hand side—a product of the constraint Jacobian, the inverse mass matrix, and the Jacobian transpose—is a square matrix with the dimensions of the constraints.[3] Once the linear system is solved, the constraint force is computed as $C_j = \lambda_i \partial c_i/\partial q_j$, and the total force $C_j + Q_j$ is plugged into equation 2 to obtain the acceleration, $\ddot{q}_j$.

2.4 Feedback

In principle, it suffices to begin with legal initial conditions, in which $c = 0$ and $\dot{c} = 0$, and ensure that $\ddot{c} = 0$ thereafter by solving equation 5 for the constraint force. In practice, an extra feedback term is needed to bring the system into a legal state initially, and to inhibit drift. Including the damped feedback term, the total force becomes

$$Q_j + C_j + \alpha c_i \frac{\partial c_i}{\partial q_j} + \beta \dot{c}_i \frac{\partial c_i}{\partial q_j},$$

where α and β are constants. This additional term is effectively a damped spring pulling the system back towards a legal state. Because it vanishes when the system is in a legal state, with $c = 0$, and $\dot{c} = 0$ the feedback is not a true force. Feedback may be incorporated into the constraint force directly by making a small modification to equation 4, as described in [9]. However, we have not found this to be advantageous, particularly in obtaining least-squares solutions to overconstrained systems.

2.5 First order systems

When the machinery of constrained dynamics is to be used as a tool for manipulating purely geometric or otherwise non-physical objects, it is often desirable to replace the second order equations of motion with a first order system of the form

$$\dot{q}_i = W_{ij} Q_j,$$

in effect replacing $f = ma$ by $f = mv$, approximately modeling the behavior of a highly damped system

[3] If it is not desired to invert the mass matrix explicitly, a larger but sparser linear system may be formed that involves the mass matrix M instead of W. See [9].

with negligible mass. The effect of this change is simply that things stop moving the moment forces are withdrawn, which facilitates accurate positioning in geometric modeling applications. One such application will be described later. The form of equation 5 changes only slightly when first order equations are adopted: the term $\partial\dot{c}/\partial q$ disappears, and instead of the second direct time derivative, we have the first, $\partial c/\partial t$. The derivation of the first-order constraint equation follows closely that of equation 5, except that $\dot{c}$ rather than $\ddot{c}$ is being held at zero, and first order equations of motion are used.

3 Decomposition

Equation 2 and equation 5 of the previous section are "universal," in the sense that the equations of motion and the constraints are represented in generic, anonymous form, rather than representing any particular constrained system. The equations are also intrinsically global: all the objects, constraints, and forces in a system are coupled, with each constraint force generally depending on every other, as well as on the applied forces, and on the positions, velocities, and mass matrices of all the objects.

How are these monolithic equations to be applied to specific systems of interacting objects and constraints? Tackling a toy problem by hand, as in most textbook examples, we would simply use the generic equations as a template, filling in the blanks with the problem specifics. One such toy problem is a dumbbell, represented as two unit-mass particles constrained to lie a distance r apart. This system's state vector, q, holds six elements, representing the two particles' positions,

$$q = [x_1, y_1, z_1, x_2, y_2, z_2],$$

and its mass matrix is the identity. The equations of motion, expressed in terms of the three components of force on each particle, are just

$$\ddot{q} = [fx_1, fy_1, fz_1, fx_2, fy_2, fz_2].$$

The single scalar constraint, to be held at zero, can be written

$$c = r - ((x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2)^{1/2},$$

or, in terms of q-components

$$c = r - ((q_1 - q_4)^2 + (q_2 - q_5)^2 + (q_3 - q_6)^2)^{1/2}.$$

Having written out these expressions, and by elementary differentiation produced a moderately ugly expression for $\partial c/\partial q$, one may then flesh out the skeletal constraint equation[4] and solve for the constraint force, which is then plugged back into the equations of motion, along with the applied force and the feedback term.

On a small-scale example such as this, it would not be difficult to complete and implement the exercise. It should be obvious, however, that this kind of substitution and expansion is *not* the way to build large-scale constrained models interactively. Each time an object or a constraint is added, modified or deleted, algebraic manipulations must be performed to derive the new equations, and the results somehow put into a form that supports efficient numerical evaluation. Obviously, a system in which attaching or detaching two objects triggers extensive algebraic manipulation, code generation, compilation, and linking would be unlikely to achieve interactive performance, even if the symbolic algebra and code generation could be automated.

Fortunately, the system of objects, constraints, and forces defining a model need not be allowed to dissolve into a massive unstructured algebraic expression. In this section we will see that all of the global quantities contained in equations 2 and 5 may be constructed by composing a small, stylized set of local quantities, each depending only on a single object or a single constraint. To exploit this mathematical regularity, we require each primitive object or constraint to compute the local quantities for which it is responsible, performing the composition dynamically. In this section we describe the mathematical decomposition of the global equations into local quantities. The structure we develop is illustrated schematically in figure 1. The next section describes the efficient implementation of dynamic composition.

3.1 State vectors and mass matrices

The state, q, of a compound model is distributed over the objects it contains. The state of each object may itself be heterogeneous, containing scalars, vectors, matrices, quaternions, or whatever, in any combination. In implementing an object it may be important to preserve this internal structure. From outside, however, it may be hidden by collapsing the state into a single vector, and providing operations to determine the length of the flattened vector, to get and set the state and its time derivative, etc, allowing the object's mapping between its internal state and the external state vector to remain hidden. The

[4] Note that in this instance $\partial\dot{c}/\partial q$ and $\partial^2 c/\partial t^2$ are both zero, and that the inverse mass matrix, W, is the identity.

global state vector is formed just by concatenating the objects' state vectors.

Similarly, each object contributes a square block to the global mass matrix, situated on the diagonal. In this special case of non-overlapping diagonal blocks, the inverse of the global mass matrix is obtained by inverting each object's matrix independently. The inverted diagonal blocks can then be combined to form the global matrix W.

3.2 Constraints and Connectors

The global constraint vector, like the state vector, is formed by concatenating the contributions of each constraint. In order to evaluate the constraint functions, and the Jacobian matrix that relates the constraints to the state, a new layer of structure must be introduced. In the global equations, the constraint vector c was given as a function of the state q and of time. Generally, though, the dependence of constraints on state is indirect, mediated by quantities, such as coordinates of points on the surfaces of objects, that may be viewed as "outputs" of the objects, pieces of geometry that "move with" the object in the sense that their values depend on state. For example, a point on a circle, with coordinates $[r\cos\theta + x_0, r\sin\theta + y_0]$ for a constant θ, tracks changes in the radius r and the center $[x_0, y_0]$. A *connector* is any such fragment of geometry, encapsulated with any constant information (such as θ for the circle point) that is required to define it. In addition to representing points on surface, connectors can also represent surface normals, areas and volumes, or anything that might be subjected to a constraint, or to which a force might be applied.

The benefit of introducing connectors is that they allow us to formulate generic constraints—e.g. attaching two points together—without the need to know anything in advance about the objects being constrained. Consider an arbitrary equality constraint c on a pair of points a and b, which could be written

$$c_i(q_j) = f_i(a_k(q_j), b_k(q_j))$$

where f is whatever function defines the constraints (just subtraction in the case of an attachment constraint,) using whatever formulae determine a and b as functions of their respective objects' state. From the standpoint of decomposition and encapsulation, it is significant that the function $f(a,b)$ is only a property of the constraint, not the constrained objects, while the position functions $a(q)$ and $b(q)$ are properties only of the two constrained objects, not of the

Figure 1: A schematized model. The three objects' state vectors are concatenated to form the global state vector, and the two constraints' outputs form the global constraint vector. The constraints depend on state through connectors, which represent outputs of the objects. The whole structure defines the global constraint function $c_i(q_j)$. Each constraint-object pair defines a block in the constraint Jacobian matrix. The block may be non-zero only when the constraint depends on the object.

constraint. We can write the constraint's Jacobian as

$$\frac{\partial c_i}{\partial q_j} = \frac{\partial c_i}{\partial a_k}\frac{\partial a_k}{\partial q_j} + \frac{\partial c_i}{\partial b_k}\frac{\partial b_k}{\partial q_j} \tag{6}$$

by simple application of the chain rule, and again each of the four derivative matrices appearing in expression belongs to exactly one object or constraint. Each pairing of a constraint with an object generates a block in the global Jacobian matrix. Only if the constraint depends on the object may the block be nonzero (figure 1).

In a similar vein, the matrix $\partial \dot{c}_i/\partial q_j$ appearing in equation 5 may be written

$$\frac{\partial \dot{c}_i}{\partial q_j} = \frac{\partial \dot{c}_i}{\partial a_k}\frac{\partial a_k}{\partial q_j} + \frac{\partial \dot{c}_i}{\partial \dot{a}_k}\frac{\partial \dot{a}_k}{\partial q_j} + \frac{\partial \dot{c}_i}{\partial b_k}\frac{\partial b_k}{\partial q_j} + \frac{\partial \dot{c}_i}{\partial \dot{b}_k}\frac{\partial \dot{b}_k}{\partial q_j} \tag{7}$$

which once again preserves modularity. If the constraint depends directly on time, this dependence is by definition encapsulated within the constraint, and so involves no composition. Finally, it remains to evaluate the constraint vector itself and its time derivative, as required in the feedback term. This is a simple matter of function composition, given as

$$c_i = f_i(a_k(q_j), b_k(q_j))$$

and

$$\dot{c}_i = \frac{\partial c_i}{\partial t} + \frac{\partial c_i}{\partial a_k}\dot{a}_k + \frac{\partial c_i}{\partial b_k}\dot{b}_k.$$

The generalization to constraints with any number of inputs is straightforward—all the above expressions become summations over the inputs.

3.3 Forces

Finally, a force f may be applied to any connector output x using the simple universal formula

$$Q_j = f_i \frac{\partial x_i}{\partial q_j}, \tag{8}$$

which is the formula for transforming an applied force into a *generalized force* on the state. The total generalized applied force is obtained by summing each applied force's contribution.

3.4 Summary

The formulae given require that only a very few distinct quantities be computed by each object and each constraint. An object must be able to report its state length L, get and set its state q and velocity $\dot{q}$, and compute its inverse mass matrix W. A connector on an object must be able to compute its output x, the time derivative $\dot{x}$, and the two derivative matrices $\partial x/\partial q$ and $\partial \dot{x}/\partial q$. A constraint must be able to evaluate its output, c, given its inputs, the direct time derivatives $\partial c/\partial t$ and $\partial^2 c/\partial t^2$, and, for each input x, the derivative matrices $\partial c/\partial x$, $\partial \dot{c}/\partial x$, and $\partial \dot{c}/\partial \dot{x}$.

Provided that each part is able to perform these evaluations, constructing the constraint equations and equations of motion governing an arbitrary system of objects, constraints, and applied forces is a comparatively simple operation, easily performed dynamically. The operations required to assemble the global equations are are just global index assignment, function composition, and matrix multiplication. The next section addresses some aspects of the efficient implementation of the process.

4 Implementation

4.1 Function blocks

The assembly of constraint equations is an instance of a larger class of problems, involving the dynamic composition of mathematical functions, and evaluation of the outputs and of their derivatives with respect to inputs. Our implementation of constrained dynamics is built on a facility, called *function blocks,* designed to handle this broader class.

A function block encapsulates a real-valued mathematical function that maps some inputs to some outputs. Each block supports operations that evaluate its outputs given its inputs, and also its Jacobian matrix—the derivative of its outputs with respect to its inputs.

The implementor's task in creating a new block type is to provide code that computes the function and its Jacobian. This task is sufficiently regular that we have automated the process to the degree that only the mathematical form a block embodies need be specified, the rest being generated by symbolic differentiation, simplification, and conversion of the expressions to code.

Complicated functions are built by creating directed acyclic graphs whose nodes are function blocks, and whose arcs, connecting inputs to outputs, denote function composition. At runtime, the function block library provides a variety of support services for creating and deleting connections, doing the bookkeeping associated with global indexing, etc.

Evaluation of an output of the graph can be a simple recursive descent, each block instructing the blocks that provide its inputs to compute their outputs, then computing its own. The recursion bottoms out at special nodes that hold the system's state.

Caching is used to avoid the redundant computation of shared quantities.

The evaluation of Jacobians involves a recursive application of the chain rule. If a block implements a function $f_i(x_j)$, then, by the chain rule, its derivative with respect to a vector of variables q_k, on which the block's inputs presumably depend, is

$$\frac{\partial f_i}{\partial q_k} = \frac{\partial f_i}{\partial x_j}\frac{\partial x_j}{\partial q_k},$$

which is just the product of the block's internal Jacobian with the Jacobian of its inputs with respect to the q's. Thus the Jacobian may be computed recursively, each block instructing its inputs to compute their Jacobians, then multiplying the collected input Jacobian by its internal one. The recursion bottoms out at the state nodes, where

$$\frac{\partial x_j}{\partial q_k} = \frac{\partial q_j}{\partial q_k} = \delta_{jk},$$

which is the identity matrix.

In practice, efficient Jacobian evaluation is far more complicated than the recursive evaluation of the function itself, because the matrices are typically sparse, and it is vital that their sparsity be preserved and exploited. Other complications arise involving, for example, issues of the allocation of storage for intermediate matrices. The naive recursive descent algorithm, even with caching, is therefore not necessarily the best. See [4] for a general discussion of sparse matrix techniques.

4.2 Physobs

Our implementation of interactive constrained dynamics employs a more specialized layer, called *physobs,* built on the generic machinery of function blocks. The classes that make up this layer correspond to the elements described in the previous section: physical objects, connectors, and constraints. In addition, *behaviors* apply forces to connectors, implementing springs, dampers, motors, and the like, and a *world* structure performs such global functions as solving the linear system and the resulting constrained differential equation.

The function block machinery automatically handles the maintenance of global coordinates for the state and constraint vectors; the dynamic composition of the constraint functions and their derivatives with respect to state; and a variety of bookkeeping and support functions.

Figure 2: The tinkertoy system is a 3D environment for interactive model construction and manipulation.

5 Applications

We are developing a number of applications of interactive dynamics. In this section we describe several of these. A major purpose in developing these experimental systems has been to explore the range of problems to which interactive dynamics applies.

5.1 Tinkertoys

A basic motivation of our research has been the desire to build and manipulate virtual 3D mechanisms. The tinkertoy system (figure 2) allows the user to build contraptions, using constraints to snap together predefined parts, with no artificial distinction between model construction and simulation. The user of the system need have no understanding of the underlying mathematics and physics.

5.2 Geometric Modeling

Another experimental system is concerned with the interactive construction and manipulation of models composed of arbitrary parametric curves (figure 3.) The idea is to convert parametric curves, which are purely geometric objects, into pseudophysical objects that respond in an intuitive way to user input. The user moves and reshapes curves by freely pushing and pulling on them, providing a consistent mode for direct manipulation of all shapes.

Each curve drawn on the screen is interpreted as a physical object by assigning it negligible mass, with uniform viscous drag along its length. Under this model, a curve responds to forces by changing shape

Figure 3: A model consisting of parametric curves attached by constraints. By grabbing and pulling arbitrary points on the curve, the user may move and reshape the model subject to the constraints.

and position in accordance with the equations that define it. For example, a circle may change radius and position. Because the user controls the object directly through its appearance on the screen, the underlying parameterization is hidden, making it easy for the user to control curves whose parameters are nonintuitive or interact nonlinearly.

Attachment constraints serve to nail curves together, while springs and other forces permit the user to express preferences that are weaker than constraints. Because they are dynamic simulations, the models created are more than static drawings: the system is proving useful as a tool for experimenting with planar mechanisms, as well as a tool for constraint-based drawing.

The steps that go from the parametric equations that define the geometry of a curve or surface to the compiled code that allows a user to interact with it as a physical object involve rote differentiation, simplification, and code generation. We have fully automated these steps as an off-line compilation process, allowing a user to add a new curve type to the system just by entering the pure mathematical equations that define it.

5.3 Interactive Optimization

An additional area of interest is the use of dynamics as a medium for the interactive solution of non-linear problems in constrained optimization. The idea is to convert local minima in the objective function into attractors, so that the model is continuously "pulled" toward some local solution. The user exercises global control by dragging the model toward the desired solution, then letting go, allowing the local attractor to take over.

An earlier application of interactive optimization to computer vision is described in [8]: a dynamic 2-D curve, called a *snake* is superimposed on an image and attracted to points of high contrast. The curve's behavior approximates that of a springy, stretchy wire. Placed near an edge, the curve locks on to it and is able to track its motion. At any time, the user may grab the curve and pull it toward features of interest.

In addition to continuing the investigation of vision applications, we are exploring other tasks involving the manipulation of nonlinear models and optimal data fitting. One experimental system allows the user to define a collection of variables, enter algebraic expressions representing constraints on the variables, functions to optimize, or user-accessible outputs. The user may then directly manipulate the system subject to the constraints, using sliders to pull on the outputs. Related investigations include the dynamic fitting of parametric models to scatter data.

5.4 Troids

Troids are simplified linearly deformable bodies. A 2-D troid may be viewed as an affine transformable sheet containing mass in some distribution. Troids are imbued with internal elastic forces that make them tend toward a rest state, and tend to preserve their original area. Because the deformations they undergo are linear, troids are extremely simple objects, simpler in fact than rigid bodies. In the case of a collection of troids that are attached together or nailed in place, the constraint matrix on the left hand side of equation 5 is constant. This simplification allows us to pre-invert the constraint matrix, eliminating the need to solve a linear system at each evaluation of $\ddot{q}$. The simulation of models built from troids is therefore very fast.

Because they are defined in terms of deformations, troids may be rendered by transforming arbitrary curves, drawings, etc. created in body coordinates (see figure 4).

We are using troids as a means of rapidly creating physical keyframe animation. Control is accomplished by constraining specified points to move along user-defined trajectories. The desired acceleration of the control point along the trajectory appears as the $\partial^2 c/\partial t^2$ term of equation 5. Subject to the keyframe constraints, and the attachment constraints that hold pieces together, the system moves with passive non-

rigid physics. The resulting behavior is best compared to that of stretchy puppet whose hands, feet, etc., are directly controlled. By adjusting the stiffness and drag of the internal forces it is possible to create behaviors ranging from highly non-rigid jello-like motion to nearly rigid forms.

Figure 4: Top: a collection of troids, rendered with polygons, and attached using constraints and rubber bands. Bottom: another still from the same animation. The animation is driven by two motion paths, one attached to the arm, the other to the shirt.

References

[1] William W. Armstrong and Mark W. Green. *Visual Computer*, chapter The dynamics of articulated rigid bodies for purposes of animation, pages 231–240. Springer-Verlag, 1985.

[2] Ronen Barzel and Alan H. Barr. *Topics in Physically Based Modeling, Course Notes*, volume 16, chapter Dynamic Constraints. SIGGRAPH, 1987.

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

[4] J. S. Duff, A. M. Erisman, and J.K. Reid. *Direct Methods for Sparse Matrices.* Oxford University Press, Oxford, UK, 1986.

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

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

[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] Michael Kass, Andrew Witkin, and Demetri Terzopoulos. Snakes: Active contour models. *Int. J. of Computer Vision*, 1(4), 1987.

[9] John Platt. *Constraint Methods for Neural Networks and Computer Graphics.* PhD thesis, Caltech, 1989.

[10] John Platt and Alan Barr. Constraint methods for flexible models. *Computer Graphics*, 22:279–288, 1988.

[11] Peter Schroeder and David Zeltzer. Dynamic simulation with linear recurive constraint propogation. *Computer Graphics*, 1990. In press.

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

[13] Andrew Witkin, Kurt Fleischer, and Alan Barr. Energy constraints on parameterized models. *Computer Graphics*, 21(4):225–232, July 1987. Proc. SIGGRAPH '87.

[14] Andrew Witkin and Michael Kass. Spacetime constraints. *Computer Graphics*, 22:159–168, 1988.