To appear in the Proceedings of the 1989 IEEE International Conference on Robotics and Automation, May 14-19, Scottsdale, AZ

# The Architecture of *Newton*, a General-Purpose Dynamics Simulator

James F. Cremer
A. James Stewart
Computer Science Department
Cornell University



## Abstract

This paper describes the architecture of *Newton*, a general-purpose system for simulating the dynamics of complex physical objects. The system automatically formulates and analyzes equations of motion, and performs automatic modification of this system of equations when necessitated by changes in kinematic relationships between objects. Impact and temporary contact are handled, though currently using simple models. User-directed influence of simulations is achieved using *Newton*'s control module, which can be used to experiment with the control of many-degree-of-freedom articulated objects.

## 1 Introduction

This paper describes the architecture of *Newton*, a general-purpose model-driven simulation system. Unlike traditional simulation systems, which concentrate mainly on integrating an unchanging set of equations, and most current CAD systems, which concentrate on geometry specification but have little in the way of analysis tools, *Newton* was designed to provide a level of automatic support thats encourage the kind of experimentation necessary for successful design. By using a model-based object representation and fully integrating geometric modeling techniques, it was possible to incorporate into *Newton* a general mechanism to deal with events (called *exceptional events*) that cause discontinuities in object behavior. Thus, Newton can automatically and incrementally modify its internal description of mechanism behavior as relationships between objects change due to events such as impacts, contact breakages, or changes in control algorithm states. Such a facility greatly increases the power and flexibility of a simulation system.

One of the goals of the *Newton* project is to make the design cycle more efficient by integrating design, prototype implementation, and testing in a single system. Attempts to integrate a control algorithm and a particular mechanical system can expose flaws in either the control algorithm or in the design of the mechanical system. The *Newton* system allows immediate redesign and testing of both of these components. For example, a designer could construct an "electronic prototype" of an anthropomorphic multi-fingered robot gripper and experiment with several different configurations before committing to a specific design. Figure 1 shows designs modeled after the Salisbury hand. The designer might first choose a three-fingered model but be unable to find control algorithms that achieve design objectives. With minimal effort the designer could test control algorithms on a four-fingered model to determine whether such a model would better meet specifications. Ease of redesign facilitates discovery of an optimal match between control algorithms and mechanical design.

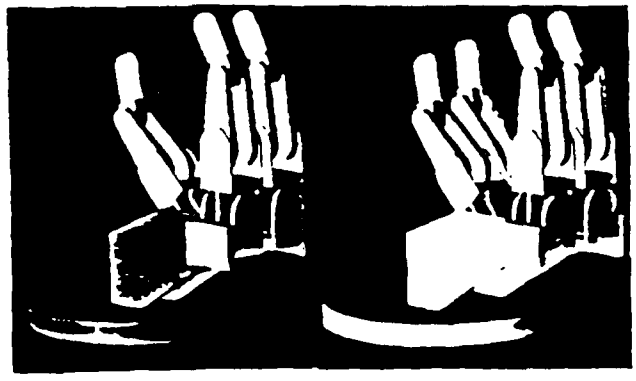

Figure 1: Different Hand Configurations

Extensive mechanical engineering research has led to many developments in physical system simulation. The ADAMS [2] and DADS [6] systems are examples of large state-of-the-art systems from the mechanical engineering domain. In many ways such systems are very sophisticated: efficient formulations of mechanism dynamics are supported, fancy numerical techniques for solving equation systems are used, object flexibility and elasticity are often handled, etc. However, from a computer science perspective, some things are lacking. Richer object representations are needed. Typically, systems have almost ignored geometric considerations and represented objects simply as point masses with associated inertias and coordinate systems. Geometric modeling techniques have matured enough to allow object representations used by dynamic simulations to include a complete geometric description usable by a geometry processing module. Furthermore,

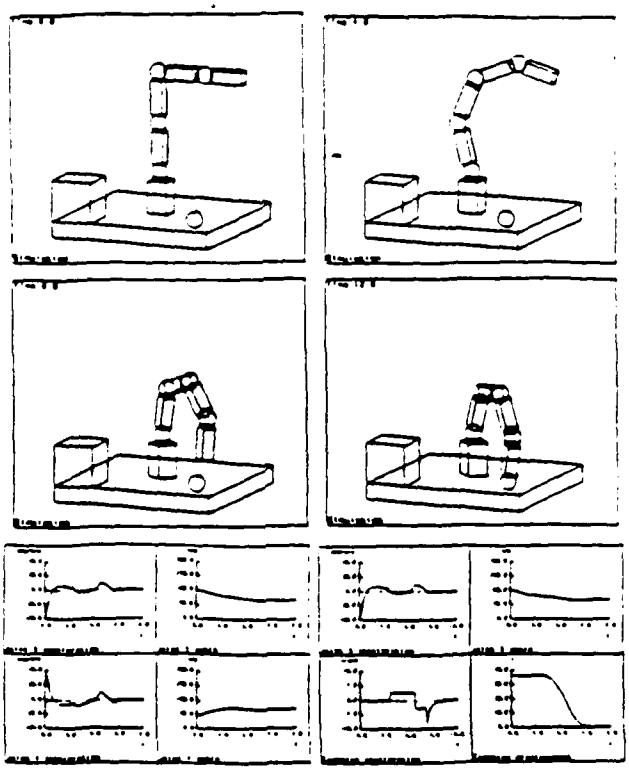

Figure 3: Redundant Arm Simulation

```
UNTIL stop-simulation DO
  save-current-state
  integrate-from-current-state
  handle-exceptional-events
  report
```

Figure 4: *Newton's* top-level loop

lation scene as execution takes place, and display of values of any quantities that the user wishes to monitor. Typically, users choose to display the evolution of values of a number of variables using a set of graphs. In Figure 3, four frames from a simulation of a redundant robot arm are shown. The graphs exhibit values of the position and acceleration of the arm's end effector, and joint angles and accelerations for the distal three joints of the arm. The report package is also responsible for the recording of information that allows later redisplay of the simulation as a real-time "movie."

## 2.3 Analysis module

*Newton*'s analysis module is responsible for overall coordination of a simulation. After defining a simulation scenario using the definition module, the main simulation loop executes according to the code in Figure 4.

At the beginning of each iteration, the current state of all objects is saved in case an unacceptable attempt at stepping forward in time by some $\Delta t$ occurs and necessitates trying again with a smaller time increment. Next, the integration subsystem is invoked in order to produce new positions and velocities for all objects from their old positions, velocities and accelerations. The call to `handle-exceptional-events` then passes control to the event-handling subsystem, which is responsible for detecting and handling collisions, contact breakages, control algorithm state changes, and other events that yield discontinuites in object velocities or accelerations. These events can invalidate the newly proposed state and necessitate restoration of the previous state, integration using a different time increment, resolution of collisions or other events, and so on. The event handler is described in more detail in Section 4 and in [3].

# 3 Dynamic Analysis

Initially, Newton-Euler equations of motion are associated with each primitive (i.e. individual rigid body).[1] At the time an object is created the equations are of the form

$$m\ddot{r} = 0$$
$$J\dot{\omega} + \omega \times J\omega = 0.$$

A specification that two objects are to be connected with a spherical hinge is met by the addition of one vectorial constraint equation and the addition of some terms to the motion equations of the constrained objects. Thus, acceleration equations become

$$\begin{aligned} m_1\ddot{r}_1 &= f_{hinge} \\ J_1\dot{\omega}_1 + \omega_1 \times J_1\omega_1 &= c_1 \times f_{hinge} \\ m_2\ddot{r}_2 &= -f_{hinge} \\ J_2\dot{\omega}_2 + \omega_2 \times J_2\omega_2 &= c_2 \times -f_{hinge} \\ \ddot{r}_1 + \dot{\omega}_1 \times c_1 + \omega_1 \times (\omega_1 \times c_1) &= \ddot{r}_2 + \dot{\omega}_2 \times c_2 + \omega_2 \times (\omega_2 \times c_2) \end{aligned}$$

where $c_i$ is the vector from object $i$'s center of mass to the location of the hinge and $f_{hinge}$ is the constraint force that keeps the objects together. Other kinds of hinges commonly used in *Newton* include revolute or pin joints, prismatic joints, springs and dampers, and rolling contacts.

If gravity is to be accounted for during the simulation the system will automatically add gravitational force terms $(m,g)$ to the objects' translational motion equations. The system keeps track of the constraints responsible for the various terms in the motion equations. Thus, constraints, and their corresponding motion equation terms, can be removed at any time without necessitating complete rederivation of the system of motion equations.

Using this method of dynamics formulation, closed-loop kinematic chains are handled as simply as open chains. Though the formulation does lead to large sets of equations, the matrices generated for solving for accelerations and constraint forces are very sparse and usually symmetric. Thus, reasonable efficiency is achieved by the use of sparse matrix techniques.

[1] *Newton* is capable of using dynamics formulations other than the one outlined here. Also, some preliminary work using non-rigid bodies has been done.

;;; The state has been saved for time $t$.
;;; New positions and velocities have been proposed for time $t + \Delta t$

1. Compute the earliest impact time $t_i$ in $[t, t + \Delta t]$.
2. Integrate from time $t$ to produce state for time $t_i$.
3. Determine and analyze all contacts.

4. WHILE there are still impacts in the contact set DO
   4.1 Formulate equations for resolving the impacts.
   4.2 Solve the equations to obtain new velocities.

Figure 5: Impact Resolution Scheme

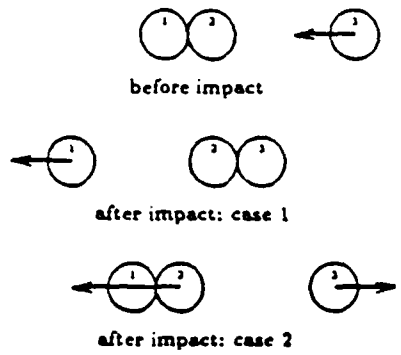


Figure 6: Different Collision Interpretations

of a set of impacts then often produces post-impact velocities for which some new subset of the contacts represent impacts. To handle this, we currently iterate the procedure until the set of contacts contains no impacts. *Newton* treats this entire process of solving a sequence of instantaneous impacts problems as occurring instantaneously. The impact handling scheme is summarized in Figure 5.

We use this impact model at present because it was relatively simple to implement and produces expected behavior in many cases. For instance, in Figure 6, using our model to resolve the collision between spheres 2 and 3 yields the normally expected behavior in which, after impact, spheres 2 and 3 are at rest and sphere 1 moves off to the left. Other impact models can be used in *Newton*. Featherstone [4], for example, details a different scheme for resolving impacts in the presence of contacts. In it, impulses are transmitted through the non-impact contacts. However, under this model, the spheres of the example behave in a less expected manner. After impact, sphere 3 moves back to the right and spheres 1 and 2 behave as if they were connected by a true hinge, moving off to the left. Still, the model does produce better results than ours in other cases, such as a large mass block falling onto a smaller mass block that is resting on a table top. It is clear that neither model is sophisticated enough to do realistic impact modeling. We are currently investigating more complex models that can better account for the elasticity properties of objects.

## 4.2 Contact

*Newton* was designed to handle continuous contacts between objects. By continuous contacts we mean contacts in which two objects remain in contact for a finite amount (not infinitesimal, as for impacts) of time. Such contact relationships — as in a block sliding or a ball rolling on a table top—are modeled in our system by extensions to hinges called *temporary hinges*. Temporary hinges generally represent one-sided, or unilateral, constraints.

During the geometric analysis of contacts, normal-direction velocities of contact points are monitored. When contact velocities are zero[2], there is continuous contact and the system creates a temporary hinge to model this relationship. During the course of simulation, the system continually monitors contact velocities, removing temporary hinges when the contact constraints are no longer met.

Determination of object accelerations is made complicated when temporary hinges exist. Constraint equations for temporary hinges are formulated in the same manner as for other hinges, and constraint force terms are again added to the motion equations of the hinged objects. For instance, for point-on-plane contact without friction the instantaneous acceleration constraint is

$$(\ddot{\rho}_1 - \ddot{\rho}_2) \cdot n_{contact} + 2(\dot{\rho}_1 - \dot{\rho}_2) \cdot (\omega_2 \times n_{contact}) = 0.$$

However, using such equality constraints when solving for object accelerations necessitates checking the results for consistency. Since the equation solving procedure calculates values for hinge reaction forces in addition to calculating object accelerations, the system is able to check that the values of the reaction forces for any temporary hinges are consistent with that hinge's intended inequality constraint. For the point-surface contact case the system needs to check that the normal-direction component of the reaction force is not tensile, since a contact hinge should only sustain compression.

For two polyhedral objects in contact, the region of contact will be either a polygon, a line segment or a point. For the case of a polygonal region, it is sufficient to use only vertices of the polygon's convex hull in formulating the temporary hinge constraints. The system first assumes that polygonal support will be maintained and searches for a "support triangle" among the convex hull vertices. If no support triangle produces accelerations and reaction forces consistent with the contact conditions, the system successively searches for supporting contacts having more degrees of freedom, i.e. it attempts to find a supporting line segment and, failing that, a single support point. If no set of contact points yield consistent solutions, the system will remove the hinge. In the simulation of Figure 7, the kinematic relationship between the small and large blocks changes twice. After sliding across the top of the large block and maintaining plane-plane contact, the kinematic relationship changes to plane-edge contact as the small blocks tips over the end of the block. Ultimately, it breaks contact altogether.

[2] Within some epsilon, of course. In this paper we avoid the crucial issue of numerical difficulties.

impact, contact, and friction are typically handled by current systems in an *ad hoc* or rudimentary manner, if at all. In some systems, for instance, any possible impacts must be specified in advance; in others, a kind of "force field" technique is used, in which between every pair of objects there is a repelling force that is negligible except when objects are very close together.

The development of *Newton* was also influenced by the recent work by graphics and animation researchers in what they term *physically-based modeling* [1,10]. The desire to create increasingly complex and realistic animations has made traditional keyframing techniques less successful and led to interest in modeling and simulating object dynamics. While the techniques currently used are less sophisticated than those used in mechanical engineering, the emphasis placed on control of high-degree-of-freedom mechanisms, such as human and animal models, makes the research interesting.

# 2 Newton Architecture

Using *Newton*, a designer can define complex physical objects and mechanisms and can represent object characteristics from a wide range of domains. An object is made up of a number of "models," each responsible for organization of object characteristics from a particular domain. In most simulations the basic domains of geometry, dynamics, and controlled behavior are modeled. A dynamic modeling system, for example, is responsible for maintaining an object's position, velocity, and acceleration, and for automatically formulating the object's dynamics equations of motion. A geometric modeling system is responsible for information about an object's shape, distinguished features on the object, and computation of geometric integral properties such as volume and moments of inertia. It must also detect and analyze object interpenetrations so that an interference modeling system can deal with collisions between objects.

With this kind of flexibility, mechanism design and analysis is made simpler; a number of simulations of a physical system might be carried out, with different sets of modeled properties being accounted for each time. New modeled aspects might be added to increase the overall accuracy of the simulation, or certain domains might be abstracted or ignored to allow the experimenter to focus on the contribution of other domains to the observed behavior.

*Newton* has three main components: the definition and representation module, the analysis module and the report system. The definition module is responsible for analysing high-level language descriptions of *Newton* entities, and for organizing information in the appropriate data structures. The analysis component implements the top-level control loop of simulations and coordinates the working of various analysis subsystems. The report system handles generation of graphical feedback to users during simulations as well as recording of relevant information for later regeneration of animations.

```
primitive link(thickness, height, rho)
  properties: (density: rho);
  geometry: cuboid (thickness, height, thickness) where
    begin
      topback: (0, height/2, -thickness/2),
      botback: (0, -height/2, -thickness/2),
      botoffmid: (0, -height/2 + .75*thickness ,0)
    end

primitive ball (radius, rho)
  properties: (density: rho);
  geometry: sphere (radius)

composite pendulum(thickness, linkheight, rho)
  components
    l0:          link(thickness,thickness,rho),
    l1,l2,l3:    link(thickness, linkheight, rho),
    ball:        ball(.75 * thickness, rho);
    j1,j2,j3,j4: ball.and.socket
  structure
    join l0 to l1 with j1 matching (botback topback),
    join l1 to l2 with j2 matching (botback topback),
    join l2 to l3 with j3 matching (botback topback);
    join l3 to ball with j4 matching (botoffmid center)
```

Figure 2: Pendulum Definition

## 2.1 Definition module

*Newton*'s definition language is used to describe a variety of simulation entities, including *objects*, *hinges*, *constraints*, *models*, *equations* and *quantities*. Objects are further divided into two subclasses: *primitives*, corresponding to single indivisible bodies, and *composites*, representing collections of objects related by constraints. The constrained relationships usually correspond to material hinges such as ball and socket or pin joints and are modeled using data structures called *hinges*. The components of composite objects can be either composites or primitives. Thus, description of complex mechanisms is made simpler by breaking the description down into natural part-component relationships. Figure 2 shows the definition of a simple five-object pendulum.

One advantage of the hierarchical object representation scheme is that it facilitates automatic, incremental reformulation of an object's motion equations. During simulation, *Newton*'s analysis module makes requests for an object's set of motion equations. The set is constructed recursively, by requesting the equation sets for each of the object's components and for each of its hinges. At every level of an object's composition hierarchy, the set of equations for that level, once derived, is stored in the appropriate dynamic model. Then, when events occur that dictate a change in the equations for a component of an object, only the equations for that component need be rederived. The other components' equation sets are still available from their dynamic models.

## 2.2 Report module

As stated above, the report system is responsible for generating output that can be of use to the experimenter in analysing simulations. This includes display of the simu-

# 4 Event handling

During the course of interesting simulations, a variety of events can occur that require special processing. Collisions need to be detected and resolved, constraints modeling contacts between objects need to be added and deleted, friction forces need to be monitored to determine when an object changes from sticking to sliding on a surface, and so on. In general, exceptional events can cause discontinuities in object velocities or accelerations and necessitate corresponding modification of the internal models of object behavior. It is crucial to the success of general-purpose simulation systems that they be able to deal with such events.

*Newton* has a general-purpose event handler that is currently responsible for coordinating collision detection and resolution, contact maintainance, and handling of events corresponding to changes in control program states. Since it deals with discontinuous changes in system behavior, the event handler is also responsible for things such as restarting parts of the integration subsystem.

For the purposes of the paper, we restrict the following discussion to impacts and contacts between polyhedral objects, though the *Newton* system is not restricted in this way. We then describe the various contacts as surface-surface, edge-surface, point-surface, edge-edge, and so on. Impacts are distinguished from other contacts as those contacts where the velocity of a contact point on one object relative to the corresponding point on the second object is directed into the second object's interior. For a contact between a point, $p_1$, of an object $O1$ and a corresponding point, $p_2$, on the surface of another object $O2$, the condition is stated more precisely as $(\dot{p}_1 - \dot{p}_2) \cdot n_{contact} < 0$, where with $n_{contact}$ is the normal to the contact surface (directed toward $O2$'s exterior). When the normal-direction relative velocity is greater than zero, the contact is in the process of breaking. When the velocity is zero, the contact will remain and may result in creation of a temporary hinge.

## 4.1 Impact

When the event handler begins its impact analysis, the integration module has just proposed a set of positions and velocities for time, $t + \Delta t$. *Newton* then uses its geometric modeling subsystem to determine whether any impacts occurred in the time interval. While there are many difficulties in properly computing the intersection between two geometric representations, the problem of determining the precise time of any impacts makes matters still more complex. To do things correctly the four-dimensional space-time swept volumes of two objects must be intersected. In the current implementation, however, we count on time steps being sufficiently small that we don't miss collisions between steps and, when it is determined that an impact does occur between times $t$ and $t + \Delta t$, the moment of impact is found by binary search of the time interval. We repeatedly halve the time increment, reintegrate, and analyze contacts for the new time, until any object interpenetrations are within a user-controllable tolerance.

After determining the time of any impacts, the geometry system is used to analyze the nature of all interobject contacts for that time. For the moment, assume that all such contacts are indeed impacts. To resolve impacts, *Newton* formulates impulse-momentum equations for each object, and contact-point velocity equations for each impact (using coefficients of restitution based on object properties), and then solves this equation system to compute instantaneous changes for the object velocities.

The equations are automatically derived in a fashion analogous to the formulation of motion equations described earlier. For point-on-surface impacts, the process involves formulating equations of the following form:

$$\begin{aligned} m_1 \Delta \dot{r}_1 &= \dot{f}_{impact} n_{contact} \\ J_1 \Delta \omega_1 &= c_1 \times (\dot{f}_{impact} n_{contact}) \\ m_2 \Delta \dot{r}_2 &= -\dot{f}_{impact} n_{contact} \\ J_2 \Delta \omega_2 &= c_2 \times -(\dot{f}_{impact} n_{contact}) \end{aligned}$$

$$\dot{\rho}_1^A - \dot{\rho}_2^A = -e(\dot{\rho}_1^B - \dot{\rho}_2^B),$$

where $c_i$ is the vector from object $i$'s center of mass to the location of the hinge, $\dot{f}_{hinge}$ is the (scalar) impact impulse, $n_{contact}$ is the surface normal at the point of impact, $\Delta r_i$ is the difference between object $i$'s center of mass velocity before and after impact, $\dot{\rho}_i^B$ and $\dot{\rho}_i^A$ are the velocities of the impact point on object $i$ before and after impact, respectively, and $e$ is a coefficient of restitution that depends on the material properties of the colliding objects.

When composite objects are involved, impulses due to impacts are transmitted through hinges by formulating impact constraint equations for the hinges and adding appropriate impulse terms to equations for the hinged objects.

Thus, if object 2, from above, and a third object are related by a spherical hinge, the hinge equation is

$$\dot{r}_2^A + \omega_2^A \times c_2 = \dot{r}_3^A + \omega_3^A \times c_3$$

and the object equations are

$$\begin{aligned} m_2 \Delta \dot{r}_2 &= -\dot{f}_{impact} n_{contact} + f_{hinge} \\ J_2 \Delta \omega_2 &= c_2 \times (\dot{f}_{impact} n_{contact}) + c_2 \times f_{hinge} \\ m_3 \Delta \dot{r}_3 &= -f_{hinge} \\ J_3 \Delta \omega_3 &= c_3 \times -f_{hinge}, \end{aligned}$$

where $f_{hinge}$ is a global coordinate system vector representing the impulse transmitted through the hinge.

Our current model of impact is extremely simple and clearly not satisfactory in some situations. When all of the contacts are impacts, use of this collision resolution scheme yields instantaneous velocity changes that do not imply any further impacts. That is, after the impact resolution procedure there are no longer any contacts that meet the conditions for being impacts. In many situations, however, only a subset of the set of contacts represent true impacts. In dealing with such situations, our model does not produce impulses for the non-impact contacts. Thus, the impact resolution scheme treats these contacts as if there were an infinitesimal separation between the objects. Resolution

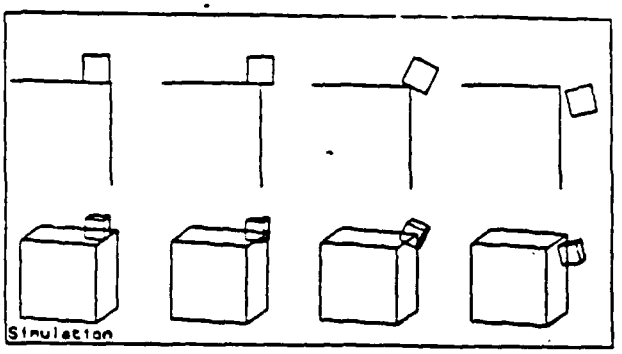


Figure 7: Changing Kinematic Relationships

For a single pair of objects in contact, determination of a consistent set of support points is simple, taking time (at worst) linear in the number of contact vertices. However, for the case of multiple objects and contacts, a naive algorithm postulates support sets for a contact independently of sets proposed for other contacts, resulting in an exponential search of the space of possible contact sets. This complexity can usually be avoided by using heuristics during the search. For example, since kinematic relationships don't usually change very often, the algorithm first attempts to use, for each contact, the same support points as for the previous timestep. Continuity considerations are also useful in most cases; the contact forces and their derivatives can be monitored to determine which (and when) contacts break. Methods for dealing with contact problems are examined in more detail in [4,8,5,3].

### 4.3 Control

*Newton*'s control subsystem permits user-directed influence of object motion through the definition of control force and torque quantities, control equations, and control programs that communicate with the dynamic analysis system through the control interface. To model an actuator in a hinge, for instance, the system associates a control torque quantity with the hinge, and, as part of the creation of this quantity, the system adds torque terms to motion equations of the two constrained objects.

In *Newton's* automatically-generated equations of motion certain quantities are considered to be *unknowns*. Typically, for what we call forward-dynamics control, the unknowns consist of accelerations and joint constraint forces, while positions, velocities and joint control torques are considered to be *knowns*. On, the other hand, inverse dynamic simulation is acheived by choosing the accelerations as knowns and solving for the control forces. Combination schemes are often used as well; given an object with $n$ degrees of freedom, with motion equations containing $q$ acceleration quantities and $f$ control force and torque quantities, a control algorithm can define and control any $n$ of the $q+f$ quantities, so long as they are independent. If fewer than $n$ quantities are controlled, the system of motion equations is underdetermined, and many different motions could satisfy the constraints of the control algorithm. In this case a control algorithm can guide the selection of a motion by providing a quadratic cost function in terms of the unknowns of the system; a solution is then chosen that minimizes this cost function.

*Newton* has been used to experiment with control of many-degree-of-freedom objects. The development of a high-level algorithm for control of a walking figure model is presented elsewhere in these proceedings[9].

## 5 Summary

Much work has been done in the past in the area of simulation of dynamics, much of it by mechanical engineering researchers. This paper has described the architecture of *Newton*, a dynamics simulator that is part of our ongoing project of applying computer science principles to the development of more powerful and flexible simulation systems. *Newton* currently supports automatic formulation and modification of object equations of motion, contains a general-purpose event-handling mechanism, allows high level description of simulated objects and scenarios, and supports experimentation with control of high-degree-of-freedom mechanisms. While the system presently handles impact, contact and friction problems using simple models, more sophisticated models can be incorporated into *Newton* without the need for major system revisions.

## Acknowledgements

This work was supported in part by NSF grant DMC 86-17355, ONR grant N0014-86K-0281 and DARPA grant N0014-85K-0591. Support for James Stewart is provided in part by U.S. Army Mathematical Sciences Institute grant U03-8300 and NASA training grant NGT-50327. The *Newton* system is being developed at Cornell and Purdue Universities using Common Lisp on Symbolics Lisp Machines.

## References

[1] A. H. Barr, J. F. Abel, R. Barzel, D. P. Greenberg, J. Platt, and C. Reynolds. Topics in physically-based modeling. ACM SIGGRAPH-88, Notes for Tutorial 27, August 1988.

[2] M. Chace. Modeling of dynamic mechanical systems. Presented at the CAD/CAM Robotics and Automation Institute and International Conference, Tucson, Arizona, February 1985.

[3] J. F. Cremer. PhD thesis, Cornell University, May 1989.

[4] R. Featherstone. *Robot Dynamics Algorithms*. Kluwer, 1987.

[5] B. Gilmore. *The Simulation of Mechanical Systems with a Changing Topology*. PhD thesis, Purdue University, August 1986. Mechanical Engineering.

[6] E. J. Haug and G. M. Lance. Developments in dynamic system simulation and design optimization in the center for computer aided design: 1980-1986. technical report 87-2, University of Iowa, February 1987.

[7] C. M. Hoffmann and J. E. Hopcroft. Simulation of physical systems from geometric models. *IEEE Journal of Robotics and Automation*, RA-3(3):194–206, June 1987.

[8] P. Lötstedt. Numerical simulation of time-dependent contact and friction problems in rigid body mechanics. *SIAM Journal of Scientific and Statistical Computing*, 5(2):370–393, 1984.

[9] A. J. Stewart and J. F. Cremer. Algorithmic control of walking. 1989 IEEE International Conference on Robotics and Automation.

[10] A. Witkin and M. Kass. Spacetime constraints. In *Computer Graphics (SIGGRAPH 88)*, pages 159–168. ACM, August 1988.