Beyond Keyframing: An Algorithmic Approach to Animation

A. James Stewart
James F. Cremer

TR 91-1207
May 1991

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

Beyond Keyframing: An Algorithmic Approach to Animation

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

Originally written January 1989

Abstract

The recent explosion of interest in physical system simulation may soon lead to realistic animation of passive objects, such as sliding blocks or bouncing balls. However, complex *active* objects (like human figures and insects) need a control mechanism to direct their movements. We present a paradigm that combines the advantages of physical simulation and algorithmic specification of movement. The animator writes an *algorithm* to control the object and runs this algorithm on a physical simulator to produce the animation. Algorithms can be reused or combined to produce complex sequences of movements, eliminating the need for tedious keyframing. We have applied this paradigm to control a walking biped. The walking algorithm is presented along with the results from testing with the *Newton* simulation system.

1 Introduction

This paper describes a new paradigm for the control and animation of complex active objects such as the human figure. This approach allows the animator to control an object through an algorithm which specifies certain "intuitive" variables as a function of time and of world state. In the case of human figure walking, the animator might write an algorithm which controls the acceleration of the figure's center of mass at one point in the animation, and which controls the angle of the knees at another point. The algorithmic approach to animation allows this to be done with ease, as demonstrated by the walking algorithm presented in Section 6.

Witkin and Kass [WK88] have combined physical simulation and key-framing to produce realistic animation of their jumping Luxo lamp. With their approach the animator uses *spacetime constraints* to specify several key points for selected variables. These variables may be positions, velocities, forces and so on. Combining spacetime constraint equations with the Lagrangian equations of motion and discretizing over time yields a system of equations that are solved to produce the motion. Since the system is generally underconstrained (having multiple solutions) a solution can be chosen to minimize the power, fuel comsumption and so on.

Our algorithmic approach is similar in that the animator can control accelerations and forces, but differs in that the constraints can be added or removed "on the fly" as the algorithm sees changes in the world state which might not be predictable. In the case of human figure walking the algorithm might, as the foot touches the ground, remove a foot positioning constraint and add a leg stiffening constraint. The exact point of contact is not predictable in advance. Additionally, the algorithmic approach frees the animator from considering the dynamics of impact and other changes in kinematic relationships, which are handled automatically by the simulation component of our system. Incorporating impact into the work of Witkin and Kass would require either guessing the impact points beforehand or incorporating a "force field" approach as described in Section 2.

Other work on combining control and simulation has been done by Barzel and Barr [BB88]. Their method of *dynamic constraints* adds fictitious forces which pull the simulated objects into specified positions. By doing this in the framework of a simulation system, the movement of complex physical objects can be simulated with little work on the part of the animator. A limited form of control is achieved by attaching forces to points on the object and dragging these points.

Various other approaches to combine control and physical simulation have been explored. Wilhelms [Wil87] blends kinematic and dynamic formulations, Isaacs and Cohen [IC87] incorporate inverse dynamics in their simulation system, and Brotman and Netravali [BN88] use dynamics and optimal control to interpolate between key frames.

Some further insights on control can be gained from examining the current literature in the field of robotics. While this field deals with controlling real, physical objects, some of the techniques can be applied to produce simpler animation.

Researchers in robotics have taken various approaches to reduce the complexity of control programs for physical objects. The computed torque method (see [Cra86]) for robot arms can be viewed as simplifying control by reducing the gripper to a unit mass. The control program can ignore the dynamics of the robot arm, only concerning itself with the position of the

end effector as a function of time.

In building his one-legged hopping machine, Raibert [Rai86] partitioned control along three intuitive degrees of freedom: hopping, forward speed and body posture. This resulted in surprisingly simple control programs for the hopping robot. For multi-legged machines, Raibert introduced the idea of a "virtual leg" which was defined in terms of the robot's physical legs. This again led to simplified control programs.

Both the computed torque method and Raibert's virtual leg demonstrate that a proper choice of control variables can lead to simplified control programs. The problem with this approach is that there is often no simple closed-form mapping of these control variables onto the forces and torques needed to control the object. In some cases a complete system of equations must be numerically solved to make this mapping. This is called "inverse dynamics" and is typically rejected by robotics researchers as being too expensive to use in real-time control. For the purposes of animation, however, it is ideal.

This is the basis of our algorithmic approach to control. This approach advocates the selection of a small set of intuitive variables which are used by the algorithm in controlling the object. The algorithm constrains these variable with *constraint equations*, which, when combined with the standard Newton-Euler equations of motion, produce a system of equations describing the motion of the simulated object. The system of equations is maintained by our general purpose physical simulator, called *Newton*. The *Newton* simulator is responsible for integrating the motion of the simulated objects over time to produce the animation. As described in the next section, *Newton* also automatically updates the system of equations as kinematic relationships in the simulation change (one such change would occur as the biped's foot touches the ground). Finally, *Newton* provides an interface to allow the algorithm to add and remove constraint equations to and from the system of motion equations.

In the event that the control algorithm underconstrains the motion of the object, constrained optimization techniques are used to choose a motion that optimizes some criterion while satisfying the constraints imposed by the algorithm. Our decision to allow control programs to underconstrain the controlled object – necessitating the use of constrained optimization techniques – is based on the realization that control algorithms often require many fewer control variables than there are degrees of freedom in the controlled object. A robot modeled after the human figure may have as many as two hundred degrees of freedom [Zel82], while the control program for such a robot would only require twenty or thirty degrees of freedom to accomplish its task. In programming our walking biped we used at most eleven of its sixteen degrees of freedom at any given instant.

In summary, the algorithmic approach presented in this paper allows the algorithm to constrain a small set of intuitive variables. The algorithm is allowed to underconstrain the motion of the object, in which case a motion is chosen which optimizes some criterion while obeying the constraints. The *Newton* simulator incorporates the constraint equations into its automatically maintained system of motion equations and integrates over time to produce realistic animation.

Section 2 outlines the relevant background of the *Newton* simulation system. Section 3 describes in detail the algorithmic approach, while Section 4 looks at some low-level controllers used by the walking algorithm. Following this, Sections 5 and 6 outline the biped model and the walking algorithm, and present results from testing the algorithm.

2 Overview of Newton

The walking algorithm described in this paper has been designed and tested using the *Newton* simulation system, part of a large research effort in modeling and simulation at Cornell University. The development of *Newton* was inspired by the need for more general-purpose, flexible simulation systems.

Extensive mechanical engineering research has led to many developments in physical system simulation. The ADAMS [Cha85] and DADS [HL87] 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, and so on. Recent work by graphics and animation researchers [BB88,IC87,MW88,Hah88] in what is termed *physically-based modeling* has generally been less sophisticated but has placed greater emphasis on animation of interesting high-degree-of-freedom mechanisms.

A number of things are still lacking in all of these systems. Typically they 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, impact, contact, and friction are typically handled by current systems in an *ad hoc* or rudimentary manner, if at all. In some cases, 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. In addition, the desire to manipulate high-degree-of-freedom objects suggests that a module for specification of control algorithms should be a significant part of a dynamics system.

2.1 Newton Architecture

Using *Newton*, a designer can define complex three-dimensional 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 also detects and analyzes object interpenetrations so that an interference modeling system can deal with collisions between objects.

Newton is composed of three main components: the definition and representation module, the analysis module and the report system. The definition module analyzes high level language descriptions of *Newton* entities and organizes the corresponding 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.

2.2 Dynamic Analysis in Newton

A complex physical object is modeled as a collection of rigid bodies related by constraints. Newton-Euler equations of motion are associated with each individual rigid body.[1] At the time an object is created the equations are of the form

$$m\ddot{r} = 0$$

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

where m is the mass, $\ddot{r}$ is the second time derivative of the position (ie. the acceleration), J is the 3×3 inertia matrix, and ω and $\dot{\omega}$ are the rotational velocity and acceleration, respectively.

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. For a holonomic constraint such as this one, the second derivative of the constraint equation can be used along with the modified motion equations

[1] *Newton* is capable of using dynamics formulations other than the one outlined here. We are also working on incorporating non-rigid bodies into the system.

to solve for object accelerations and reaction forces. Thus, the equations above become

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

$$\ddot{r}_1 + \dot{\omega}_1 \times c_1 + \omega_1 \times (\omega_1 \times c_1) = \ddot{r}_2 + \dot{\omega}_2 \times c_2 + \omega_2 \times (\omega_2 \times c_2),$$

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. Note that the last equation above is the second time derivative of the holonomic constraint equation $r_1 + c_1 = r_2 + c_2$ for spherical joints. Other kinds of hinges commonly used in *Newton* include revolute or pin joints, prismatic joints, springs and dampers, and rolling contacts.

If gravity is present during the simulation the system will automatically add gravitational force terms 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 a large set of equations, the matrices are very sparse and often symmetric. Thus, acceptable efficiency is achieved by the use of sparse matrix solution techniques.

2.3 Event handling, impact and contact

Newton, unlike many other simulation systems (though see [Fea85]), can automatically and incrementally reformulate the motion equations as exceptional events occur during simulations. One kind of exceptional event is a change in kinematic relationship between objects. Figure 1 shows a block that was initially sliding along a table top. After some time the edge of the table is reached and the contact relationship changes from a plane-plane contact to a plane-edge contact. Still later the contact is broken altogether. These changing contact relationships are automatically detected by *Newton*. The system of motion equations and the related constraint equations are automatically maintained by *Newton* to reflect these changing relationships.

During the course of a simulation, a variety of events can occur that require special processing. *Newton*'s event handler is primarily responsible for detection and resolution of impacts, for analysis of continuous contacts

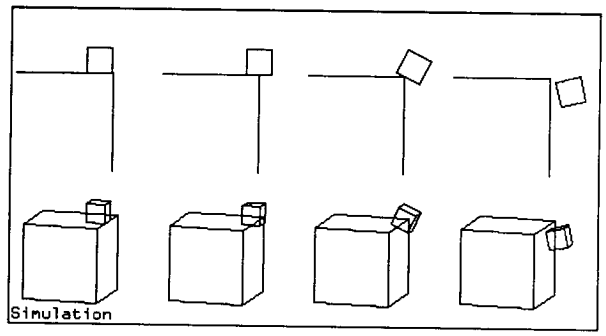

Figure 1: Changing Kinematic Relationships

between objects and corresponding maintenance of *temporary hinges*, special kinds of hinges that model one sided constraints between objects in contact, and for handling of events specified by control programs that necessitate changes in the constraint set. For example, the walking algorithm might tell the event handler to notify it when the biped's foot touches the ground so that it can change the constraint equations.

The geometric modeling subsystem is responsible for detecting and analyzing impacts and interpenetrations. In the usual method of handling impacts, the dynamic analysis module formulates impulse-momentum equations in a manner completely analagous to the formulation of the basic dynamics equations, and solves these equations to produce the instantaneous velocity changes caused by the impact. The details of *Newton*'s methods for handling impact, contact and other exceptional events are given in [HH87,HH88,CS88,Cre89].

3 The Algorithmic Approach

In *Newton's* automatically-generated equations of motion certain quantities are considered to be *unknowns*. A system of simultaneous linear equations is solved at each time step to produce values for the unknowns. These values are integrated over time to produce the simulated motion. Typically, the unknowns consist of accelerations and joint constraint forces, while positions, velocities and joint control torques are *knowns*.

In the algorithmic approach, the programmer controls "intuitive" quantities defined as linear combinations of the unknowns. The programmer might, for example, want to control the acceleration of the center of mass of a biped without explicitly controlling each component of the biped. To do this, the algorithm must define the acceleration of the center of mass in terms of the accelerations of the centers of mass of the primitive components of the ob-

```
procedure initialize

  begin
  add-equation " r̈_cm = (1/M) Σ m_i r̈_i "
  end

procedure controller( time )

  begin
  r̈_cm = f( time )
  end
```

Figure 2: The Format of an Algorithm

ject. Over the course of execution, the algorithm must supply the desired acceleration of the center of mass at each point in time.

Figure 2 shows the format of a control algorithm. For the sake of clarity the algorithms will be described in a Pascal-like notation[2]. Two procedures are always present: one to initialize the algorithm (called `initialize`) and one to be executed repeatedly over the course of the task (called `controller`). The `controller` procedure has access to the complete state of the system. The algorithm of Figure 2 trivially defines and controls the acceleration of the center of mass of an object (the function f must be defined elsewhere).

Defining and controlling a three-dimensional vectorial quantity like the acceleration of the center of mass has the effect of adding three constraint equations to the system of simultaneous linear equations that describe the instantaneous motion of the object. By considering joint torques as unknowns in this augmented system of equations, the system can be solved to produce motion that satisfies the additional constraint equations. This is a simple application of inverse dynamics.

For an object with n degrees of freedom the control algorithm can define and control up to n independent scalar quantities[3]. If fewer than n equations are added the system of motion equations is underdetermined, and many different solutions could satisfy the constraints of the control algorithm. In this case the algorithm must guide the selection of a solution by providing a cost function which is quadratic in the unknowns. A standard numerical optimization technique is used to compute a solution that instantaneously (for each point in time) minimizes the cost function while obeying the algorithm's constraints. This is different from the approach of Witkin and Kass

[2]The algorithms are, for now, written in Lisp.

[3]The additional definitional equations could make the system of motion equations inconsistent. This would be an error on the part of the control algorithm.

[WK88], who optimize over the whole animation. This reflects the different philosophies of the two systems: Witkin and Kass specify all of the information beforehand, while we let the control algorithm make decisions *during* the animation. Such "on the fly" decisions make it impossible to do global optimization, but allow much more versatility in the control algorithm by not requiring *a priori* knowledge of impacts and other exceptional events.

In summary, the programmer designs an algorithm in a high-level computer language to control intuitive degrees of freedom of the object. These degrees of freedom are defined as linear combinations of the unknowns in the object's equations of motion. An augmented linear system of equations describes the instantaneous behavior of the object; this system can be solved to produce the object's configuration at each point in time. If the system is underdetermined, the algorithm can provide a cost function to guide the choice of a solution.

In the remaining sections we describe the application of this approach to the design of a simple walking algorithm.

4 Low-level Controllers

In designing algorithms with *Newton* we found ourselves frequently using PD controllers[4] and curve-fitting controllers to control the "trajectory" of many of the defined quantities. In controlling the biped, for example, a quintic interpolation was used to plot the trajectory of the heel, and a PD controller was used to orient the foot before it struck the ground. A small library of these controllers is used in the biped algorithm, and will be described here.

PD controllers are used in the biped algorithm to control orientation, position and joint angle. Each controller adds an equation to the system of motion equations which defines the second derivative of the quantity in terms of the first derivative and the quantity itself. The procedure in Figure 3 produces accelerations to move an `object` to within 1% of a position `x-desired` within a given time `delta-time`. The quantities x, v and a are data structures representing state variables of the controlled object. These data structures are used by the `add-named-equation` function to create the appropriate equation.

Execution of the procedure in Figure 3 causes a named equation to be

[4] A PD controller (Proportional, Derivative), also known as a "spring and damper" controller, relates the second derivative of a variable linearly to the error in the variable's first derivative and to the error in the variable itself. The equation is $\ddot{x} + \frac{2}{\tau}\dot{x} + \frac{1}{\tau^2}(x - x_{\text{desired}}) = 0$ for some appropriate τ. PD controllers are used extensively in robotics to move robot joints into specified positions by calculating the joint acceleration as a function of the position and velocity errors. A good explanation can by found in [Cra86]. Barzel and Barr [BB88] use a form of PD controller to achieve their dynamic constraints.

```
procedure position-with-PD( constraint-name, object,
                            x-desired, delta-time )

  var x, v, a:  quantity
      τ:  real

  begin
  x = get-position-quantity( object )
  v = get-velocity-quantity( object )
  a = get-acceleration-quantity( object )

  τ = - delta-time / log( .01 )

  add-named-equation( constraint-name,
                      " a + (2/τ) v + (1/τ²)(x - x-desired) = 0 " )
  end
```

Figure 3: PD Controller Used in Positioning

added to the system of motion equations. This equation will continue to affect the motion of the object until it is explicitly removed by the control algorithm.

A complete list of controllers available to the biped walking algorithm is shown in Figure 7 at the end of the paper. Those with `quintic` in their name do quintic interpolation to achieve the desired position and velocity in the desired time. Quintic interpolation was chosen over cubic interpolation to eliminate "jerk" (discontinuous acceleration) from the beginning and end of the trajectory.

5 The Biped Model

The simulated biped is composed of a torso, two legs with knee joints and two feet with toe joints. This model was adapted from a description in [McM84] and is shown in Figure 4. The hips and ankles are three degree of freedom spherical joints, while the knees and toes are one degree of freedom revolute joints, making a total of sixteen degrees of freedom. The biped is about six feet tall with moments approximating those of a human being.

We hope to improve this model by incorporating joint limits and elastic tendons. McMahon suggests that, during walking, energy is stored in stretched tendons and is released when the stretched leg swings forward [McM84]. This idea might be used to simplify the walking algorithm described in the next section.

Newton's impact handling capabilities have not yet been extended to

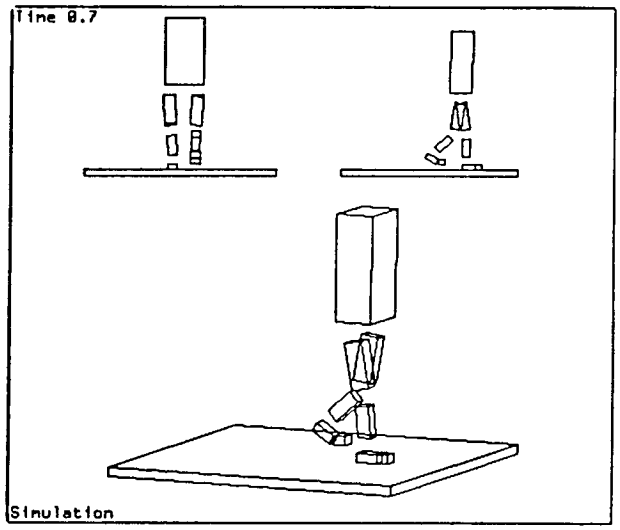

Figure 4: Simulated Biped Model

accurately model the impact of the feet upon the ground. Instead, impact is simulated by adding an external force and torque to the feet that holds them level with the ground until they are released with an explicit command from the control algorithm. This is as though the biped was walking with magnetic shoes on a steel plate. Very shortly we expect to adapt the algorithm to incorporate realistic impact.

6 The Walking Algorithm

An abbreviated version of the walking algorithm is shown in Figures 8 and 9, which can be found at the end of this paper. The algorithm cycles through a set of six states: swing the right leg, land the right foot, lift the left foot, swing the left leg, land the left foot, lift the right foot and then repeat the cycle. In the *swing* phase, a quintic trajectory is plotted for the swing foot with `move-heel-to-target`, while the stance leg is stiffened with `set-angle-with-PD` and the foot is oriented for landing with `orient-with-PD` (shown under **START** in Figure 9). In the *landing* phase, the leading leg is stiffened as the foot nears the ground. Following this, the *takeoff* phase flexes the trailing leg, causing the trailing foot to lift from the ground. Once the trailing toe is bent to 10° the flexing constraint is removed and the *swing* phase begins for the trailing leg.

The largest number of constraints are applied during the *swing* phase, as shown in Table 1. Since the biped has sixteen degrees of freedom (DOF) it remains underconstrained at all times. A quadratic cost function is therefore defined (in `initialize` of Figure 9) in order to fully determines the motion

Constraint Name	DOF	Constrained Item
TORSO-CONSTRAINT	3	torso orientation in 3 dim
L-KNEE-ANGLE	1	angle of revolute knee joint
R-HEEL-TRAJ	3	heel acceleration in 3 dim
R-FOOT-ORIENTATION	3	foot orientation in 3 dim
R-TOE-ANGLE	1	angle of revolute toe joint

Table 1: Swing Phase Constraints

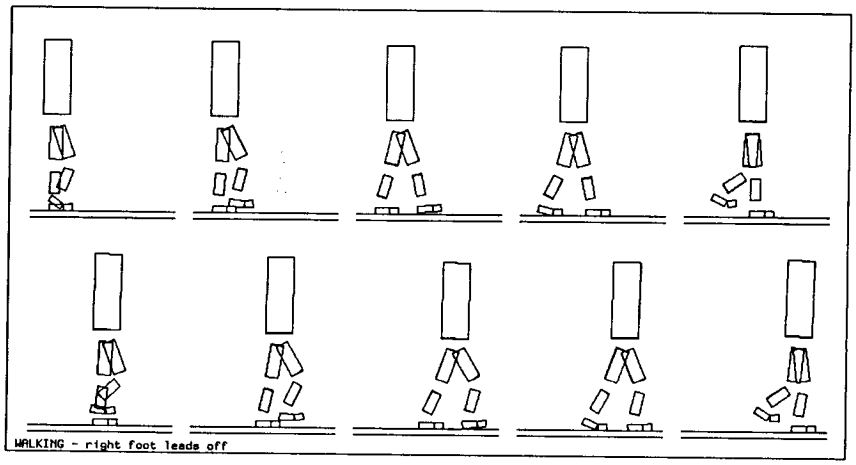

Figure 5: Walking Cycle

of the biped. The cost function is a weighted sum of the translational and angular accelerations, and of the difference between the torso translational acceleration and some acceleration defined by a function **F** which tries to keep the torso mid-way between the two feet.

We found that a cost function which minimizes instantaneous translational and rotational acceleration usually produces smooth motion. In the case of the simulated biped, the cost function causes the constrained heel acceleration to be achieved by a linear combination of small accelerations of many components of the body, rather than a few large accelerations of those components which are near the heel. We have observed that the combination of many small accelerations yields more stable motion than large, local accelerations.

The walking algorithm was tested with the *Newton* simulation system. Figure 5 shows ten frames in which the biped completes a full cycle of the six phases described above. The full simulation consisted of twenty seconds of straight-line walking on a flat surface and generated the statistics shown in Figure 6. The version of the algorithm that produced these statistics had the biped increase speed at 4.0 seconds, as can be seen on the graphs.

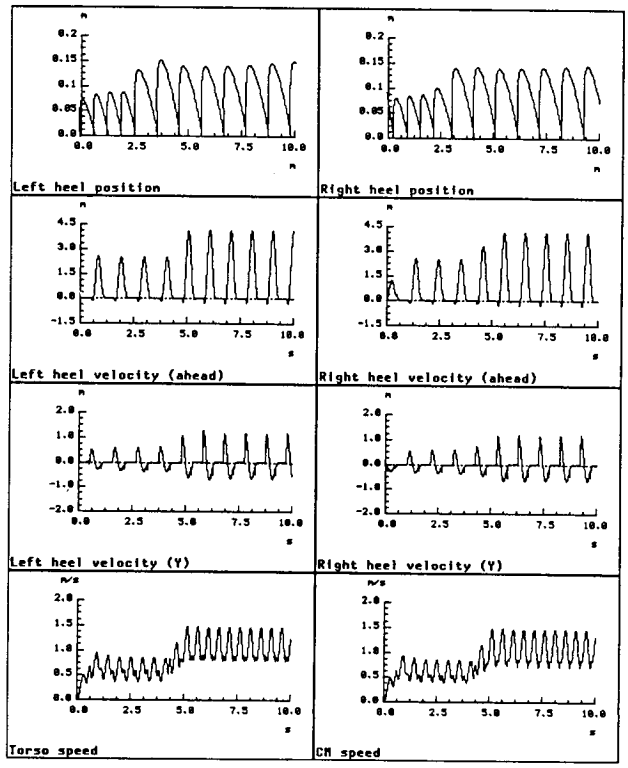

Figure 6: *Newton* Statistical Output

Due to the simplicity of our current biped model, this algorithm is forced to use too many constraints to achieve the desired motion. In particular, the trajectory of the heel must be specified. However, if the biped model were extended to include elastic tendons the number of constraints might be reduced. In this case, the swing phase would not have to specify a trajectory for the heel. Instead, no torque would be applied in the swing leg; it would be pulled forward by the stored energy of the stretched tendons. This might approximate the "ballistic walking" described by McMahon[McM84].

We feel that a high-level algorithm should *greatly underdetermine* the motion of the controlled object. Our philosophy is to incorporate in the model many "passive elements" – such as springs, dampers and joint limits – which reduce the number of constraints needed by the control algorithm. The algorithm then has the job of guiding, rather than forcing, the motion of the object.

7 Summary

We have presented an algorithmic approach to control. This approach allows the animator to choose intuitive degrees of freedom by which to control an object. The control algorithm adds and removes constraint equations "on the fly" as the world state changes; *a priori* knowledge of the exact moment of each state change is not required. With the algorithmic approach, all consideration of dynamics and impact is left to the *Newton* simulation system. The burden on the animator is further reduced by allowing underdetermined specification of motion through the use of constrained optimization techniques.

We have presented an algorithm to control a simulated biped, along with results from its execution on the *Newton* simulation system. The algorithm has the advantage of being intuitive, simple to program, and reusable.

Unlike keyframing, the algorithmic approach does not require the animator to repeat the work of creating new key frames for every walking sequence. Unlike keyframing, the algorithmic approach allows various algorithms to be combined to produce long animated sequences. We believe that in the future, animating complex physical objects will require a structured, algorithmic approach similar to that presented in this paper.

8 Future Work

We will incorporate elastic tendons and joint friction into the *Newton* simulation system and modify the walking algorithm accordingly. From there we hope to develop a suite of algorithms to allow a biped to walk, turn, climb stairs, manipulate objects, and so on. In keeping with the structured approach presented in this paper we will attempt to combine these algorithms to have the biped perform complicated tasks. In carrying an object up a flight of stairs the high-level algorithm would combine subroutines to pick up the object, walk to the stairs, climb the stairs and deposit the object.

Acknowledgements

This work was supported in part by NSF grant DMC 86-17355, ONR grant N0014-86K-0281 and DARPA grant N0014-88K-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 in Common Lisp on Symbolics Lisp Machines and can be used on other machines supporting Common Lisp.

References

[BB88] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constraints. In *Computer Graphics (SIGGRAPH 88)*, pages 179–188. ACM, August 1988.

[BN88] Lynne S. Brotman and Arun N. Netravali. Motion interpolation by optimal control. In *Computer Graphics (SIGGRAPH 88)*, pages 309–315. ACM, August 1988.

[Cha85] M. Chace. Modeling of dynamic mechanical systems. Presented at the CAD/CAM Robotics and Automation Institute and International Conference, Tuscon, Arizona, February 1985.

[Cra86] John J. Craig. *Introduction to Robotics: Mechanics and Control.* Addison Wesley, 1986.

[Cre89] James F. Cremer. PhD thesis, Cornell University, in preparation, 1989.

[CS88] James F. Cremer and A. James Stewart. Using the *newton* simulation system as a testbed for control. In *Proceedings of the 3rd IEEE International Symposium on Intelligent Control*, 1988.

[Fea85] Roy Featherstone. The dynamics of rigid body systems with multiple concurrent contacts. In O. D. Faugeras and G. Giralt, editors, *Robotics Research: The Third International Symposium*, pages 191–196. The MIT Press, 1985.

[Hah88] James K. Hahn. Realistic animation of rigid bodies. In *Computer Graphics (SIGGRAPH 88)*, pages 299–308. ACM, August 1988.

[HH87] 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.

[HH88] C. M. Hoffmann and J. E. Hopcroft. Model generation and modification for dynamic systems from geometric data. Presented at the NATO Workshop on CAD-based Programming for Sensor-based Robots, Il Ciocco, Italy, July 1988.

[HL87] 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.

[IC87] Paul M. Isaacs and Michael F. Cohen. Controlling dynamic simulation with kinematic constraints, behavior constraints and inverse dynamics. In *Computer Graphics (SIGGRAPH 87)*, pages 215–224. ACM, July 1987.

[McM84] T. A. McMahon. Mechanics of locomotion. *The International Journal of Robotics Research*, 3(2):4–28, 1984.

[MW88] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. In *Computer Graphics (SIGGRAPH 88)*, pages 289–298. ACM, August 1988.

[Rai86] M. H. Raibert. *Legged Robots That Balance.* The MIT Press, 1986.

[Wil87] J. Wilhelms. Using dynamic analysis for realistic animation of articulated figures. *IEEE Computer Graphics and Applications*, 7(6):12–27, 1987.

[WK88] Andrew Witkin and Michael Kass. Spacetime constraints. In *Computer Graphics (SIGGRAPH 88)*, pages 159–168. ACM, August 1988.

[Zel82] D. Zeltzer. Motion control techniques for figure animation. *IEEE Computer Graphics and Applications*, 2(9):53–59, 1982.

```
position-with-PD( constraint-name, object, x_d, Δt )
position-point-with-PD( constraint-name, object, point-on-object, x_d, Δt )
orient-with-PD( constraint-name, object, Φ_d, Δt )
set-angle-with-PD( constraint-name, joint, θ_d, Δt )

position-with-quintic( constraint-name, object, x_d, v_d, Δt )
position-point-with-quintic( constraint-name, object, point-on-object, x_d, v_d, Δt )
orient-with-quintic( constraint-name, object, Φ_d, Φ̇_d, Δt )
set-angle-with-quintic( constraint-name, joint, θ_d, θ̇_d, Δt )
```

Figure 7: Low-level Controllers

```
const   time-in-air              = 0.5 s
        stride                   = 0.5 m
        direction                = (1 0 0)
        inside-step-fraction     = 20 %
        heel-Y-strike-speed      = −0.05 m/s
        heel-X-strike-speed      = 0.02 m/s
        foot-strike-orientation  = 10° about (0 0 1)
        torso-orientation        = −10° about (0 0 1)

var     phase:  {start r-swing r-land l-lift l-takeoff l-swing l-land r-lift r-takeoff }

procedure move-heel-to-target( constraint-name, foot, other-foot, hip, other-hip )

  var target-x, target-v, hip-to-hip:  vector

  begin
  hip-to-hip = get-position( TORSO, hip ) - get-position( TORSO, other-hip )

  target-x = get-position( other-foot, HEEL ) + stride × direction
             + inside-step-fraction × hip-to-hip

  target-v = heel-Y-strike-speed × (0 1 0) + heel-X-strike-speed × direction

  position-point-with-quintic( constraint-name, foot, HEEL, target-x, target-v, time-in-air )
  end
```

Figure 8: Definitions for the Walking Algorithm

```
procedure initialize

  let F = K_p((1/2)(r_{l-foot} + r_{r-foot}) - r_{torso}) + K_v((1/2)(ṙ_{l-foot} + ṙ_{r-foot}) - ṙ_{torso})

  begin
  quadratic-cost = Σω̇² + Σr̈² + 20 (r̈_{torso} - F)²
  phase = START
  end

procedure controller( time )

  begin
  case phase of

    START:
      phase = R-SWING
      orient-with-PD( TORSO-CONSTRAINT, TORSO, torso-orientation, 2.0 s )
      move-heel-to-target( R-HEEL-TRAJ, R-HEEL, L-HEEL, R-HIP, L-HIP )
      set-angle-with-PD( L-KNEE-ANGLE, L-KNEE, 175°, 0.1 s )
      orient-with-PD( R-FOOT-ORIENTATION, R-FOOT, foot-strike-orientation, time-in-air )
      set-angle-with-PD( R-TOE-ANGLE, R-TOE-JOINT, 0°, time-in-air )

    R-SWING:
      if distance-to-target( R-FOOT ) < 0.01 m then

        phase = R-LANDING
        remove-constraint( R-HEEL-TRAJ )
        set-angle-with-PD( R-KNEE-ANGLE, R-KNEE, 175°, 0.05 s )

    R-LANDING:
      if heel-has-touched( R-FOOT ) then

        phase = L-TAKEOFF
        remove-constraints( R-FOOT-ORIENTATION, R-TOE-ANGLE, L-KNEE-ANGLE )
        set-angle-with-PD( L-KNEE-ANGLE, L-KNEE, 160°, 0.1 s )

    L-TAKEOFF:
      if joint-angle( L-TOE-JOINT ) > 10° then

        phase = L-SWING
        remove-constraint( L-KNEE-ANGLE )
        move-heel-to-target( L-HEEL-TRAJ, L-HEEL, R-HEEL, L-HIP, R-HIP )
        orient-with-PD( L-FOOT-ORIENTATION, L-FOOT, foot-strike-orientation, time-in-air )
        set-angle-with-PD( L-TOE-ANGLE, bL-TOE-JOINT, 180°, time-in-air )

    Cases L-SWING, L-LANDING, and R-TAKEOFF
    are analogous to the preceding three cases.

    end
  end
```

Figure 9: Abbreviated Walking Algorithm