Beyond Keyframing: An Algorithmic Approach to Animation

A. James Stewart
Department of Computer Science
University of Toronto

James F. Cremer
Computer Science Department
Cornell University

Abstract

The use of physical system simulation has led 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 both 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 keyframing. We have applied this paradigm to control a biped which can walk and can climb stairs. The walking algorithm is presented along with the results from testing with the *Newton* simulation system.

CR categories: I.3.7 [Computer Graphics]: Three dimensional graphics and realism – animation; I.6.3 [Simulation and Modeling]: Applications

Keywords: physical simulation, human figure animation, animation control, constraints, dynamics

Introduction

This paper describes a paradigm for the control and animation of complex active objects such as the human figure. In this approach the animator develops an *algorithm* which controls the object by specifying certain intuitive variables as a function of time and of world state. The algorithm is able to continuously monitor the world state as it is being automatically updated by an underlying dynamics simulation system, and the algorithm is able to react when it sees changes in the world state.

For example, in the case of biped walking, the animator might write an algorithm that controls the angle of the knees at one point in the animation, and that controls the trajectory of the foot at another point. The algorithm might monitor the world state and, when it notes an event such as a foot touching the ground, stop controlling the trajectory of the heel and start controlling the angle of the knee.

Most animators would probably be comfortable with the idea of "programming" a human figure to walk. The algorithmic approach to animation allows this to be done with ease. This is demonstrated by the walking algorithm presented below.

Other Dynamics Work in Graphics

In some of the first work in this area, **Armstrong and Green** [1] present the equations of motion for tree-structured linkages of rigid bodies and discuss an efficient method of solving them.

Witkin and Kass [25] have combined physical simulation and keyframing 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 at specific times. 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. Our algorithmic approach 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.

Herr and Wyvill [13] describe a dynamics simulation system which allows easy user control through a simulation language and several high level control primitives. Our work is similar in that the user can define and control arbitrary variables, but we concentrate more on developing algorithms to control complex objects in an intuitive manner.

van de Panne, Fiume, and Vranesic [23] build state-space controllers to provide control torques that achieve desired goal states from arbitrary initial states. Such controllers can be concatenated to produce movement, including cyclic movement like walking.

Other approaches to combine control and physical simulation have been explored: **Wilhelms** [24] and **Barzel and Barr** [3] blend kinematic and dynamic analysis, **Moore and Wilhelms** [20] and **Baraff** [2] discuss the collision and contact problems, **Isaacs and Cohen** [16] incorporate inverse dynamics in their simulation system, and **Brotman and Netravali** [5] use dynamics and optimal control to interpolate between key frames.

Other Work in Walking and Control

The algorithmic approach is meant as a general method by which to control complex mechanisms. In this paper we use the walking problem as an example of an application of the algorithmic approach. Some other approaches to walking are briefly described here.

Kearney, Hansen, and Cremer [17], in an approach very similar to ours, examine the control of mechanical systems as a constraint programming problem. **Bruderlin and Calvert** [6] have developed an effective goal directed approach to dynamic walking in which the animator specifies a few high-level walking parameters. **McKenna and Zeltzer** [18] develop a gait controller and low-level motor programs to generate legged motion. **Zeltzer** [26] analyzes various approaches to the control of complex animated objects and considers their integration. **Raibert and Hodgins** [22] describe control systems for several legged creatures. **Brooks** [4] produces complex walking behavior in a physical, insect-like robot from a distributed network of low-level finite state machines.

Other Work in Robotics

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

Researchers in robotics have taken various approaches to reduce the complexity of control programs for physical objects. The computed torque method for robot arms (see **Craig** [8]) 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** [21] 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. Our application of inverse dynamics will be described in the next section.

The Algorithmic Approach

In the algorithmic approach, the animator's algorithm selects a small set of intuitive variables with which to control the object over the course of the simulation. The algorithm can control predefined variables, such as the forces and torques at the joints, or the instantaneous translational and rotational accelerations of the various components of the object. The algorithm can *also* control variables that it defines as linear combinations of these predefined variables.

For example, the algorithm could, with the appropriate subroutine call to the underlying simulation system, define the acceleration of the center of mass of an object as

$$a_{cm} = \frac{1}{M} \sum_i m_i \cdot a_i,$$

where m_i is the mass of the i^{th} component of the object (a constant), and a_i is the translational acceleration of the i^{th} component. Then at each time step of the simulation, the algorithm could supply a value for a_{cm}.

The underlying simulation system, called *Newton*, is a general purpose physical simulator. Given a description of a complex object (in, say, a computer file), *Newton* will automatically generate the corresponding system of Newton-Euler equations of motion which describe the instantaneous behavior of the object. *Newton* can then integrate these equations of motion over time to produce the animation. *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).

The animator's algorithm interacts with Newton in the following ways:

- The algorithm can add arbitrary equations and variables to *Newton's* system of motion equations. In the example above, the algorithm added a variable, a_{cm}, and a equation defining that variable in terms of other variables of the system. The algorithm can remove equations that it previously added to the system of motion equations.
- The algorithm can set the value of a variable at any time step of the simulation. In the example above, the algorithm could supply a value for the a_{cm} variable at each time step.
- It may be that the algorithm manipulates *Newton's* system of motion equations such the system becomes underconstrained, admitting many solutions. In this event, the algorithm can tell *Newton*, through an appropriate subroutine call, to select a motion that instantaneously minimizes some quadratic function of the variables of the system.
- The algorithm can observe the world state and act upon it. For example, a walking algorithm might observe that the heel has touched the ground and react by moving into a new state of its execution (like a finite-state machine).

At each time step of the simulation, *Newton* evaluates the current system of equations to determine values for any unknown variables, including the translational and rotational accelerations of the individual components of the object. *Newton* integrates these accelerations to produce the state at the next time step, and this process is iterated.

Format of a Control Algorithm

A control algorithm can be considered as a set of finite state machines. Each machine has an initial state and a transition between states is made when some user-defined predicate become true.[1]

In the algorithm of Figure 1 there is a single machine having initial state `START` and having one transition `START -> CM-ACCEL`. The transition is made immediately, and defines a new unknown variable, $\ddot{r}_{cm}$, causes an equation[2] to be added to the system of motion equations, defines a function f which will be called whenever *Newton* needs a value for $\ddot{r}_{cm}$, and defines a quadratic minimization function. Note that the object which is being simulated must be defined elsewhere.

initial-states { **START** }

transition START -> CM-ACCEL when TRUE

  begin
  new-unknown " $\ddot{r}_{cm}$ "
  add-equation " $\ddot{r}_{cm} = \frac{1}{M}\sum m_i\,\ddot{r}_i$ "
  add-function " $\ddot{r}_{cm} = f($ **time** $)$ "
  add-quadratic " $Q = \sum \ddot{r}_i + \sum \dot{\omega}_i$ "
  end

Figure 1: A Simple Control Algorithm

Overview of Newton

The walking algorithm described in this paper has been designed and tested using the *Newton* simulation system [9] developed 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 [7] and DADS [12] systems are examples of large state-of-the-art systems from the mechanical engineering domain. Such systems are sophisticated in many ways: they support efficient formulations of mechanism dynamics, they use fancy numerical techniques for solving equation systems, they often handle object flexibility and elasticity, and so on. The recent work by graphics and animation researchers (discussed above) has generally been less sophisticated but has placed greater emphasis on animation of interesting high-degree-of-freedom mechanisms.

Still, none of these systems combines the full range of features required to make dynamics simulation as powerful and useful as it could be. 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.

Newton Architecture

Using *Newton*, a designer can define complex three-dimensional physical objects and mechanisms and can represent object characteristics from various domains. An object consists 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 has 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.

[1] For the sake of clarity the algorithms will be described in a Pascal-like notation (however, they are currently written in Lisp).

[2] We use quotation marks to indicate that the actual equations must be represented in some internal manner.

Dynamic Analysis in Newton

A 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. 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 (i.e. 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 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.

Event handling, impact and contact

Newton, unlike many other simulation systems, can automatically and incrementally reformulate the motion equations as *exceptional events* occur during simulations.

Figure 2: Changing Kinematic Relationships

One kind of exceptional event is a change in kinematic relationship between objects. Figure 2 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.

Newton's event handler is primarily responsible for detection and resolution of impacts, for analysis of continuous contacts between objects, for the corresponding maintenance of *temporary hinges* that model unilateral 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 analogous 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 elsewhere [14, 15, 11, 10].

Event definition and control

Support for control programming is provided by allowing users to define their own event types. Events provide the mechanism for state transitions in control programs. Event definition consists of a specification of how to detect the event (including information about how accurately the time of event occurrence should be isolated) and how to resolve it.

```
procedure position-with-PD( equation-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( equation-name,
           " a + (2/τ) v + (1/τ^2)(x − x-desired) = 0 " )
  end
```

Figure 3: PD Controller Used in Positioning

Figure 4: Simulated Biped Model

Low-level Controllers

In designing algorithms with *Newton* we found ourselves frequently using PD (proportional–derivative) controllers and curve-fitting controllers to control the "trajectory" of many of the defined quantities. In controlling the biped, for example, 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 adds an equation which produces accelerations to move an `object` to within 1% of a position `x-desired` within a given time `delta-time`. The equation continues to affect the object's motion until it is explicitly removed by the control algorithm. 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.

The Biped Algorithms

We have developed two algorithms to control a biped: one for straight-line walking and one for walking up stairs. An abbreviated version of the walking algorithm is shown in Figure 8.

The simulated biped consists of a torso, two legs with knee joints and two feet with toe joints. This model was adapted from a description of McMahon [19] and is shown in Figure 4. The hips are three degree of freedom spherical joints, the ankles are two degree of freedom universal joints, while the knees and toes are one degree of freedom revolute joints, making a total of fourteen degrees of freedom. The biped is about 180 centimeters tall, weighs 85 kilograms, and has moments approximating those of a human being.

Walking Algorithm

For ease of exposition, the walking algorithm of Figure 8 is an abbreviated version of our actual algorithm. We have hidden many of the lower level procedures (in particular, those which compute the trajectory of the heel). The actual algorithm is written is Lisp; a simulation language like that of Herr and Wyvill [13] will be implemented in the future.

The algorithm has three states: `START`, `SWING` and `DOUBLE-SUPPORT`. Consider the `START -> SWING` transition in Figure 8. After this transition (that is, during the `SWING` phase) the torso is forced to remain in a fixed orientation by the `TORSO-ORIENTATION` constraint. The swing foot follows a trajectory defined by an equation called `SWING-HEEL-TRAJECTORY` which was determined by the procedure `move-heel-to-target`, the stance leg is stiffened with `set-angle-with-PD`, the foot is oriented for landing with `orient-with-PD`, and the angle of the toe is set with `set-angle-with-PD`.

In the `DOUBLE-SUPPORT` phase, the constraints on the swing foot are removed, the names of the swing and stance legs are swapped, and the torso is constrained to accelerate slightly forward, which helps the trailing heel to lift.

The largest number of constraints are applied in the `SWING` phase, during which eleven scalar equations have been added to *Newton's* system of motion equations. Since the biped has fourteen degrees of freedom, it remains underconstrained at all times. A quadratic cost function **Q** is defined in order to fully determine the motion of the biped (a motion is chosen to minimize **Q**). The cost function is a weighted sum of the translational

Figure 5: *Newton* Statistical Output

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.

A slightly more complex walking algorithm was actually implemented and tested with the *Newton* simulation system. Figure 6 shows ten frames in which the biped completes a full cycle. The full simulation consisted of twenty seconds of straight-line walking on a flat surface.

Stair Climbing

Another version of the algorithm was developed for stair climbing. The principal differences between the walking and climbing algorithms were: a more complicated function to determine the trajectory (it has to avoid the steps), a "loose constraint" holding the torso upright, which allowed the torso to sway in a natural manner (this is explained below), and various parameter changes (for example, the foot strike orientation will be different when climbing stairs than when walking).

Figure 7 shows six frames (side view and back view) in which the biped lifts the right foot. Note that the torso sways slightly (the degree of sway can be changed trivially) and that the torso moves from side to side to be over the supporting foot.

Discussion

The walking algorithm of Figure 8 looks almost too simple to be true. While a lot of the underlying procedures have not been described in this paper, the real reason for this simplicity is that *Newton* automatically handles almost all of the underlying dynamics, and, if we choose, can also automatically handle the detection and resolution of impact and contact.[3]

Due to the simplicity of our current biped model, the algorithms are forced to use too many constraints to achieve the desired motion. In particular, the trajectory of the heel must be exactly specified, yielding motion which can sometimes appear unnaturally stiff. Experiments have shown that the best way to avoid this stiffness is to "loosely constrain" the heel trajectory by adding a weighted term to the minimization function **Q**. This weighted term is the square of the difference between the actual toe acceleration and a computed acceleration which guides the toe along the desired trajectory.

Future Work

We will experiment with elastic tendons in the hope that the swing phase will not have to specify an explicit 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 "ballistic walking" as described by McMahon[19].

The algorithms will be extended to include downstairs walking and turning on a level surface. Once a suite of such algorithms has been developed, we will be able to define a set of high level commands such as "walk forward" and "step up". With these commands, the animation of walking bipeds should be a simple task for the animator.

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.

[3] For the sake of efficiency, two additional finite state machines — one for each foot — are used to deal with impact and contact, rather than allowing *Newton* to do so in a more general, and hence more expensive, manner. These finite state machines are hidden from the animator.

Figure 6: Walking Cycle

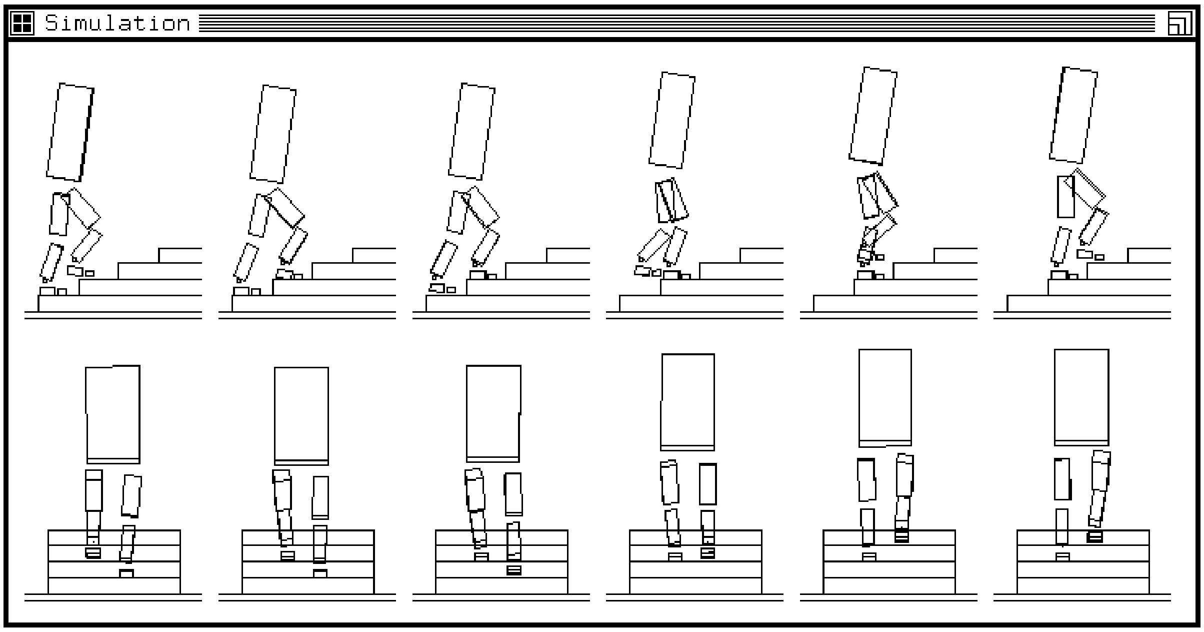

Figure 7: Climbing Cycle

```
const   time-in-air              = 0.5 s
        foot-strike-orientation  = 10° about (0 0 1)
        torso-orientation        = −10° about (0 0 1)
```

let **F** = $K_p \ (r_{torso} - \frac{1}{2}(r_{lfoot} + r_{rfoot})) \ + \ K_v \ (\dot{r}_{torso} - \frac{1}{2}(\dot{r}_{lfoot} + \dot{r}_{rfoot}))$

let **Q** = $(\sum \dot{\omega}^2 \ + \sum \ddot{r}^2 \ + \ 20 \ (\ddot{r}_{torso} - \mathbf{F})^2)$

```
initial-states = { START }

transition START -> SWING when TRUE

  begin
  add-quadratic( Q )
  orient-with-PD(      TORSO-ORIENTATION,       TORSO, torso-orientation, .2 s )
  move-heel-to-target( SWING-HEEL-TRAJECTORY,   swing-heel )
  set-angle-with-PD(   STANCE-KNEE-ANGLE,       stance-knee, 175°, 0.1 s )
  orient-with-PD(      SWING-FOOT-ORIENTATION,  swing-foot,  foot-strike-orientation, time-in-air )
  set-angle-with-PD(   SWING-TOE-ANGLE,         swing-toe,   0°, time-in-air )
  end

transition SWING -> DOUBLE-SUPPORT when hits-ground( swing-foot )

  begin
  remove-equations( SWING-HEEL-TRAJECTORY, SWING-FOOT-ORIENTATION, SWING-TOE-ANGLE )
  swap-swing-and-stance()
  accelerate-torso( TORSO-ACCELERATION )
  end

transition DOUBLE-SUPPORT -> SWING when leaves-ground( swing-foot )

  begin
  remove-equation( TORSO-ACCELERATION )
  remove-equation( STANCE-KNEE-ANGLE )
  move-heel-to-target( SWING-HEEL-TRAJECTORY,   swing-heel )
  set-angle-with-PD(   STANCE-KNEE-ANGLE,       stance-knee, 175°, 0.1 s )
  orient-with-PD(      SWING-FOOT-ORIENTATION,  swing-foot,  foot-strike-orientation, time-in-air )
  set-angle-with-PD(   SWING-TOE-ANGLE,         swing-toe,   0°, time-in-air )
  end
```

Figure 8: Abbreviated Walking Algorithm

References

[1] W. W. Armstrong and M. W. Green. The dynamics of articulated rigid bodies for purposes of animation. *The Visual Computer*, 1:231–240, 1985.

[2] D. E. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *Computer Graphics (SIGGRAPH 89)*, pages 223–231, 1989.

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

[4] R. A. Brooks. A robot that walks: emergent behaviors from a carefully evolved network. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, pages 692–694c, May 1989.

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

[6] A. Bruderlin and T. W. Calvert. Goal-directed, dynamic animation of human walking. In *Computer Graphics (SIGGRAPH 89)*, pages 233–242, 1989.

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

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

[9] J. F. Cremer. *An Architecture for General Purpose Physical System Simulation — Integrating Geometry, Dynamics, and Control.* PhD thesis, Cornell University, May 1989.

[10] J. F. Cremer. *An architecture for general purpose physical system simulation – Integrating geometry, dynamics, and control.* PhD thesis, Cornell University, 1989. also as Cornell technical report TR 89-987.

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

[12] 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.

[13] C. Herr and B. Wyvill. Towards generalised motion dynamics for animation. In *Graphics Interface*, pages 49–59, 1990.

[14] 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.

[15] C. M. Hoffmann, J. E. Hopcroft, and M. S. Karasick. Towards implementing robust geometric computations. In *ACM Annual Symposium on Computational Geometry*, pages 106–117, June 1988.

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

[17] J. K. Kearney, S. Hansen, and J. F. Cremer. Programming mechanical simulations. In *Proceedings of the 2nd Eurographics Workshop on Animation and Simulation*, pages 223–243, September 1991.

[18] M. McKenna and D. Zeltzer. Dynamic simulation of autonomous legged locomotion. In *Computer Graphics (SIGGRAPH 90)*, pages 29–38, 1990.

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

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

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

[22] M. H. Raibert and J. K. Hodgins. Animation of dynamic legged locomotion. In *Computer Graphics (SIGGRAPH 91)*, pages 349–358, 1991.

[23] M. van de Panne, E. Fiume, and Z. Vranesic. Reusable motion synthesis using state-space controllers. In *Computer Graphics (SIGGRAPH 90)*, pages 225–234, 1990.

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

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

[26] D. Zeltzer. Towards an integated view of 3-d animation. *The Visual Computer*, 1:245–259, June 1985.