Using Kinematic Clones to Control the Dynamic Simulation of Articulated Figures

By

William Charles Westenhofer

B.S.E.G. May 1990, Bucknell University

A Thesis submitted to

The Faculty of

The School of Engineering and Applied Science
of The George Washington University in partial satisfaction
of the requirements for the degree of Master of Science

May 14, 1995

Thesis directed by

James Kwangjune Hahn
Assistant Professor of Engineering and Applied Science

ACKNOWLEDGMENTS

The author would like to thank *Rhythm & Hues, Inc.* for the generous use of their rendering and editing resources. All of the material in the demonstration video was created at their facility and with their proprietary rendering and compositing software. Thanks especially to Steve Ziolkowski and Ian Hulbert for their modeling support.

Additional thanks goes to Larry Gritz who helped with insight in choosing an articulated figure dynamics algorithm and in proofreading this thesis.

TABLE OF CONTENTS

INTRODUCTION

Over the past century, classical animators have striven to endow their characters with "life." All movements and actions had to be made to work in concert in order to suspend the disbelief of a character's existence, allowing it to transcend from a set of sequential drawings to a personality that an audience could identify with. Animators at groundbreaking studios like Disney and Warner Brothers learned early on that such a quality could be reliably attained by following a set of principles and guidelines [1]. Many of these principles, including such things as *proper staging*, *exaggeration*, and *appeal*, are artistic metrics that are important for the unambiguous portrayal of the intended actions of a character. There are others, however, like *squash-and-stretch*, *follow-through*, and *overlapping action* which are really formalizations of phenomena that occur naturally as a result of gravity, inertia, and the deformations of viscous material. Such physical behavior is well understood and capable of being calculated which is why it is tempting, in light of the relatively recent use of the computer as an animation medium, to let a dynamics simulation handle their incorporation into a character's movements. Animations to date that have employed physically-based models have produced incredibly realistic motion of both rigid and deformable objects [2][3].

The dynamics of rigid bodies have been known for quite some time. *Classical Mechanics* by H. Goldstein [4], was originally published in 1950 and is still considered a "bible" on the subject. The use of dynamics in computer applications is not new either; robotics, mechanical and civil engineering have all used computer driven rigid body dynamics simulations in some form or another. Computer graphics and animation applications of dynamic simulations have also been presented and produced. Some of them, especially particle systems have enjoyed widespread use at both the research and professional levels [5]. While the use of dynamic simulations may not be new, the issue

of how to *control* them is still a concern for their practical applicability in a production environment. Professional animation tends less to be an act of discovery and more the result of planning and execution. Any use of simulations must therefore provide adequate control to be acceptable within this paradigm. This is especially true when dealing with *character animation* since the actions and intentions of central characters receive the most scrutiny from both directors and audiences alike. The goal of this thesis is to present a technique that makes the incorporation of dynamic motion practical in a production environment. A new control strategy, one that uses a structure called a *kinematic clone*, is presented. This strategy gives an animator detailed control over a dynamic simulation while still allowing the animation to benefit from the realism the dynamics provides.

In terms of rigid bodies, this paper focuses on articulated figures. An articulated figure is a set rigid bodies connected by flexible joints, analogous to the limbs, joints and torso of a living creature. The concept of an articulated figure is actually born out of robotics from which is also borrowed the specialized equations of motion for their use [6]. For the purposes of this thesis, the term "dynamics simulation" will be assumed to be a simulation involving articulated figures unless otherwise specified.

Articulated figures have appeared before in computer graphics in kinematic control systems. Jointed characters are typically represented as hierarchical trees of links. An animator in a kinematic system has complete control over the positioning of each joint and link by manipulating joint parameters within a figure's degrees of freedom. The hierarchical structure allows an animator to specify the relative motion between a link and its parent (e.g. a hand in relation to a forearm). Using a parent-child hierarchy also maintains link connectivity and allows an animator to move substructures (e.g. an arm) as a unit. Of the kinematic systems, *keyframe* systems are the most widely used in production animation work. Their popularity undoubtedly stems form their similarities with the traditional 2D cel animation technique by the same name. Most professional

animators are classically trained (.i.e. 2D) and must transition to 3D when they begin animating on a computer. Using a keyframed system allows direct application of the principles learned in 2D. This is good from a control standpoint but bad from a tedium standpoint, especially considering the large number of degrees of freedom in a typical character. Adding the subtleties of dynamic motion to an animation typically requires the inclusion of a much larger number of keyframes and these "key positions" must be set for each degree of freedom. This tedium, coupled with the ever increasing pressure to cut production times justifies the search for a technique which allows the control of kinematic systems with the automation benefits of a dynamic simulation.

To do this one must consider what makes realistic motion *real* and what makes expressive motion *expressive*? The assumption here is that realism can be tied to physical equations which can be computed while expression deals with perceptions and qualitative measures that are best left to the animator. The quality of "realism" presumably comes from our experienced expectations of how an object should move when subjected to the physical laws of motion. Arms under muscular control still sway somewhat as a result of gravity. A heavy person sags a little more on his/her joints than a thin person and shows a bit more jiggle in the fatty layers. A dog, on landing after a big jump shows tremendous follow through as its muscles strain against inertial forces. On the other hand, expressions and emotions defy such precise explanations. Depicting personality in animation is basically "acting" where a sequence of images replaces movement about a stage. A quote by Saint Exupéry pulled from *Disney Animation: The Illusion of Life* [1] reads "It's not the eyes, but the glance—not the lips, but the smile…" Excitement bursts with outstretched arms and legs and wide open fingers. Gloom recedes with slumped limbs and a slouched head. Such things would be very difficult to code algorithmically in all but the most contrived cases.

For a dynamics simulation to work in character animation then, a system must allow an animator to move a character expressively while allowing the simulation to do

the work of adding physical realism. The *kinematic clone* technique achieves this by giving an animator access to a clone of the articulated figure representing a character. This clone is moved kinematically through the keyframing process. This clone then "directs" the movements of the dynamically controlled character through spring forces at each of the joints. Each joint of the kinematic clone is connected to the corresponding joint of the dynamic figure by virtual springs whose tension can be controlled by the animator depending on how closely the dynamic figure must approximate the positions of the clone.

With this setup, the intention is for the dynamic simulation to enhance and refine the motions of the kinematic clone. Thus when a figure lands after a jump, inertia causes the links to swing forward a little on impact. When an animator waves an arm, the arm will move closely to the suggested path, but the dynamic solution will add tiny nuances to the motion, including gravitational sag and follow through. The spring tensions can be adjusted to vary the degree to which the dynamics matches the kinematics. The hope is that with the assistance of the dynamic simulation, an animator can greatly reduce the number of keyframes required. In purely kinematic systems, follow-throughs and oscillations must be accounted for through the use of extra keyframes. These extra keyframes would be unnecessary if the dynamics simulation could account for them.

Chapter One will present the background and related material that was researched in the development of this technique. Chapter Two is devoted to the dynamics equations used for the simulation of articulated figures. This paper uses a very efficient recursive dynamics formulation called the *Articulated Body Method* developed by Roy Featherstone [6]. After that, the implementation details of a current system that incorporates the kinematic clone concept will be given in Chapter Three, followed by the results of experiments conducted on the system designed to measure its potential in Chapter Four. The final chapter will discuss areas for future development to improve the performance of the system.

CHAPTER 1

PREVIOUS WORK

Controlling a dynamic simulation in order to produce a desired motion has been a concern of robotics research for quite some time [7]. In robotics, a motor control program must be able to calculate the torques and forces required to drive the servos at each joint. Since an articulated figure in computer graphics is in one sense a virtual robot, a naive control technique would be to allow an animator to "play the role" of the motor at each link. For each frame of an animation, the animator could supply the set of torques and forces that drive them. To make an arm lift and grab an object, for example, an animator would supply the appropriate time-varying torques to the shoulder, elbow, and wrist to make the arm perform the task. Unfortunately, the work demanded of an animator in this case is far more difficult and less intuitive than the work required to specify a similar motion kinematically. While the final result will probably look more realistic (once the correct forces are found), an animator will have to spend more time working and "tweaking" the torques and forces before the figure behaves as desired. The added difficulty stems from the fact that people tend to think kinematically when planning motions. The conscious mind tends to visualize goals like "move arm to point X to grab the apple." or simply "jump over there." The subconscious mind has been trained through experience to respond with the appropriate muscular tugs and pulls. In the less frequent occasions that forces are considered on a conscious level (e.g. "pull *hard enough* to open the door"), it tends to be an inexact, trial-and-error process requiring force testing and feedback. It is very hard to guess the exact outcome of a command like "move the shoulder with a force of 10 Newtons for 10 seconds" Such difficulties carry into animation control. This need has driven the research described below.

Work to date can be grouped into two categories based upon the nature of animator input. *Direct simulation control* schemes seek to control a simulation by manipulating parameters of the simulation itself. They attempt to mitigate the difficulty of force and torque inputs by either providing higher-level, abstract parameters to control, or lessening the number of forces and torques that must be dealt with. Examples include techniques that allowing some of the links to be controlled kinematically with the rest remaining dynamic. Also included are more advanced methods that attempt to replicate the motor control programs of real creatures in software. *Induced simulations*, in contrast, are controlled through objective functions that describe the general characteristics of what a motion is supposed to do. The system then employs dynamic simulation to find a motion that satisfies the objective functions. The kinematic clone concept belongs to the first category, but work from both will be described here to provide a comparison for judging the concept's utility.

Direct Simulation Control

When designing a simulation, it seems natural to grant control by building 'hooks' into the simulation that an animator can control. A wide array of terms are readily available for this purpose, including the force of gravity, mass of the links, and the torques and forces themselves that drive an articulated structure. In addition to manipulation of terms, an animator could add an additional layer to the simulation by building motor control programs that are patterned after the biomechanical processes of a real creature.

The defining quality for all of these modes of control is that the animator directly manipulates some part of the simulation itself. To do this, the animator accepts the responsibility of understanding the fundamental principles of the dynamic simulation in order to supply appropriate values for the parts that are being controlled.

Kinematic Approaches

The use of kinematics to control dynamic simulations can be found in many early papers. The techniques used keyframing to control important parts of a figure while other parts would rely on the simulation. The kinematically controlled links move without regard to external forces or with sufficient additional forces applied to make it appear so. The remaining links would be controlled by the dynamic simulation with the effects of motion of the kinematic links being included in their calculations.

Total Kinematic Articulation

In his paper, Hahn [2] places all of the joints of an articulated figure under kinematic control, seeking only to determine the net effect of their motions on the motion of the body as a whole. This is equivalent to the "floating base" systems in the robotics literature [6][7], as is used to find the net motion of a body in free-fall or in orbit like a robotic arm on a satellite. The motion of internal links effects the rigid body motion of the figure by imparting forces and torques due to the conservation of linear and angular velocity.

Hahn's technique proceeds by finding a single rigid body inertia tensor (see Chapter 2) from the sum of all the inertia tensors of the links transformed to the body fixed coordinate system. Once a single instantaneous inertia tensor has been found, the forces imparted on the body by the kinematic motion are calculated such that the net change in linear and angular velocity is nullified. The resultant angular velocity of the rigid body is given by:

(1) $$d\mathbf{W}' = -\mathbf{I}_t^T[(\sum_j \mathbf{I}_j)\, d\mathbf{W}_j + (\mathbf{R}_j \times (d\mathbf{W}_j \times \mathbf{C}_j))(\sum_j m_j)]$$

$d\mathbf{W}'$ resultant angular velocity of the body
$\mathbf{I}_t^T$ transpose of total inertia tensor of body
$\mathbf{I}_j$ inertia tensor of link i (including children of i)
$d\mathbf{W}_j$ angular velocity of link j
$\mathbf{R}_j$ location of joint in body fixed coord.
$\mathbf{C}_j$ location of center of mass in body fixed coord.
m_j mass of link i

The total linear velocity is found by comparing the displaced location of the new center of mass in comparison to its position from the previous frame. For linear velocity to be conserved, these values must coincide. The "conservation" force is then the force required to move the new center of mass to the location of the old one.

Hahn's method works very well for situations where an articulated figure can be treated as a single rigid body. These situations are generally limited, however, to figures in free-fall. Figures in contact with the ground or a wall do not behave in this manner since they are conceptually part of a single rigid body composed of the union of themselves and the immovable surface with which they are in contact. Since the ground is generally implemented as a mass of infinite size, conservation of momentum is not assured.

Partial Kinematic Articulation

Another means of realizing kinematic control is to classify certain links as kinematic and others as dynamic. This technique was proposed by several researchers [8][9][10]. Each implemented the technique in a slightly different way, however.

Wilhelms [8] and Armstrong et al. [9] both proposed a method whereby links could be assigned to certain states, each of which represents a different control method for the animator to choose. For example, in Wilhelms' paper, a link could be assigned

one of four states: 1) *direct dynamic* control where an animator supplies a torque function by building a splinar curve; 2) *relaxed control* where no internal forces are given and the link hangs loosely governed only by gravity and collisions; 3) *frozen control* where the system supplies the forces to "freeze" the relative joint angle with a link's parent; and 4) *oriented control* where forces are supplied by the system to hold a link in a fixed world coordinate position/orientation. Importantly, these states can be changed over time, so an arm after moving under direct dynamic control to place an object on a shelf, for example, can revert to a relaxed state and fall limply at the actor's side when it is done. She also introduced a concept called *positional control* which is very similar to kinematic clones. Her technique did not allow an animator to vary the degree to which the dynamic motion would approximate the kinematic positions and did not allow links to be controlled completely through kinematics, the value of which is discussed in Chapter 4.

Armstrong et al. [6] proposed a similar system, but added the concept of local vs. global control. Local control effects the links themselves; they can be positioned, oriented, or allowed to hang freely, just like Wilhelm's version. The paper also adds an additional local control mechanism called a simple move. A simple move gives a new angular position between a link and its parent. The system sets up a smooth transition function from rest at a link's current position to rest at its new position by calculating the appropriate torque by:

$$(2) \qquad T(x) = \begin{cases} \alpha\,(e^{\beta(x-\gamma)} - 1) & \text{if } x \geq \gamma \\ \alpha\,(e^{\beta(\gamma-x)} - 1) & \textit{otherwise} \end{cases}$$

The parameter γ is the desired angle between a link and its parent while x is the current angle. The other constants, α and β, must be tweaked to prevent the link from moving or accelerating too quickly. A smooth "ease-in" and "ease-out" are performed during this transition using the following formula:

$$(3) \quad maximum\_torque = center\_torque \sqrt{(x - \theta)/(\phi - \theta)}$$

Here, *center_torque* is the torque at the angular midpoint of the motion, *x* is the current angle at this time-step, θ is the final angle, and ϕ is the angular midpoint. This process is also conceptually similar to the work accomplished by kinematic clones. The two significant differences are that (3) requires that links be at rest at the start and end of a movement, and both (2) & (3) require a continuous rotation about a fixed axis from the start of the move to the end. These limitations are not acceptable for general animation.

As mentioned earlier, Armstrong also introduced the concept of global control. Global control encompasses processes the effect the entire figure to achieve a certain goal. Examples include a processes to introduce balance and another to introduce ground collisions. Armstrong suggested several approaches for realizing these effects such as a force that keys off the relative angle with the body to the ground. The ground-to-figure collision process is used in the current kinematic clone implementation.

While the previous two techniques allow the simultaneous control of both kinematic and dynamic links, both limit the animator to the set of control methods which were set a priori. A more general approach is presented in [10]. Isaacs and Cohen's DYNamic MOtion system (DYNAMO) incorporates kinematics indirectly through a "behavioral motion" layer[1]. This layer is completely general and allows any procedure (including keyframing) to supply the forces, torques, and accelerations to the dynamically controlled links. Examples of a behavioral motion controller might be a keyframe system, or a system that applies a braking force when a speeding car passes too close to a cliff. The behavioral motion layer then can pass both forces and accelerations into the dynamics system. For each link, the data that is passed, be it a force or an

[1]Not to be confused with "behavioral motion control" as defined in C. Reynolds, "Flocks, Herds, and Schools: A Distributed Behavioral Model," *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21, no. 4 (July 1987): 25-34.

acceleration, determines whether the link will act as a kinematic link or a dynamic one. Both the forces and the accelerations are combined in a set of linear equations. Links for which the accelerations are known are subtracted out, removing them from the set of linear equations. Gaussian elimination is then used to solve the system. This process is an example of a *Composite Body Method* discussed in Chapter 2.

Summary of Kinematic Control

Each of the papers presented above have desireable features that have actually been incorporated into the kinematic clone technique. They all share the advantage of allowing an animator to control some or all of the links kinematically. The kinematic clone technique distinguishes itself by formalizing the relationship between the kinematic process and the dynamic process and by letting the animator choose the degree to which the final motion is effected by either. This point is crucial in allowing an animator to portray a "tense" character versus a "relaxed" one.

Motor Control Schemes

If a higher level of control of a dynamic simulation is desired, a useful approach is to study biological systems to find how they handle the motion control process. Real creatures locomote by stimulating muscles on command from a body's nervous system. Researchers have been able to mimic natural motor control by writing programs that imitate these systems, either by direct application of biomechanical research, or indirectly through similar research that has already been performed in robotics.

One of the interesting results of recent biomechanical studies has shown that certain periodic motion is controlled through distributed neural processes, as opposed to a completely centralized system as was once thought [11]. For the purposes of control, this implies that complex motion can emerge from the coordinated workings of simple mechanisms. This discovery has been exploited in the field of robotics [12]. Brooks

developed the robot *Ghengis* which walks about with the simulated intelligence of an insect. The motor controls are distributed; there is no master processor controlling each of the legs. Each leg has its own agenda and must solve problems on its own when trying to reach a goal. For example, a leg has a preprogramed goal of lifting, then swinging forward, then planting, the levering backwards. If an obstacle prevents this, it is up to the leg to lift itself higher to overcome it. For such meager intelligence, it is amazing that Ghengis was one of the first robots to be able to adeptly navigate a room filled with moving objects and debris.

Statically Stable Walking

McKenna and Zeltzer developed a motor control scheme patterned after the work done in biomechanics and robotics for statically stable walking motions [13]. *Statically stable* motion refers to motion that can be halted and maintained at any point in time without causing a creature to fall over. Insects with their six-legged stance fall into this category (rather appropriate since the distributed control technique they used is very similar to Brooks' robots that are said to have the intelligence of insects).

Their simulation of a cockroach uses a distributed oscillator network that accepts input from reflex sensors. The authors cite five observables that have been compiled from the studies of the walking gaits of real insects:

1. A wave of steps runs from rear to head (and no leg steps until the one behind is placed in a supporting position).
2. Adjacent legs across the body alternate in phase.
3. Stepping time is constant.
4. Leg step frequency varies.
5. The interval between steps of adjacent legs on the same side of the body is constant, and the interval between the stepping of the foreleg and hindleg varies inversely with the stepping frequency.

Experimental data of insect nervous response has confirmed that each leg has its own "pacemaker" termed an oscillator in the literature, that fires every time the leg is supposed to step under normal conditions. These oscillators are coupled to cause the "wave of steps" to occur and to synchronize the motion on each side of the body. Interestingly, the method used by the oscillator motor control to move the legs is identical to the kinematic clone concept of linking a kinematic potential to the dynamic link via spring forces. The oscillator moves the "rest angle" of the spring in order to induce movement of the legs. The key distinction between the kinematic to dynamic link in McKenna and Zeltzer's paper and the kinematic clone concept is the control that is afforded the animator. With the motor control system, the animator is not directing the kinematics; the "animator" is actually writing control programs which do so. In actuality, these motor control programs need not incorporate kinematics at all. The kinematic interface just happened to be a means of directing the legs in this particular instance. All of this is not unexpected; it was not McKenna and Zeltzer's intent to provide a kinematic interface to the animator, but rather to create virtual autonomous creatures that could locomote on their own.

As far as autonomous locomotion is concerned, the oscillators alone are not robust enough for uneven terrain. Feedback support is essential and is supplied through a set of triggers attached to each leg. A step trigger causes a leg to step automatically when it reaches some maximum rearward extension. A support trigger sends an inhibitor signal to a leg's step reflex whenever that leg is supporting some minimum weight of the creature. Finally, an elevator reflex causes a leg to lift higher when its forward swing is obstructed. Amazingly, these three simple responses are enough to allow a virtual cockroach to maneuver on rough terrain and to surmount obstacles.

In a companion paper [14], McKenna, Pieper, and Zeltzer discuss higher levels of control processes that rely on the roach's walking ability to carry out their tasks. The

animated roach is placed in a virtual environment that an animator has access to through typical virtual reality interfaces (e.g. Spaceball or DataGlove). The animator issues simple commands like follow the cursor, and run and hide. The roach responds by running towards the cursor, or running rapidly until it runs into a "safety zone," respectively. These commands are interpreted by the highest level and converted into commands that influence the walking level. The walking level follows these commands when possible, but feedback can readily override them to force the roach to first accomplish an intermediate task. Thus, "follow the cursor" is sometimes interrupted by "step over the brick." The result resembles the "virtual actor" described by Zeltzer [15], albeit a very simple one.

Statically Unstable Walking

Statically unstable walking differs from its stable counterpart in that a creature exhibiting this type of walk is in a constant state of imbalance. It literally "falls" from one step to the next, catching itself every time a leg hits the ground. Bipedal and quadrupedal motion are examples of statically unstable walking. Raibert and Hodgins [16] proposed some motor control schemes for this type of motion in an analogous fashion to McKenna's and Zeltzer's static models.

A complete dynamic solution is much more important for hopping and running creatures than it is for slower moving ones. Raibert and Hodgins point out that for running creatures, a much smaller percentage of the energy required for motion is contributed by the muscles themselves. This is readily apparent when one observes the effort required for a running creature to stop. A large amount of energy is actually stored as potential and kinetic energy in the elastic tendons of a leg and the inertial forces of the body and limbs. To account for the elastic energy, the authors added compliant legs, that can telescope and have longitudinal springs able to store potential energy. These

structural changes were needed before any consideration was made for the motor controllers themselves.

Similar to McKenna and Zeltzer, Raibert and Hodgins fashioned models from analytic analyses of robots and real creatures. They were successful in generating balanced and cyclic gaits for monopedal, bipedal, quadrupedal, and kangaroo-like creatures. They even compared experimental measurements of the simulated creatures to actual data of robots and kangaroos. The comparisons showed a very close correlation between simulated walking and the real creatures it was imitating.

Summary of Motor Control Techniques

Fashioning motor control programs after real world counterparts is a very interesting pursuit from an intellectual standpoint. The results of McKenna & Zeltzer and Raibert & Hodgins studies produced some very lifelike and experimentally validated motion. Motor control techniques can be very useful for simulating robot designs before they are actually implemented. For animation, however, some difficulties persist. First of all, the process of building motor controllers by hand involves a great deal of trial and error. Raibert and Hodgins admitted that their running creatures repeatedly fell over before the leg and torso controllers had been tweaked correctly. Cyclic motion is also difficult because the animation relies entirely on initial data to manage the timing of events.

A larger issue makes the motor control paradigm even less suited for general animation. The motion produced to date has been limited to relatively simple creatures performing a very small number of tasks. While this is acceptable for insects, the interplay between controlling programs and sensory input in more intelligent animals is vastly more complex. Automated control at that level ventures into artificial intelligence in areas for which the field has not yet advanced. As discussed in the introduction, the creation of expressive motion is still best served by the hand of an animator.

Induced Simulations

The category of induced simulations presents an entirely different approach for controlling dynamic motion. With direct simulation control, an "animator" is commanding the forces and accelerations either directly or through a motor control program. The final resultant motion is the product of the iterations of a forward simulation. Induced simulations reverse this relationship. An animator gives the computer a set of goals and the simulations seeks out a solution of forces and accelerations that best satisfy these goals. For example, an animator can command a character "to move to point X without tripping over this obstacle." With induced simulation techniques, the goals are expressed as a set of objective functions that have to be satisfied. The computer then finds a motion that simultaneously satisfies all of the objective functions while adhering to all of the physical laws of the dynamic simulation.

Several researchers have implemented methods of realizing induced simulations. The primary difference between the different types is the way in which a solution is found.

Optimal Path Searching

A unique approach for generating induced simulations was presented by van de Panne, Fiume, and Vranesic [16]. They noticed that the act of finding torques which move a joint from an arbitrary point in space to a destination using only an optimal path is similar to solving an all-pairs shortest path problem. This type of problem is readily solved using dynamic programming. One could conceivably build a table indexed by object state (i.e. position and velocity of all of the degrees of freedom) where the table entry holds a vector of torques that must be applied to the joints to move the object along the optimal path to the destination. Such a table could drive a simple motor controller for each joint. This is exactly what the authors present in their paper and they called the construction a *state-space controller*.

A state-space controller for a given object is built through dynamic programming. In an optimal path problem, if, for example, *AD* is an optimal path, then an optimal path from another point *B* that lies along *AD* is the portion of *AD* between *B* and *D*. Thus if all of the optimal paths within a region *X* (which also contains the destination) are known, the optimal path from a location *S* outside of *X* is found by choosing the shortest path that uses any optimal path from *S* to the boundary of *X*. The real algorithm first quantizes state-space into regions. For each region, it finds optimal paths to the boundaries of the region by sampling from the set of all possible torque vectors. This is possible because both state space and the torques are assumed to be bounded. The dynamic simulation is used at this point to evaluate the effects of applying a certain set of torques on an object in a given state. This determines the point on the boundary of a region that the object will hit if a certain torque vector is applied. The algorithm continues until optimal paths from all the sample points are known. When the controller is used, the sample points are used as interpolants to find the values from an arbitrary position.

In order to use dynamic programming, the function to be optimized must be monotonically increasing. The authors chose to hard-code an optimization of the time and energy expended. This is one of the biggest limitations of the state-space controller method. The subsequent induced simulation techniques allow more general objective functions. Another disadvantage is that the time and space complexity of a state-space controller increases with the cube of the total degrees of freedom of an object. Thus for highly complex objects, this technique is no longer practical. For simpler objects, however, this technique does have a distinct advantage in that the controllers can be calculated ahead of time. During the animation process, a state-space controller exhibits real-time performance. One can also link several state-space controllers in a sequence to cause an object to move through various destination states along an optimal path.

Induced Response to Stimulus

McKenna and Zeltzer [12] showed the benefits of allowing stimulus feedback to play a part in the motor control of an object. One of the difficulties with theirs and all motor control techniques is that the animator is forced to program and hand-adjust the motor controllers themselves. An animator might have a better idea of where to put things like sensors and muscles, but might be unable to specify how they should relate. Two recent works [18][19] have demonstrated techniques of connecting networks of sensors and responses and automatically establishing the relationships between them such that a creature can meet a set of objective functions.

Neural Network Approach

van de Panne and Fiume [18] solved the relational problem using neural networks. Neural networks, patterned after our biological nervous system, relate a set of inputs to a set of outputs through neurons that are interconnected by a series of weighted connections. A signal from an input to an output is triggered when the weights along a path meet some minimum threshold. The massive inter-connectivity allows for the formation of complex relationships which is desired in this case. The difficulty with neural networks lies in the task of setting the weights for the connections. It has proven more practical to "teach" a neural network how to relate the inputs to outputs through repeated trial and error and constant readjustment. To use neural networks for motor control, the authors had to find an effective way of doing this.

The neural nodes in a system consist of sensor nodes (touch, angle limit, sight, length extension), internal nodes, and actuator nodes (implemented PD controllers in robotics parlance). Every node has some parameters associated with it, based on the type of node, e.g. sensing range for a sensor. All nodes also have a *delay* and *hysteresis* value that corresponds to the delay of propagation of a signal, and an amount of time that a node continues to propagate a signal the initial stimulus is gone, respectively. The

authors found that these values are essential for cyclic motion as they allow for the oscillation patterns found in real creatures [10].

The set of nodes are joined together by a connection network with each node of a given type being connected to all nodes of the other two types. As stated previously, the settings of the connection weights is an important problem. The authors found that a randomized generation and objective evaluation function worked best for this purpose. The evaluation function can be user defined; they chose a simple function of distance traveled and time expended. The random search effectively creates a generation of basic locomotion schemes for a sensor-actuator network. The top 5% of these are refined through a stochastic ascent algorithm that works on the node parameters only. The authors found that modification of the weights at this stage was not productive since it tended to create unwieldy fluctuations in the class of motion within an individual family. Rather, the parameters like sense thresholds, delays, etc. are tweaked for a set number of iterations to produce a final motion.

Parameter Approach

Another way of handling a stimulus-response technique is to link stimuli and responses together as pairs and to redefine stimuli as some function of the combined sensory input. Ngo and Marks [19] proposed this solution and came up with a technique that works analogously to sensor-actuator networks. In this case, a stimulus function weights all of the inputs using the following function:

$$(4) \qquad \mathbf{w}\left\{1 - \max_{i=1}^{\nu} \left[(\lambda_i (\nu_i - \nu_i^\circ)\right]^2\right\}, \quad \mathbf{w} = \sum_{i=1}^{\nu} \log\left(\frac{\lambda_i}{\lambda_i^{min}}\right)$$

$\{\nu_1, \nu_2, \nu_3, \ldots\}$ "sense" variables

ν_i° and λ_i parameters set by search

λ_i^{min} predetermined minimum range

Likewise, a response function is a "prescription" for motion in that is provides a critically damped transition from one location to another:

$$\tau^2 \ddot{\theta}_i + 2\tau\dot{\theta}_i + (\theta_i - \theta_i^o) = 0 \quad (5)$$

τ	user defined time constant
$\{\theta_1, \theta_2, \ldots \theta_N\}$	current joint angles
$\{\theta_1^o, \theta_2^o, \ldots \theta_N^o\}$	target joint angles

Ngo and Mark employed a parallel genetic algorithm to find the parameter weights for the stimulus functions. Like van de Panne and Fiume, their evaluation function was concerned primarily with distance traveled and time elapsed.

Summary of Stimulus-Response Methods

The advantage of these techniques is that they can come up with modes of locomotion that an animator neither anticipated nor considered. They come up with these with minimal operator input as well. Just like the motor control programs, however, the complexity of the motion that they seem to be capable of is limited. Their search spaces are bounded which places some unknown maximum complexity on their actions. These methods are less constrained in time and space complexity than the state-space controllers (see below), however, because the random searches do not increase in complexity as more joints are added to a figure.

Space-Time Constraints

The induced simulation paradigm allows an animator to control the basic objectives of a motion that behaves under the rules of a dynamic simulation. If, in mechanical terms, a direct simulation is an *initial-value* problem, then an induced simulation is a *boundary-value* problem where the values of various parameters are given

for the beginning, end, and perhaps, in-between points of a motion. These points can be viewed as constraints in both space and time, and have been thus aptly named *spacetime constraints*. These constraints express both the components and their limits that need to be optimized. To express the goal of a generalized induced simulation in a canonical form, it is to "find the minimum R(S) where C(S)=0, where S are the parameters of the simulation, R is the set of objective functions, and C is the set of constraints." [20]. The group of control philosophies that follow are placed here because of the generality with which they approach this goal. I have subdivided them by the way in which they try to find an optimal solution.

Sequential Quadratic Programming

It was Witkin and Kass [20] who first introduced this control philosophy to computer graphics in their landmark paper by the same name. The stated mission of their technique was to build a system that would automatically produce a motion the exhibited as many of the principles of traditional animation as possible. In practice their system did indeed demonstrate many of these including *anticipation*, *squash-and-stretch*, and *follow-through*. The difficulty with their system, however, was that it was very cumbersome. Its internal engine used a variant of Sequential Quadratic Programming (SQP) to perform a constrained optimization of the user's objective functions. In order for SQP to work, the objective functions must be supplied as program functions that can be evaluated repeatedly. The authors admitted that the algebraic equations being programmed could, in general, become prohibitively complex. In fact, they had to develop a set of precompiled functions and add a graphical interface which linked them together to make the problem tractable. To their credit, however, their system was able to produce some incredibly convincing animations of a lamp hopping and skiing over a jump and is one of the few systems that automatically introduces anticipation into a characters movements.

The initial space-time constraint method was enhanced to a more workable and complete form by Cohen [21]. From a procedural standpoint, Witkin's and Kass's implementation was limited by the fact that objective functions had to be optimized over the entire span of an animation. Symbolically, it is as if the object being animated had complete fore-knowledge of where it had to go. Cohen used the analogy of a cat chasing a mouse to illustrate his point. If a cat had complete fore-knowledge of the motion of the mouse, in an attempt to minimize its time and energy, it would simply walk over to where it knew the mouse would end up and wait. More realistically, however, a cat optimizes its motion over discrete steps, or "windows" in time, projecting the motion of the mouse forward in its mind and setting a target for it at each step. Cohen enhanced spacetime constraints with a concept called *spacetime windows*, which are literally windows in both space (degrees of freedom) and time (segments of an animation). Different objective functions can be specified for each window, allowing for the setting of interim goals. Windows can overlap to help smooth the transition between them as well. The degrees of freedom and the degree functions themselves are interpolated with a cubic B-spline function to ensure continuity between adjoining windows. Spacetime windows are also critical for the realization of complex motion without a related increase in the requisite complexity of the objective functions.

In addition to the concept of a spacetime window, Cohen introduced several features in his system that were created to help overcome the difficulty of specifying the objective functions, a problem inherent in Witkin's and Kass' version. A symbolic manipulation process allowed an animator to supply mathematical functions in symbolic form as opposed to program code. A graphical display also displayed the results of the objective functions and allowed an animator to interactively edit them. Cohen found that the ability to see a graphical representation of the functions gave important insight into which functions were causing a particular unwanted motion. Keyframing was also

available to add additional constraints where necessary in order to "nudge" the system towards a desired motion.

Genetic Programming

One limit of the Sequential Quadratic Programming method is that the complexity of the solution is limited to the complexity derived from the objective functions. Gritz and Hahn [21] propose a *Genetic Programming* approach to overcome this. Genetic programming uses the concepts commonly used in genetic algorithms to write programs (LISP S-expressions in this case). Gritz and Hahn developed a system that uses this technique to write programs that act as the motor controllers of the joints of an articulated figure. Since the programs are generated randomly from a set of functions and variables, the potential complexity is bounded only by the theoretical bound of all programs derivable from the particular alphabet of components.

Their method proceeds by creating an initial generation where each individual contains a random program for each link. The terminals used in their system contained only simple arithmetic operators, constants, a conditional function, and system variables such as position, velocity, etc. The system is not limited to these, but the authors found these to be sufficient for all of the examples that they tried. Their choices allowed for sufficiently complex motion and allowed implicit coordination caused by the possibility of randomly incorporating system variables in a program.

The "fitness" of an individual is tested by running a dynamic simulation using the generated motor programs to drive the links, and comparing the results to the set of supplied objective functions. One drawback with this and the SQP method is that the functions must still be supplied as program subroutines. Fortunately, the authors stated that most of their examples only required a few lines of code.

Following the process of genetic programming, the most fit individuals are combined and mutated (optionally) to spawn a new generation of individuals. The act of

evaluating fitness and mating only the best, drives the system towards a solution that satisfies all of the constraints and optimizes appropriate objective functions, just like natural selection selects for the most "fit" individuals. The authors were able to get a physical model of a lamp to hop and then move under a "limbo" bar using this technique. The system was able to find and generate motor programs that allowed the lamp to drop and crawl under the bar when it got too low to hop under. The system showed anticipation, squash-and-stretch, and follow-through as well, all generated automatically.

Summary of Space-Time Constraints

Space-time constraint techniques have shown the ability to automatically create pleasing and realistic motion. They have shown to be capable of generating motion that satisfies several of the basic principles of animation. They still seem to have a long way to go, however. The motions created are limited by an animator's ability to write/program mathematical objective functions that meet the goals of an animation. Writing a mathematical function for a "melancholy walk" would be very difficult, for example.

Summary and Conclusions

This chapter has presented the two distinct paradigms for controlling dynamic simulations of articulated figures that has shown up in literature to date: *direct simulation control*, and *induced simulations*. Direct simulation control gives control of various parameters of a simulation to the animator. The animator is in effect directly manipulating the simulation itself. This can be done by kinematically controlling various links, or by building motor programs that drive the joints of an articulated linkage. Induced simulations give an animator the ability to create objective functions that the system uses to find a path that best satisfies these functions within the confines of a dynamic simulation.

In terms of satisfying the goal of granting an animator the control to portray expressive motion over a dynamic simulation, none of them completely satisfy this task within the context of commercial animation production. Interestingly, in comparison, the relatively more complex induced simulations which are very successful in producing motions that demonstrate self-actuating characters, seem less suitable as a result of their complexity; their complexity currently limits them to simple cases and hampers the ability of an animator to guide the simulations to a desired result. The kinematic clone technique would be said to use direct simulation control if one considers the literal interpretation of a kinematic armature from which the dynamic figure is suspended by springs. As described above, several components of the direct simulation techniques are used in the kinematic clone method. The uniqueness of the method presented in the following chapters is in the interface that it provides an animator.

CHAPTER 2

THE DYNAMIC SIMULATION

This chapter will present information required to understand and implement a dynamic simulation of an articulated figure. Since an articulated figure is really a collection of interconnected rigid bodies, the discussion will start with an overview of rigid body dynamics. From there, it will progress to the details of the formulations required to simulate an articulated figure and will describe in detail the formulation by Roy Featherstone [6] used for the current implementation of the kinematic clone system.

Rigid Body Dynamics

H. Goldstein defines a rigid body as "a system of mass points subject to the holonomic constraints that the distances between all pairs of points remain constant throughout the motion." [4]. He also demonstrates that these distance constraints reduce the number of degrees of freedom for a rigid body to six. The following overview of rigid body dynamics comes from [23]. These six degrees of freedom include three for *position* and three for *orientation*. These degrees of freedom are generally assumed to represent three Cartesian coordinates and three Euler angles, respectively.

For a calculation of rigid body dynamics, we must be able to find the rates of change of the position and orientation. Classical treatment (in contrast to Featherstone's spatial algebra below) considers the linear and angular velocity about a bodies *center of mass*. Assuming contact free motion, the axis of rotation about which an object rotates will always pass through the center of mass [4]. Thus, the center of mass is always unaffected by the change in orientation. This allows the rate of change of the position

and orientation to be considered separately which is not true of other parts of a body. Given this, the *linear velocity* of a rigid body is expressed as a function of the position:

$$v(t) = \dot{x}(t) \tag{6}$$

The *angular velocity* is expressed as a vector whose direction is the axis of rotation of the rigid body and whose magnitude is its rate of change. Unfortunately, the parameters used to express orientation cannot generally be differentiated directly to get the angular velocity. In fact, the number of parameters used to define orientation is not necessarily three. Quaternions and rotation matrices require four and nine parameters respectively. Quaternions are attractive for a number of reasons which are beyond the scope of this thesis, and have in fact been used in the implementation of the kinematic clone system. If ω represents the current angular velocity, the corresponding rate of change of the quaternion representing a body's orientation is:

$$\dot{q}(t) = \frac{1}{2}\,\omega(t)q(t) \tag{7}$$

where $\omega(t)q(t)$ is shorthand for the multiplication between two quaternions $[0,\omega(t)]$ and $q(t)$. The multiplication between two quaternions $[s_1,v_1]$ and $[s_2,v_2]$ is:

$$[s_1s_2 - v_1 \cdot v_2,\ s_1v_2 + s_2v_1 + v_1 \times v_2] \tag{8}$$

Changes in velocity (both linear and angular) are caused by forces. Forces such as gravity, spring connections, and contacts all cause the body to accelerate as illustrated in Newton's famous formula *F=ma*. When dealing with rigid bodies, the nature of the reaction depends upon the location of the force's application on the body. Forces actually impart *torques* on a body where the value of the torque is:

(9) $$\tau = \vec{r} \times F$$

$\vec{r}$ is the vector from the origin to the point at which F is applied

The final component required to build the rigid body equations of motion is momentum. Linear momentum is a simple function of mass and velocity:

(10) $$p = mv \quad or \quad v = \frac{p}{m}$$

It can also be shown that:

(11) $$\dot{p} = F$$

where *F* is the total force acting on a body. *Angular momentum* does not share the same self-explanatory construction as does linear velocity. In order to map angular velocity to angular momentum, a structure called an *inertia tensor* is used. An inertia tensor is a symmetric tensor of the third rank [2] and has the form:

(12) $$I = \int \rho \, (\vec{R}^2 \vec{U} - \vec{R}\vec{R}) \, dv$$

ρ is the density of the rigid body
$\vec{R}$ vector from origin to volume element dv
$\vec{U}$ unit dyadic
$\vec{R}\vec{R}$ dyad product

which expressed in 3x3 matrix form is:

$$I = \begin{bmatrix} \int \rho (y^2 + z^2) dv & \int - \rho xy \, dv & \int - \rho xz \, dv \\ \int - \rho xy \, dv & \int \rho (x^2 + z^2) dv & \int - \rho yz \, dv \\ \int - \rho xz \, dv & \int - \rho yz \, dv & \int \rho (x^2 + y^2) dv \end{bmatrix}$$

Fortunately, the rate of change of the angular momentum is much simpler:

$$\dot{L} = \tau \tag{13}$$

where L is the angular momentum and τ is the total torque being applied to the body.

Now that all of the components have been defined, the complete set of rigid body equations of motion can be presented. Given a state vector Y(t) whose components are:

$$Y(t) = \begin{bmatrix} x(t) \\ q(t) \\ \rho(t) \\ L(t) \end{bmatrix} \tag{14}$$

compute the following by:

$$v(t) = \frac{\rho(t)}{m} \tag{15}$$

$$\omega(t) = I(t)^{-1} L(t) \tag{16}$$

and finally find the derivative of Y(t):

$$\frac{d}{dt}\,Y(t) = \frac{d}{dt}\begin{bmatrix} x(t) \\ q(t) \\ p(t) \\ L(t) \end{bmatrix} = \begin{bmatrix} v(t) \\ \frac{1}{2}\omega(t)q(t) \\ F(t) \\ \tau(t) \end{bmatrix} \tag{17}$$

Articulated Body Dynamics

As mentioned previously, there are optimizations that can be made when dealing with articulated figures over general rigid bodies. With the goals of a fast and numerically stable dynamic simulation, the optimization method used must be chosen carefully. The kinematic clone implementation employs Roy Featherstone's Articulated Body Method (ABM) [8] which is an iterative method that utilizes spatial algebra to lower the number of equations required to specify motion. Featherstone's method has been cited in other works [13] for its efficiency. Dynamics algorithms fall into two categories: 1) ones that solve a set of simultaneous equations, and 2) ones that use recursive formulas. The simultaneous equation solvers (of which the Walker-Orin is the most efficient [23]) typically have $O(n^3)$ time complexity or worse, but can actually be efficient for small values of n where n is the number of links in a figure. Featherstone states that the Walker-Orin method is actually more efficient than his ABM for values of $n<9$. This efficiency for small systems explains why the simultaneous equation algorithms are so popular for robotics systems. For the purposes of an articulated figure in animation, however, one often desires figures with arbitrarily large numbers of links which justifies the use of a recursive algorithm like the ABM.

Spatial Algebra

In his work, Featherstone developed specialized mathematics called *spatial algebra* which is based on screw calculus. The sole purpose of spatial algebra is to allow for the simultaneous handling of the linear and angular components of the rigid body equations (see previous section). This simplifies both the number of equations required and the amount of code required to implement it. The foundation of spatial algebra is the *spatial vector* which is a six element vector that encodes the linear and angular components of such quantities as rigid body velocity and acceleration. The advantages gained are primarily of an algebraic nature, but such gains follow through into simplicities in the coded implementations of the formulas.

Spatial algebra makes use of the fact that a rigid body, in a mathematical sense, need not have any physically defined boundaries. The motion of a rigid body can actually be described about any arbitrary point in space. That point need not be fixed to nor even be a part of the body itself. Traditional schemes describe rigid body motion the combined effect of a linear and an angular velocity component about a body's center of mass. This practice is used for the very purpose of simplifying the separation of the two components. Featherstone's spatial algebra shows that such measures are unnecessary and actual lead to more overhead. Spatial vectors reformat the linear and angular components that would have been expressed about the center of mass to the corresponding representation about the origin which is fixed in space. Angular velocity is constant for all points about a body, so is untouched by this change in basis. The linear component is transformed by the following relation:

$$\hat{\mathbf{f}} = \dot{\mathbf{r}} + \mathbf{r} \times \omega \tag{18}$$

The symbol "^" is used to denote spatial quantities. Using this methodology, *spatial velocity* and *spatial acceleration* are defined as:

(19) $$\hat{v} = \begin{bmatrix} \omega \\ \dot{\mathbf{r}} + \mathbf{r} \times \omega \end{bmatrix}$$

(20) $$\hat{a} = \begin{bmatrix} \dot{\omega} \\ \ddot{\mathbf{r}} + \dot{\mathbf{r}} \times \omega + \mathbf{r} \times \dot{\omega} \end{bmatrix}$$

Spatial rigid body inertia is defined as:

(21) $$\hat{I} = \begin{bmatrix} m\,\vec{\mathbf{r}}\times & m\,1 \\ I^* + \vec{\mathbf{r}}\times m\,(-\vec{\mathbf{r}})\times & \vec{\mathbf{r}}\times m \end{bmatrix}$$

I^* is regular rigid body inertia about center of mass

$\vec{\mathbf{r}}$ is vector from origin to center of mass

$a\times$ is cross operator defined as:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

Another important construction used in the following sections is a *vector sub-space*. This represents a mapping from a sub-space with less or equal dimensions to another with more or equal dimensions. In Featherstone's equations, sub-space matrices are used to map joint parameters to spatial vectors. The generality of a sub-space matrix allows any type of joint to be used without a need to reformulate the equations.

Finally, there are two important spatial functions that must be defined. A *spatial transpose*, denoted a^S is defined as:

(22) $$\hat{a}^S = \begin{bmatrix} a \\ b \end{bmatrix}^S = [b^T a^T] \qquad \text{and,}$$

$$\hat{A}^S = \begin{bmatrix} A & B \\ C & D \end{bmatrix}^S = \begin{bmatrix} D^T & B^T \\ C^T & A^T \end{bmatrix}$$

The other is the *spatial cross operator*:

$$\begin{bmatrix} a \\ b \end{bmatrix}\times = \begin{bmatrix} a\times & 0 \\ b\times & a\times \end{bmatrix} \tag{23}$$

The Articulated Body Method

To establish a recurrence relation, Featherstone develops a quantity known as an *Articulated Body Inertia* (ABI). ABI is the apparent instantaneous inertia of a body comprised of a set of flexible joints. Once the ABI of an articulated figure is known, the entire figure can be treated as a single rigid body with a rigid body inertia equal to the ABI and with a fixed point rotation about the first joint of the figure. Assuming that an articulated figure has the hierarchical structure of a tree (i.e. no loops), we can recursively define any branch or sub-branch as a single rigid body with an ABI built from all of its constituent links. Put simply, the Articulated Body Method finds the ABI of each branch and sub-branch of an articulated figure and uses these to propagate accelerations from the base out to all links in the figure.

Step 1: Finding Articulated Body Inertias

The process of finding all of the ABIs for the various sections and subsections requires a recursive step starting from the root node a propagating out the leaves. The purpose of this step is to propagate the velocity of the base along with the cumulative velocities of each of the joints outwards in order to calculate the momentum of the links. Momentum will be propagated back from the leaves so it is efficient to calculate it at the same time as the ABIs. This equation for the velocity of a link *i* (expressed in spatial coordinates) is:

$$\hat{v}_i = \hat{v}_{i-1} + \hat{s}_i \dot{q}_i \tag{24}$$

$\hat{s}_i$ is the subspace matrix of joint i

The ABIs are propagated back up the tree of an articulated figure starting from the leaf nodes. The ABI of a leaf node is simply the rigid body inertia of that link. From there, the following equations are employed for the ABI and for the momenta:

$$\hat{I}^A_i = \hat{I}_i + \hat{I}^A_{i+1} - \frac{\hat{I}^A_{i+1}\,\hat{s}_{i+1}\,\hat{s}^S_{i+1}\,\hat{I}^A_{i+1}}{\hat{s}^S_{i+1}\,\hat{I}^A_{i+1}\,\hat{s}_{i+1}} \tag{25}$$

$$\hat{p}_i = \hat{v}_i \times^* \hat{I}_i \hat{v}_i + \hat{p}_{i+1} + \hat{I}^A_{i+1}\,\hat{v}_{i+1} \times^* \hat{s}_{i+1}\,\dot{q}_{i+1} + \frac{\hat{I}^A_{i+1}\,\hat{s}_{i+1}\,(Q_{i+1} - \hat{s}^S_{i+1}\,(\hat{I}^A_{i+1}\,\hat{v}_{i+1} \times^* \hat{s}_{i+1}\,\dot{q}_{i+1} + \hat{p}_{i+1}))}{\hat{s}^S_{i+1}\,\hat{I}^A_{i+1}\,\hat{s}_{i+1}} \tag{26}$$

where Q_{i+1} is the active joint forces of the next link.

Step 2: Propagating Accelerations

Once the ABIs and momenta are known, the spatial accelerations and resultant joint accelerations can be found. This last step propagates back out from the base to the leaves. These equations are respectively:

$$\hat{a}_i = \hat{a}_{i-1} + \hat{v}_i \times^* \hat{s}_i\,\dot{q}_i + \hat{s}_i\,\ddot{q}_i \tag{27}$$

$$\ddot{q}_i = \frac{Q_i - \hat{s}^S_i\,(\hat{I}^A_i\,(\hat{a}_{i-1} + \hat{v}_i \times^* \hat{s}_i\,\dot{q}_i) + \hat{p}_i)}{\hat{s}^S_i\,\hat{I}^A_i\,\hat{s}_i} \tag{28}$$

Implementation Details

The current implementation uses all spherical joints. Spherical joints were desired in this case to allow for maximum experimentation with the effects of spring forces on the dynamic simulation. The use of sub-matrices and Featherstone's ABM makes the coding of spherical joints more efficient than others algorithms that require them to be modeled as a collection of single degree of freedom joints. Numerical singularity problems typically associated with spherical joints are resolved by using quaternions to represent joint positions while using joint angular velocities directly in the dynamics equations. This yields a sub-space matrix of:

$$\hat{S} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{29}$$

which is clearly simple to code for. The relationship between angular velocity and the change in the positional quaternion is determined using (7). Fortunately, the animation system within which the dynamics simulation already stores angular positions with quaternions which minimizes representation conversions. Also, since these values are already stored in link coordinates, Featherstone's equations are also carried out there. Featherstone claims that calculations using a link by link coordinate system (as opposed to representing everything in world coordinates) can yield a 10% improvement in performance.

Numerical Integrations

Finally, to numerically integrate the dynamics equations, a fifth order Runge-Kutta method is used with adaptive step-size. For the sake of completeness, a Euler-step method was attempted but proved to be very unstable. The particular Runge-Kutta method used is an embedded Runge-Kutta formula originally invented by Fehlberg [24], which has error estimation built into the function evaluation. The formulation is fifth-order with error estimation that bounds the fourth order calculations. The extra accuracy makes up for its cost by allowing larger step sizes. Its stability has also held up throughout the testing of the simulations; no simulations had to be rerun on account of numerical inaccuracies.

CHAPTER 3

KINEMATIC CLONES

This chapter presents the detailed workings of the *kinematic clone* technique. The goal of this thesis as described in the introduction is to provide animators adequate control over a dynamic simulation to reap the realism of a simulation while retaining the ability to create expressive motion. It was argued in chapter 1 that such control should be kinematic in nature since the alternatives thus far proposed, while successful in several specific areas, do not yet suitable for the purpose of portraying personality and emotion. A *kinematic clone,* as the name implies is an identical copy, link for link, of an articulated figure that is accessible to an animator for the purposes of kinematic control. It is controlled using traditional keyframe methods which accommodates time-saving enhancements such as inverse kinematics. The following sections will describe the connection between the kinematic clone and the articulated figure and will also discuss the important elements as they have been implemented in the first version of the system.

Kinematic Clones: Technical Description

From a high level, a kinematic clone is a wire frame skeletal clone of the articulated figure that imparts control over a dynamically simulated articulated figure by forces generated from virtual springs connecting the two (see figure 1). There is a one-to-one correspondence between the links and joints of the kinematic clone and the links and joints of the articulated figure. This correspondence, and the formalization of the two structures distinguishes this method from other kinematic-spring systems that have appeared in other works (e.g. [9][13]).

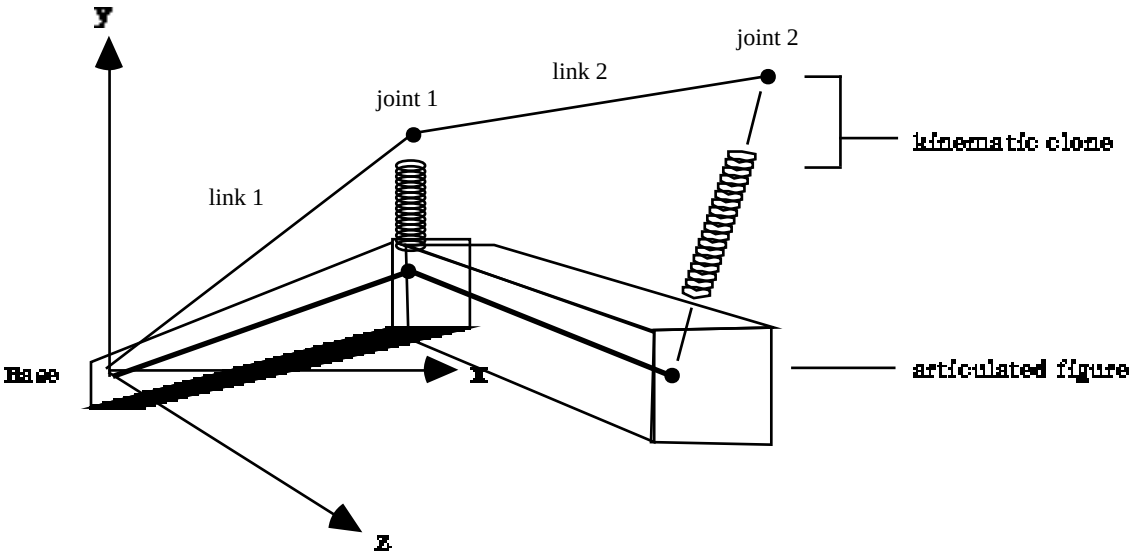

figure 1

The structural correspondence makes the relationship between control and reaction very intuitive. The general behavior of an object suspended by a spring from another is well understood and should be a part of a person's typical world experience, especially for someone who's vocation is animation. This alone is important from a usability standpoint since it would enable an animator to predict the outcome, to some degree, of moving the kinematic clone through a series of keyframes.

In addition to the setup of keyframes for the kinematic clone, an animator has the ability to adjust several other parameters that effect the degree to which the articulated figure is effected by the kinematic control. The most important of these are the spring constants which determine the relative freedom that a dynamic joint has to move in relation to its kinematic counterpart. The following sections will describe each of these parameters in turn.

Spring Connections

The connection between the kinematic clone and the articulated figure is through spring connections that connect the corresponding joints of the two structures. For control purposes, the components of the spring force between a pair of corresponding joints are broken out along the coordinate axes of the dynamically controlled link on the posterior side of the joint. This allows an animator to optionally select different spring coefficients along certain axes, to impart directional favoritism for example. Such a setup can be used to simulate a more realistic "one degree of freedom" joint that still has some give in the restricted directions. The rest lengths of all of the springs is zero which follows the notion that link and joint displacements should only exist when there exists a physical force to move them so.

The springs in the simulation can be either simple linear springs (Hooke's Law) or can be made exponential to achieve certain effects. The general force equation is:

(30) $$\partial = \beta\,(kd^{\alpha})m$$

k	is the Hooke's law spring constant
d	is the linear joint offsets
m	is the mass of the link

The two constants α and β determine the exponential behavior of the springs. α is the spring's exponent and determines how steeply the force rises with respect to distance. Exponential springs also exhibit reduced strength (i.e. looseness) when the distance approaches zero. β determines the relative width of this "valley." The use of exponential springs can be desirable over stiff linear springs from a numerical stability standpoint.

There were two options of how to apply the forces to the articulated figure; (1) apply an external linear force to the joint in question, or (2) apply a torque to the interior joint. For reasons of convenience of implementation, the second option was chosen. The torque resulting from a linear spring acting on the link between a parent and its child is:

(31) $$\tau = r \infty \beta (kd^{\alpha})m_p$$

r	is the vector to child link in parent coords.
m_p	mass of parent link

At the time of implementation, applying a torque to a link's parent seemed to be efficient because it reduced the number of joint space transformations slightly. One limitation of this method is that the base joint of the figure cannot move with respect to the kinematic location of the base. This modification is recommended for future enhancements in Chapter 5.

Some special consideration must be given to the end-effectors of a figure. The orientation of a joint is actually a by-product of the position matching forces of the joint-to-joint connection springs. Since there is no joint to connect at the end of an end effector, it falls limply unless extra forces are introduced. To combat this, an additional virtual spring connects the geometric centers of each end-effector. It has been found desirable to treat this spring separately from the spring connecting the anterior joints. From an animation standpoint, it is often the case that the "wrist" of a character must be more carefully controlled than the exact orientation of the hand. Slightly freer hand movements seem to accentuate a gesture and add extra believability to a motion.

There are other choices for joint connections beyond those that have been implemented. Experiments have suggested that torsion springs whose force would be dependent on the angular differences between a kinematic clone and its articulated figure would be beneficial. During periods of extreme rotation, the dynamic simulation must rely on positional restoring forces to rotate a body. When these rotations are coupled with linear accelerations of the body as a whole, equilibriums can occur that prevent the figure from rotating into position. While physically correct, this behavior can sometimes result in awkward looking movements and postures. This is another topic that is discussed in chapter 5.

Damping Forces

Given spring forces alone, an articulated figure in this system will exhibit oscillations and vibrations that are not characteristic of a creature under autonomous power. Equally as important to the simulation is motion damping. Sufficient damping is required to prevent excessive oscillations and to eliminate the "marionette effect." One of the concerns during the research and planning of this thesis was that characters under this kind of control would move like marionettes suspended by springs, which given the nature of the setup is very close to what is really happening. Marionette motion is characterized by excessive follow-through in the links and a limp feel which denies self motivated actions. Fortunately, as chapter 4 will discuss, applying the correct amount of *damping force* to the movement of the articulated figures prevents this from happening. Sufficient damping makes the resultant movements seem more deliberate. Even under periods of extreme force, the damping effect of resisting joint angle changes makes a character feel more in control of its actions. This was a very welcome discovery.

As with the spring constants, an animator can supply varying degrees of damping to the various degrees of freedom. They can also be made to be linear or exponential. It was found that global viscous damping was all that was required to achieve the desired effect. The formula for the damping force of a link is:

(32) $\tau = - \omega p m$

p is the damping coefficient
ω is the angular velocity of the link
m is the mass of the link

The mass factor in the equation, while not physically correct (it should be dependent on surface area), seems to work for the purposes here. Damping is also important from a

numerical standpoint to guard against numerical instability. Damping forces act in direct opposition to joint velocities which helps to curtail their blowing up due to numerical errors.

Animating Parameters

One very important feature is that both the spring constants and damping coefficients can be animated, i.e. changed as a function of time. Overall muscular exertion varies depending on the attitude of a character. At one time during an animation, a character may be lethargic which is generally depicted with limp, slumped-over movements. During this stage, both the spring and the damping coefficients would be drastically lowered, approaching the unchallenged swing of an unpowered limb. However, if that same character became suddenly exited, the spring and damping forces would be set to a much higher value to produce the quick and snappy movements produced by a character in that frame of mind. One of the examples in chapter 4 displays this use of animated values.

Another important use of animating spring constants is to allow an animation to be tweaked and adjusted to modify the resulting motion. Such action would be an alternative to modifying keyframes when an animator desires to add some extra follow-through to a turn, for example.

Spring Force Release

During testing, it was often found desirable to lower spring coefficients at the end of a rapid movement to exaggerate the reaction. In an attempt to automate this process, a threshold was introduced which is used to compare with the magnitude of the spring forces and reduce the spring coefficients when its value was surpassed. The spring coefficients are reduced by the following formula:

$$k' = k \left(\frac{f - \alpha}{\beta - \alpha} \right)^{\gamma} \quad (33)$$

k	normal coefficient value
∂	current spring force
α	force threshold
β	maximum force allowed (k=0)
γ	exponent of decay

Exponential decays have been found to behave more naturally since they "ease-in" to the fall-off function making the discontinuity less obvious.

External Forces

Very few external forces have been implemented in the current system. Gravity, is implemented by introducing a fictitious acceleration in the *opposite* direction of the true gravitational force to the base of the articulated figure. The only other external force is a ground reaction force that helps keep legs from passing through the ground. This is implemented through an exponential spring force that is based on the amount of penetration into the ground. Interesting results could come from the addition of wind forces, force-fields, etc.

Kinematic Links

In practice, there are generally situations where an animator would like to limit the effects of the dynamics simulation to a certain subset of the entire figure. An animator may require precise control of a limb for some action, or he or she may be animating a creature with a huge number of articulations. Even though Featherstone's Articulated Body Method is O(n) [6], the computational costs can become quite large for a figure with many joints. For these reasons, an animator has the option to make certain links, or certain subsets of links completely kinematic. The effect of this action on the position of the link of the articulated figure depends on where that link is in the structure

and whether there are any dynamic links in the path from it to the base. For links with no dynamically controlled links interior to it, the position of the link will exactly match the position of the corresponding link in the kinematic clone. If dynamic links exist on the path, however, the positions of the links will not, in general, coincide. This is because the parameters under kinematic control relate the position of a link with respect to its parent. Controlling these parameters kinematically does insure, however, that the relationship between a joint and its parent are exactly positioned.

Kinematic links require significantly less calculation in the context of the dynamics equations (see chapter 2). Their accelerations and Articulated Body Inertias are still calculated passed into the simulation so that the effects of their motions are reflected in the dynamic links. Actually, in the case where no dynamic link exists between a link and the base, even the ABI can be ignored. ABI's are passed "up" the tree structure of an articulated figure during the dynamics calculations. If all of the ancestors of a link are kinematically controlled, the ABI is not required.

Kinematic control can be very useful if an animator wishes to enhance *motion-capture* data. Motion-capture is a process in which a real figure is fitted with sensors that detect and record the motion of key parts of the body. This data is used to control an articulated figure representing some 3D character. If an animator wishes to add additional limbs or a tail, these must be animated by hand. An alternative would be to use the kinematic clone technique on the additional limbs while keeping the joints controlled by motion capture under kinematic control.

Interface Design

To enable positioning of the kinematic clone, it must be displayed in some fashion along with the dynamic figure. In the current implementation it is displayed as a linear skeleton connecting the joints. To aid visualization prior to the execution of the dynamic simulation, rotations and translations of links within the kinematic skeleton are also

applied to the dynamic figure interactively. Prior to the execution of the simulation, the kinematic and dynamic figures are coincident, so like rotations will maintain this coincident positioning. After a simulation is run and joints are offset, as the animator repositions a keyframe, the dynamic link also moves. By applying identical transformations, a crude estimate of the effects on the dynamic simulation as a result of moving the keyframe are visualized.

CHAPTER 4

RESULTS

The kinematic clone system has proven to be very successful at generating realistic motion that approximates the kinematic motion designed by an animator. Movements automatically exhibit mass and momentum, especially around rapid movements and quick changes in velocity. The motion also exhibits subtle reactionary motions that are difficult to animate directly and that help create the realistic feel, by eliminating "frozen poses" and creating what Disney animators call a "moving hold" [1]. This chapter describes the results of a set of animation examples that were conducted with the kinematic clone system. The last section describes the system that it was implemented upon and the resulting calculation times.

Chain Link Tests

The first experiments were done with a simple object comprised of a set of three links connected in a chain. Tests included drooping the chain from a stationary kinematic clone with varying spring strengths. Another test included varying the spring strength over time to simulate a relaxation of a character. Two final tests included animated articulation of the kinematic clone and animated motion of the base of the clone to inspect how the clone interpolates an animator's motion.

The tests showed expected performance. With the articulating chain, it was found that a very low number of keyframes were able to guide very natural and organic sweep of the chains. One important find was noted in that the motions of the articulated figure

can have a different timing feel. This must be addressed in situations where precise choreography of the figure is required. This topic is discussed in Chapter 5.

Ice Skater

This example demonstrates the effect of rapid body movements on the dynamic simulation. It also demonstrates the need for joint constraints that is discussed in chapter 5. In this example, the legs and hips are actually controlled kinematically. This was done to avoid problems with ground collisions as the legs move rapidly back and forth. The animation depicts an ice skate making the final few steps of a "speed skate" and then coming towards camera. His arms are rocking back an forth speed-skater style. The arms tend to lag behind the kinematic model considerably. If this were a problem in the animation, the animator could increase the spring coefficients or start the kinematic arm swings earlier. The other more major problem occurs when the arms swing backwards. The spring forces alone are not enough to prevent the elbow from bending backwards somewhat. It seems that some degree of joint constraint checking will have to be added to prevent this type of occurrence. Fortunately, the violation is not that extreme suggesting that the computational cost of joint constraints is probably not worth the expense for slow moving animations.

Mannequin Jump

This proved to be a deceptively simple yet remarkably complex animation of a mannequin jumping into the air and landing a few feet forward. With only seven keyframes, a complex motion is generated that clearly demonstrates follow-through and momentum. The initial crouch is accentuated with a slight bob of the head and drop of the arms. As the mannequin leaps, the legs of the kinematic clone crouch causing the dynamic legs to snap up. Upon landing, ground reaction forces cause the legs to bend and the arms head and torso lean forward. The additional work required to animate the

follow throughs using traditional keyframe would more than double the number required here.

Walk Cycle

This test demonstrated the need for joint limits even more so than the skater example. When the arms swing backwards, the spring forces cause the elbow to bend backwards. What is worse, is that it stays in this position as the arm hangs momentarily before the return swing. The walk cycle also proved the advantage of exponential springs. With the rapid leg and foot swings, linear springs lagged so far behind that they had not yet caught up when the leg or arm swung back to meet them. This killed the feel of a walking character. Simply by setting the spring exponent to two eliminated this problem and produced the cycle displayed in the demonstration video.

System Performance

The kinematic clone system was implemented on a Commodore Amiga 3000 with a 16MHz 68030 microprocessor. This system includes a 68881 math co-processor for doing double-precision floating point operations.

Calculation times on this system were longer than expected.

Animation	***Avg. Frame CPU Sec***	***Avg. # Steps***
drop: stiffness 1	3.44	8.28
drop: stiffness 90	19.32	45.97
chain swing	9.79	31.83
skater	110.79	65.08
jumper	120.94	66.75
walker	149.04	65.31

Table 1

The dynamics algorithm is similar to those that have been used to produce near real-time dynamic simulations [9]. In light of this fact, an implementation is planned for a Silicon Graphics Indigo II Extreme with a 150MHz R4400 Risc processor to gauge the technique's performance at the workstation level. Ideally, simulations should take no more than a few minutes to be practical. This would allow it to be used along side of linear and splinar keyframe interpolation methods in an animation system.

CHAPTER 5

FUTURE WORK

The kinematic clone system presented herein successfully met the objectives as described in the introduction. During the course of its implementation, however, and during the execution and analysis of the results from the examples presented in chapter 4, several areas were identified that merited future research but that could not be incorporated at this time. This final chapter presents each topic in turn, presenting the issue and gives consideration to possible solutions/implementations.

Alternative Spring Configurations

As was suggested in chapter 4, there may be alternative and/or additional spring configurations that could produce desirable effects. The first configuration to consider are *torsion springs*. The spring forces included in the current implementation are proportional to the distances between the joints of the kinematic clone and the articulated figure. Given sufficient spring coefficients at each of the joints, this configuration will drive an articulated figure to match the kinematic clone. Any torques that are required to rotate sections of the figure are the result of these linear spring forces acting on a parent link. Unfortunately, the path taken as a result of these forces can twist a figure abnormally. One solution might be to supplement the positional springs with torsion springs that directly impart torques that cause a link to match the *orientation* of its kinematic counterpart. Such springs would have the additional benefit of eliminating the need for extra positional springs at the end-effectors.

In addition to torsion springs, the concept of a *body rest position* might be useful. Additional springs would be added which relate the joints of the dynamic model to

themselves. Currently, as the joint-to-joint spring forces are relaxed, the dynamic figure starts to droop like an unpowered rigid body. Worth investigating is whether intra-joint forces which cause links to favor a certain orientation would benefit the simulation in this case. One would have to investigate whether such forces would help or hamper an animator's intuitive ability to judge the effects of changing spring coefficients.

Joint Constraints

A few of the examples in chapter 4 demonstrated the need for joint constraints in the dynamic simulation. Roy Featherstone's Articulated Body Method [6] accommodates the limiting of degrees of freedom in a joint. The current implementation of the kinematic clone method uses spherical joints exclusively, primarily for the purposes of testing the kinematic control techniques. It would not be too difficult to impart more of the ABM's generality into the dynamics simulation allowing for varying degrees of freedom at the joints.

Joint constraints, however, includes more than just limits on the number of degrees of freedom. Within a degree of freedom, the range of angles attainable by a joint may be restricted. Featherstone talks briefly about some techniques for incorporating this type of constraint into his algorithms. It would be interesting to see what effects torsion springs have on the need for such constraints. Another option to investigate would be the use of resistant forces to handle constraints in the fashion described in [8] and [9]. Such approaches have traditionally suffered from numerical problems, but might be worth considering here since the violations in the current system were not that bad.

"Floating Bases"

"Floating Bases" refer to the base of an articulated figure when it is not fastened to the ground or some other object. In the kinematic clone system, the base is locked in that the dynamics equations have no effect on its position (it may be moved kinematically

by the animator, however). There are times when it might be desirable to have this base move as a result of the forces generated by the articulated figure, such as when it is in freefall. Both Featherstone [6] and Hahn [2] present methods for doing this.

Collision Detection

Currently, there is no joint-to-joint or figure-to-object collision detection. Collision with the ground is cheaply implemented with a height compare. Such tests are very expensive but might be worth considering especially when very low spring constants are used to connect the joint pairs since this can increase the probability of joints falling through one-another.

Advanced Kinematic Joint Control

Considerable work is needed to enhance the ability of mixing kinematic joints into the simulation. Currently kinematically controlled joints are controlled in link space as opposed to being positioned in world space. Disregarding the practical advantages of limiting calculations with kinematic links, the primary advantage of using them are for precise placement of certain links in world space. An animator should have the option of making a joint kinematic and have that joint exactly coincide with the corresponding joint in the kinematic clone, whether or not there are dynamic links between that link and the base. Doing so would require that the dynamic links receive forces that connect them to the position of an exterior link. The difficulty is that such forces could easily become very stiff and produce numerical instability. One option might be to use forces to get the dynamic links close to the correct position and then use inverse kinematics to place them exactly.

Another feature of kinematic link control should enable an animator to toggle between kinematic and dynamic control during the course of an animation. Such transitions are easy when going from kinematics to dynamics, but some consideration

must be given about how to handle the transition back from dynamics to kinematics. There should be some sort of blending function that would prevent links from "popping" from their positions under the dynamic simulation to the position of the link in the kinematic clone. The two positions would not generally coincide.

Finally, the issue of *timing* must be addressed. Animations are often choreographed first to get the timing of motions correct. Timing is very important in animation and lends a lot to the interpretation of the intentions and emotions of a character. Several of the examples exhibited relative large changes in the timings set down in the kinematic animation. Perhaps a blending function is needed in areas where timing is critical that interpolates between the kinematic and dynamic positions is needed. Such a function would smoothly blend from the dynamic motion to the kinematic such that the articulated figure arrives at a key frame at the time prescribed.

WORKS CITED

(In order of reference)

1. F. Thomas, and Ollie Johnston. 1981. Disney animation: the illusion of life. New York: Abbeville Press Publishers.

2. J. K. Hahn. 1988. Realistic animation of rigid bodies. Computer Graphics (SIGGRAPH '88 Proceedings). 22, no. 4 (August): 299-308.

3. D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. 1987. Elastically deformable models. Computer Graphics (SIGGRAPH '87 Proceedings). 21, no. 4 (July): 205-214.

4. H. Goldstein. 1980. Classical Mechanics. Reading, MA: Addison-Wesley Publishing Co.

5. W. Reeves. 1983. Particle systems-A technique for modeling a class of fuzzy objects. ACM Transactions On Graphics. 2, no. 2 (April): 359-376.

6. R. Featherstone. 1987. Robot Dynamics Algorithms. Boston: Kluwer Academic Publishers.

7. J. J. Craig. 1989. Introduction to Robotics Mechanics and Control. Reading, MA: Addison-Wesley Publishing Co.

8. J. Wilhelms. 1987. Using dynamic analysis for realistic animation of articulated bodies. IEEE Computer Graphics and Applications. 7, no. 6 (June): 12-27.

9. W. Armstrong, M. Green, and R. Lake. 1987. Near real-time control of human figure models. IEEE Computer Graphics and Applications. 7, no. 6 (June): 52-61.

10. P. Isaacs, and M. Cohen. 1987. Controlling dynamic simulations with kinematic constraints, behavior functions and inverse dynamics. Computer Graphics (SIGGRAPH '87 Proceedings). 21, no. 4 (July): 215-224.

11. K. Pearson. 1991. Sensory elements in pattern-generating networks. Making Them Move: Mechanics, Control and Animation of Articulated Figures. San Mateo, CA: Morgan Kaufmann Publishers. 111-127.

12. R. Brooks. 1991. A robot that walks: emergent behaviors from a carefully evolved network. Making Them Move: Mechanics, Control and Animation of Articulated Figures. San Mateo, CA: Morgan Kaufmann Publishers. 99-108.

13. M. McKenna and D. Zeltzer. 1990. Dynamic simulation of automonous legged locomotion. Computer Graphics (SIGGRAPH '90 Proceedings). 24, no. 4 (August): 29-38.

14. M. McKenna, S. Pieper, and D. Zeltzer. 1990. "Control of a virtual actor: the Roach." from the Proceedings of the 1990 Symposium on Interactive 3D Graphics (Snowbird, Utah). Computer Graphics. 24, no. 2 (****): 165-174.

15. D. Zeltzer. 1991. Task-level graphical simulation: abstraction, representation, and control. Making Them Move: Mechanics, Control, and Animation of Articulated Figures. San Mateo, CA: Morgan Kaufmann Publishers. 3-33.

16. M. Raibert, and J. Hodgins. 1991. Animation of dynamic legged locomotion. Computer Graphics (SIGGRAPH '91 Proceedings). 25, no. 4 (July): 349-358.

17. M. van de Panne, E. Fiume, and Z. Vranesic. 1990. Reusable motion synthesis using state-space controllers. Computer Graphics (SIGGRAPH '90 Proceedings). 24, no. 4 (August): 225-234.

18. M. van de Panne, and E. Fiume. 1993. Sensor actuator networks. Computer Graphics (SIGGRAPH '93 Proceedings) Annual Conference Series. (August): 335-342.

19. J. Ngo, and J. Marks. 1993. Spacetime constraints revisited. Computer Graphics (SIGGRAPH '93 Proceedings) Annual Conference Series. (August): 343-350.

20. A. Witkin, and M. Kass. 1988. Spacetime constraints. Computer Graphics (SIGGRAPH '88 Proceedings). 22, no. 4 (August): 159- 168.

21. M. Cohen. 1992. Interactive spacetime control for animation. Computer Graphics (SIGGRAPH '92 Proceedings). 26, no. 2 (July): 293-302.

22. L. Gritz and J. K. Hahn. 1995. Genetic programming for articulated figure motion. Journal of Visualization and Computer Animation. (to appear in 1995).

23. D. Baraff. 1994. Rigid body simulation. in SIGGRAPH '94 Course Notes #32 (An Introduction to Physically Based Modeling). G1-G68.

24. W. Press, S. Teukolsky, W. Vetterling, B. Flannery. 1992. Numerical Recipies in C. New York: Cambridge University Press. 710-721.